Accelerating Deep Reinforcement Learning for Autonomous Racing

Benjamin David Evans

Dissertation presented for the degree of Doctor of Philosophy at the University of Stellenbosch

Promoters: Dr H.W. Jordaan & Prof. H.A. Engelbrecht
Department of Electrical and Electronic Engineering

March 2023

Acknowledgements

Oh, the depth of the riches and wisdom and knowledge of God!
How unsearchable are his judgments and how inscrutable his ways!
"For who has known the mind of the Lord,
or who has been his counselor?"
"Or who has given a gift to him
that he might be repaid?"
For from him and through him and to him are all things.
To him be glory forever. Amen.

Romans 11:33-36

- **God:** I am thankful to God for the opportunity and ability to study for many years of my life. I am thankful for the strength that He has given me through the last few years and perseverance.

- **Magdaleen:** Thank you, Magdaleen, for standing by me and supporting and encouraging me. When I started the PhD, I did not even know you, and now you are my wife; what a wonderful journey it has been getting to know and marry you. Thank you for your love and support through it all. As Proverbs says, through finding a wife, I have truly received a blessing from the Lord.

- **Parents:** Thank you, Mum and Dad, for always telling me I could do anything. Thank you for educating me and encouraging me, and bearing with my constant taking things apart. Dankie Pappa en Mamma vir julle ondersteuning en aanmoediging om my beste te gee.

- **Supervisors:** Thank you, Willem and Herman, for your guidance through this project. Thank you for the meetings, the comments, the corrections, and the encouragement.

- **Teachers:** I am thankful for the many teachers who taught me what I know. From Mr. Markus, who taught me the algebra and calculus I depend on, to Dr. Roux, who taught me applied maths, and Dr. Owen, who cemented my understanding of differential equations.

- **Community:** I am thankful for all my friends who walked the journey with me, prayed for me and encouraged me along the way.

Stellenbosch UNIVERSITY IYUNIVESITHI UNIVERSITEIT

forward together
sonke siya phambili
saam vorentoe

Plagiarism Declaration

1. *Plagiarism is the use of ideas, material and other intellectual property of another's work and to present is as my own.*
2. *I agree that plagiarism is a punishable offence because it constitutes theft.*
3. *I also understand that direct translations are plagiarism.*
4. *Accordingly, all quotations and contributions from any source whatsoever (including the internet) have been cited fully. I understand that the reproduction of text without quotation marks (even when the source is cited) is plagiarism*
5. *I declare that the work contained in this assignment, except where otherwise stated, is my original work and that I have not previously (in its entirety or in part) submitted it for grading in this module/assignment or another module/assignment.*

	Signature:
Initials and surname: B.D. Evans	*Date:* 1st of November 2022

Abstract

The F1/10$^{\text{th}}$ racing problem is to use the onboard LiDAR scan to calculate speed and steering references to move a 1/10th scale car around the track as quickly as possible. While planning has typically used perception, planning and control pipelines, recently, deep reinforcement learning (DRL) has grown in popularity due to its advantages of not requiring explicit state representation and environmental flexibility. Current approaches have suffered from poor performance at low speeds, safety concerns exacerbated by sim-to-real transfer, and few approaches have considered obstacle avoidance.

The first contribution of this work is the development of high-speed learning formulations for autonomous racing. A comprehensive evaluation of previous approaches concludes that current learning formulations train agents to select infeasible speed profiles, resulting in the agents being unable to race using the vehicle's full speed profile. This problem is overcome by using analytical vehicle models to develop learning formulations for improved speed selection. The performance evaluation shows that the novel formulations enable the vehicle to learn a feasible speed profile using the vehicle's full speed range and achieve lower lap times than previous methods in the literature. This result indicates that using vehicle models improves high-performance racing behaviour.

The second contribution of this work is to enable online learning by using a supervisory safety system (SSS). A safety system is designed that uses viability theory to ensure vehicle safety, irrespective of the planner used. The SSS is incorporated into the learning formulation and used to train DRL agents to race without them ever crashing. The novel learning formulation is extensively evaluated in simulation, demonstrating that online training can train agents to race without ever crashing, achieve a $10\times$ improvement in sample efficiency and that the trained agents select conservative speed profiles. The proposed method is validated at constant speed on a physical vehicle, demonstrating that an agent can be trained from random to drive around a track without ever crashing.

The final contribution of this work is to explore how DRL agents can be used to expand the ability of current classical planners to avoid unmapped obstacles. Three hybrid architectures that combine classical and learning components are presented and evaluated. The modification planner, which combines a path follower and DRL agent in parallel, demonstrates the ability to track a reference path while avoiding unmapped obstacles. The results indicate that combining classical and DRL components can improve the performance of DRL agents while enabling classical solutions to avoid obstacles.

Lay Summary

F1/10$^{\text{th}}$ autonomous racing has been approached using classical methods that use vehicle models to generate and follow a plan, resulting in high-performance racing. While deep reinforcement learning (DRL) agents, which learn from experience, have been used for racing, approaches have suffered from poor performance and safety concerns due to training the agent in a simulator before transferring it to a physical vehicle. This work combines classical vehicle models and machine learning techniques to accelerate DRL methods for autonomous racing by making the three contributions of, (1) novel learning formulations resulting in feasible, high-speed racing behaviour, (2) a method for safely training agents to race onboard physical vehicles, with them never crashing, and (3) investigating hybrid architectures for obstacle avoidance. The novel learning formulations enable the agent to learn an appropriate speed profile of slowing down and speeding up, resulting in lower lap times, using higher top speeds than in previous work. The online learning method can train agents to race without ever crashing, which enables agents to be trained onboard physical vehicles, resulting in better driving performance. Finally, the hybrid architectures for obstacle avoidance demonstrate that DRL agents can expand the capability of classical planners to avoid obstacles.

Contents

Acronyms

Cth	cross-track and heading error.
DNN	deep-neural network.
DQN	deep-Q network.
DRL	deep reinforcement learning.
E2e	end-to-end.
EIL	expert intervention learning.
GTS	Grand Turismo Sport.
IAC	Indy Autonomous Challenge.
IL	imitation learning.
ML	machine learning.
Mod	modification.
MPC	model predictive control.
PP	pure pursuit.
PPO	proximal policy optimisation.
PPPS	pure pursuit policy search.
RL	reinforcement learning.
ROS	robot-operating-system.
SAC	soft-actor-critic.
SSS	supervisory safety system.
Std	standard.

Chapter 1

Introduction

1.1. Background

Autonomous robots are gaining popularity due to their advantages; self-driving cars could promise to reduce road fatalities while not requiring humans to drive, factory robots can improve efficiency, and mining robots can perform dangerous tasks. Robotic systems comprise of computer programs (software) that are used to control mechanical systems (hardware). For robotic systems to be beneficial, they must be able to perform complex tasks, operate at a high level of performance, guarantee safety, and be robust to environmental uncertainty [1].

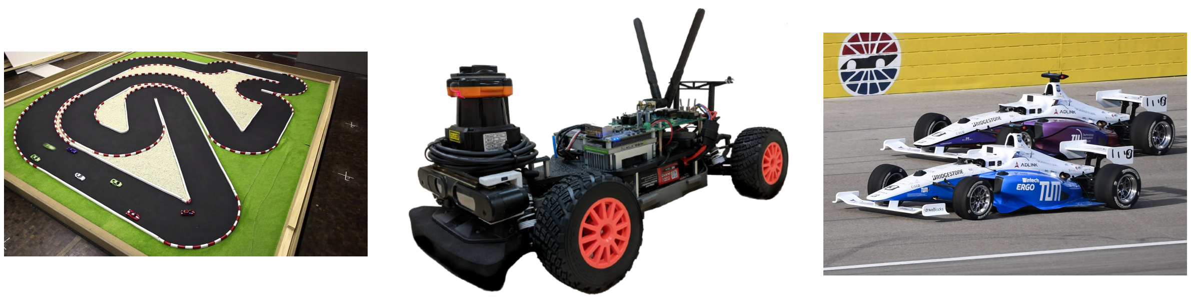


Figure 1.1: Left: 1:43 scale autonomous racing experiment [2]. Middle: an F1/10th racing car (f1tenth.org). Right: an autonomous race car at the Indy Autonomous Challenge (IAC), from Getty Images.

Racing cars are a popular testbed for high-performance autonomous algorithms due to the high-performance nature of the competition, and easy-to-measure performance metrics, such as lap time and completion rate [3]. Scaled autonomous racing, such as 1:10 [4], 1:18 [5], 1:20 [6], 1:43 [2], provides a simple, safe platform for the development of autonomous racing planners [7]. F1/10th racing cars, shown in Figure 1.1 (middle), are ideal for high-performance control research due to fixed hardware requirements[1], well-developed-simulators [4, 8], and extensive comparative research [9].

[1]f1tenth.org/build

1.1.1. Problem Identification

The aim of racing is simple; move the vehicle around the track as quickly as possible. Autonomous racing planners must use onboard sensor measurements to generate steering and speed references that operate the vehicle on the edge of their performance limits to move around the track in the minimum time. In F1/10th racing, the primary input sensor is a light detection and ranging (LiDAR) scanner with 1080 range finder beams, giving the planner a two-dimensional scan of the surrounding environment. Figure 1.2 illustrates the planning problem of using a LiDAR scan of the track to select speed and steering references for the vehicle to follow. The image shows the planning loop of selecting commands, that move the vehicle, which leads to new sensor readings for the planner.

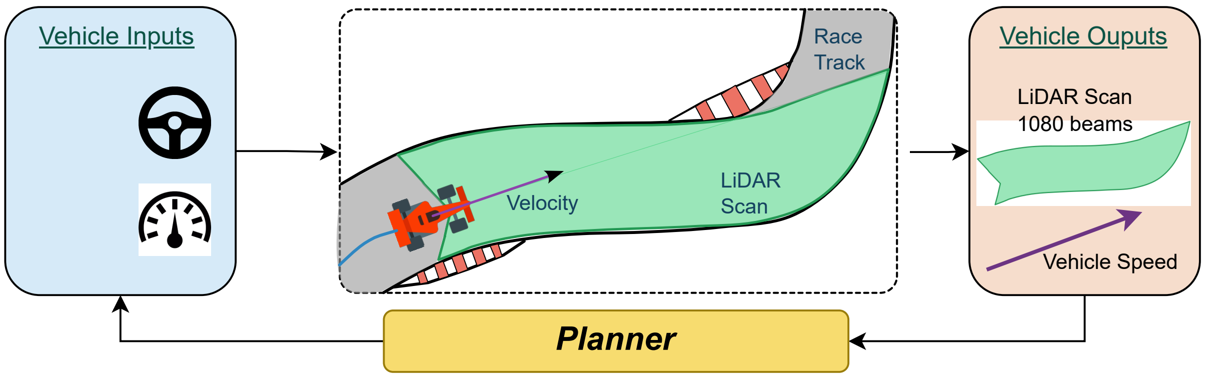


Figure 1.2: The racing problem is to design a planner that uses the LiDAR scan and speed to determine speed and steering references to move the vehicle around the track.

The two central challenges to racing are (1) performance, operating the vehicle at the limits of handling, and (2) safety, ensuring that the vehicle does not crash. An inherent tension exists between high-performance and safe systems. Inherent to high-performance racing is operating the vehicle at the physical limits, which is the boundary of safety. Conversely, the safest behaviour is to go slowly, which is a terrible racing strategy. The challenge of racing is to drive as fast as possible while ensuring that the vehicle remains safe. These challenges are complicated by the vehicle's momentum, meaning it takes time for the car to change its velocity; i.e. a vehicle driving fast must slow down before turning a sharp corner. An extension of the problem is to race while avoiding unmapped obstacles on the track.

This dissertation approaches the problem of high-performance, safe autonomous racing using DRL agents to race using LiDAR scans as input to generate speed and steering references.

1.1.2. Motivation

The classical method for autonomous racing is to model the world (perception), generate and optimise a plan (planning) and then execute the plan (control) [3]. Generally, a track map is built before the race to be used for particle filter localisation [10]. Classical approaches have demonstrated exceptionally good performance [11], yet are limited to contexts with large amounts of processing [12] and cannot operate on unmapped tracks [13].

In recent years, deep reinforcement learning (DRL) has led to excellent performance in many games such as Atari [14,15], Go [16], Starcraft [17], and Dota [18]. In racing games, super-human performance has been demonstrated in Grand Turismo Sport (GTS) [19], where DRL agents have outperformed the world's best gamers [20]. These approaches to racing games have shown the power of DRL agents to learn high-performance, computationally efficient policies that map raw data to fine-grained control commands. However, games make many assumptions, such as being allowed to slide along the walls and having access to all the vehicle state variables, that do not transfer to real-world racing.

While DRL algorithms have many advantages, it is difficult to use them for high-performance, real-world systems [21]. Current approaches have used only part of the vehicle's speed range [9], and even at low speeds still crashed [22,23]. While classical F1/10th planners can race up to 8 m/s, the fastest learning approach uses a maximum speed of 5 m/s, indicating the need for work to develop high-performance learning formulations. In addition to low-performance solutions, current racing approaches have not yet considered the problem of unmapped obstacle avoidance.

A key problem in developing DRL agents for real-world problems, including racing, is safety; the agent might crash the vehicle. The general method of using a DRL agent is to train it in simulation and then transfer the policy to the physical vehicle [5,24]. This transfer, known as the sim-to-real problem, exacerbates the safety problem by the agent performing differently on the physical robot than the simulation, resulting from the difference in dynamics. Unsafe policies are a restrictive problem because deep-neural networks (DNNs) are inherently uninterpretable; therefore, the policy cannot be evaluated for safety without testing it on the physical platform.

1.2. Contributions

In light of the lack of feasible approaches to high-speed autonomous racing, this dissertation aims to accelerate DRL methods for F1/10th racing through the following novel contributions:

1. **High-speed Learning Formulations:** The development of learning formulations for high-speed racing through a comprehensive evaluation of current methods, identification of current limitations, and development of new learning formulations that harness vehicle models to aid the training.

2. **Safe Online Training:** The development of a supervisor that can ensure vehicle safety and be used to train DRL agents without them ever crashing, enabling agents

to be trained online physical vehicles, thus bypassing the sim-to-real gap.

3. **Obstacle Avoidance:** The extension of the racing problem to include avoiding un-mapped obstacles on the track and investigating how the flexibility of DRL agents can be combined with classical solutions for high-performance obstacle avoidance.

1.2.1. Design Methodology

The approach in designing solutions integrates classical, vehicle-model-based solutions with DRL agents to overcome the shortcomings of pure machine learning (ML) methods. Classical approaches are limited in their flexibility and computation requirements, and ML approaches are limited in their performance and safety. Combining these two techniques enables improved solutions that operate safely at high performance while not requiring explicit state representation.

1.2.2. Objectives

The contributions described above are achieved through the following objectives:

1. Provide an extensive analysis of current DRL methods of autonomous racing, including a comparison of reward signals, and detailed performance evaluation using comprehensive metrics to understand the limitations of current approaches.

2. The development of DRL learning formulations to enable agents to learn high-performance, variable speed F1/10th racing behaviour using the vehicle's full speed range of up to 8 m/s.

3. The design of a supervisory safety system (SSS) that guarantees vehicle safety by allowing safe actions to be implemented and preventing unsafe actions from being implemented for the vehicle's full speed range.

4. The integration of the supervisory safety system (SSS) into the learning formulation to train DRL agents without ever crashing, with extensive tests comparing reward signals and maximum speeds in simulation and constant speed validation on a physical vehicle.

5. Expand the racing problem to include obstacle avoidance, by designing hybrid planners that combine classical and learning components to enable a vehicle to track a reference trajectory while avoiding obstacles.

1.2.3. Publications

Part of this work has been published in the following articles:

1. "Reward Signal Design for Autonomous Racing." In *2021 20th International Conference on Advanced Robotics (ICAR)*, pp. 455-460. IEEE, 2021 [25].

2. "Learning the Subsystem of Local Planning for Autonomous Racing." In *2021 20th International Conference on Advanced Robotics (ICAR)*, pp. 601-606. IEEE, 2021 [26].

1.3. Dissertation Overview

In light of the lack of feasible approaches to high-speed autonomous racing, this dissertation aims to accelerate DRL methods for F1/10$^{\text{th}}$ racing through the following chapters.

Literature Study: Chapter 2 explores current techniques for classical racing, DRL advances in racing games and DRL methods for real-world racing. Classical racing, in §2.2, is studied in the categories of full-scale and miniature-vehicle racing. DRL advances in racing games, presented in §2.3, are analysed according to their input modality, performance and feasibility for real-world robotics. An extensive analysis of learning methods for real-world racing considers end-to-end approaches in §2.4.1, partial end-to-end in architectures in §2.4.2, and an in-depth analysis of the taxonomies of performance, safety, evaluation metrics, reward signals and sample efficiency in §2.4.3. In §2.4.4 safe learning for racing and control systems is examined and in §2.4.5 the problem of obstacle avoidance is investigated.

The F1/10$^{\text{th}}$ Platform: Chapter 3 presents preliminary information regarding the F1/10$^{\text{th}}$ vehicle setup and simulation configuration. The kinematic and single-track vehicle models are described and compared in §3.2. The evaluation methodology that is used for the remainder of the dissertation is described in §3.3.

Evaluation of DRL for F1/10$^{\text{th}}$ Racing: Chapter 4 provides an extensive evaluation of current methods of DRL for F1/10$^{\text{th}}$ autonomous racing. The overview of reinforcement learning (RL) presented in §4.2 is used to explain the baseline F1/10$^{\text{th}}$ formulation in §4.3. The baseline is evaluated at constant speed in §4.4 and variable speed in §4.5. The extensive evaluation compares reward signals, comments on the agents' training, performance, and repeatability, and uses comprehensive metrics of distance, curvature, deviation from the centre line, and action profile analysis.

High-speed Learning Formulations using Vehicle Models: Chapter 5 uses vehicle models to design high-speed learning formulations that address the problem of speed profile selection. The optimal trajectory (racing line) is used to develop improved reward signals that train the agent to select the optimal speed in §5.2. The link architecture, presented in §5.3, uses the agent to select a steering angle and a friction model to calculate the speed reference. The racing reward signals and link architecture methods are compared with a classical planner and previous approaches in the literature in §4.5.1.

Supervisory Safety System: Chapter 6 addresses the problem of ML safety on physical vehicles by developing a vehicle-model-based supervisory safety system that ensures that the actions implemented on vehicles are safe. §6.2 provides an overview of the supervisory system, §6.3 explains what Viability Theory is, and §6.4 describes how the racing kernels are formulated. The evaluation in §6.5 validates the system's ability to keep a vehicle safe, while not affecting safe planner performance.

Online Learning using a Supervisor: Chapter 7 uses the supervisory safety system that was developed in Chapter 6 to train agents online without them ever crashing during the training process. The modifications to the learning formulation (architecture, episodes configuration and reward signal) are explained in §7.2. The simulation evaluation in §7.3 measures how the supervisor trains agents at constant speeds in §7.3.1 and variable speeds in §7.3.2, compares different reward signals in §7.3.3 and compares end-to-end and online learning in §7.3.4. The formulation is validated on a physical vehicle by training agents from random to drive around a track in §7.4.

Avoiding Un-mapped Obstacles: Chapter 8 expands the problem from racing from moving around the track to additionally avoiding unmapped obstacles. The end-to-end, serial and modification architectures for obstacle avoidance are presented in §8.2. The architecture design aims to build hybrid solutions that combine the advantages of classical and learning-based components. The obstacle avoidance evaluation methodology is described in §8.3 and the results are presented in §8.4.

Chapter 2

Literature Study

Approaches to autonomous racing are studied in the categories of classical methods, learning advances in racing games, and real-world racing approaches. Classical methods, using a perception, planning and control pipeline, demonstrate excellent performance with speeds of up to 8 m/s for F1/10$^{\text{th}}$ racing but are limited by requiring localisation, being inflexible to map changes and being unable to avoid obstacles. While deep learning has achieved super-human performance in games, these methods are infeasible for physical vehicles due to the lack of safety considerations and the assumption of accurate, explicit state representation. End-to-end and partial end-to-end approaches to real-world racing are studied, and LiDAR is shown to be the best input sensor for DRL agents. An in-depth study on racing taxonomies shows that current methods have focused on low (and often constant) speeds, that safety is a significant concern, and that there is a lack of evaluation and comparison of current approaches. Safe online learning and obstacle avoidance are poorly studied fields with few approaches to either problem existing.

2.1. Introduction

The autonomous racing problem is to use onboard sensor readings and computation to generate control inputs for the actuators to move the vehicle safely around the track as quickly as possible, subject to the physical constraints [3]. There are many challenges in this process, the sensor readings are usually large and difficult to use, the vehicle must be accurately modelled, the limits must be identified and defined, and a plan must be generated and executed quickly while ensuring vehicle safety.

Figure 2.1 shows the different planning approaches with classical planning blocks in red and learning components in green. The classical, optimisation-based approach splits the racing problem into perception, planning and control [27]. Perception is the task of interpreting the sensor data into a format that can be used for planning; generally, a method of sensor fusion that produces localisation data. Planners use the map and location data to generate a plan using a vehicle model, subject to the vehicle operating limits. The controller implements this plan on the hardware.

In contrast to classical planning, end-to-end learning approaches focus on replacing

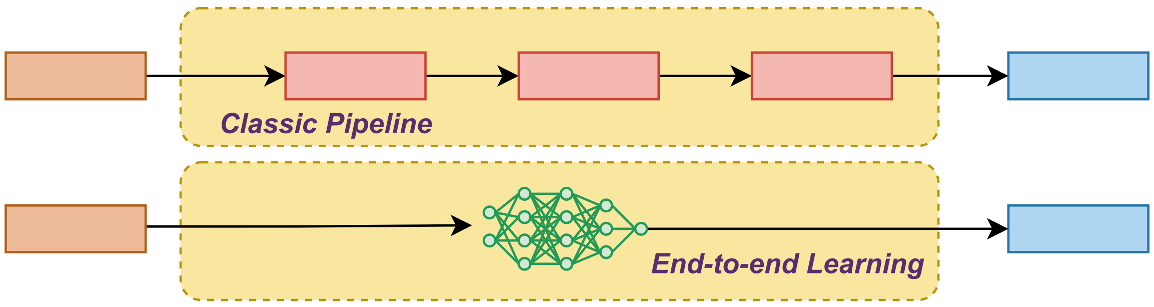


Figure 2.1: The classical racing pipeline using perception, planning and control modules, compared to end-to-end learning that replaces the whole process with a DNN.

the entire planning pipeline with a neural network that can fulfil all three roles. Neural networks have shown an impressive ability to map complex state readings to fine control inputs and can learn policies to control vehicles [20]. End-to-end learning uses input directly from the sensors and partial end-to-end approaches modify the input or output from the agent.

2.2. Optimisation-based Racing

A common approach to autonomous driving and racing is to use a vehicle model to plan a route and then follow it [13, 28]. This process is split into the well-studied perception, planning and control framework [27]. Figure 2.2 shows a detailed view of the generic framework highlighting the role of each block in the pipeline. Classical approaches to autonomous racing are studied in the groups of full-sized vehicles [12, 13, 29, 30] and miniature (scale) vehicles [31–34].

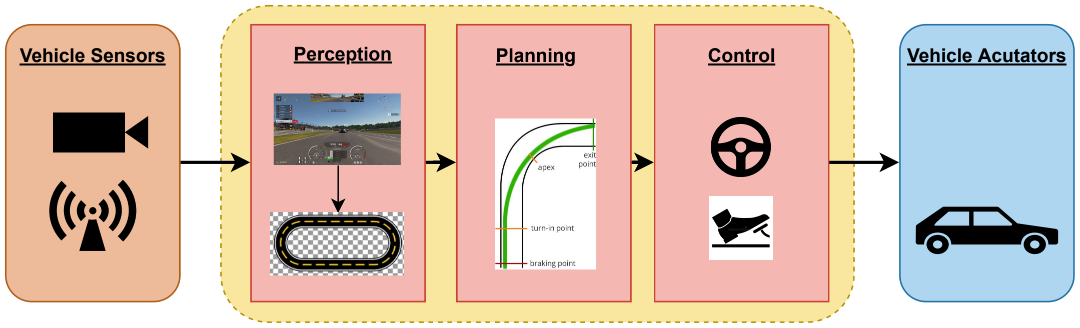


Figure 2.2: The classical pipeline illustrating the role of perception in generating a map, planning to generate a trajectory and control to follow the trajectory.

2.2.1. Full-stack Racing

Full-stack racing refers to full-sized vehicles that drive at the performance limits autonomously. There are several racing competitions for full-sized autonomous vehicles, such as Roborace [13], Formula Student Driverless [30], and the IAC [29]. The full-sized race cars are generally equipped with a collection of LiDAR, camera and radar sensors for perception, and drive motors to control the vehicle. The teams compete by writing software stacks to process the raw data and output control commands to the motors to make the vehicles drive. Methods for perception and planning are studied, while control systems are neglected due to being outside the scope of this project.

Perception

Perception refers to converting the raw data from multiple sensors into intelligible information that can be used to plan. In the Formula Student Driverless (FSD) competition, the teams had to race on an unknown track [30]. The winning team, from ETH Zurich [30], used two perception modes, a SLAM-based mode for the first lap that generated a map of the track and then a localisation mode for the rest of the laps. This system is motivated by their remark that it is not computationally possible to build a map in real-time that is suitable for planning. To increase robustness, camera and LiDAR pipelines that can function independently are used and merged to identify the track boundaries defined by specific traffic cones. The computing is split over two computers, namely a PIP39 rugged computer and a Jetson TX2, both of which are high-performance computers.

Another method of perception was implemented by the TUM (Technical University of Munich) team in the Indy Autonomous challenge to combine LiDAR, camera and radar data for object detection [12]. Their method uses a mixture of detection, clustering and deep learning processing steps before fusing the data to create a list of tracked objects. A separate global navigation satellite system (GNSS) system with inertial measurement units (IMUs) provides localisation. Despite having large rugged onboard computers, the authors still highlight a larger requirement for onboard computing power.

While [12, 30] and other full-stack approaches [35] have the space for large computers that can process immense amounts of data, small vehicles are unable to manage the computational burden required.

Planning

Once a map of the track and the vehicle's current location is available, possibly supplemented by a list of objects, then the autonomous stack must plan a trajectory that it will follow. Heilmeier et al. [13], developed a method of planning a minimum curvature or time-optimal trajectory before the race begins and then using a straightforward control system to follow the plan during the race. An advantage of their method is that they can shift the computational burden of optimisation to offline before the race begins. The TUM team plans online in a receding horizon approach using a tube-model predictive control (MPC) algorithm [12]. MPC has been widely used as a planning approach in software stacks for autonomous racing and has seen much success [3, 30].

Full stack racing with model predictive controllers shows the value of planning with accurate state estimation and that it is possible to push the vehicle to high levels of performance. The AMZ driverless vehicle drove at 85% of the vehicle limits, with accelerations of 1.5g [30]. In Roborace, Heilmeier et al.'s approach drove at 150 km/h and 80% of the acceleration limit [13]. The TUM team's vehicle drove at speeds of 270 km/h and accelerations of 28 m/s^2 in the IAC [12]. These results demonstrate that given enough computation, accurate modelling and optimisation can produce incredibly fast racing performance at the edge of the physical limits.

2.2.2. Miniature Vehicle Racing

In contrast to full-stack racing, where all the variables are easily available, miniature vehicle racing uses smaller vehicles with smaller computers and fewer sensors. The advantage of these vehicles is that they allow for easy and cheap testing of high-performance algorithms. A challenge in miniature vehicle racing is that they have fewer sensors and smaller computers and thus require more efficient solutions.

In work on 1:43 scale autonomous racing cars, Liniger et al. presented an MPC formulation for driving the vehicle at the performance limits [31]. An optimisation routine maximises the distance travelled by the car along the track centreline. The experimental setup used an external motion capture system to provide the locations of the vehicles, and all the optimisation for the planner was done on an external computer. The motion capture system can detect the ego vehicle's location and speed along the x and y axes and the location and speeds of obstacles or opposing vehicles. The control commands for the actuators were sent to the vehicle via Bluetooth. The miniature cars drove at speeds of 3 m/s with saturated rear tyre forces (drifting), showing that their control strategy can operate the vehicles at the performance limits. However, this attempt does not run onboard the vehicle and requires external sensing and computation. Despite later work to speed up the optimisation formulation [11, 32], these algorithms are still not feasible without additional external sensors and computers.

Classical F1/10th Racing

Classical approaches to F1/10th racing have studied methods of optimal trajectory generation and path following. Pure pursuit, model predictive control, and lattice planners have been used to follow the optimal trajectories generated. Localisation is often performed using a particle filter designed for F1/10th racing in [10].

Pure pursuit, originally presented in [36], uses a geometric model with a lookahead distance to calculate a steering angle for the vehicle to take to track a path. Adaptive pure pursuit, which adjusts the lookahead according to the vehicle's speed, has been successfully used for F1/10th racing [33]. The latest iteration of a pure pursuit controller for F1/10th racing has incorporated a model of the vehicle to limit the lateral acceleration to within the friction limit and has achieved the highest level of F1/10th racing performance, winning the German Grand Prix and travelling at speeds of up to 8 m/s on a physical vehicle [37].

MPC has been widely applied as a control strategy that uses a model to prepare an optimal trajectory before executing the first step of the plan [38, 39]. A problem identified within MPC solutions is the computation requirements to run the optimisation [40]. In recent work, a MPC approach using a non-linear vehicle dynamics model has been used for F1/10th racing [41]. However, while the onboard requirements can be improved, the fact remains that requiring localisation and onboard optimisation is computationally expensive.

Another problem noted in model-based solutions is the requirement for accurate vehicle models [42]. Approaches focused on overcoming the modelling challenges have generally focused on integrating learning components to estimate model parameters accurately. A method for using RL to learn tyre parameters for a vehicle has been successfully applied to F1/10th racing [43]. The most successful approach has used a data-driven framework to learn latent dynamics that has enabled a 1/12th scale car to travel at up to 7 m/s [44, 45].

Method	**Localisation**	**Max Speed**	**Physical Vehicle**
MPC [39]	Motion capture, position and velocity	3 m/s	Yes
Non-linear MPC [41]	-	5 m/s	No
Pure pursuit following optimised trajectory [33, 34]	Particle filter	7 m/s	Yes
Model-based pure pursuit [37]	Particle filter	8 m/s	Yes
Data-driven MPC [44, 45]	Particle filter	7 m/s	Yes

Table 2.1: Summary of classical methods for F1/10th racing with inputs required and maximum speeds in evaluation.

Table 2.1 summarises the classical methods for F1/10th racing with the inputs required, maximum speeds and an indication of physical results. The table indicates that classical solutions are required by running either online [37, 44] or offline [39] localisation. A significant advantage of classical methods is they demonstrate high-performance racing, operating the car at the physical limits.

Optimisation-based racing offers high-performance solutions that have demonstrated pushing cars to high speeds. However, these algorithms either require excessive computational power or are not flexible to changes in the map, both of which are significant limitations. An additional challenge is model identification, which requires a learning component to estimate accurately.

2.3. Deep Reinforcement Learning in Racing Games

Deep reinforcement learning has been developed mainly in games such as Go [16], Starcraft [17], and Dota [18]. Approaches to racing games are studied in the categories of input modalities, and performance and realism.

Deep learning is a method of solving control problems that uses an agent, comprising a deep neural network, to learn a policy that maps a state vector to a control action. Deep learning is a method of learning a mapping of inputs to outputs using deep (multi-layered) neural networks. Deep learning has often been used to learn a mapping of sensor data to control actions to solve engineering problems, such as quad-rotor flight [46, 47], controlling a quadruped [48], autonomous navigation [49], and autonomous racing [20]. Reinforcement learning is a method of training the agent from experience using a reward signal that indicates how good or bad the action is [50]. The training aims to learn neural network weights and biases to values, such that the mapping from input to output through the agent results in the desired behaviour.

2.3.1. Input Modalities

Video games are an ideal context for DRL methods because they use simulations that give the agent access to the full state of the simulator. The input modalities are grouped into image-based solutions and methods using hybrid state vectors of multiple simulated sensor readings.

Camera Images

The most popular environment for advances in DRL has been the Atari environments [51], which are commonly accepted as the benchmark tests for how well algorithms perform [14, 15]. Solutions to Atari games use the game images as input to the agent, which must return actions to the game engine. Included in the Atari suit is a racing game, “car_racing”, where the agent must learn to control the steering, accelerating and breaking. These Atari environments are designed as games to promote research and lack many forms of realism, such as representing physical systems.

In the game World Rally Championship 6, an approach using DRL to learn to race, using the video game image feed as input, was presented [52, 53]. Their method uses an actor-critic algorithm with a convolutional neural network (CNN) and long-short-term memory (LSTM) layers to train an agent to control the steering and speed actions. Actor-critic algorithms use an actor-network to select actions and a critic network to estimate the value of a state and train the actor. After training their agent for 190 million steps, they show that it learns to race and take many corners. The paper demonstrates a significant advantage of DRL; agents can learn from experience to perform highly complex tasks using raw input data without any explicit state estimation.

However, the authors note in [53] that their algorithm still crashes regularly, thus not being able to achieve any level of feasibility. In their follow-up paper [52], they mention that the agents cannot learn any optimal trajectory. An interesting point that they demonstrate is that adding speed limits results in significant performance improvements. The agents' lack of ability to select an optimal trajectory and the improvement in speed limits show that high-speed racing, even in unrealistic simulators, is a complex task, especially due to the agents' inability to plan. A further challenge when using images is that parameters such as lighting conditions that are constant in a game can easily change in the real world, making it specifically difficult to transfer image-based work to reality [24].

Hybrid State Vectors

Another approach to game-based racing is to use the hybrid state vectors consisting of the game state variables that are used in the simulator. State variables are the quantities used by the simulator to record the current game context. For example, the most basic state variable is the vehicle's position, and more complicated variables are the speed, acceleration, orientation distance from walls etc. In contrast to single source state vectors that have a single input, hybrid state vectors include multiple different values such as range finders, velocity and acceleration.

DRL agents have been used to learn the complex dynamics of high-speed drifting, which are difficult to model using classical methods due to a high degree of non-linearity [54]. Their method is developed and evaluated in the Speed Dreams racing simulator. While the simulator uses realistic dynamics to represent a vehicle, their method relies on many detailed measurements, such as the derivative of slip angle, which are nearly impossible to measure accurately on a physical vehicle. Their work demonstrates that DRL agents can learn to control vehicles with highly non-linear tyre dynamics from experience without human domain knowledge. However, their approach is far from feasible since the state estimation that they require is not currently possible.

The popular racing game GTS has been the subject of several DRL research projects [19, 20, 55]. While some finer details differ between approaches, the general method uses a state vector containing range finders, upcoming waypoints, and the vehicle's velocity and acceleration in three dimensions. This forms a hybrid state vector rich with information for how the agent should perform. In addition to complex, hybrid state vectors, multiple hand-crafted rewards that rely on having full access to the simulator are used. While their work demonstrates excellent racing results in outperforming humans, it is impractical to measure or estimate all the state variables used, thus rendering these solutions infeasible.

2.3.2. Performance

The approaches considered in racing games are analysed for their behaviour across the two key challenges in racing performance and safety. Racing games lend themselves towards developing high-performance solutions that outperform humans. However, due to the nature of games, safety restrictions present in the real world are neglected to improve the user experience. Table 2.2 presents a list of the methods studied, with their safety considerations and performance levels.

Methods	**Safety**	**Performance**
End-to-end learning for WRC6 [52, 53]	Crashes every 0.8 km	Travel 1.2 km without crashing
Autonomous drifting in Speed Dreams [54]	No consideration	Drifts smoothly on complex tracks
Super-human performance in GTS [19]	Allowed to crash into barriers and other cars	Outperforms humans drivers
Outracing champions in GTS [20]	Allowed to crash and slide along barriers	Outperforms world-champion gamers

Table 2.2: A study of the approaches to games with their safety considerations and performance levels.

Table 2.2 shows the safety considerations and performance levels of DRL approaches in popular racing games. The end-to-end learning from images in [52] performs badly, regularly crashing, even when fully trained. Using a DRL agent for autonomous drifting in [54] demonstrates good results, drifting smoothly on all their test tracks. Their study also shows that the method highly depends on the vehicle's slip angle (angle between orientation and velocity). The work in GTS has demonstrated excellent performance in exceeding human capability in various competitions and outperforming the world champions. A key limitation in all these studies is the lack of any safety considerations; either safety is completely neglected, or the vehicle is explicitly allowed to be in contact with the boundaries as a part of its strategy.

The study of DRL approaches to racing games demonstrates the power of learning agents to master complex domains and outperform humans. However, the methods applied to games require much work before they can be used for physical robotic systems. The two most significant hindrances to robotic use are the lack of safety considerations and the extensive state variables required by high-performance methods.

2.4. Learning Real-world Racing

Real-world racing approaches include all those that are designed specifically for a hardware platform and thus are physically feasible. While approaches with physical experimental results are the basis, papers with studies in realistic simulators, such as the official F1/10$^{\text{th}}$ simulator, are also included. End-to-end approaches that map sensor readings directly to control actions are studied, followed by partial end-to-end solutions that combine classical components with learning agents. The study then analyses the taxonomies of racing performance, safety, the evaluation metrics used, reward functions and sample efficiency. A further study on safety considers methods focused on training agents without them crashing and consults the literature on safe learning. Finally, the problem of obstacle avoidance is considered.

2.4.1. End-to-end Learning

End-to-end learning focuses on using raw sensor data as input to a learning agent that outputs control actions. As the name implies, the agent connects to the output and input ends of the vehicle system. End-to-end methods are categorised according to the input modalities of camera vision and LiDAR scans.

Camera Vision

Research into the problem of using camera vision as an input into a deep learning controller for autonomous racing is focused on replicating how humans drive by looking at what is approaching and making decisions [56]. Classical methods require perception algorithms to convert images to a location on a map before planning can take place. Therefore, mapping an image directly to a control action represents a significant advantage over previous methods.

The DeepRacer platform, which features 1/18$^{\text{th}}$ scale vehicles with a corresponding simulator, was developed by researchers at Amazon to encourage deep-learning research [5]. The general focus is to develop and train DRL algorithms using simulated images and then transfer them to the physical vehicles where a single camera is used. Their original work uses a five-layer convolutional neural network to map the images to the steering angle references. The results demonstrate that it can learn a policy to control a vehicle in reality by training on simulated images. However, their results demonstrate many shortcomings, firstly, the vehicles are evaluated at slow, constant speeds of 1.33 and 2 m/s, far from the performance limits. Secondly, their models are sensitive to changes in lighting conditions, and many of their results require a human to reset the vehicle regularly. This research demonstrates that the reality gap is significant even for a simple autonomous driving setup, and current approaches cannot guarantee safe behaviour.

Cai et al. [6] present one of the best studies in vision-based control of a racing vehicle to date. They present a hybrid training method that uses imitation learning (IL) to learn from an initial driving data set and then RL to improve the policy. The imitation learning algorithm trains the agent to imitate the actions selected in the data set. Their algorithm, called deep imitative reinforcement learning (DIRL), has improved sample efficiency due to the IL and is robust due to the RL improvement. They train their algorithm in simulation, where it can achieve a 100% completion rate and then transfer it to a physical vehicle and test it on a track with obstacles. They use a 1/20$^{\text{th}}$ scale vehicle and run it at speeds of less than 2 m/s, which is relatively slow. While they show that it is possible to overcome the sim-to-real gap, it is reported that they require an average of between 1.3 and 32 interventions per lap. They conclude that their method, despite all the improvements, cannot match the performance of human experts.

Another paper called "Sim-To-Real Transfer for Miniature Autonomous Car Racing" studied the sim-to-real problem in simulation by investigating how perturbing different variables such as lighting affects the transferability of agents [57]. The paper does not transfer their policies to a physical vehicle but only measures simulation results. The author's work starts with the acknowledgement that transferring agents trained on simulated images to real vehicles with real images is difficult because small changes such as different colours or lighting conditions can greatly affect neural network performance. Their method involves training a teacher policy in simulation combined with domain randomisation [58] and is used to train a student policy in different conditions. While they demonstrate an improvement in overcoming the reality gap, their best result has a completion rate of 52%, which is far from safe. This work indicates that the agents that learn from images do not transfer well from simulation to reality, and even advanced methods present poor results.

All these results in using vision as input to DRL agents make a clear point, vision is not yet a viable solution for autonomous racing in simulation or reality. While vision-based systems have displayed mediocre performance, they are not currently a viable solution for high-performance behaviour that is safe and robust to different environments.

LiDAR Scans

LiDAR scanners, known as range finders that use beams of light to determine how far away the boundaries of an environment are, have become an increasingly popular sensor for autonomous racing algorithms. Classical systems use LiDAR scans for localisation [10], and reactive algorithms like follow the gap (FGM) use the geometric information from LiDAR scans [59]. Autonomous navigation pioneered the use of range finders for navigation [49]. Deep learning agents use LiDAR scans as input to a neural network [19, 22, 60, 61]. LiDAR scans are less dependent on lighting conditions than cameras, and several approaches have been successfully implemented on physical vehicles [22, 60].

Several approaches have successfully demonstrated that LiDAR scanners on physical vehicles can be used for autonomous racing. One of the first successful approaches trained an agent to complete a single racing corner using LiDAR [62]. While the experiment was focused on validating a network using formal methods, it demonstrated that agents using LiDAR scans could transfer from simulation to reality.

Brunnbauer et al. [22] used a model-based algorithm (Dreamer [63]) for F1/10th racing using LiDAR as input. Their work was focused on comparing model-free and model-based algorithms for autonomous racing. They claimed that model-based RL was required for racing and showed that their policy generalises well between tracks and that they can transfer the model trained in simulation to a physical vehicle. The authors note that a limitation of their method is slaloming, which they improve upon with action regularisation and reward hacking but do not completely overcome. The evaluation of the sim-to-real transfer is limited, with no data on the lap times or success rates of the physical vehicle. This result demonstrates the advantages of using LiDAR scans as DRL input, namely, the ability to transfer between different maps and the ability to transfer the policy trained in simulation to a physical vehicle with no further training required. However, further investigation into how trained policies transfer to the real vehicle and a solution to the problem of slaloming are still required.

Another paper positioned itself in competition to [22] by claiming that it is possible to use model-free learning to race autonomously using LiDAR [60]. They use a similar learning configuration, with the model-free DQN DRL algorithm that takes LiDAR beams as input and outputs steering angles. The DQN algorithm uses a deep neural network to implement Q-learning using a value function. They use a fixed speed for their experiments, which is a proportion of the vehicle's maximum speed. They present two methods of training an agent, either on the vehicles, using a simplistic safety mechanism or in simulation and then transferring the policy to the physical vehicle. In simulation, they claim they can outperform the work in [22] by producing faster lap times on two common maps. The results in [60] do not indicate the sim-to-real gap apart from saying it worked to transfer a policy to a physical vehicle.

Imitation learning trains a neural network from demonstrations to learn a mapping between observations and actions that replicates human behaviour [64]. The most basic IL algorithm is behavioural cloning (BC) which aims to copy the behaviour shown in a data set. One of the key challenges in IL is the robustness of the trained policies to states that were not well represented in the data set. The data-aggregation (DAgger) algorithm tried to overcome this by expanding the data set by using the oracle policy to label actions selected by the agent trained with behaviour cloning [65]. Human-gated approaches [66] and expert intervention learning (EIL) [67] have tried to use a human or other policy to further train the agents in a safe manner.

A study to benchmark imitation learning methods for F1/10th racing compared popular methods, such as behavioural cloning [68], DAgger [65], human-gated-DAgger [66], and EIL [67] for their ability to learn to race using LiDAR as input [23]. Their results show that IL cannot train agents to complete a lap of their track, even at a constant speed. Their paper has useful metrics, but the only algorithm to complete a lap of the track is human-gated DAgger that is improved with proximal policy optimisation (PPO) [69]. Another study compared IL to DRL through evaluation in simulation and on physical vehicles travelling at 1 m/s [9]. They concluded that DRL was more robust than IL and should be preferred.

Using LiDAR beams as input to a DRL agent has been shown to produce high-performance racing behaviour [19] that adapts to different maps [22] and can be transferred to physical F1/10th vehicles [60]. Current limitations in LiDAR-based DRL racing are the problem of slaloming [3, 22, 61], the lack of safety guarantees [19, 61], and a lack of evaluation as to the sim-to-real gap.

2.4.2. Partial End-to-end Learning

Partial end-to-end methods use classical components in conjunction with DRL agents. Partial end-to-end approaches are split into those that use classical components to condition the input into the agent and those that use classical components on the agent's output. Methods that condition the inputs typically combine a classical localisation method with a DRL agent that replaces the planner. Approaches that use control outputs have harnessed the power of neural networks for perception and tried to reduce the learning complexity by using control components after the agent. Figure 2.3 shows the difference between conditioned input and control output architectures.

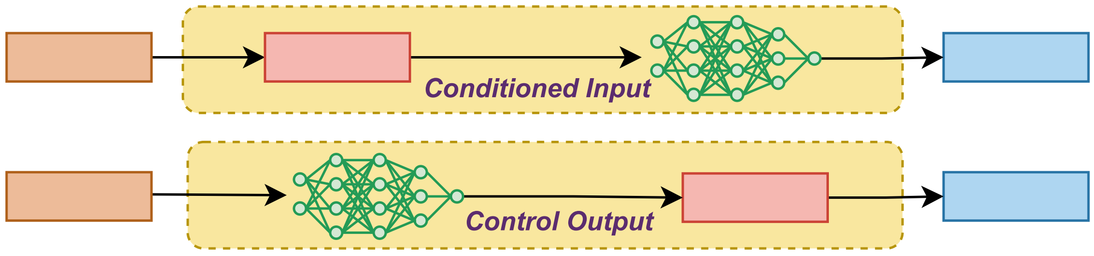


Figure 2.3: Partial end-to-end architectures comparing conditioned input to the network and using a control system to follow the neural network output.

Conditioned Inputs

The typical method of conditioning inputs for a DRL agent uses a classical perception algorithm, such as a particle filter for localisation, to generate a hybrid state vector for the agent. The aim of replacing the classical planner with a DRL agent is to reduce computation times, since MPC algorithms are expensive and neural networks execute quickly [39].

Chisari et al. [70] use a state that comprises the vehicle's location relative to the track centre line, velocities in the lateral and longitudinal directions and the vehicle's yaw and steering angles, which are obtained using an external camera-based motion capture system. They train the policy in simulation and further refine it on the physical vehicle. Their high-speed results and improvement through policy refinement on the physical vehicle show the value of training online on the physical vehicle. Their work is limited by requiring a high-frequency, external motion capture system and by indicating that for a portion of the driving time, the vehicle operates outside the constraint bounds, causing a safety concern. Other similar methods have used a particle filter for localisation [38, 39].

Zhang et al. [71] claimed that using LiDAR as a sole input did not perform well enough and proposed that a modified artificial potential field (MAPF) controller should be used in conjunction with a DRL agent. They aim to simplify the problem to one of local planning where the MAPF controller can identify a target that the agent must navigate towards. Their results showed that their method could outperform the Dreamer algorithm [22, 63] and human drivers. While they show the paths taken by the vehicle, they do not show the vehicle's speed or make any remarks as to how close to the operating limits they are. Their work demonstrates that aiding the learning formulation for additional information improves the quality of the learning.

While replacing the planner with a neural network has improved calculation speed, it has the critical disadvantage of requiring localisation. One of the main advantages of DRL algorithms is that they are flexible to unmapped contexts and do not require localisation. Therefore, methods requiring localisation remove the advantages that neural networks bring.

Control Output

The most in-depth study that has been done into different architectures for autonomous racing is by Weiss and Behl in [72, 73]. The authors note that replacing the entire perception, planning and control problem can exacerbate problems like over-fitting and make solutions difficult to debug. Therefore, they propose methods for learning a smaller part of the navigation pipeline, namely to generate trajectories that a path follower can track. Their results show that learning trajectories defined by Bezier curves produces the best performance. The authors state that future work should investigate the application of their work to using reinforcement learning and the ability to avoid obstacles. This work demonstrates that it is possible to improve racing performance by learning a more abstract representation of racing and then using a classical system to implement it.

Another approach to using a controller after the agent output was proposed in [74]. This method uses an encoder-decoder network to convert raw image input into a cost map for the road that can be used by an MPC planner for planning. This method allows the network to maintain the optimality of an optimisation-based planning approach. The success of this approach demonstrates that using networks in the pipeline to handle the areas with large variance (i.e. images), and an optimiser to plan and maintain optimality is a good strategy.

2.4.3. Racing Taxonomies

The study now shifts from considering different approaches to racing to analysing the racing behaviour generated by each method. End-to-end methods using LiDAR scans as input are the focus since they are the widest body of research. The behaviour is studied according to the two challenges in racing; performance and safety. Thereafter, the study looks at the evaluation metrics, reward functions and sample efficiency of learning-based approaches.

Racing Performance

Racing performance is the simple metric of how fast the vehicle can move around the track. Since autonomous racing is a new research domain that has gained popularity in the last five years, much of the work is focused on proving that DRL is a viable solution, as opposed to generating high performance.

While it is difficult to compare lap times since most papers use different maps, top speed is easy to compare since all the F1/10$^{\text{th}}$ methods use the same hardware platform and similar simulators. Constant speed solutions operate the vehicle at a constant speed for the entire lap, and variable speed solutions allow the planner to vary the speed. The methods studied are categorised according to the speeds used in simulation and on physical vehicles.

Approach	**Simulation**	**Physical Vehicle**
RL vs IL [9]	1.5 m/s (constant)	1 m/s (constant)
IL benchmarks [23]	8.24 m/s (variable)	3 m/s (constant)
DQN for F1/10$^{\text{th}}$ racing [60]	5 m/s (variable)	1.7 m/s (constant)
Model-based DRL [22, 75]	4 m/s (variable)	Not presented
ResRace [71]	5 m/s (variable)	-

Table 2.3: Maximum speeds used by DRL approaches in simulation and on physical vehicles. All speeds are in m/s.

Table 2.3 reports the maximum speeds in m/s used in simulation and reality by the methods studied. The work in comparing RL and IL in [9] used slow constant speeds of 1 and 1.5 m/s. While the study on benchmarking IL algorithms claims the highest speed of 8.24 m/s, only the EIL algorithm was able to complete a lap at this speed [23]. Additionally, for their physical experiments, only the HG-DAgger agent improved with PPO could complete a single lap of their test track, with all the other algorithms crashing.

Using the Dreamer algorithm showed good performance with the agent learning a valid speed profile of speeding up and slowing down up to 4 m/s [75]. Using a DQN outperformed Dreamer in terms of lap time, but this might be simply due to using a higher top speed of 5 m/s[1].

Many classical solutions used top speeds of up to 7 or 8 m/s [44]. Therefore, the speeds used in DRL racing are far below what classical methods can achieve. One of the challenges is that there is very little study in the literature on the speed profiles selected by the agents. Another difficulty is that many solutions only operate at a constant speed, and the agent does not select a speed profile which is required for the vehicle to be able to reach higher speeds. While the preprint [75] shows trajectories and speed profiles of the agent, Bosello et al. [60] make no mention of how the vehicle learns to speed up or slow down. Zhang et al. [71] do not mention the speeds used in their work. Future work should study the profiles selected by the vehicle and use the information to enable high-speed racing that can compete with classical solutions.

Safety

The second key challenge for racing agents is that they perform safely. Safety in DRL is a significant concern that has resulted in many methods not being feasible for use on physical vehicles [24]. From their own experience and surveying the current state of training agents for robotics tasks, Ibarz et al. report that safety is a “bottleneck” for real-world implementation of learning systems [21]. They point out that real systems are significantly more complicated than simulations and that there is a lack of methods for guaranteeing safety. Another paper on the challenges and opportunities of applying RL to autonomous racing notes that the sim-to-real gap for racing is large as small model differences can cause big changes in the results [76].

Method	**Vehicle Safety**
Model-based DRL [75]	Complete 1.4-2.1 laps before crashing
ResRace [71]	Complete up to 3.5 laps before crashing
IL benchmarks [23]	All algorithms except EIL crashed
DQN for F1/10th racing [60]	No information given
RL vs IL [9]	Safety score between 0-100%

Table 2.4: Safety levels of DRL agents used for F1/10th racing.

Table 2.4 shows the safety level for work studied on DRL methods for autonomous racing. Brunnbauer et al. [22] report that the best performance achieved is 2.1 laps of two tracks and only 1.5 laps on Austria. Bosello et al. [60] make no mention of the safety or

[1]Found in associated repository: https://github.com/MichaelBosello/f1tenth-RL

repeatability of their approach. The study on bench-marking IL algorithms shows that many of their algorithms crash; there is no consideration of how repeatable this result is or what the crash rate of the algorithms is. Hamilton et al. [9] report a safety percentage which is the number of completed laps, which is a useful metric. In their results, the agents achieve a safety score that is often below 100% indicating that their agents crash a lot. A key problem is that all these methods report multiple crashes. While an F1/10th vehicle crashing does not pose a big safety risk to humans, for as long as these algorithms are unsafe, they are unsuitable for use on any performance critical system as they might crash.

There is a significant lack of investigation into how safe and repeatable DRL methods are for autonomous racing. As long as methods do not produce safe solutions, or the safety is unknown, they cannot be implemented on physical vehicles. Future work should investigate how safe current methods are and develop solutions to prevent DRL agents from crashing.

Evaluation Metrics

A significant problem in the racing literature is the lack of evaluation of DRL agent's racing performance. This problem is particularly acute in understanding how the learning formulation affects the trained agent's behaviour. As an example in [13], Heilmeier et al., in calculating a minimum curvature trajectory, use the metrics of path curvature, lateral and longitudinal acceleration, speed profile, and lateral deviation from the racing line to evaluate their method. In contrast, the learning literature lacks many of these metrics, making it difficult to understand the problems.

The problem of slaloming, also known as warbling, of swerving rapidly from left to right, has been identified in many works [22, 23, 61]. However, little measurement to understand the problem, such as plotting the curvature, has been conducted. Another problem is that many DRL solutions crash and have poor completion rates. An investigation should analyse why agents display poor completion rates so that the problem can be addressed.

Another open challenge in robotics is the sim-to-real problem of transferring policies trained in simulation to reality. While many papers have overcome the sim-to-real problem [9, 22, 60], there has been little study comparing how the methods compare differently in simulation to reality.

Reward Functions

One of the largest areas where there is a lack of comparison is the learning formulation used by the agent. Reinforcement learning uses a reward function to teach the agent how good a certain action was. Reward signals encode the desired behaviour in an equation that can be easily calculated at each timestep. While many reward functions have been used, there has been little comparison between reward methodologies. This is especially surprising when studies have noted that it is a challenging task to design an accurate reward signal [60, 77].

Progress rewards are adapted from the navigation literature [49] and reward the agent for the vehicle's progress made in the direction of the centre line. In the racing literature, progress rewards have produced high-performance racing [5, 19, 22]. While using the progress indirectly takes the vehicle's velocity into account, there is no explicit reward for velocity. Considering that autonomous racing is aimed at high-speed driving, this is an obvious weakness.

Velocity-based rewards have been widely used in DRL solutions to autonomous racing [52, 60, 61, 77]. Velocity rewards are referred to as intrinsic rewards since the ultimate aim of racing is fast lap times which is the sum of the integral of velocity along the track centre line. The classic formulation is to reward the velocity in the direction of the centre line and punish lateral deviation from the centre line [53]. While several other methods for using the velocity have been tested, none of them have outperformed the classic formulation [52].

Other interesting reward signals that have been used include hand-crafted [78], kinetic energy-based (when crashing) [19], overtaking [55], and steering rewards [62]. In the world-class agent Sophy, eight reward terms were used to encourage different behaviours, from good sportsmanship to overtaking [20]. Yet none of these approaches have been used by more than a single study or shown to outperform other reward functions, so they are not considered important.

There is a significant lack of study in reward signals in three areas, firstly in how different signals compare to each other, secondly, on the training performance created by the reward (especially with regards to speed), and thirdly in the development of custom reward signals for the task of autonomous racing.

Sample Efficiency

An analysis of the sample efficiency is presented in a table showing how many training steps were used by each method. Table 2.5 shows a list of different methods and the number of training steps used.

Method	**Input**	**Algorithm**	**Training Steps** ($\times 10^6$)
End-to-end racing in WR6 [52]	Images	SAC	140
Learning in simulation [70]	State variables	SAC	18
TORCS autonomous racing [61]	LiDAR	SAC & DQN	6
Model-based learning [22]	LiDAR	Dreamer	2
Generalised F1/10th RL [60]	LiDAR	DQN	0.55

Table 2.5: Number of training steps used in the end-to-end RL literature.

In Table 2.5, the number of training steps for methods with different inputs is presented. Methods using images are trained for the largest number of training steps (140 million in [52]). Using LiDAR scans presents a significant improvement by using less than 6 million training steps. The most sample-efficient result found in the literature is Bosello et al. [60] that train their agents for 550,000 steps. The continuous soft-actor-critic (SAC) algorithm and the discrete deep-Q network (DQN) algorithm have been the most popular choice for racing. It is difficult to compare the number of training steps since every configuration has a different frequency used in the planner, algorithm, and input vector. However, these DRL algorithms require many samples to learn how to race.

2.4.4. Safe Autonomous Racing

The literature study in end-to-end learning for autonomous racing showed that the problem of safety, and thus agent implementation on physical vehicles, remains. In response to these problems, methods for safe autonomous racing are studied. Approaches using formal methods of neural network verification [62, 79], and probabilistic methods [80, 81] are briefly mentioned. A study on supervisory learning approaches [82–85], safe learning for autonomous racing [60, 86, 87], and methods to improve safe learning through Viability Theory [32, 88] are presented.

Supervisory Autonomous Racing

Only three papers have been found that contain approaches for safe learning for autonomous racing, one in simulation published in 2020 [86], and two using physical vehicles published in 2022 [60, 87]. The first approach [86] uses a two-stage supervision approach process, with the first stage using a simplified dynamics model and low speeds, while a learned model is trained. The second stage uses the trained model as a supervisor to ensure that the vehicle does not crash, even at the handling limits. This approach is demonstrated in the TORCS simulator but lacks physical data to validate it. While the results show that it can keep their agent safe, conceptually using a neural network for safety is dangerous because there are no guarantees that it performs in a certain way.

Bosello et al. noted the problem of transferring agents from simulation to reality and thus used a safety mechanism to enable training onboard an F1/10th vehicle [60]. The safety mechanism is simple and reverses the vehicle if it is near a boundary. The results show that this approach can successfully train a DRL agent to drive at a constant speed onboard a vehicle and bypass the sim-to-real gap.

Another approach by Musau et al. [87] implements the simplex control structure [89] on an F1/10th vehicle. Reachability theory is used to ensure that the vehicle remains on the track. This is done by calculating future trajectories for the vehicle and ensuring that there exists a sequence of control commands that lead the vehicle to remain on the track. This work also runs the vehicle slowly on a simple track. They conclude that the idea of supervisory learning for online RL is a good idea, but that Viability Theory would be a better approach instead of Reachability Theory. This is a true statement since Reachability Theory does not guarantee recursive feasibility, and it requires a planning run-time of how far the algorithm looks ahead, which is resource intensive and limited by onboard computation. Recursive feasibility means that for every safe state, there exists an action that leads to another safe state. Thus, if you are in a recursively feasible state, you can select control actions that keep you in the set of feasible states.

Viability Theory, which is concerned with finding sets of states that remain within a constraint while evolving according to a set of dynamics, has been used to ensure the safety of driving controllers [88]. Liniger et al. [32] developed a formulation for a viability kernel for autonomous racing to speed up the search of an MPC algorithm. Liniger calculated a set of safe states offline, before a race, and then used the kernel during the race to look-up if a specific state was safe or not. Viability Theory shows promise as a method for guaranteeing the safety of a racing vehicle, but this has yet to be evaluated.

Safe Learning Methodologies

Since there have been few approaches to safe autonomous racing, an exposition of work in safety and safe learning in other fields is presented. Safe learning methodologies are grouped into formal verification methods, probabilistic methods and supervisory methods.

Formal verification methods that aim to guarantee that a network will act in a certain manner for a set of possible states, have been used to verify the safe performance of a DRL autonomous racing agent [62,79]. In [62], the authors train a DRL agent to race an F1/10th vehicle around a single corner, then validate the network for safety before transferring the policy to a physical vehicle. Their results show that the verification method is successful, and the agent can drive the physical vehicle safely. However, verification approaches are limited by (1) validating the policy using the simulation model (i.e. the sim-to-real problem remains), (2) taking a long time to validate simple networks, (3) not scaling well to large networks, (4) not providing any way to fix a policy if it is not validated.

Probabilistic methods have been used to estimate the probability of an action being unsafe and if the probability is above a certain threshold, then using a safe policy [80,81]. This solution is limited to ensuring safety because probabilities inherently do not provide guarantees. Conceptually, this is a poor solution to the problem of safety. This is demonstrated by the work in [81] still achieving a success rate of less than 100%, i.e. their vehicle still crashes. Any chance of crashing means the system cannot ensure safety in all circumstances.

Using a supervisor has been widely proposed to ensure safety in validating experimental vehicle planners [82], safety-critical learning [90], and cruise-control systems [83]. The role of a supervisor is to monitor a potentially unsafe planner and ensure that safe actions are taken. In 1998, the simplex architecture for safe online control system upgrades was proposed with the three components of an experimental policy, a safe policy, and a decision module [89]. The safe policy was a fall-back option if the decision module decided that the experimental policy had selected an unsafe control. Since this paper, the idea of using a supervisor to ensure that controllers and planners operate according to a set of specifications has grown.

Safety in Control Systems

The problem of safe reinforcement learning has been widely approached in control systems. The general approach has been to use a supervisor with access to a safe set to ensure that the control system performs in a certain region.

Dalal et al. introduced a method for using a supervisor during the training of an RL agent to ensure that it remains within a safe set [84]. Their method adds a safety layer after the agent that ensures that the agent remains within a predefined region. They pre-train a dynamics model on random actions taken in the environment and then use it in the safety layer to ensure that an action will result in a state within the constraints. Their evaluation shows that they can solve the classic RL problems, like moving a ball to a goal location while keeping it within a constraint set during training. This idea of training within a safe region (or a shield [91]) presents a great opportunity for learning to race safely. An additional advantage of this method is more efficient exploration, leading to greater sample efficiency.

Many approaches to safe reinforcement learning have been developed in control systems [83,85,92,93]. Control barrier functions (CBFs) ensure that a control system remains within a constraint set [94]. Several approaches have used control barrier functions with dynamics models to create a safe region for an agent to learn in [85, 90]. Other approaches to safe learning in control systems have used constraint-admissible sets for linear systems [83], Hamilton-Jacobi methods for uncertain systems [92], and reachability theory [93].

These approaches are applicable to control systems where a control input must be chosen in such a way that the state variables remain within a certain limit. CBFs contribute the valuable idea of recursive feasibility or forward in-variance, the quality of all states within a set to contain an action that leads to another state within the set. However, in the racing setup, the constraints are not a static limit on a variable but rather a map of the track boundaries, and thus, these methods are not directly applicable. For these methods to be useful in autonomous racing, a definition of a recursively feasible safe set of states is required.

2.4.5. Obstacle Avoidance

In their paper on the F1/10$^{\text{th}}$ simulator, O'kelly et al. [4] cite the three benchmark tests for autonomous racing performance, (1) racing around a known track, (2) avoiding static obstacles, and (3) head-to-head racing. The first challenge of racing around a track has been widely studied, and the third problem of head-to-head racing has been occasionally studied, but little work has been done on static obstacle avoidance. The problem of static obstacle avoidance is defined as having obstacles (such as blocks or cones) placed on the track at random, un-mapped locations that must be avoided. The few approaches related to obstacle avoidance in F1/10$^{\text{th}}$ racing are studied, followed by the literature on hybrid navigation architectures.

Obstacles in F1/10$^{\text{th}}$ Racing

Several F1/10$^{\text{th}}$ approaches have evaluated their methods by adding several static obstacles on the track and showing that their solution still works [6]. Even these simplistic tests have presented poor results; for example, Cai et al. report the average number of human interventions required per lap between 4.3 and 32.7 for a track with 8 obstacles [6].

In studying the safety of planners for overtaking in simulation, Bak et al. [95] conclude that overtaking other vehicles is a surprisingly difficult problem, producing many more crashes than anticipated. In their evaluation of the several popular algorithms, such as the follow the gap algorithm [59], disparity extender and graph planner, they found that vehicles crash on average between 33.5 and 83.4 times during their experiments. Another study targeting head-to-head racing using game theory also noted that they cannot achieve a 0% crash rate [96]. The conclusion from the literature is that few approaches have studied obstacle avoidance and preliminary studies indicate that it is a difficult problem.

Hybrid Navigation Architectures

DRL agents have been a popular method of low-speed obstacle avoidance for holonomic vehicles [49, 97, 98]. The navigation problem is to move from one point to another in an unknown environment while avoiding any obstacles. Obstacle avoidance is a difficult problem for any planning-based approach because obstacles must be detected (perceived), and the plan must be updated to avoid them in real time. Obstacle detection, perception and re-planning is a computationally intensive task that is difficult to do onboard. The difficulty in obstacle avoidance in navigation is demonstrated in Tai et al.'s robot requiring human intervention [49], and de Villiers et al. concluding that high-level planners should be used to guide the low-level DRL agents [98].

In a survey paper on autonomous navigation, Xiao et al. [99] recommend using learning techniques at a subsystem level due to their analysis showing improved performance.

Another survey on end-to-end learning notes that it has limitations in safety and performance, and further research into subsystem architectures should continue [100]. While DRL agents have the advantage of being flexible to unstructured environments, they are limited by their lack of ability to plan at any level. Therefore, a common approach has been to combine a classical high-level path planning module with a lower-level DRL agent that learns the subsystem of obstacle avoidance [101–103]. Table 2.6 shows a study of long-range navigation techniques that have combined classical high-level planners with lower-level learning-based planners.

Method	**High-level**	**Low-level**
Deployment of RL-based obstacle avoidance in conventional systems [101]	A* planner generates waypoints	DRL agent trained for path following obstacle avoidance
Long-range Navigation using sampling planning and DRL [104]	Probabilistic road map generates sub-goals	DRL agent trained to plan according to vehicle dynamics
Delivery robots [102]	Hybrid A* to generate “intentions”, such as go forward or turn left	DRL agent to enact intentions around obstacles
Navigation combining visual SLAM and DRL [103]	A* planner supplies local sub-targets	RL planner takes map detail into account

Table 2.6: Hybrid planners that combine high-level classical planning with low-level DRL flexibility.

High-level path search algorithms, such as A* [103], can find paths for long missions. The decision to use RL for local planning and obstacle avoidance is motivated by agents' flexibility and computational efficiency [101]. The general trend is to use learning agents to respond to the uncertainty in the environment, usually in the form of obstacles [102]. The ability of hybrid architectures to combine the performance advantage of classical solutions and the flexibility of learning-based planners show promise for obstacle avoidance in autonomous racing.

2.5. Summary

The literature review started by studying optimisation-based approaches to full-scale and miniature autonomous racing [2, 29]. Optimisation-based approaches bring high-performance racing [30], with speeds of up to 8 m/s on F1/10$^{\text{th}}$ vehicles [37, 45]. However, classical methods are limited by being unable to avoid obstacles or react to map uncertainty and requiring computationally expensive localisation [13, 41].

Racing games have pioneered much DRL research with methods using camera images and hybrid state vectors as input. While DRL agents have demonstrated super-human performance in games [19], the methods used are not physically feasible. The two significant limitations are the lack of safety considerations, as the cars can crash into barriers in many games [20], and the assumption of accurate, explicit state variable estimation [54].

Real-world racing approaches have used fully end-to-end DRL architectures with camera images and LiDAR scanners and partial end-to-end configurations that have replaced part of the planning pipeline with a neural network. Using LiDAR scans as input to the DRL agent has produced better results than camera images and policies have been shown to transfer to other tracks [60]. Studying the performance of learning approaches showed that current methods have only been evaluated at low speeds of up to 5 m/s [9, 71], and even at these speeds, many approaches are unsafe [22, 23]. A significant problem identified in the learning literature was the lack of comprehensive metrics to analyse current behaviour; for example, while multiple reward signals have been used, there has been no quantification of the effect of the reward signal.

A study on safe methods for autonomous racing showed that while some recent approaches have recommended safety supervisors so that DRL agents can be trained online vehicles, there is little work in this direction [87]. While the safe learning literature in control systems was consulted, current methods have focused on learning linear systems with affine dynamics where a safe set is easily identifiable [83]. Viability Theory has been proposed as a feasible method to ensure vehicle safety on a race track and should be further investigated [2, 88]. The problem of obstacle avoidance has been poorly studied in the literature with few approaches considering it. Current approaches have noted that it is a difficult problem [6]. Investigation into hybrid navigation architectures showed that combining high-level classical planners with low-level DRL agents is a promising direction for obstacle avoidance in racing [99].

Chapter 3

Preliminaries: The F1/10th Platform

This chapter describes the preliminary information used as a basis for the rest of the work in this dissertation. The F1/10th platform is introduced with a description of the hardware and the simulation platforms. The Gym simulator that is used for a majority of the testing is described, and the ROS communication bridge is explained. The bicycle and single-track vehicle dynamics models are presented, along with the equations governing them. Aspects of the evaluation methodology that are general throughout the dissertation are described. The test maps are presented, and notes on their characteristics are given. The evaluation metrics are defined, and the classical baseline planner is explained. The classical methods for map preparation, optimal race line generation and pure pursuit path following are explained.

3.1. The F1/10th Platform

The F1/10th racing platform has been developed to promote research into autonomous control algorithms at the edge of performance. The platform consists of vehicles that are 1/10th the size of normal F1 cars (shown in Figure 3.1), and an accompanying, open-source,[1] high-fidelity simulator that allows for rapid algorithm development and deployment.

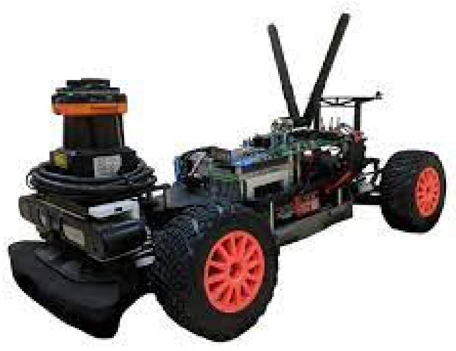


Figure 3.1: An F1/10th vehicle equipped with hardware for autonomous navigation.

The aim of designing a racing planner is to use the incoming data to generate a sequential set of drive messages that cause the vehicle to move around the race track as

[1]Available at: https://github.com/f1tenth/f1tenth_gym

quickly as possible without crashing. Planning is done in the velocity space of selecting a speed and steering angle and implemented using a control system that implements the references on the hardware.

Development Simulator

Most of the algorithms were developed in a sandbox simulator that used the kinematic bicycle model. The simulator was built in-house and kept simple to facilitate easy prototyping of planners. The sandbox simulator is simple and used in prototyping and not described in detail. This is the simulator that is used in the papers [25, 26] that were published.

3.1.1. Physical Vehicle Description

The cars use standard remote-control car chassis, with a drive motor to power the rear wheels and a servo to control the steering. The car is equipped with a light detection and ranging (LiDAR) sensor to sense the environment, and an inertial measurement unit (IMU), to detect its own movement. A Jetson TX2 computer runs the perception and planning algorithms, and an electronic speed controller (ESC) runs the low-level control system.

F1/10th cars use the Robot-Operating-System (ROS) software stack [105]. ROS is a communication platform that uses nodes (independent parts of code) that communicate with each other over topics. ROS provides the planner access to the incoming data from all of the sensors. The planner controls the vehicle by sending a message containing a reference steering angle (for the front wheels) and longitudinal speed.

3.1.2. Gym Simulator

The testing platform is an open-source Gym-style simulator. The simulator was originally built by a research group at the University of Pennsylvania [4, 34] and used in other research [9, 22, 60]. The simulator is built on the Gym-style environments [106]. The Gym environments are commonly used to develop and compare different DRL algorithms and feature a standard format of step, reset and render methods. At each timestep, the simulator takes an action containing steering angle and speed references and returns a state containing a LiDAR scan, position, orientation, velocity and steering angle.

This simulator uses a 7-dimensional state with the single-track bicycle model described in §3.2.2. The simulator includes a control system, similar to the one used on the physical vehicles, that transforms the steering angle and speed references into accelerations that the model can use as inputs. Planning happens at a frequency of 10 Hz, and the dynamics model is updated at a frequency of 100 Hz, meaning that for each planning step, 10 model updates are run. This simulator is the most widely used in this dissertation due to it being easy to use and providing a high degree of realism.

All of the simulator results are seeded so that the exact numerical results can be reproduced by running the code. For the results where the experiment is repeated, the random seed is updated to a new value for each test. The random seeds start at 1000 and are incremented by 10 for each consecutive experiment.

3.1.3. ROS Simulator

A robot-operating-system (ROS) simulator is used to bridge the Gym simulations and the physical vehicle. The ROS simulator runs the Gym simulation inside a Docker container and uses the ROS topic system to interact with the agent. The simulator uses the ROS topic system in exactly the same configuration as the car to make the simulation-to-reality transfer seamless. The ROS simulation allows for the particle filter that is used on the vehicle for localisation to be run in real-time to provide an accurate simulation of the localisation quality available on the vehicle.

Topic Name	Publisher	Type	Description
`/drive`	Planner	AckermannDrive	Steering and speed reference for the car to follow
`/scan`	LiDAR	LaserScan	1080 range beams from LiDAR
`/odom`	Particle filter	Odometry	Pose and Twist messages describing vehicle's position and movement

Table 3.1: List of ROS nodes used by the vehicle for planning.

Table 3.1 lists the main topics that are used for controlling the vehicle. The vehicle is controlled by publishing AckermannDrive messages to the `/drive` topic. The LiDAR scanner publishes a LaserScan message, that contains a 1080 beam scan. A particle filter is used for localisation that uses the LiDAR scans to estimate the vehicle's location. The particle filter publishes an Odometry message with the estimated pose and twist of the vehicle. The planner has access to the odometry and the laser scan.

3.2. Vehicle Modelling

This section presents two models used for F1/10$^{\text{th}}$ vehicles. The models are presented to help the reader understand the dynamics of racing and to lay the mathematical foundation for the solutions that are later developed. The single-track model is used in the simulator and is used to develop the safety system in §6.3. First, the kinematic bicycle model is presented to provide an intuitive understanding of the parameters used. The equations are then expanded to take the vehicle slip angle into account in the single-track model. This work is adapted from the models presented in [107] and described in detail in their online repository documentation.[2]

3.2.1. Kinematic Bicycle Model

The kinematic bicycle model has been used in previous autonomous racing literature [2], and is favoured due to its simplicity. The kinematic model is highly accurate at low speeds, but since tyre slip is neglected, loses some of its accuracy as the vehicle tends towards the tyre saturation point [2].

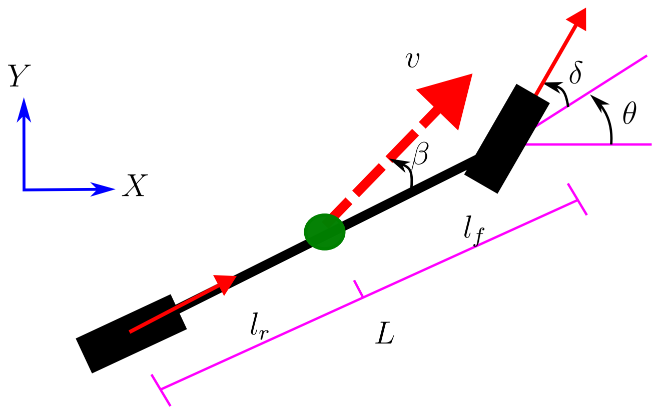


Figure 3.2: Kinematic Bicycle Model: $l_{r,f}$ are the rear and front axle to centre distances respectively, L is the wheelbase length, X and Y are the coordinate frame, v is the velocity, β is the slip angle, δ is the steering angle and θ is the orientation angle (yaw).

Figure 3.2 shows a schematic of the model with the parameters used. The wheelbase L is the sum of distances from the centre of gravity (CoG) to the front and rear axles, l_r, l_f. The inputs into the bicycle model are velocity v, and steering angle δ. The state of the vehicle consists of position in the X and Y directions and the orientation angle θ. The slip angle β is an intermediate quantity that describes the direction of the velocity at the CoG, relative to the orientation of the vehicle. In the kinematic bicycle model, β does not form part of the state and is assumed to be geometrically related to the vehicle and the steering angle; an assumption which is removed in the single-track model. The equations

[2]https://gitlab.lrz.de/tum-cps/commonroad-vehicle-models/-/blob/master/vehicleModels_commonRoad.pdf

describing the vehicle's dynamics are given by

$$\begin{aligned}
\beta &= \arctan(l_r \tan(\delta)/L), \\
\dot{X} &= v\cos(\beta+\theta), \\
\dot{Y} &= v\sin(\beta+\theta), \\
\dot{\theta} &= \frac{v}{L}\tan(\delta)\times\cos(\beta).
\end{aligned} \tag{3.1}$$

The X and Y positions are updated based on the velocity in the appropriate direction. The yaw rate is calculated using the steering angle and current velocity.

3.2.2. Single-track Model

The single-track (ST) model provides a more realistic approximation of the vehicle dynamics, specifically accounting for the acceleration and steering velocity and tyre slip. The ST model adds the four state variables of yaw-rate $\ddot{\theta}$, slip angle β, and steering velocity v_δ. The ST dynamic equations are

$$\begin{aligned}
\dot{X} &= v\cos(\beta+\theta), \\
\dot{Y} &= v\sin(\beta+\theta), \\
\dot{v} &= a_{\text{long}}, \\
\dot{\delta} &= v_\delta, \\
\dot{\theta} &= \dot{\theta}, \\
\ddot{\theta} &= \frac{\mu m}{I_z(l_f+l_r)}\Big(l_f C_f(gl_r - a_{\text{long}}h_{cg})\delta \\
&\quad + (l_r C_r(gl_f + a_{\text{long}}h_{cg} - l_f C_f(gl_r - a_{\text{long}}h_{cg}))\beta \\
&\quad - (l_r^2 C_r(gl_f + a_{\text{long}}h_{cg}) + l_f^2 C_f(gl_r - a_{\text{long}}h_{cg}))\frac{\dot{\theta}}{v}\Big), \\
\beta &= \frac{\mu}{v(l_f+l_r)}\Big(C_f(gl_r - a_{\text{long}}h_{cg})\delta \\
&\quad . - (C_r(gl_f + a_{\text{long}}h_{cg}) + C_f(gl_r - a_{\text{long}}h_{cg}))\beta \\
&\quad + (C_r(gl_f + a_{\text{long}}h_{cg})l_r - C_f(gl_r - a_{\text{long}}h_{cg})l_f)\frac{\dot{\theta}}{v}\Big) - \dot{\theta}.
\end{aligned} \tag{3.2}$$

In the equations above, h_{cg} is the vehicle's height to the centre of gravity, $C_{f,r}$ is the front and rear cornering stiffness, μ is the coefficient of friction, m is the mass and I_z is the moment of inertia. The standard parameters from the simulator that have been identified for F1/10th vehicles are used.

3.2.3. Model Comparison

The kinematic (KS) and single-rack (ST) models are now compared to each other. While being more complicated, the single-track model is known to result in more accurate behaviour [2].

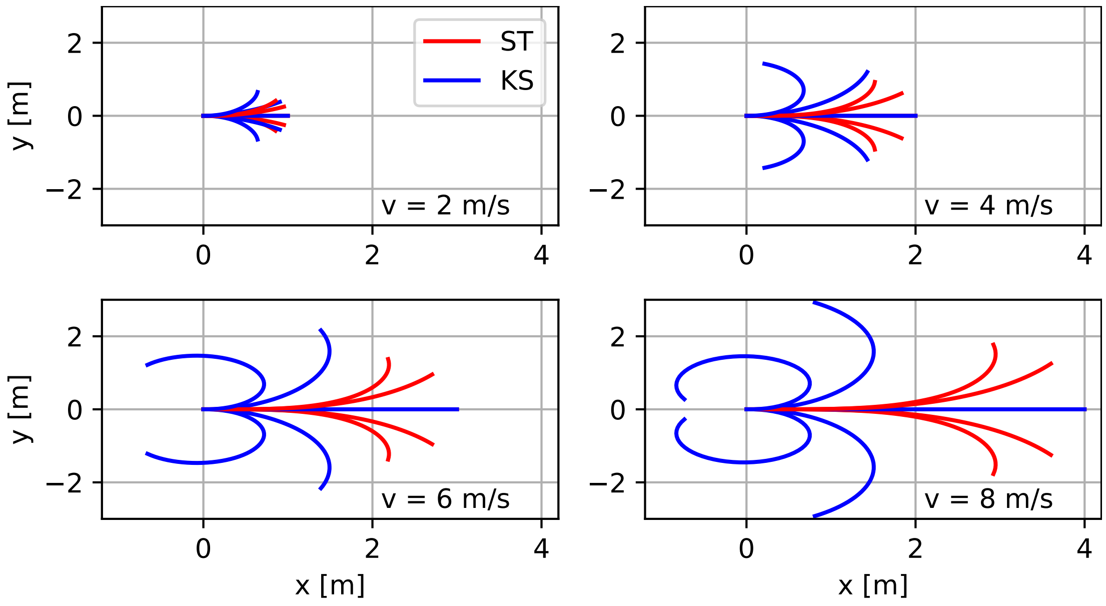


Figure 3.3: A comparison of the kinematic and single-track models at the speeds of 2, 4, 6, and 8 m/s for steering actions ranging from -0.4 to 0.4 radians.

Figure 3.3 shows the trajectories that the kinematics and single-track models produce, simulated for 0.5 seconds. The trajectories are generated by setting the initial state to have the speed used and the steering angle to 0. For the single-track model, the initial yaw rate and slip angle are set to 0. Actions of 5 different equally spaced steering angles are then fed into the model, and the resulting vehicle positions are plotted.

Figure 3.3 shows that at slow speeds, the kinematic model matches the single-track model accurately, but as the speed increases, the models deviate further. This means that to represent the dynamics at high speeds accurately, the single-track model must be used. The main source of deviation is the update of the vehicle orientation angle, which is why the positions differ so radically between the models.

3.3. Evaluation Methodology

The bulk of the evaluations presented in this work uses the Gym simulator. Therefore, the Gym simulator is described in detail. The general pattern that is followed is to run a training loop where the agent selects an action, the action is implemented in the simulator and the simulator returns a new state. The state transition storing, reward calculation and record keeping are all done by an intermediate class that controls both the vehicle and the simulator. All the code used in this work is available online and all the simulation results are repeatable. The results from Chapter 4 and 5 are in a single repository.[3] The supervisory safety system has its own repository,[4] as do the planners for obstacle avoidance.[5]

3.3.1. Map Data

Throughout the dissertation, a total of 7 different maps are used. Four of the maps are scaled versions of Formula 1 circuits, called AUT (Spielberg in Austria), GBR (Silverstone in Great Britain), MCO (Monaco), and ESP (Spain). The physical vehicle evaluation in §7.4 uses two maps that were used in the experiments, Levine (environment 1) and Lobby (environment 2). Chapter 8 focuses on obstacle avoidance and thus uses a more simple map called Columbia. The information about the four F1 maps is listed in Table 3.2.

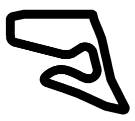
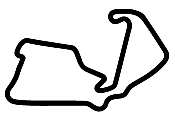
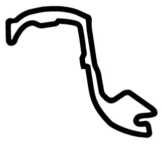
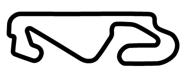
Metric	**AUT**	**GBR**	**MCO**	**ESP**
Length	93.7 m	202.2 m	178.3 m	236.8 m
Mean Curvature	0.16	0.14	0.16	0.12
Total Curvature	19.03	36.52	34.85	36.80
Max Curvature	0.76	0.87	0.98	0.90
Image				

Table 3.2: Length, curvature and image of the AUT, GBR, MCO and ESP maps used in testing.

Table 3.2 shows a table of each map used in training. The maps have been used in previous research [22, 60] and are repeated here for comparability. The table shows that the AUT map is the shortest at only 93.7m and the ESP map is the longest at 236.8m. The MCO map has the highest mean curvature, meaning that the map has the most turns relative to straight sections. The GBR and ESP maps have the highest total curvatures, indicating the greatest number of sharp corners. The MCO track has the most difficult corner with the highest curvature.

3.3.2. Evaluation Metrics

Throughout the dissertation, several important metrics are used to measure the performance of the vehicles. Here is a list of the important metrics, how they are measured and what they indicate:

[3]https://github.com/BDEvan5/RacingRewards
[4]https://github.com/BDEvan5/SuperSafety
[5]https://github.com/BDEvan5/hybrid_planners

- **Lap-time (seconds):** The time taken to complete a lap of a race track. This is the ultimate performance metric of racing.
- **Distance (meters):** The difference between the positions at each timestep. Useful to evaluate how far the vehicle drove compared to the track length to evaluate the path efficiency.
- **Curvature (radians per meter):** The difference between the vehicle's direction of motion at each timestep. The direction of motion is calculated from the gradient in between consecutive positions. In racing, lower curvatures allow for higher lateral accelerations and thus good racing performance will have low curvature.
- **Completion rate (percentage):** The number of successful (finished without crashing) laps as a percentage of the total number of test laps.
- **Average progress (percentage):** The mean of the vehicle progress when the vehicle either crashed or completed the lap. This is a useful metric for analysing how quickly a learning agent is training.
- **Average speed (meters per second):** The mean of the velocities at each planning step. This metric measures what the average was relative to the vehicle's maximum to show the vehicle's speed selection behaviour.
- **Average steering (radians):** The average of the absolute steering values. This metric indicates the normal behaviour of the planner in selecting steering actions.

3.3.3. Classical Planner

This section describes the classical planner that is used as a baseline for the work in this dissertation. The classical planner uses an offline global planner to calculate a globally optimal trajectory and a pure pursuit path follower to follow the trajectory as closely as possible. The results from the classical planner are treated as the theoretically best possible results.

Map Preparation

The first step of the optimisation planner is to prepare the map for optimisation. The map comes in the form of a binary file indicating free space or a boundary and a `.yaml` file indicating resolution and start location.

The first step of map preparation starts with finding the centre line. The centre line is found by taking the Euclidean distance transformation of the map. A manual gradient search algorithm is used to find the line of the lowest points on the map from the starting point. Once the centre line is found as a set of points, the normal vector direction is calculated by taking the line perpendicular to that joining the centre points. The normal vector is stored as a unit vector indicating the direction. Finally, the track widths are found by searching at set intervals along the normal direction until the boundary is reached.

After this, the map is stored as a 6D set of points as x, y of the centre line, normal vectors and track widths at each point. The left image in Figure 3.4 shows what a track that has undergone this procedure looks like.

Trajectory Optimisation

Once the map has been formulated as a set of points, then the optimisation routine is set up. The optimisation routine is performed in two steps to ensure a time-efficient solution. The routine described here was presented in [13] and consists of generating a minimum curvature path and then calculating a speed profile for the path. In the experiments, the original code version is used with minor modifications.

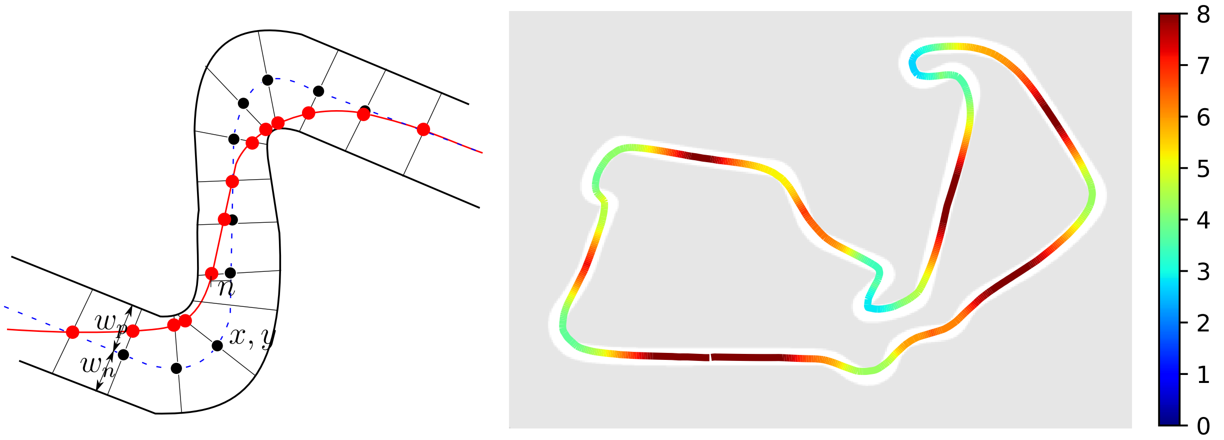


Figure 3.4: Left: Race track model comparing the centre line (black) and minimum curvature line (red). Right: An optimal trajectory for the GBR map with the speed represented by the colour bar in m/s.

The minimum curvature optimisation finds the path with the lowest curvature by selecting a distance along each normal vector for each track point. The centre line is represented as a set of third-order splines, and a quadratic program is formulated to find the path with the lowest total curvature. An approximate formulation of the optimisation is presented as,

$$\begin{aligned} \min_x : f(x) &= \Sigma_i(\theta_{i+1} - \theta_i)^2 \\ \text{s.t.} : w_{\text{left},i} &\leq n_i \leq w_{\text{right},i} \\ \theta_i &= \arctan\left(\frac{y_{i+1} - y_i}{x_{i+1} - x_i}\right). \end{aligned} \tag{3.3}$$

Once a minimum curvature path has been found, the next step is to generate a velocity path using a model of vehicle kinematics. A forward-backwards solver is used to generate a velocity profile subject to the vehicle's acceleration limits and the physical friction limit.

The result of these two steps is an ordered set of way points consisting of an x and y position in the global coordinate frame and the velocity at each point. This trajectory is referred to as the racing line. The trajectory is visualised for the GBR map in Figure 3.4, showing the minimum curvature line, with the speeds calculated by the optimiser shown using different colours.

Pure Pursuit Path Following

A pure pursuit path follower, which navigates towards a waypoint situated at a fixed look-ahead distance in front of the vehicle, is used as a simple means of following a precalculated trajectory [36]. Figure 3.5 shows how the vehicle and waypoints are modelled so that the steering angle can be calculated.

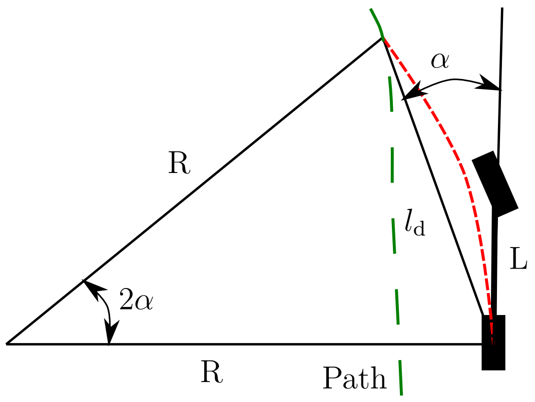


Figure 3.5: Pure Pursuit Path Follower: R is the turning radius, g is the goal point, α is the angle to the goal and δ is the steering angle which is calculated.

Using the notation from Figure 3.5, the desired steering angle is calculated according to,

$$\delta_{\text{pp}} = \arctan\left(\frac{2L\sin(\alpha)}{l_{\text{d}}}\right), \tag{3.4}$$

where L is the length of the vehicle body, l_{d} is the look-ahead distance to the waypoint being pursued, and α is the angle between the vehicle's current heading and the heading to the waypoint. The pure pursuit planner uses the speed at the upcoming waypoint as the speed reference to control the vehicle.

3.4. Summary

This chapter presented the preliminary information used to build the solutions developed in later chapters. An overview of the F1/10th platform was presented, detailing the hardware on the vehicle and simulation platforms. The Gym simulator provides fast development and the ROS simulator provides a useful bridge between development and physical testing. The kinematic and single-track vehicle dynamics models were presented. General aspects of the evaluation methodology were described. The maps used in testing were presented and their attributes recorded, the evaluation metrics were defined and the classical baseline was detailed. The classical methods for map preparation, optimal race line generation and pure pursuit path following were explained.

Chapter 4

Evaluation of DRL for F1/10$^{\text{th}}$ Racing

This chapter evaluates current reinforcement learning formulations for F1/10$^{\text{th}}$ racing. The evaluation fulfils three roles; (1) explaining how DRL is formulated for F1/10$^{\text{th}}$ racing, (2) investigating how well current learning formulations perform, thus providing a baseline for future results, and (3) providing novel insights into how DRL agents perform for racing, such as comparing reward signals. The constant speed evaluation shows that the cross-track and heading reward outperforms the progress and standard reward regarding training consistency and accuracy in tracking the centre line, while the progress reward selects shorter smoother paths. The variable speed investigation shows that current learning formulations are unsuitable for racing at speeds above 5 m/s due to the high crash rate. The specific problem identified is that the agent does not learn to select an appropriate speed profile but only selects the maximum speed and thus drifts around the track with a high slip angle.

4.1. Introduction

Reinforcement learning is the task of training an agent to perform a task from experience. Deep reinforcement learning using neural networks has shown to be an effective method for controlling racing games and physical vehicles at low speeds. The challenge of training an agent is to transfer knowledge, via a reward signal, to the agent so that it can map an input state vector to an output action that results in fast, safe racing behaviour.

Currently, there is a lack of in-depth studies on the behaviour learned by the agent. While different studies have rewarded the agent for progress [19] or proportionally to the cross-track and heading error [52], there has been no direct comparative study. DRL agents for F1/10$^{\text{th}}$ racing have not been evaluated at speeds above 5 m/s, and there has been little work studying the effect of the maximum speed or the velocity profiles.

This chapter presents and evaluates current learning formulations for F1/10$^{\text{th}}$ racing. An introduction to reinforcement learning is presented, followed by an explanation of how it is applied to the task of racing. Most of the chapter contains an in-depth evaluation of end-to-end DRL for autonomous racing and constant and variable speed. The evaluation compares the progress and cross-track and heading reward signals, measures the curvature of the trajectories taken, considers the effect of maximum vehicle speed on performance, and plots the speed profiles. The study concludes by highlighting the limitations of current learning formulations.

4.2. Reinforcement Learning

Reinforcement learning (RL) is a convenient method for training agents because it can generate its own samples (learn from experience), and the only knowledge it requires is a reward signal [50]. RL uses the agent's policy to collect experience to train the agent to produce the desired behaviour. The work in this section is presented as preliminary information and is adapted from recent developments in the field, such as [98, 108].

4.2.1. Definitions

A reinforcement learning problem is modelled as having an agent that receives a state and selects an action in an environment. After each action has been implemented, the environment returns a reward indicating how good or bad the action was. The agent is trained to select actions that maximise the amount of reward received.

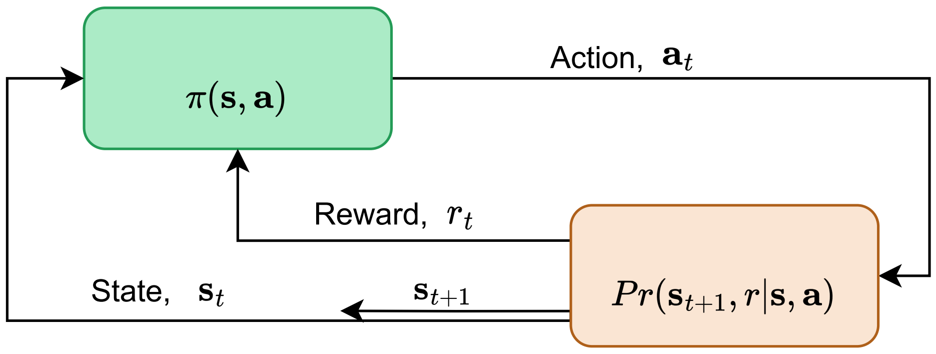

$\pi(\mathbf{s},\mathbf{a})$
Action, $\mathbf{a}_t$
Reward, r_t
State, $\mathbf{s}_t$
$\mathbf{s}_{t+1}$
$Pr(\mathbf{s}_{t+1}, r|\mathbf{s},\mathbf{a})$

Figure 4.1: RL agents select an action that is implemented in the environment, and a new state and reward are returned to the agent.

Figure 4.1 shows the generic layout of the components used for training an RL agent. At each timestep, t the agent observes the state of its environment $\mathbf{s}_t$ from the state space $\mathcal{S}$, which the agent uses to select an action. The agent uses its policy $\pi(\mathbf{s},\mathbf{a})$, to select an action $\mathbf{a}_t$ from the action space $\mathcal{A}$. The agent's policy is the probability of selecting an action, based on being in a certain state, written as $\pi(\mathbf{s},\mathbf{a}) = Pr(\mathbf{a}_t = \mathbf{a}|\mathbf{s}_t = \mathbf{s})$. Environments are used with stationary transition dynamics (a fixed probability density) of a next state and reward based on a current state and action combination. After each step, the agent receives a reward to encourage or punish the selected action in the current state according to a reward function $r : \mathcal{S} \times \mathcal{A} \rightarrow \mathcal{R}$.

The RL problem is finding the policy that maximises the expected discounted sum of future rewards when followed. The agent's policy is written as π_θ, with θ referring to the policy parameters. The expected discounted sum of future rewards is written as $G_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+2} + \gamma^3 r_{t+4} + \ldots$. The return can be rewritten in terms of the reward and return of the next timestep as, $G_t = r_{t+1} + \gamma G_{t+1}$. The discount factor $\gamma \in [0, 1)$ determines how much the agent favours immediate reward ($\gamma = 0$) as opposed to future reward (as γ approaches 1).

An action-value function $q^{\pi}(\mathbf{s}, \mathbf{a})$ is defined as the expected value of the discounted return, by taking an action $\mathbf{a}$ from state $\mathbf{s}$ under the policy π_θ. The action-value of the state-action pair is used to improve the policy via policy improvement.

Deep reinforcement learning uses a deep neural network to represent the policy. Deep neural networks use sets of neurons (layers) to calculate an output for a given input. For input to a given layer $x \in \mathcal{R}^n$, the output of the layer is calculated as $\mathbf{y} = \sigma(W \times \mathbf{x} + \mathbf{b})$, where σ is the activation function W are the layer weights, and $\mathbf{b}$ are the layer biases. Deep neural networks using multiple layers connect the outputs of one layer to the inputs of the following layer. The parameters of the network are the weights and biases of each layer that determine how the inputs map to the outputs.

4.2.2. Policy Based Reinforcement Learning

In this work, policy-based methods are described since they can select outputs for continuous control [109]. Specifically, deep deterministic policy gradient (DDPG) methods are described [110].

Deterministic policy gradient algorithms are part of the actor-critic algorithm family that uses two networks, an actor and a critic. The critic network, $Q(\mathbf{s}, \mathbf{a}|\theta^Q)$ is used to approximate the action-value function q^{π} (the Q-value of a given state-action pair). The actor-network, $\mu(\mathbf{s}|\theta^{\mu})$, is the policy network that maps a state to an action.

A standard method to improve the stability of neural networks during training of using separate model (or online) and target networks is employed. The model network is used to select actions and is updated during training. The target networks, denoted by parameters $\theta^{Q'}$ and $\theta^{\mu'}$, are changed using a soft update to slowly track the model networks during training, written as,

$$\theta^{Q'} \leftarrow \tau\theta^{Q} + (1-\tau)\theta^{Q'} \quad \text{and} \quad \theta^{\mu'} \leftarrow \tau\theta^{\mu} + (1-\tau)\theta^{\mu'}.$$

The updates use a temperature parameter where $0 < \tau \ll 1$.

Experience is collected by allowing the agent to interact with the environment and storing the collected state transitions in a replay memory. During the sample collection phase, random policy noise is added to the action selected by the network to ensure adequate exploration. After every step, the networks are trained by sampling mini-batches of N transitions from the buffer. For each transition, i, the improved estimate of the action-value function is calculated using the target networks by

$$y_i = r_i + \gamma Q'(\mathbf{s}_{i+1}, \mu'(\mathbf{s}_{i+1}|\theta^{\mu'})|\theta^{Q'}). \tag{4.1}$$

The online critic network is updated by minimising the mean squared error (loss, L) between the target estimate of the action-value function and the current estimate of the Q-value:

$$L(\theta^Q) = \frac{1}{N}\sum_i (y_i - Q(\mathbf{s}_i, \mathbf{a}_i|\theta^Q))^2. \tag{4.2}$$

The policy network (actor) is trained to maximise the expected return estimated by the critic network, $J(\theta) = \mathbb{E}[Q(\mathbf{s}_t, \mathbf{a})|_{\mathbf{a}=\mu(\mathbf{s}_t)}]$. The gradient of the policy's performance is obtained by applying the chain rule to calculate the derivative of the objective function with respect to the policy parameter as proven in [109]:

$$\nabla_{\theta^\mu} J(\theta) \approx \nabla_{\mathbf{a}} Q(\mathbf{s}_t, \mathbf{a}|\theta^Q)|_{\mathbf{a}=\mu(\mathbf{s}_t)} \nabla_{\theta^\mu} \mu(\mathbf{s}_t|\theta^\mu). \tag{4.3}$$

The policy network is updated by calculating the mean of the policy gradients in the mini-batch:

$$\nabla_{\theta^\mu} J(\theta) \approx \frac{1}{N}\sum_i \nabla_{\mathbf{a}} Q(s_i, \mathbf{a}|\theta^Q)|_{\mathbf{a}=\mu(\mathbf{s}_i)} \nabla_{\theta^\mu} \mu(\mathbf{s}_t|\theta^\mu), \tag{4.4}$$

to perform stochastic gradient ascent (to maximise the expected return).

The twin-delayed deterministic policy gradient algorithm (TD3), an improved version of DDPG, is used in this work [108]. As the name implies, the TD3 algorithm uses twin value networks, of which the smallest value is used in training the policy gradient. Selecting the smallest of two networks helps to prevent over-approximation bias. An additional improvement is that they delay policy updates compared to value updates. Thus, the actor-network is trained half as much as the critic network, meaning that for every training step, two mini-batches are sampled to train the critic, while only one mini-batch is used to train the actor.

The experiments all use neural networks with two hidden layers of 100 neurons each. The *ReLU* activation function is used after each hidden layer and the *tanh* function for the output layer to scale the output to the range [-1, 1].

4.3. Racing Learning Formulation

This section describes how the learning is formulated for the problem of autonomous racing. The state and action vectors are defined and motivated, followed by an extended presentation on reward signals for racing.

4.3.1. State and Action Vectors

End-to-end learning involves replacing the entire processing pipeline with a learning agent. Therefore, the sensors and actuators on the vehicle define the state and action spaces. The sensor readings are used as input to the agent (the state), and the agent must output actions that can be implemented on the vehicle.

State Space

The agent uses raw LiDAR scans as input into the neural network. F1/10th vehicles are fitted with Hokuyo or Sick LiDAR scanners that provide a 1080 beam scan with a field of view of $3/2\ \pi$ radians. For the state vector, a slice of 20 evenly spaced beams with a field of view of π radians is used, similar to other work [61]. A tuning test was done on the number of beams and showed that using less than 15-20 beams results in poorer performance while using more than 20 beams increases the time required to train the network but does not change the performance.

Several LiDAR scans are stacked together so that the agent can determine its motion. This is done because no explicit vehicle quantities (i.e. position, speed, etc.) are calculated or given to the agent. This is in contrast to many approaches in racing games where detailed vehicle measurements and upcoming track waypoints are given to the agent [19]. A tuning test showed that stacking two state vectors together is enough for the agent to learn effective policies. Stacking more states together did not show any differences.

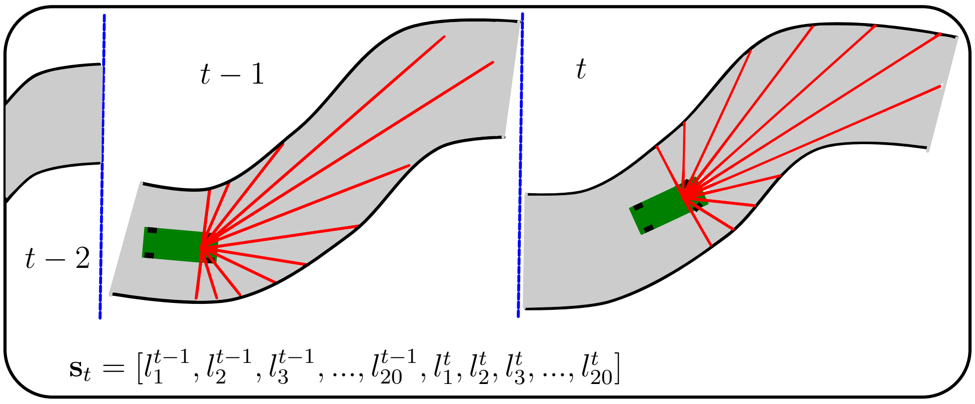

$t-1$
t
$t-2$
$\mathbf{s}_t = [l_1^{t-1}, l_2^{t-1}, l_3^{t-1}, ..., l_{20}^{t-1}, l_1^t, l_2^t, l_3^t, ..., l_{20}^t]$

Figure 4.2: Illustration of two consecutive states with the LiDAR vectors stacked together.

Figure 4.2 shows graphically how the LiDAR scans are stacked together. Each beam has a maximum range of 10m, which is why some stop when not touching a wall. Each beam is scaled according to the maximum, resulting in a value between 0 and 1 used as input into the neural network. The state used by the agent is written as $\mathbf{s} \in [0, 1]^{40}$.

Action Space

Two forms of racing are considered; constant speed and variable speed racing. Constant speed racing uses the agent to output the steering angle and a constant speed for the vehicle. Variable speed racing uses two control variables of steering angle and longitudinal speed. All outputs from the neural network are in the range $[-1, 1]$ and are scaled before use. The steering action is scaled according to the maximum steering angle as $\delta = a[0] \times \delta_{\text{max}}$. The speed action is scaled to the range $[0, v_{\text{max}}]$ using the formula $v = (a[1] + 1)/2 \times v_{\text{max}}$. In the above formulas, a represents the output value from the neural network and δ and v are the action quantities that are executed to control the vehicle.

4.3.2. Reward Signal

As explained in §4.2, RL algorithms train the agent to maximise the reward signal. Therefore, the reward signal must accurately convey the desired behaviour to the agent. Many approaches to DRL observe that it is difficult to define an appropriate reward signal [53, 60].

The primary desired behaviour that must be communicated to the agent is not to crash but complete laps of the race track. This behaviour is encoded in an equation called the *standard* reward, as a positive reward for completing a lap r_{complete} and punishment for crashing r_{crash}, written as,

$$r_{\text{standard}} = \begin{cases} r_{\text{crash}} = -1 & \text{if crash} \\ r_{\text{complete}} = 1 & \text{if lap complete.} \end{cases} \tag{4.5}$$

Due to the sparsity of the standard reward, shaped intermediate rewards are used to aid the learning process. In the literature, the progress and cross-track and heading rewards have been widely used, but no comparison has been done.

Progress Reward Signal

The first racing reward considered is to use an equation relative to the progress the vehicle has made along the track, as used in [19, 22]. The progress reward uses the progress made by the vehicle along the track centre line at each timestep. The difference in progress between the current and previous timesteps is scaled according to the track length and used as a reward.

The progress is measured by projecting the vehicle's position onto the centreline. We write the *progress reward* as,

$$r_{\text{progress}} = \beta_{\text{distance}} \frac{(p_{\text{t+1}} - p_{\text{t}})}{L_{\text{track}}} \tag{4.6}$$

where p_t is the progress along the track at time t, and L_track is the total length of the track. The hyperparameter β_distance is the amount that the progress reward is scaled by. In this work, β_distance is set to a value of 100. Figure 4.3, shows the progress using colours, starting with blue to represent the start, and moving through the spectrum until the lap is complete.

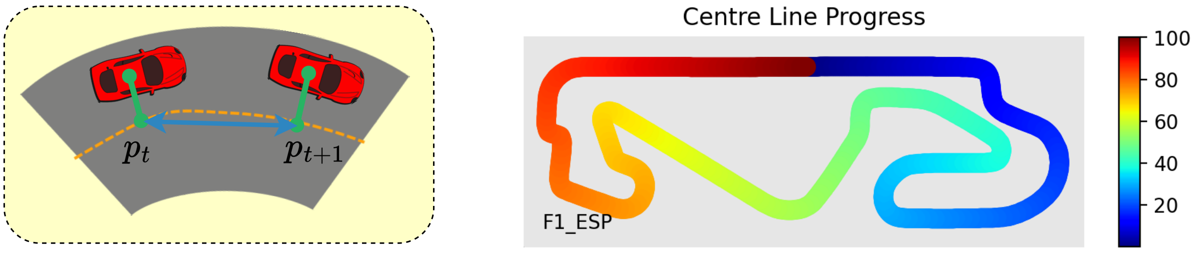
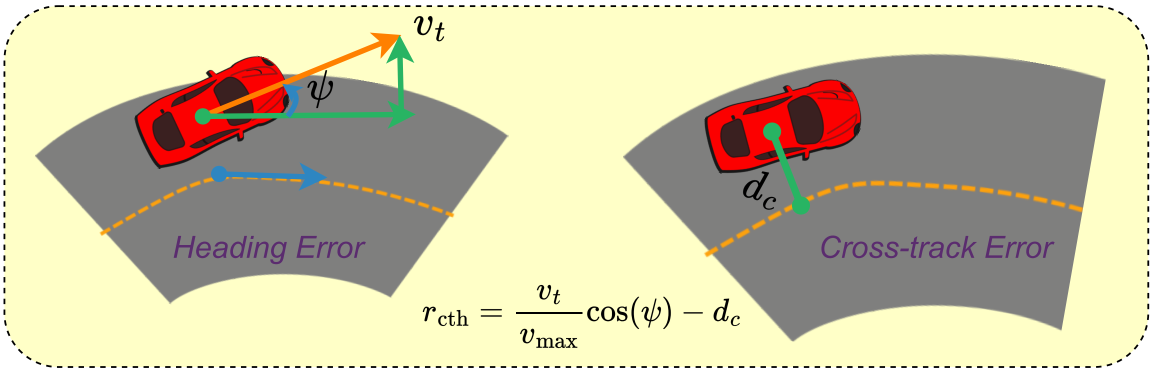


Figure 4.3: Left: Illustration of how the progress is measured. Right: The cumulative centre line track progress is shown by colours ranging from 0-100%.

The progress reward is a good measure of the progress made through the race track, with explicit encouragement for the agent to make as much progress as possible at each timestep. However, a limitation of the progress reward is that the total reward is always 1 (or β_progress) for a completed lap. The fact that the total reward always sums to 1 means that the progress reward does not quantify how well the agent drove the lap.

Cross-track and Heading Rewards

The second reward that is considered uses the vehicle's velocity in the direction of the race track and the vehicle's lateral deviation from the centre line to reward the agent, as used in [52, 61]. The vehicle's velocity is a good variable to use since velocity is the derivative of position and thus contains information about where the vehicle will be at the next timestep.



v_t
ψ
Heading Error
d_c
Cross-track Error
$r_\mathrm{cth} = \frac{v_t}{v_\mathrm{max}} \cos(\psi) - d_c$

Figure 4.4: Illustration of how cross-track distance, d_c, and heading error, ψ, are measured.

Figure 4.4 shows how the cross-track distance, d_c, and heading error, ψ, are measured.

The velocity along the line is calculated according to the cosine of the difference in angle between the vehicle and the track centre line. The distance from the centre line is measured perpendicular to the centre line heading. The cross-track and heading reward is written as,

$$r_{\mathrm{cth}} = \frac{v_{\mathrm{t}}}{v_{\max}} \cos\psi - d_{\mathrm{c}}, \tag{4.7}$$

where v_{t} is the speed of the vehicle, ψ is the heading error and d_{c} is the cross-track error. The velocity is scaled according to the vehicle's maximum velocity.

This reward has the advantage of directly encouraging the agent to select the maximum speed possible. It also encourages the agent to follow the centre line closely by penalising the agent for moving away. It is expected that this reward will cause the agent to follow the reference path at the highest velocity possible.

4.4. Constant Speed Reward Comparison

The learning formulation is evaluated at a constant speed to highlight the effect of the path selected by the reward signals, detached from the speed considerations. The study consists of studying the training behaviour shown by the different reward signals, comparing the performance of the trained agents and measuring how well the agents generalise to other tracks. For the constant-speed evaluation, the speed is kept at 2 m/s since the friction is negligible at this speed.

4.4.1. Training

An investigation into the training considers how effectively the agents learn. The constant speed agents are trained for 30,000 training steps, during which policy noise is added to the action to encourage exploration. To understand the training behaviour, the total reward earned by the agent per episode is plotted against the training steps and the episodes. The episode begins with the agent at the starting position on the track and ends when the vehicle crashes or completes a lap.

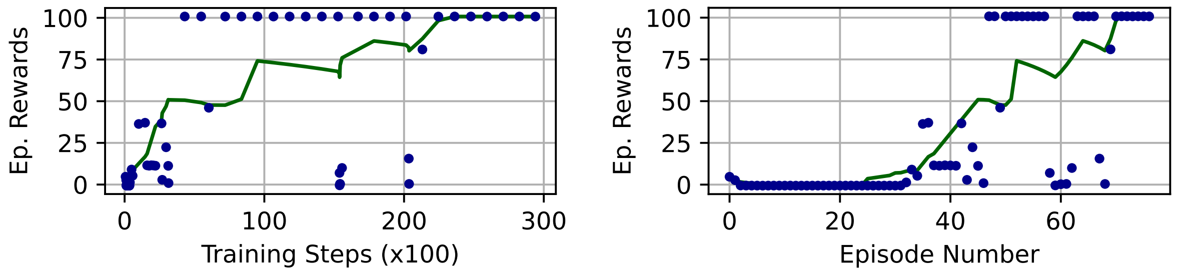


Figure 4.5: The rewards from the progress reward signal, trained on the MCO map, are plotted against the training steps (left) and the episode number (right).

Figure 4.5 shows the rewards earned by the progress agent on the MCO map. The agent starts by crashing quickly and earning little reward. Over time, the agent learns to complete more of the race track and, after several thousand steps, can drive around the entire track successfully. The rewards plotted against the number of episodes (right-hand graph) shows that the agent crashes many times before learning how to drive without crashing.

The effect of the reward signal on the training is studied by training agents with the standard, progress and cross-track and heading reward signals on four maps. Since the quantity of reward earned by the different agents is difficult to compare, the average progress made by the agents is used to compare the different reward signals. The agent's progress before crashing is the best indicator of how well it performs during training because it can be directly compared across various reward signals and provides a reliable assessment of the agent's performance. Experiments that converge to less than 35% average progress are neglected from the analysis. The reason for doing this is that some of the experiments did not converge and thus had extremely poor values. Including these outliers greatly skews the results, leading them to be inaccurate.

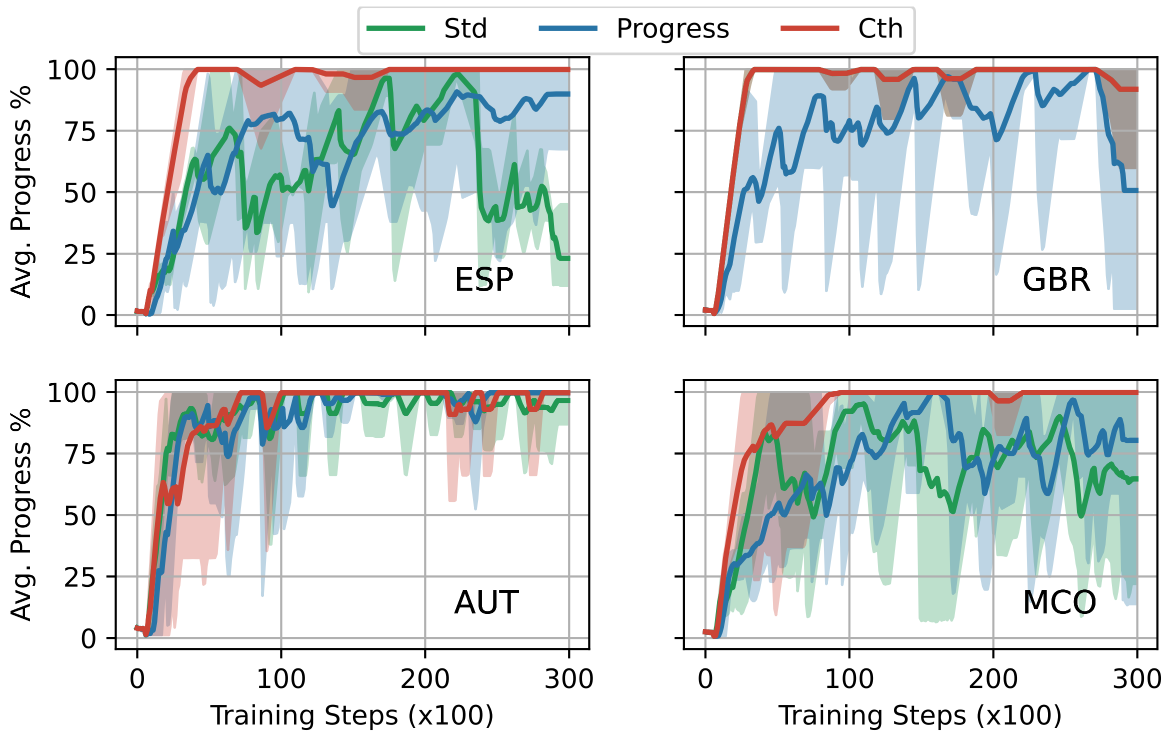


Figure 4.6: Training progress using the standard (Std), progress and cross-track and heading error (Cth) rewards on the ESP, AUT, GBR, and MCO tracks. The lines are the average from 5 runs, with the shaded area representing the minimum and maximum from all the experiments.

Figure 4.6 shows the progress made by the standard, progress and cross-track and heading reward signals on the ESP, AUT, GBR, and MCO tracks. The graph shows that for some of the simpler tracks, such as AUT, all the reward signals effectively train agents to complete most of the laps, achieving average progress of near 100%. For the ESP and MCO maps, which are more complicated, with many sharp corners, the progress and standard reward signals do not converge well, taking many dips and never rising to 100%. In contrast, the cross-track and heading reward results in the agents reaching near 100% average progress on all four test maps.

The standard and progress rewards are considered to be highly dependent on the map that they are trained on. This reliance on map structure makes them poorly suited to learning general racing policies because they only work in certain settings. The cross-track and heading reward demonstrates the ability to train agents to race on all maps to almost always complete the laps. Therefore the cross-track and heading reward is the most suited to the racing problem.

4.4.2. Performance

The performance of agents trained with each reward signal is evaluated by testing the agents on the track they were trained on. The metrics of path distance, curvature and deviation from the centre line are used to analyse the behaviour. The results are compared to a pure pursuit planner following the centre line and driving at the same speed. The results are the average of 10 test laps run on each track.

Metric	**Std**	**Progress**	**Cth**	**PP**
Total Distance (m)	94.51	91.98	94.50	93.57
Mean Curvature (m^{-1})	0.39	0.36	0.42	0.17
Total Curvature (m^{-1})	184.72	167.33	200.77	80.62
Mean Deviation (m)	0.38	0.16	0.07	0.04
Total Deviation (m)	178.55	74.54	34.63	18.86

Table 4.1: Distance travelled, curvature and deviation from the centre line of agents trained with the standard (Std), progress and cross-track and heading error (Cth) reward signals on the AUT map, compared to the pure pursuit (PP) planner following the centre line.

Table 4.1 compares the metrics of distance travelled, curvature and deviation from the centre line of the pure pursuit planner and agents trained with the standard, progress and cross-track and heading rewards. The standard and cross-track and heading agents took a longer path around the track of 94.5 m in comparison to the pure pursuit planner of 93.57 m. The progress agent took a shorter path, even than the pure pursuit planner, of 91.98 m, suggesting that the progress reward trains agents to select shorter paths.

Figure 4.7 shows a section of the MCO track with several corners and the paths taken by each agent. This figure confirms that the progress reward travels less distance in completing a lap because it learns to cut corners consistently. The standard agent does not

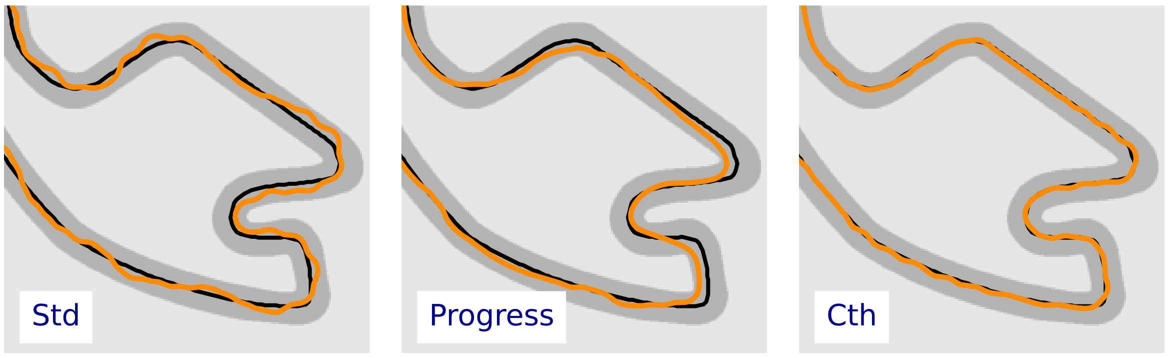


Figure 4.7: A segment from the paths taken by agents trained with the standard (Std), progress and cross-track and heading error (Cth) reward signals on the MCO track.

follow any specific pattern between staying on the centre line, cutting the corner narrow or driving outside the corner. The cross-track and heading agent normally drives on the centre line, as demonstrated by the orange trajectory covering most of the black centre line.

The deviation from the centre line is calculated by measuring the perpendicular distance from the centre line to the vehicle at each timestep. The deviation metric assesses where the vehicle is driving on the track. The cross-track and heading agent's driving behaviour on the centre line is demonstrated in Table 4.1, by the cross-track and heading planner having the smallest total deviation of 34.6m. The cross-track and heading planner performs slightly worse than the pure pursuit planner, suggesting that the cross-track and heading reward signal can train agents to follow the centre line accurately. The progress agent has a deviation of 74.54, and the standard agent has a massive deviation of 178.55. Figure 4.7 shows that the progress agent has a larger deviation because it learns to cut corners and maximise progress on the track without reference to the deviation. The standard agent follows no distinct pattern apart from not crashing.

The curvature is the difference in the heading angle of the vehicle at consecutive timesteps, divided by the distance travelled during the interval. The total curvature measures how much the vehicle turned during the lap. The DRL agents all have significantly higher mean curvatures of 0.39, 0.36, and 0.45 between planning steps, compared to the pure pursuit planner's average of 0.17. Of the DRL planners, the progress agent has the lowest mean curvature and the lowest total curvature of 167, which is 7% smaller than the standard agent. This problem of high-curvature paths, known as slaloming, can be visually seen in the path in Figure 4.7, and has been noted by other studies [22].

The performance study concludes that both the progress and cross-track and heading rewards improve upon the standard reward signal. The cross-track and heading reward showed to train agents to convergence with the greatest sample efficiency and demonstrated the best ability to track the centre line used as a reference. The standard and cross-track and heading agents had higher curvature due to the problem of slaloming, swerving from side to side. The progress reward selected more moderate actions, slaloming less and cutting corners, resulting in finding a short path around the race track.

4.4.3. Generality

The robustness of the planners to different tracks is measured by training agents on the ESP track and then evaluating them on all of the tracks. A reduced set of metrics is used and averaged across 10 test laps on each track. Table 4.2 presents the results from the evaluation for standard, progress and cross-track and heading reward signals.

Map	Metric	Std	Progress	Cth	PP
AUT	Distance m	NAN	90.21	93.87	92.74
	Curvature rad/m	NAN	143.06	213.13	103.89
	Deviation m	NAN	123.99	31.16	28.63
	Progress %	32.03	100.00	100.00	100.00
MCO	Distance m	175.43	172.25	178.73	176.07
	Curvature rad/m	392.39	296.92	475.89	181.16
	Deviation m	224.58	231.67	69.85	47.88
	Progress %	100.00	100.00	100.00	100.00
GBR	Distance m	198.59	200.95	203.64	200.48
	Curvature rad/m	462.38	374.56	505.09	190.29
	Deviation m	264.06	206.53	71.42	44.90
	Progress %	100.00	100.00	100.00	100.00
ESP	Distance m	235.88	230.58	238.55	235.51
	Curvature rad/m	550.85	279.45	525.43	192.54
	Deviation m	309.40	303.94	67.93	44.17
	Progress %	91.38	100.00	100.00	100.00

Table 4.2: Distance travelled, curvature and distance from the centre line of agents trained on the ESP map and tested on the AUT, GBR, MCO and ESP maps.

Table 4.2 show that DRL agents trained on one map can transfer to a different map and complete laps. The agent trained with the standard reward signal could not complete the sharp turn on the AUT map, but the agents trained with the progress and CTH rewards could complete all the test laps on the test maps. The "NAN" values in the table mean that these values could not be calculated. This shows that the standard reward signal is unsuitable for training agents that can transfer to other tracks.

The same patterns identified in the agent's performance on the training map are seen on different maps. The progress reward signal leads to the agent taking a shorter path around the track. For all of the maps considered, the progress agent takes a shorter path than the CTH agent and often than the pure pursuit agent. Figure 4.8 shows an example trajectory of the progress agent on the different maps.

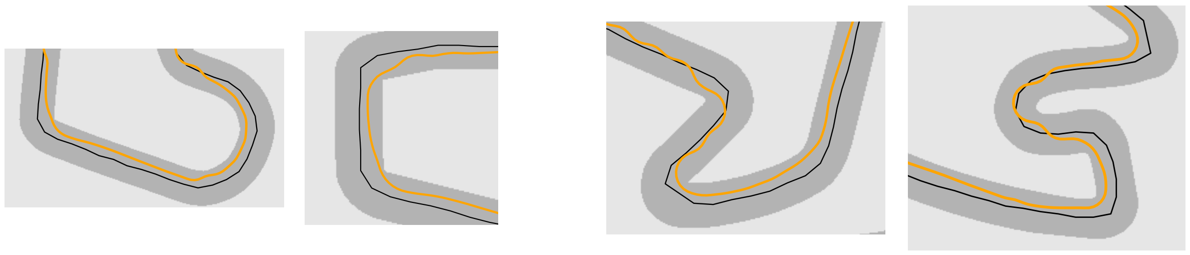


Figure 4.8: Path segments from agents trained with the progress reward cutting corners on the ESP, AUT, GBR and MCO tracks.

The cross-track and heading agent can track the centre line of the other tracks tested. This is shown by the cross-track and heading agent having a significantly smaller deviation than the progress agent. The cross-track and heading agent deviation is still higher than the pure pursuit, but it is much closer than the other reward signals. Figure 4.9 shows the cross-track and heading agent's path overlaid on top of the center line. It can be seen that the agent tracks the centre line closely.

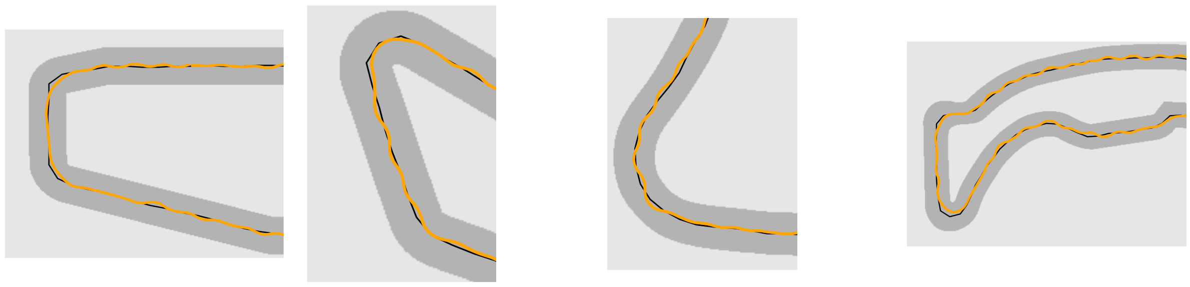


Figure 4.9: Agents trained with the cross-track and heading reward signal tracking the centre line accurately and showing slaloming behaviour on segments of the AUT, ESP, GBR, and MCO tracks.

The problem of slaloming shown by the standard and cross-track and heading reward signals is also present on other tracks. The total curvature measurements are significantly higher for the cross-track and heading reward signal than the progress reward signal for all tracks. Figure 4.9 demonstrates this behaviour by showing track segments from the different tracks with the agent swerving from side to side.

The study on reward signals at constant speed concludes that the cross-track and heading reward trains agents to drive with better sample efficiency, more consistently, and to higher average progress than the standard and progress rewards. The performance evaluation showed that agents trained with the progress reward signal select shorter paths, often cutting the corners and slaloming less. Agents trained with the cross-track and heading reward track the centre line accurately, but have a higher curvature, often swerving side to side. The standard reward signal produces unpredictable behaviour, with the agent sometimes swerving, cutting the corners or driving straight. Their behaviours were validated to transfer to different maps other than the training map.

4.5. Variable Speed Evaluation

Previously, the rewards were compared at a single speed to demonstrate the different paths taken. The dimension of speed is now added, making the racing problem more difficult since the agent must select two control signals. The evaluation compares the progress and cross-track and heading reward signals before investigating how the maximum speed affects the training. Since selecting speed and steering is a more complicated task, agents are now trained for 100,000 steps.

4.5.1. Reward Signal Comparison

Standard Reward Signal

The standard reward signal is only briefly considered since it demonstrates terrible racing performance. The training graph and an example trajectory are shown in Figure 4.10. The training graph shows that the agent learns to complete laps of the track early in the training process. The trajectory shows that the policy learned is to select slow speeds that are overly conservative, and thus the agent does not crash.

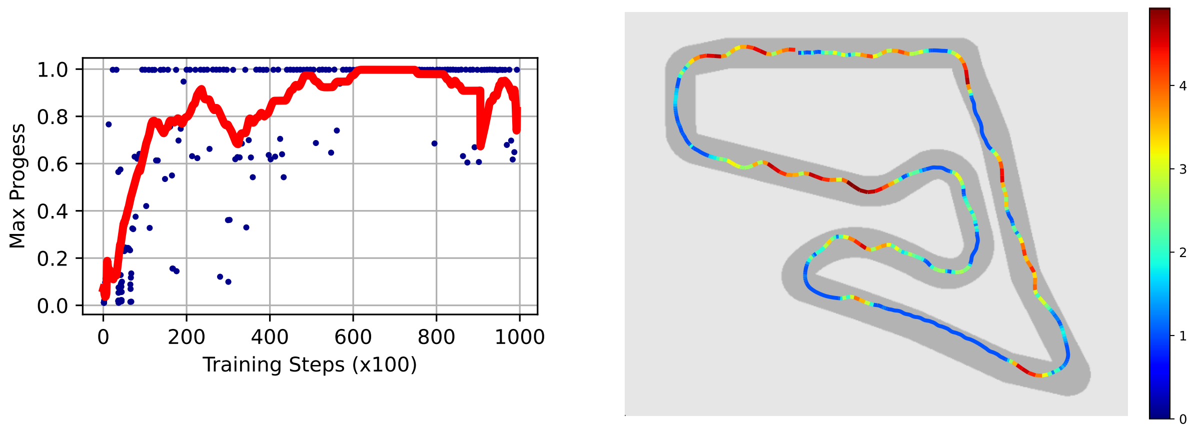


Figure 4.10: The vehicles progress around the track during training (left) and an example trajectory (right) from training an agent using the standard reward signal. The colour bar represents the vehicle speed.

The example trajectory in Figure 4.10 shows that the agent typically selects low speeds of around 2 m/s to 3 m/s and unsystematically selects high speeds. As a result, the agent can finish many laps but with an incredibly slow lap time of around 60 seconds (compared to the pure pursuit of around 20 seconds). Further, there is no definite pattern in the behaviour displayed by the agent, resulting in this reward signal being unsuitable for racing.

Progress and Cross-track Reward Comparison

The progress and cross-track and heading reward signals are now compared with each other. While a detailed analysis of the effect of maximum speed is provided in §4.5.2, in this analysis maximum speeds of 5 m/s and 8 m/s are considered. The average progress during training of five agents with the progress and cross-track and heading rewards on the ESP and MCO maps, with a maximum speed of 5 m/s is recorded.

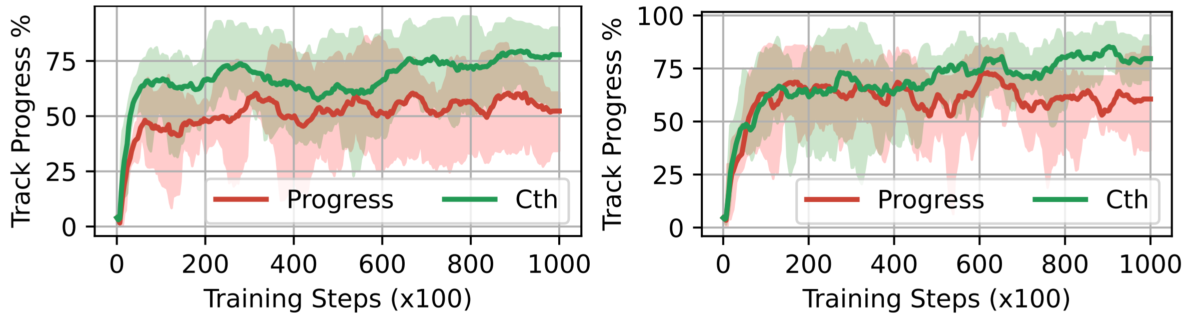


Figure 4.11: Plot showing progress during training five agents using the progress and cross-track and heading error (Cth) reward signals on the ESP (left) and MCO (right) maps with a maximum speed of 5 m/s.

In Figure 4.11, the training graph on the ESP map (left) shows that the cross-track and heading reward trains the agent to around 75% average progress while the progress reward trains the agents to around 50%. The training graph on the MCO map shows that the cross-track and heading reward trains the agents to an average above 75%, while the progress reward trains the agents to around 60%. The first observation is that the average progress of both reward signals is significantly lower than for the constant speed driving with none of the runs reaching near 100% completion. The cross-track and heading reward outperforms the progress reward during training to reach a higher level of average completion. The reward signals are compared using the vehicle's full speed range of up to 8 m/s.

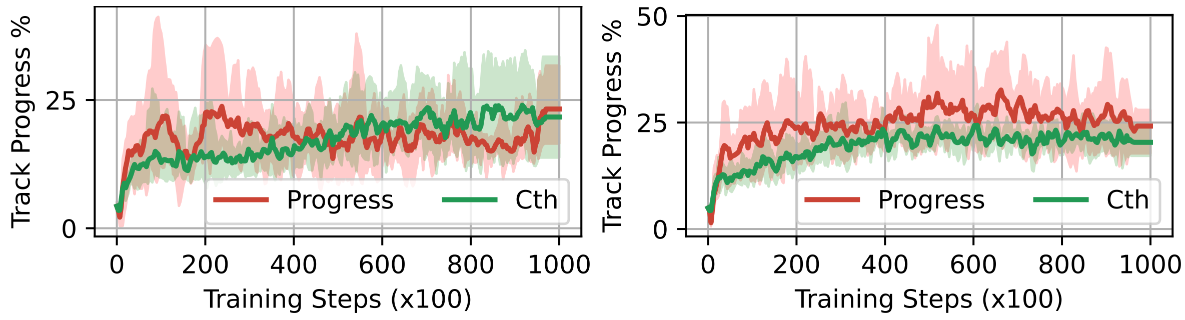


Figure 4.12: Plot showing progress during training five agents using the progress and cross-track and heading error (Cth) reward signals on the ESP (left) and MCO (right) maps with a maximum speed of 8 m/s.

In Figure 4.12, the training graph on the ESP map (left) shows that the cross-track and heading and progress rewards train the agent to a similar level of around 20% average completion. The training graph on MCO shows a similar pattern, with both agents reaching around 25%. This indicates that neither reward is adequate for training agents to race using the vehicle's full speed range. The training behaviour of the rewards is further investigated by analysing the total reward earned by each agent per episode.

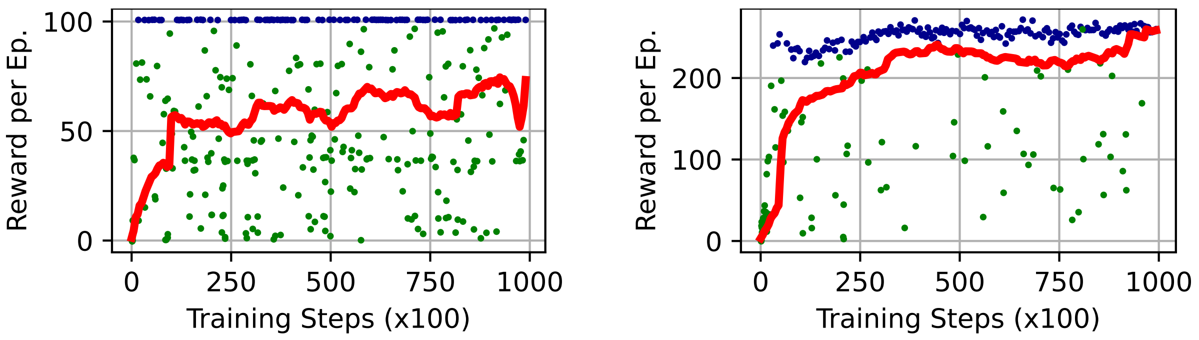


Figure 4.13: Training graphs showing the total reward per episode achieved by the progress (left) and cross-track and heading (right) reward signals on the ESP map. Green dots are crashes, blue dots are completed laps, and the red line is a moving average of the reward.

Figure 4.13 shows the rewards earned by the progress and cross-track and heading agents trained on the ESP map. Comparing the two graphs shows that the cross-track and heading agent crashes a lot less during training, explaining why the average reward is higher from earlier in the training process. The progress agent reward graph (left) shows that when the agent completes a lap, then the total reward of 100 (the β_{progress} hyperparameter) is obtained. It is interesting to see that the progress reward training performance comprises numerous completed and unfinished laps throughout the training process. This is why the average progress made by the progress agent is lower than the cross-track and heading agent.

The cross-track and heading reward receives different rewards for each complete lap, though they are grouped around 220. There is a slight increase in the reward received by the cross-track and heading agent for complete laps from the first laps that are completed until the last laps. It is suggested that this is due to the cross-track and heading reward encouraging the agent not just to complete laps, but to select high velocities that keep the agent on the centre line. Further analysis is done on the lap times generated by the agents to understand this better.

Figure 4.14 shows the lap times recorded during training for completed and crashed laps. In the beginning, many of the laps end with a crash, and as the training progresses, the agent can complete more laps. The cross-track and heading reward signal demonstrates a slight improvement in lap times as the training progresses. The progress reward shows a constant lap time (shown by the straight red line) throughout the training. Both agents achieve similar lap times during training and there is no significant change in the lap time between the initial and final laps. It is concluded that neither reward trains the agents to complete all the laps with the trained agents still crashing regularly.

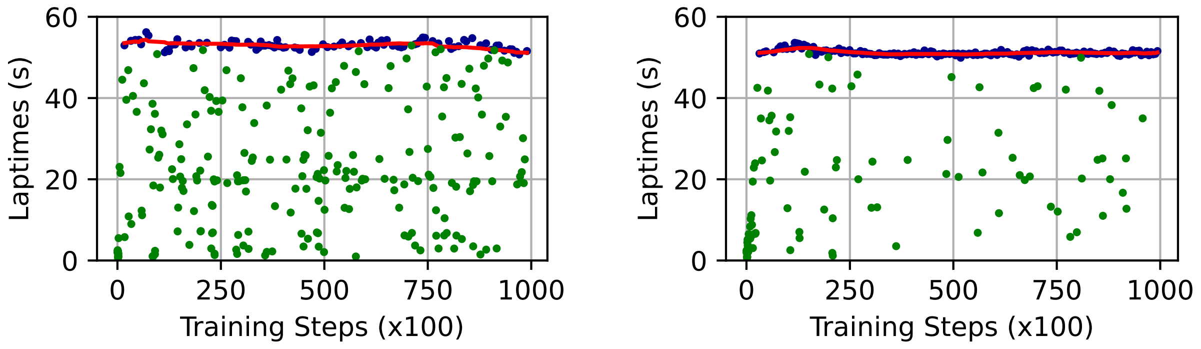


Figure 4.14: Lap times of agents trained on the ESP map for the progress (left) and cross-track and heading (right) reward signals. Green dots are crashes, blue dots are completed laps, and the red line is a moving average of the lap time.

Reward Signal Performance Comparison

The performance of agents trained with each reward signal is measured by training and testing agents on the ESP track. The progress and cross-track and heading reward signals are considered and compared to the pure pursuit planner. The pure pursuit planner is set to follow the optimal race line and is thus treated as the theoretical best. The pure pursuit planner is run with a maximum speed of 5 m/s (PP5) and the normal 8 m/s (PP8) to show the effect of limiting the maximum speed.

Metric	Progress	Cth	PP5	PP8
Lap Time (s)	50.60	50.36	52.10	46.80
Total Distance (m)	238.07	240.67	228.51	228.73
Total Curvature (m^{-1})	127.86	123.26	71.38	70.68
Avg. Velocity (m/s)	4.72	4.80	4.40	4.92

Table 4.3: Lap-time, Distance travelled, curvature and average velocity of agents trained with progress and cross-track and heading error (Cth) reward signals on the ESP map. The results are compared to the pure pursuit planner following the racing line with a maximum speed of 5 m/s (PP5) and 8 m/s (PP8).

Table 4.3 presents the results from measuring the performance of the trained agents on the ESP track. The metrics are averaged across 10 test laps, but only completed laps are used to calculate the averages. The progress agent completes laps in 50.60 s and the cross-track and heading agent in 50.36 s. The first observation is that the progress and cross-track and heading agents perform similarly on the laps they complete, which is interesting given the difference in training results.

The surprising result in Table 4.3 is that both the DRL agents achieve a faster lap time than the pure pursuit planner, which is limited to 5 m/s. What makes the result more surprising is that the planners travel 10-12m further within this time. The DRL agents have much higher total curvatures and a higher average speed, meaning that the agents swerve a lot while keeping a high speed. This result is further investigated by analysing the speed and slip angles selected by the agents.

The first conclusion about the cross-track and heading and progress reward signals is that they both produce poor racing performance. This was shown by the agents only training to 75% average progress, with them continuing to crash throughout the training. Further, the performance investigation showed that the trained agents have high curvatures and select speeds near to the maximum for the entire lap. Therefore, the speed profiles are further investigated in the following section.

4.5.2. Maximum Vehicle Speed

Training

The agents take more training steps to converge now that they have to select two control actions. The agents are thus trained for 100,000 steps. The effect of the maximum speed on training and performance is now considered. Agents are trained at maximum speeds ranging from 4-8 m/s on the GBR map, and the average progress during training is recorded. The experiments are repeated 10 times, and the average, minimum and maximum from the 10 runs are presented. The minimum and maximum are used to show the variance between different random seeds.

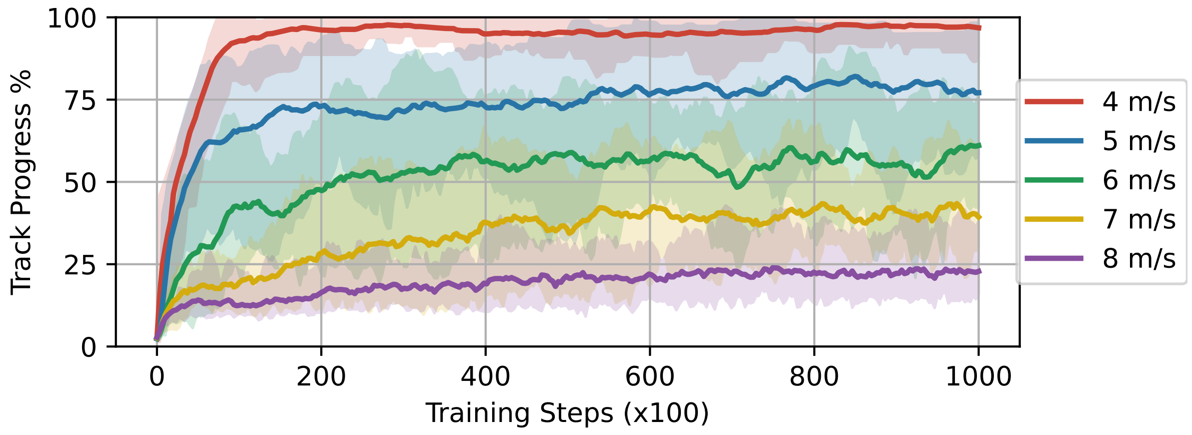


Figure 4.15: The progress of agents trained with maximum speeds of 4, 5, 6, 7, and 8 m/s using the cross-track and heading reward signal on the GBR map.

Figure 4.15 shows the progress made by the agent after training it for 100,000 steps at 5 maximum speeds using the cross-track and heading reward. The graph shows that for higher maximum speeds, the agent can complete little of the race track. The prime example of poor performance is when the vehicle has the true maximum speed of 8 m/s and averages around 15 completion after the training. As the maximum speed is decreased, the portion of the track that the agent can complete increases. The agent with a maximum of 7 m/s can complete around 30%, and the agent with a maximum of 6 m/s can complete around 55%. Both the agents with maximum speeds of 4 and 5 m/s can learn to complete an average of 95% of the track. The progress reward demonstrates similar behaviour; therefore, the results are not shown.

It is concluded that end-to-end DRL agents can only be trained to race F1/10$^{\text{th}}$ vehicle with maximum speeds of 4 m/s. Selecting high maximum speeds leads to the training converging to a solution that cannot complete laps of the race track. This limitation resonates with the results by Brunnbauer et al., who claimed that it was impossible to race autonomously using model-free DRL [22]. In their work, they presented graphs similar to Figure 4.15, where they trained agents for 8 million steps, with the agents never being able to complete a single lap.

Maximum Vehicle Speed

The performance of the agents with varying maximum speeds is measured. While many of the training runs were unsuccessful (the agents did not converge to useful racing behaviour), the performance is nonetheless recorded as a baseline to be improved upon. Table 4.4 shows the success rates and average progresses achieved by the agents trained with maximum speeds ranging from 4 to 8 m/s. The success rate is the number of completed laps out of 50 test laps. The average progress is a useful metric for measuring where in the lap the agent crashes, even if the lap was not completed.

Metric	**4 m/s**	**5 m/s**	**6 m/s**	**7 m/s**	**8 m/s**
Success Rate (%)	92	66	38	4	2
Avg. Progress (%)	98.59	87.47	77.65	36.49	40.93

Table 4.4: Success rate and average progress percentage for agents trained with maximum speeds of 4, 5, 6, 7 and 8 m/s on the ESP map.

The results in Table 4.4 show decreasing success rates and average progress as the maximum speed increases. The best average progress is at 4 m/s with 98.53 and a 92% success rate, meaning that many of the laps are completed, or at least mostly completed. The agents with a maximum speed of 7 and 8 m/s completed only 4% and 2% of the test laps, respectively, with terrible average progresses of 36.49% and 40.93%. This result indicates that the agent completion rate gets worse with increasing maximum speeds. There is a significant drop in performance after 6 m/s, with the agents able to complete few laps.

Table 4.5 shows the lap time, distance, curvature and average velocity for the agents with maximum speeds of 4, 5, and 6 m/s. The 4 m/s agent achieves a lap time of 63.30 seconds, the 5 m/s of 50.00 seconds and he 6 m/s of 41.85 seconds. Unsurprisingly, the agents that have faster maximum speeds complete laps faster. The 4 m/s agent completes laps with an average distance of 239.1 m, the 5 m/s agents with 240.92 m and the 6 m/s with 241.93. This shows agents with faster maximum speeds travel a slightly longer

Metric	4 m/s	5 m/s	6 m/s
Lap Time (s)	63.30	50.00	41.85
Total Distance (m)	239.41	240.92	241.93
Avg. Velocity (m/s)	3.79	4.84	5.80

Table 4.5: Lap time, distance travelled and average velocity for completed laps on the ESP track by agents with maximum speeds of 4, 5 and 6 m/s.

distance. The average velocity selected by all agents is near the maximum velocity; for example, the 5 m/s agent has an average velocity of 4.86 m/s, and the 6 m/s agent has an average velocity of 5.80 m/s. This indicates that the agents always select velocities near the maximum.

The study on maximum speed concludes that the higher the maximum speed, the worse DRL agents perform. At maximum speeds of 4 m/s, the agents exhibit feasible behaviour of completing many laps, but as the speed increases, the behaviour worsens, as shown by the success rate decreases. Using the vehicle's maximum speed of 8 m/s results in infeasible behaviour, with the agents having a lap completion rate of less than 5%. Studying the average velocity indicated that a potential problem is that the average velocity is close to the maximum speed. The speed profiles are further investigated in the next section to understand better why high-speed agents perform so poorly.

4.5.3. Speed Profile Investigation

An investigation is done into the speed profiles selected by the agents to understand the behaviour learned by the agents, specifically why the average velocity is so high and the lap times are lower than the pure pursuit planner. All the analysis in this section is on agents trained using the cross-track and heading reward since it produced better results in §4.5.1. Trajectories recorded by agents with different maximum speeds are presented with the colour bars indicating the speed the vehicle is travelling at.

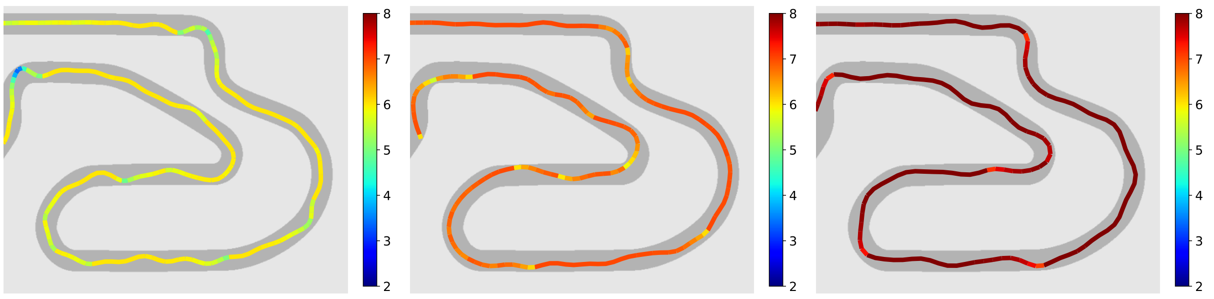


Figure 4.16: Trajectory segments on the ESP map taken by agents with top speeds of 6, 7, and 8 m/s using the colour bars to represent the vehicle speed.

Figure 4.16 shows example trajectory segments taken by agents trained with the cross-track and heading reward signal on the ESP map with maximum speeds of 6, 7, and 8 m/s. The plots show that the vehicle remains at a similar speed for most of the trajectory, showing that the agent cannot learn to select an appropriate speed profile of speeding up and slowing down. The 6 m/s agent (shown in the left image) has a mainly yellow trajectory, showing that the speed is almost always 6 m/s. Occasionally, the agent slows down when near to the boundaries. All of the agents demonstrate high-curvature behaviour of swerving a lot. This is due to the vehicle drifting since turning at high-speeds causes large lateral forces on the tyres. This is further investigated by plotting the speed profiles of an agent trained with the cross-track and heading reward compared to the pure pursuit planner following the racing line.

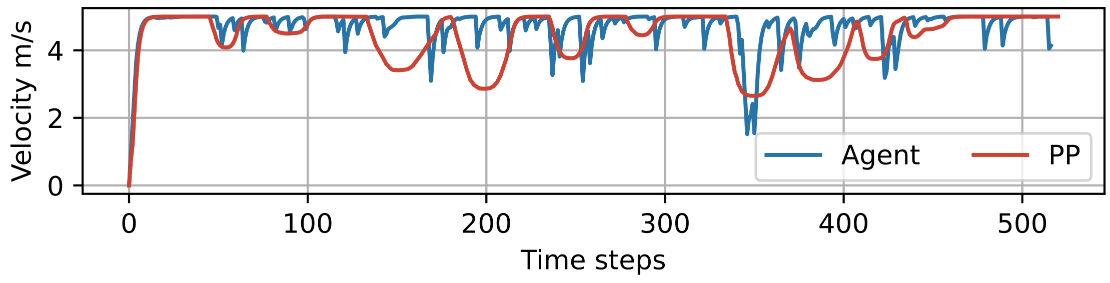


Figure 4.17: Speed profile graph of the DRL agent and the pure pursuit (PP) planners plotted against the planning steps on the ESP map.

Figure 4.17 shows the speed profiles selected by an agent trained with the cross-track and heading reward signal compared to the profile of a pure pursuit planner following the racing line. The graph shows that the pure pursuit planner smoothly speeds up and slows down during the lap. In contrast, the agent maintains the maximum speed for nearly the entire lap, occasionally sharply slowing down. This behaviour is visually seen in Figure 4.16 by the lines having a constant colour. The effect of the vehicle travelling at a high speed for the entire lap is investigated by plotting the slip angle profile of the agent and the pure pursuit planner. The slip angle is the angle between the direction of the vehicle body and the velocity of the vehicle. If the vehicle is turning a corner, then the slip is non-zero since the vehicle's velocity is measured at the centre of mass.

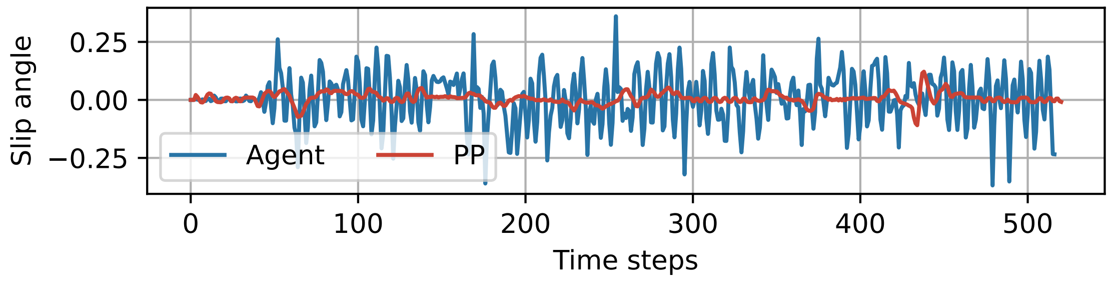


Figure 4.18: A comparison of the slip angle profiles for the pure pursuit (PP) and DRL agent planners on the ESP track.

Figure 4.18 shows the slip angles (recorded as part of the simulator state) for the cross-track and heading agent and pure pursuit planner trajectories. The trajectories shown here are the same as in Figure 4.17. The pure pursuit planner has a normal slip of around 0.05. The agent planner has an average slip of 0.1, reaching 0.3. This shows that the vehicle is drifting a lot and that this is not a feasible method for racing a physical vehicle. While from a cursory glance, the DRL agent may outperform the classical planners, that view would be misguided since the trajectories taken by the agent are not feasible in reality. The vehicle agent learns a policy that depends on the vehicle sliding across the track. This style of driving, where the agent learns to exploit the simulator, has been seen in other learning literature [111], and is problematic since it is practically infeasible.

The investigation into speed profile selection concludes that the leading problem with agents trained with the cross-track and heading reward is that they do not learn to select an appropriate speed profile. Rather, they mainly select the maximum speed for the entire lap. This leads to high slip angles, making the learned policies infeasible. This is a significant problem that must be addressed.

4.5.4. Discussion of Results

The results in the section provide many interesting insights into how DRL agents learn to control F1/10th cars in simulation. The constant velocity tests showed that agents can be trained to select a steering angle repeatably with excellent sample efficiency. The cross-track and heading reward signal converges the fastest, followed by the progress and then standard rewards. The progress reward signal learns to cut the track corners, achieving a shorter, smooth path. While the cross-track and heading reward track the centre line well, it has a significant problem of slaloming. It was shown that the agents can be transferred to different tracks where they display similar performance to the training track.

The high-speed evaluation showed that training agents to select speed and steering constraints is much more difficult than just steering. The standard reward signal was shown to be ineffective for training agents to race because it generates poor, erratic performance. A study on the maximum speed that the agent may select showed that end-to-end DRL agents are unable to learn to race at speeds above 5 m/s, which is significantly lower than the vehicle's true maximum speed of 8 m/s. In comparing reward signals, the cross-track and heading reward signal demonstrated significantly better training performance. While neither of the reward signals were able to generate perfect, repeatable success, the progress reward signal was plagued explicitly with the problem of crashing.

The trained agents also demonstrated many issues. The most significant problem was that the agents learned to drive at exclusively high speeds, not slowing down for the corners. The result is that they drift for a significant part of the race track. The problem of learning a policy that drifts and the imperfect success rate render current advances in

DRL impractical for physical systems.

These results show that current learning formulations do not perform well in the challenge of realistic variable speed racing. The key problem is that the agent cannot select an appropriate speed to stay within the friction limits. This problem has been seen in the literature with the authors of [111] noting that maximising a single objective can lead to undesirable behaviour, such as high-slip angles and infeasible trajectories. The authors of the world-class racing agent Sophy noted that their agent could not act strategically [20], and the problem of slaloming is persistent in DRL solutions as experienced by [22, 61].

Limitations

The limitations of this study were the consideration of only one RL algorithm. While future work could consider other algorithms, previous approaches using a variety of algorithms, DQN, SAC, DDPG, and PPO have all produced similar results. While only one algorithm was used, the TD3 algorithm is a state-of-the-art algorithm for continuous control, and it is expected that other algorithms will produce similar results.

4.6. Summary

This chapter introduced the learning formulation used in F1/10th autonomous racing. The approach to designing the reinforcement learning formulation was discussed, and the standard, progress and cross-track and heading reward signals were presented. The constant speed evaluation demonstrated that DRL agents could be trained in 30,000 steps to drive at a constant speed and follow a reference line. The cross-track and heading reward outperformed the progress and standard rewards regarding vehicle performance during training and consistency between maps. The progress reward selected shorter paths and swerving less than the standard and cross-track and heading rewards. The variable speed evaluation demonstrated that current learning formulations are unsuitable for high-speed racing using the vehicle's full speed profile. The maximum speed comparison showed that while agents can complete all the laps with a maximum speed of 4 m/s, as the maximum speed increases, the performance worsens. The specific problem identified is that the agent does not learn to select an appropriate speed profile and thus drifts around the track with a high slip angle. This problem should be rectified by designing learning formulations for high-speed racing that train the agent to select an appropriate speed profile.

Chapter 5

High-speed Learning Formulations using Vehicle Models

Chapter 4 concluded that current learning formulations obtain low completion rates for variable speed F1/10th racing due to poor speed selection resulting in high-slip paths. This chapter presents two methods to overcome these problems using analytical vehicle modelling. The optimal trajectory, with velocity profile, is used to design improved reward signals that train the agent to select an appropriate speed. Secondly, the link architecture is presented, which uses the agent to calculate a steering angle and a friction model to calculate the speed. The evaluation shows that both methods select appropriate speed profiles with low-slip trajectories, overcoming the problem of low-completion rates. The racing reward signals achieve faster racing performance, the link architecture achieves a 100% completion rate, and both methods achieve faster lap times on four maps compared to the two leading methods in the literature.

5.1. Introduction

5.1.1. Problems in High-speed Racing

While learning policies have many advantages for autonomous racing, such as not requiring localisation, current variable speed methods are limited in their ability to select appropriate speed profiles. A vehicle's speed profile refers to the speeds the vehicle travelled at while driving around the track. An appropriate speed profile is one where the vehicle remains within the friction limits for the entire trajectory, i.e. not drifting. Visually, an appropriate speed profile looks like the optimal trajectory that speeds up in the straights and slows down around the corners.

The evaluation in §4.5 showed that variable speed solutions for autonomous racing perform very poorly at high speeds, achieving low completion rates (Table 4.4). The root cause of the low completion rate is the poor speed profile selection (Figure 4.17), where it was seen that the agent selects a speed near the maximum for most of the track. Selecting a high speed causes the vehicle to drift (as seen in Figure 4.18) with a high slip angle. When a vehicle is drifting it is difficult to control due to small actions having a big effect on the vehicle's motion. Therefore, solutions to the problem of speed selection should enable the vehicle to speed up and slow down to keep the vehicle within the friction limits.

5.1.2. Design Methodology

This chapter addresses the problem of poor learned performance by end-to-end agents for the task of autonomous racing by using analytical vehicle models. The design methodology for overcoming these problems is to use additional information about the vehicle dynamics to aid the learning. Vehicle dynamics are well studied, and this information can be useful to build better learning formulations for training DRL agents.

The first application of this methodology, in §5.2, is to redesign the reward signal using the optimal racing line. Current reward signals have assumed that the track is available but have simply used the centre line to reward the agent. While the centre line is only a set of waypoints going around the map, the racing line is a set of the optimal waypoints to follow and includes the optimal velocity reference at each point.

The second application, in §5.3, is to design a new architecture that uses a friction model to calculate a speed reference for the vehicle. The *link architecture* uses a DRL agent to select a steering angle and then analytically calculates how fast the vehicle should go. This is motivated by the constant speed evaluation in §4.4 showing that agents are good at selecting only steering references but struggle with also selecting speed profiles.

In this chapter, the focus is on improving the learning formulation and therefore, many details in the training are kept constant from the previous chapter and not redescribed. Components like the neural network, training algorithm, etc., are not explained here so that the focus can fall on the learning formulation.

5.2. Racing Line Reward Signals

In the context of racing, the best solution is available using a pure pursuit controller following the optimal racing line. However, this requires the vehicle to localise itself and is not flexible to new environments. The advantage of neural network-based agents is that they do not require localisation, are flexible to new environments, and can select control commands directly from the LiDAR scan. Therefore, this approach aims to train the DRL agent to replicate the behaviour of a pure pursuit planner with a DRL agent that maps LiDAR scans to control commands.

5.2.1. Design Considerations

In RL, the only method of information transfer from the designer to the agent is through the reward signal. Therefore, the reward used is critical since it is the only means of changing the agent's behaviour. Further, RL algorithms will always maximise the reward function, and while this sounds good, it can cause unexpected results, such as falling into poor local minima.

Designing a reward signal for tasks with the binary outcomes of winning or losing, such as chess, is straightforward. However, when there are multiple, continuous, competing metrics, such as in racing, selecting a reward signal that accurately encompasses them is more difficult.

In racing, the first aim is that the vehicle learns not to crash, and the secondary aim is that the vehicle achieves good lap time performance. These metrics are opposed to each other; if the vehicle is travelling fast, then it is achieving good performance but might crash. If the vehicle travels slowly, it is less likely to crash but performs badly.

Current Reward Analysis

In Chapter 4, an extensive comparison of the cross-track and heading and progress reward signals at constant and variable speed was performed. The cross-track and heading error reward encourages large velocities in the direction of the centre line. While this appears to be a good reward signal, the result is that the selected velocities are too large. The expected mitigating factor is that if the vehicle travels too fast, it will crash. But the reality is that these conflicting reward terms do not produce good behaviour; namely, the agents always select speeds that are too high.

The progress reward does not provide any measure of how good a path is compared to another path. Any behaviour that results in a lap being completed receives the same reward. It is worth noting that due to the discounting of future rewards (γ) inherent in RL algorithms, there is a motivation to complete laps quickly. An additional limitation is that for small differences in heading angle, the difference in progress made along the centre line is small.

Neither of these reward signals communicates useful information to the agents, explaining the poor performance.

Rewarding States Vs Actions

The progress and cross-track and heading reward signals are based on the vehicle's state, i.e. position and speed. The primary reason is that it is easier to quantify how good or bad a state is, and it is difficult to assess the value of an action. While rewarding states is simple, this method is limited by the lack of direct information to the agent. The agent is left to discover the link between states and actions for itself.

Another method called guided policy search (GPS) uses an optimal policy to guide the policy search by calculating a reward based on the difference between the optimal policy's actions and the agent's actions [46]. This method is useful in contexts where an optimal policy is available. GPS has not yet been applied to F1/10th racing but appears to have a possible benefit.

While imitation learning has been used for F1/10th racing [9], it has demonstrated poor results with the specific problem of crashing [23]. Reinforcement learning is better suited to racing than imitation learning since there are a large number of possible states and it is difficult to represent enough states to result in a robust policy in a data set.

Simulator Exploitation

One of the problems with training DRL agents in simulation is that they learn infeasible policies due to exploiting the simulator dynamics. A prime example of this was identified by Zheng et al. [111], who use several gradient-free optimisation algorithms to jointly optimise a trajectory and controller parameters. They show that while their solution learns incredibly fast racing performance, its final strategy is to drift around the outside of the corner with slip angles of up to 80°. The key problem with this strategy is that the vehicle only completes 60-70% of the laps. The authors identify in their discussion that the behaviour of learning algorithms to exploit the simulator dynamics is an ongoing problem.

In Chapter 4, the same exploitation was seen in the agent learning to race at high speeds and drift around the corners. Methods that do this are practically infeasible because the dynamics in the real world are not constant, and drifting is not as repeatable.

Preventing learning agents from exploiting the simulator is a difficult problem because exploitation is inherently high-performance. A possible solution is to add additional noise to the simulation during training, also known as domain randomisation [112], but this is a poor solution to this problem since the noise in the real world, such as a change in friction coefficients is difficult to model. The authors of [111] propose using additional objectives (rewards) for the learning algorithm to maximise.

Racing Line Description

The racing line is the fastest trajectory, set of waypoints and velocities around the race track. The calculation of the racing line was explained in the preliminaries chapter on F1/10th racing, §3.3.3.

Figure 5.1 shows a segment of the racing line for the MCO on the left and the speed profile for the entire track on the right. The racing line typically slows down around the corners and speeds up along the straighter sections. This slowing down is shown by the blue on the trajectory, and the higher speeds are shown in red. The racing line can go faster around the track than the centre line since the waypoints represent the line of minimum curvature, as shown by the racing line coming close to the apexes of the corners.

This racing line is used as the basis for the novel reward signals.

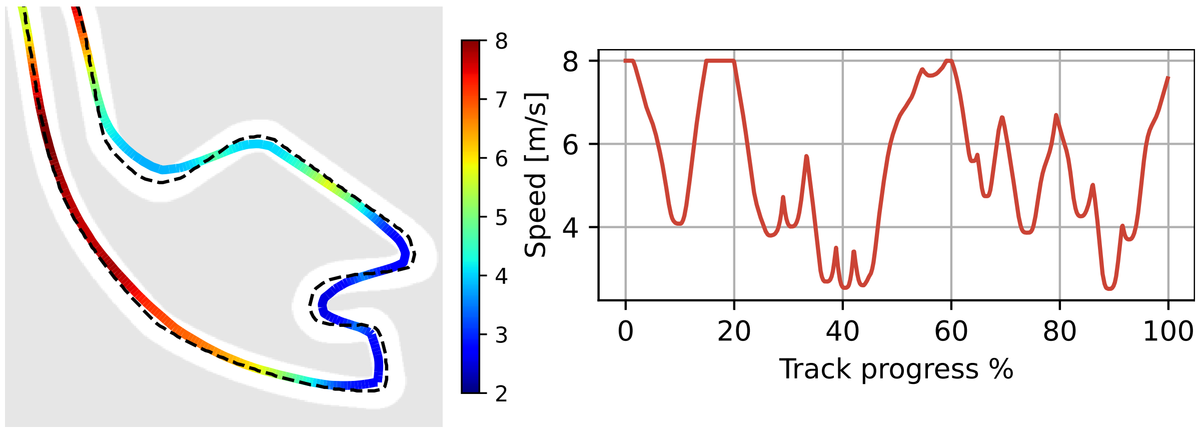


Figure 5.1: Left: Dotted centre line compared with the racing line for a section of the MCO track with the colours indicating the speed at each point. Right: The speed profile for the MCO track.

5.2.2. Reward Candidates

The previous section focused on improving the path selected by the agent. This section concentrates on the agent's speed selection. It is proposed to use the speed at each waypoint in the optimal racing line in the learning process to help guide the agent to select an appropriate speed. This method aims to train the agent to select an intelligent velocity profile, unlike the behaviour in the end-to-end agent.

Velocity Line Rewards

The first approach followed in integrating the racing line into the reward signal is to modify the cross-track and heading reward. Where the cross-track and heading reward trains the vehicle to go as fast as possible, the velocity line rewards aim to train the vehicle to select the racing line speed. A new term is introduced as the difference in speed between the vehicle and the racing line at a position along the track. The speed difference dV is calculated as,

$$dV = \frac{|v_{\text{vehicle}} - v_{\text{raceline}}|}{v_{\max}}, \tag{5.1}$$

using the subscripts to represent the vehicle's speed, the speed of the racing line at the vehicle's location and the vehicle's maximum speed. The speed of the racing line is found by projecting the vehicle's current location onto the racing line and using the speed at the closest waypoint. The speed difference is a value between 0 and 1, representing how far away the vehicle's speed is from the racing line speed. Three candidate rewards that use the speed difference are proposed.

The first reward changes the velocity term that is multiplied by the heading angle from being the vehicle's speed, to be proportional to the difference between the vehicle's speed and the racing line speed. The first velocity line reward, called v1 is written as,

$$r_{\text{v1}} = (1 - dV) \cdot \cos(\psi) - d_c, \tag{5.2}$$

with ψ being the heading error, and d_c being cross-track distance. The reward aims to minimise the speed difference in the direction of the racing line. The reward gradient increases as the speed difference, heading error and cross-track distance all decrease.

The second reward uses the speed difference as an additional penalty to the agent. The reward v2, is written as,

$$r_{\text{v2}} = v_{\text{vehicle}} \cdot \cos(\psi) - d_c - dV. \tag{5.3}$$

The reward aims to minimise the speed difference in the same way that the cross-track distance should be minimised.

The third reward is a combination reward that only gives the reward for velocity in the direction of the track (first term of the cross-track and heading reward) if the vehicle's speed is below the racing line speed. This reward is specifically targeted at overcoming the problem of the agent selecting speeds that are too high. The reward v3, is written as,

$$r_{\text{v3}} = \begin{cases} v_{\text{vehicle}} \cdot \cos(\psi) - d_c & \text{if } v_{\text{vehicle}} < v_{\text{raceline}} \\ -d_c & \text{otherwise} \end{cases} \tag{5.4}$$

Pure Pursuit Policy Search

A different approach to training the vehicle is to guide the policy search by comparing the actions to a pure pursuit planner. The pure pursuit policy search (PPPS) reward uses the difference between the actions selected by a pure pursuit planner and the agent as the reward. This reward aims to train the agent to select the same actions as the pure pursuit planner. The reward for this reward signal is written as

$$r_{\text{PPPS}} = \beta_c - |v_{\text{pp}} - v_{\text{agent}}| \cdot \beta_{\text{v}} - |\delta_{\text{pp}} - \delta_{\text{agent}}| \cdot \beta_{\text{steering}}. \tag{5.5}$$

In the equation, v_{pp} and δ_{pp} are the pure pursuit planners speed and steering angle, v_{agent} and δ_{agent} are the agent's speed and steering angle, and β refers to a hyperparameter. The parameters used are: $\beta_c = 0.5$, $\beta_{\text{v}} = 0.4$, $\beta_{\text{steering}} = 0.4$. It is noted that the steering and velocity values used here refer to the action selected and not the vehicle's state.

5.2.3. Racing Line Reward Evaluation

The novel learning formulations aim to train an agent to replicate the behaviour of a pure pursuit planner in following the racing line and selecting speeds similar to the racing line speed. Therefore an evaluation is constructed to measure the effect of the racing line reward signals on the learning formulation in terms of the path taken, the training behaviour and the speed profiles generated. The focus on the evaluation in this section is a comparison between agents trained with the racing reward signals and the pure pursuit planner. In §5.4 a comparison with conventional learning is presented.

Constant Speed Path Evaluation

The first investigation compares the effect of using the centre line (as used in Chapter 4) as opposed to the racing line for constant-speed driving. To show the effect, the cross-track and heading reward signal is used to train agents with the centre line and racing line as the reference. Constant-speed driving is used to highlight the difference in the path taken by using the different reference lines. The paths taken by the agents are compared to each other numerically in Table 5.1 and visually in Figure 5.2. The centre line label refers to the agent trained with the cross-track and heading reward signal using the centre line as the reference line, and the racing line label refers to the agent trained using the racing line as the reference. Both agents are trained and tested at a constant speed of 2 m/s. The results are compared to the pure pursuit planner (abbreviated as PP) that follows the racing line at a constant speed of 2 m/s.

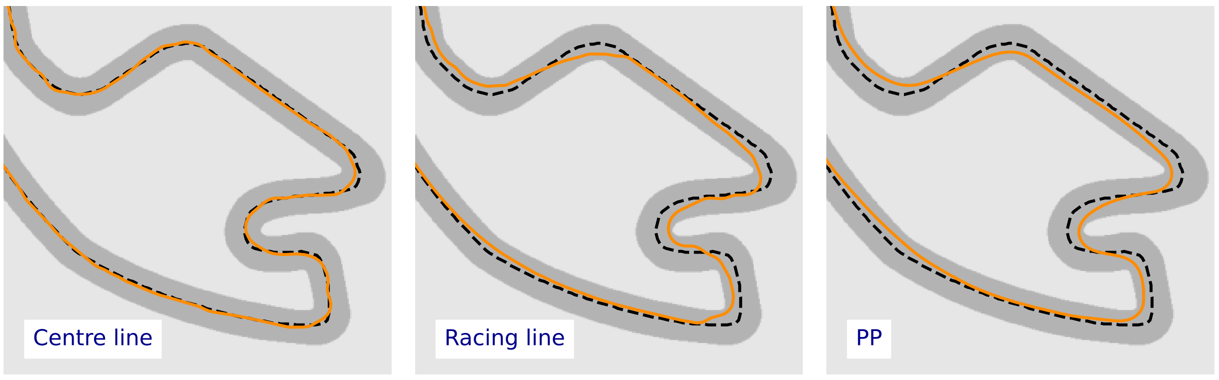


Figure 5.2: A comparison of the paths taken by agents trained to follow the centre (left) and racing (middle) lines using the cross-track and heading reward on the MCO map. The path taken by the pure pursuit (PP) planner (right) is shown for reference.

In Figure 5.2, it can be seen that both agents learn to track the reference lines. Therefore, using the minimum curvature line as the reference results in the vehicle following the racing line, which is the desired behaviour. The difference in the path taken by using the centre line and minimum curvature line is explored numerically in Table 5.1. The agents are referred to as the centre and racing agents, and the pure pursuit planner is provided for comparison.

Table 5.1 shows that the racing line agent completes laps of the track with a total distance of 168.44 m, which is 8.53 m shorter than the centre agent. The centre agent has a considerably larger total curvature of 239.76 m^{-1}, compared to the racing agent's

	DRL Agents		*Classical*
Metric	**Centre Line**	**Racing Line**	**PP**
Total Distance (m)	176.97	168.44	167.63
Total Curvature (m^{-1})	239.76	186.77	119.22
Centre L. Deviation (m)	51.17	173.76	175.63
Racing L. Deviation (m)	198.95	68.23	21.62
Mean Steering (rad)	0.13	0.11	0.05

Table 5.1: The total distance, curvature, centre and racing line deviation, and mean steering angles for agents trained with the cross-track and heading reward signal using the centre and racing lines on the MCO track. The pure pursuit planner following the racing line is shown for comparison.

curvature of 186.77 m^{-1}. While using the racing line improves the total curvature from the centre line, the classical pure pursuit planner still outperforms both DRL agents with a total curvature of 119.22 m^{-1}.

The deviation of the agents from the centre line and the racing line is provided to illustrate the agents' ability to follow the reference. The centre agent tracks the centre line accurately with a deviation of 51.17 m. The centre agent's learned policy is far from the racing line, with a deviation of 173.76 m. The racing agent tracks the racing line with a deviation of 68.23 m and the centre line with a deviation of 198.95 m. The pure pursuit planner tracks the racing line more accurately than the DRL agents, with a total deviation of 21.62 m.

These results indicate that using the racing line as the reference line for constant speed driving leads to the agent taking shorter paths with lower curvatures that result in a smaller mean steering angle. The differences are due to the racing line having less curvature than the centre line, thus requiring the vehicle to steer less. While the cross-track and heading reward trains agents to accurately track the reference line used, the pure pursuit planner still outperforms agents trained with the racing line as a reference by achieving a shorter distance, lower curvature, and smaller mean steering angle.

Candidate Reward Comparison

After establishing that the racing line improves the path taken by constant speed agents, the reward signals presented in §5.2.2 are compared against each other for variable speed racing. Variable speed racing requires the agents to select a speed between 2 m/s and 8 m/s that results in the vehicle moving around the track as quickly as possible while not crashing. The velocity line rewards v1, v2, and v3 are considered and compared to the pure pursuit policy search (PPPS) reward. This evaluation aims to determine which reward signal trains DRL agents to select the speed profile most similar to the racing line.

The evaluation starts by analysing the training performance of agents trained with the different rewards before considering the behaviour of the trained agents. The v1, v2, v3 and PPPS rewards are each trained for 100,000 steps, and the average progress during training is recorded.

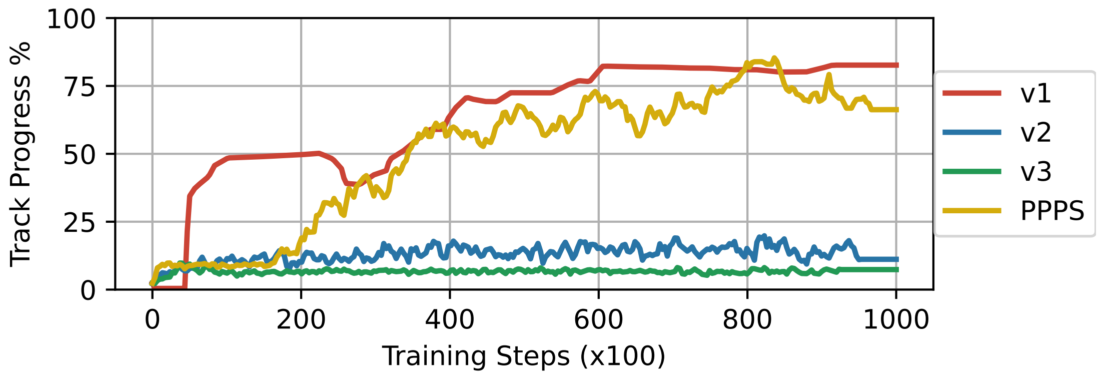


Figure 5.3: The average progress during training of the v1, v2, v3 and PPPS rewards trained on the ESP map for 100,000 steps.

Figure 5.3 shows the training progress for each reward signal being trained on the ESP map. The graph shows that the v1 and PPPS rewards both produce average performance above 60%. The v2 and v3 rewards achieve under 20% throughout training. This indicates that these rewards are poorly suited to racing since the agent cannot learn to complete any laps. An investigation into the reason for the average progress being low showed that the agents still select the maximum possible speed and quickly drift, losing control of the vehicle and crashing.

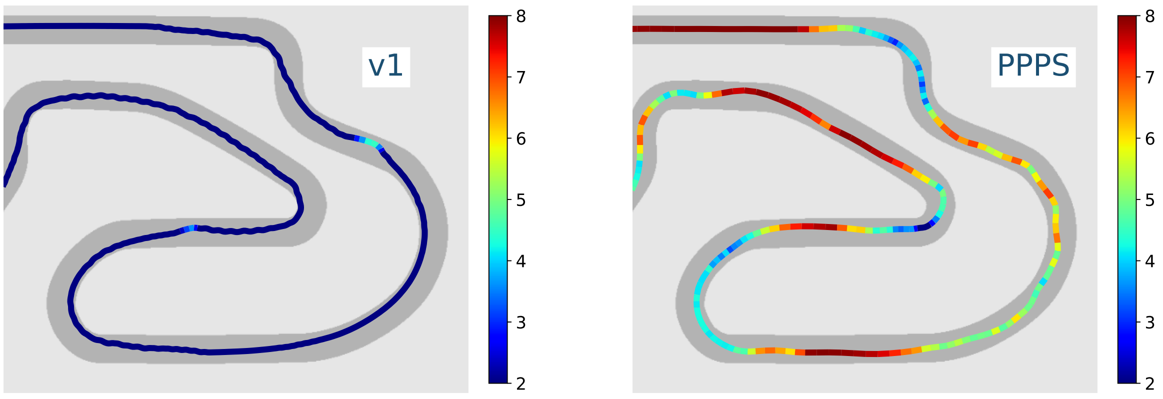


Figure 5.4: Comparing the v1 and PPPS rewards for a portion of the ESP track.

Since the v1 and PPPS rewards achieve over 60% average progress during training, the trajectories selected by the agents are further investigated. Figure 5.4 shows trajectory segments for the v1 and PPPS reward on a portion of the ESP track. The v1 reward mainly selects a speed of 2 m/s, which is near the slowest possible speed. In contrast, the PPPS reward shows the ability to select a valid speed profile, speeding up in the straights and slowing down in the corners.

Therefore, it is concluded that none of the velocity line reward signals can be used to train agents because they either result in agents that complete few laps or the agents learn a policy of selecting low speeds to complete laps. This result demonstrates that designing reward signals for racing is a difficult problem, and solutions that appear to promote good behaviour are often unsuccessful. In contrast, the PPPS reward was shown to train agents to complete many laps and select a speed profile of speeding up and slowing down. Therefore, the PPPS reward signal is further studied to measure its training performance on different maps and investigate the speed profile selected by the agents.

Training Investigation

The training behaviour of agents trained using the PPPS reward signals is done to understand the effect of the map and the random seed used during training. The random seed affects the initial values in the neural network and the samples that are selected for training the agent. Ten agents are trained to race by selecting speed and steering actions on the AUT, MCO, ESP and GBR maps and the progress made by the agents during training is recorded. The average progress along the track is plotted for the ten runs with the shaded regions indicating the minimum and maximum for each map.

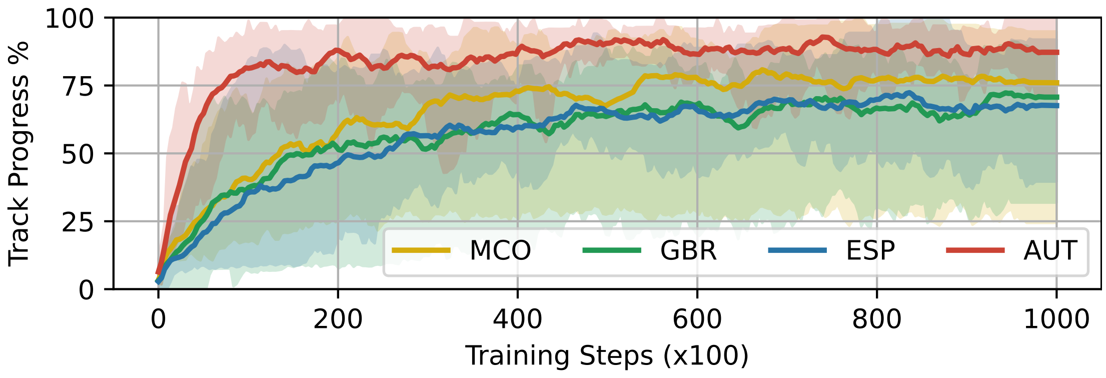


Figure 5.5: The average progress during training of ten agents trained with the pure pursuit policy search algorithm on the AUT, MCO, ESP and GBR maps.

Figure 5.5 shows the average progress during training using the pure pursuit policy search reward signal. On the AUT map, the agent learns faster and to a higher average progress of around 85%. On the MCO, GBR and ESP maps the agents achieve an average track progress of between 70% and 80%. While the agents were trained for 100,000 steps, there is little change in the average progress after 60,000 steps, indicating that the agents could possibly be trained for fewer training steps. On the graph, there is a wide shaded region, showing that there is a large difference between the minimum and maximum training runs. The discrepancy between training runs is further investigated by plotting all the different runs for the MCO map in Figure 5.6.

Figure 5.6 shows the average progress during training using different random seeds. Most of the runs reach above 75% by 60,000 steps, with two of them reaching around 95%.

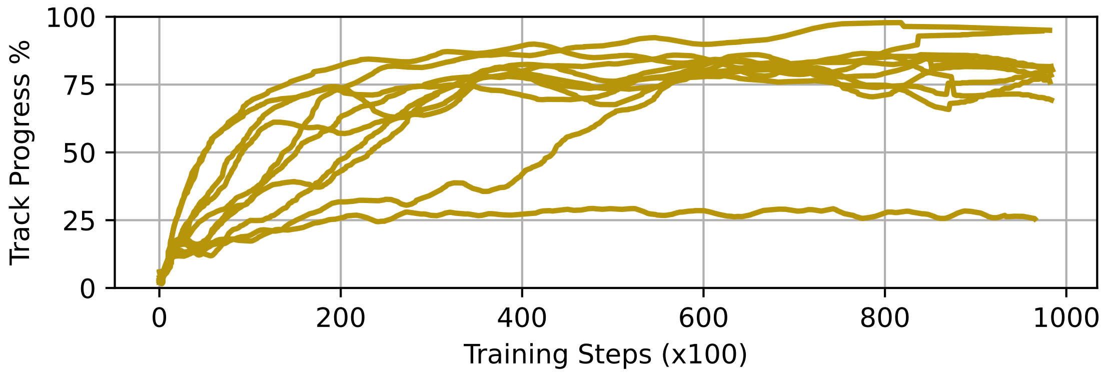


Figure 5.6: Ten repetitions of the training on the MCO map with different random seeds are plotted for comparison.

However, one of the runs only reaches around 25% for the entirety of the training. The single run that achieved only 25% explains why the shaded region for the MCO map in Figure 5.5 is so large. In addition to resulting in a large difference between the minimum and maximum, the single negative run also decreases the average in Figure 5.5.

The discrepancy between training runs indicates that the random seed used plays a significant role in how well the agents perform, with certain seeds resulting in excellent behaviour and others not converging. The conclusion from the training investigation is that while the random seed plays a significant role in the performance, for most seeds, the pure pursuit policy search reward signal trains agents to complete most of the laps. The difference between the maps is significant with the PPPS reward being the most effective on the AUT map.

Speed Profile Analysis

The next evaluation investigates the similarity between the pure pursuit planner following the racing line and the agent trained with the pure pursuit policy search (PPPS) reward signal. The analysis consists of visually comparing the trajectories in Figure 5.7, comparing the actions select by the planners in Figure 5.8 and the vehicle's speed and steering states in Figure 5.9. The actions shown in Figure 5.8 are the actions selected by the planner, while the states shown in Figure 5.9 are the vehicle states of speed and steering.

Figure 5.7 shows the trajectories taken by the pure pursuit planner (left) and an agent trained with the PPPS reward signal (right). The first observation is that the trajectory taken by the pure pursuit planner is a lot smoother than the trajectory taken by the PPPS agent in terms of the path taken and the speeds selected. The pure pursuit planner smoothly follows the racing line, slowing down and speeding up in proportion to the curvature. While the PPPS agent's trajectory is not as smooth, it does select an appropriate speed profile, as shown by the darker red line in the straighter section of the track and the blue and green line segments around the corners. Visually it appears that the PPPS agent has more red and less blue regions than the pure planner, indicating that

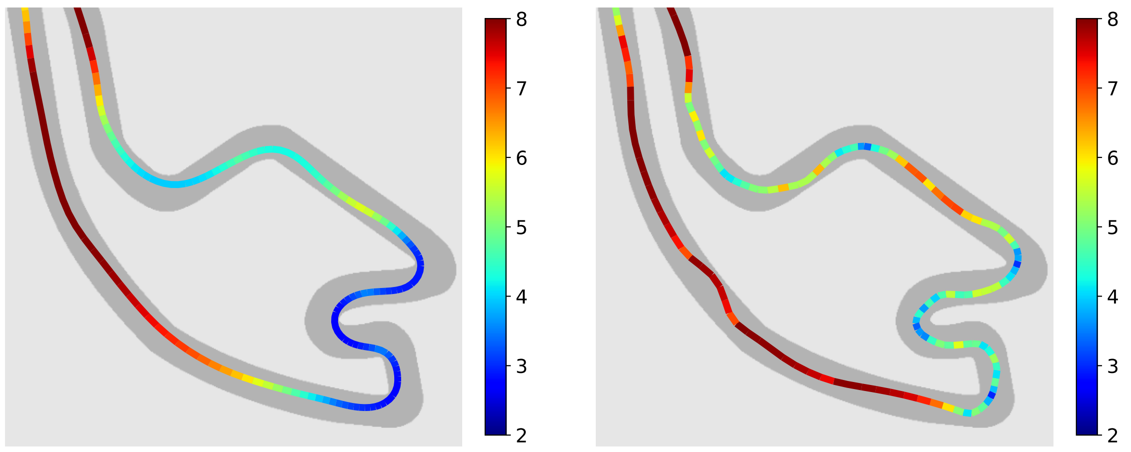


Figure 5.7: The trajectory from the pure pursuit planner (left) compared to the PPPS agent (right) on the MCO map. The colour bar represents the vehicle speed in m/s.

the PPPS agent selects higher speeds. This is further investigated by considering the actions selected by the two planners.

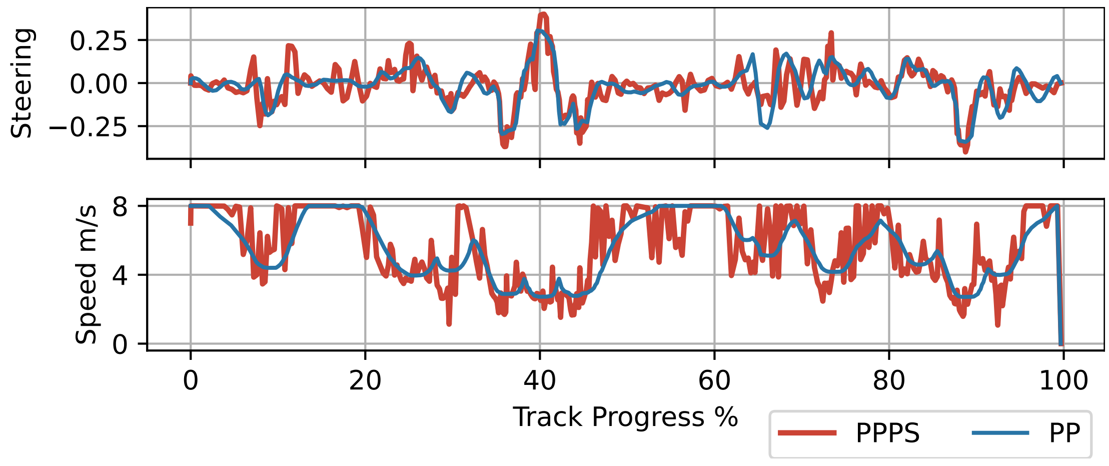


Figure 5.8: A comparison of the actions selected by the trained PPPS agent (blue) and the pure pursuit planner (red) on the GBR track.

In Figure 5.8, the steering and speed actions for the pure pursuit planner and PPPS agent are compared. As seen in Figure 5.7, the pure pursuit planner smoothly selects steering and velocity actions that gradually change. The PPPS agent tracks the steering angle (top graph) accurately, often selecting similar steering angles. This pattern is specifically seen around the 40% mark, where the vehicles steer negatively, positively, then negatively again, and the agent tracks the pure pursuit planner's steering angle.

The bottom graph in Figure 5.8 shows the speed actions selected by the PPPS agent and pure pursuit planner. The PPPS agent shows many fluctuations in speed selection, where speeds higher and lower than the pure pursuit planner's speed are regularly selected. While the PPPS agent's speed selection is not smooth, it roughly tracks the pure pursuit planner's speed selection. For example, around 20%, both planners select the top speed of 8 m/s, and around 40%, both planners select a speed of around 3 m/s. The differences between the two speed profiles often result from the PPPS agent selecting a higher speed than the pure pursuit planner, such as around the 70% mark, where the PPPS agent selects 8 m/s, while the pure pursuit planner selects around 6 m/s. While the actions selected by the planners are important, the states experienced by the vehicle are more important, because the states determine the vehicle's racing performance.

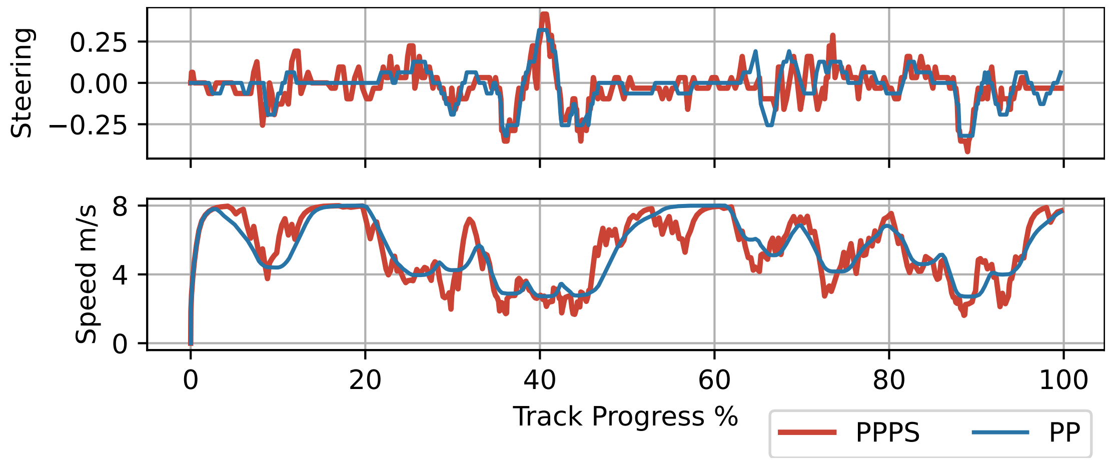


Figure 5.9: Steering and speed profile comparisons for the pure pursuit (PP) and pure pursuit policy search (PPPS) agent on the MCO track.

Figure 5.9 shows the steering and speed profile comparison between the pure pursuit planner and the agent trained with the PPPS reward signal. As with the action graph, the PPPS agent tracks the steering angle of the pure pursuit planner accurately. The only difference between the pure pursuit planner and PPPS agent's steering graphs is the PPPS agent's steering angle occasionally fluctuates around the pure pursuit reference. The speed graphs for the states in Figure 5.9 are more similar than the action graphs in Figure 5.8. This is because the high-frequency fluctuations in speed selection are partially filtered out by the vehicle's momentum, meaning that the vehicle required time to speed up or slow down. This result indicates that the PPPS reward is effective for training an agent to speed up and slow down, roughly tracking the pure pursuit planner.

The investigation into the trajectories selected by the pure pursuit planner and PPPS agent shows that the PPPS reward signal is effective for training an agent to replicate the behaviour of the pure pursuit planner. While the actions selected by the agent fluctuate around the pure pursuit planner's actions, the vehicle states of speed and steering accurately track the pure pursuit planner. The accurate tracking of the pure pursuit planner's vehicle speed indicates that the PPPS reward signal is effective in training agents to produce similar behaviour as the racing line. This result indicates that the racing line can be used to train agents to select an appropriate speed profile of speeding up and slowing down.

5.3. Linked Action Architecture

In §5.2, methods of training the agent to select a better speed profile were considered using an optimal trajectory. This section takes a different approach to overcome the problem of speed selection for DRL agents by aiming to simplify the learning formulation. The approach followed uses the agent to select the steering angle and a friction model to select a speed reference. The section starts by explaining the problem of action complexity before describing architecture design.

5.3.1. Architecture Design

Action Complexity Problem Discussion

In §4.4, the constant speed DRL agents demonstrated good performance, tracking the reference line well, not crashing, and selecting good paths. They could be trained in fewer than 30,000 steps, were highly repeatable and could follow the reference line closely. In contrast, the variable speed agents performed very poorly, with high crash rates. The key problem was poor speed selection; the agent could not select an appropriate speed profile for speeding up and slowing down, resulting in high-slip trajectories. This result indicates that DRL agents demonstrate good performance in selecting a single action, but they perform less well when having to select two actions. Therefore, the design methodology followed is to use the agent to select only the steering angle to reduce the complexity of the learning problem.

Linked Action Architecture Overview

In response to the problem of speed profile selection, the linked action architecture is presented. A method is presented for calculating the agent's speed based on the steering angle selected by the agent. The link architecture aims to retain the sample efficiency and training stability of the constant-speed agents while achieving good racing performance by selecting appropriate speeds. This approach is partially inspired by optimal trajectory generators that often use a two-step process of calculating a minimum curvature path and subsequently calculating a minimum speed profile. In optimal trajectory generators, the velocity is selected as the highest speed that remains within the friction limit.

Figure 5.10 shows a schematic of the linked action architecture. The agent (shown in green) receives the LiDAR scan and selects a steering angle using the same setup as the constant-speed agents. The calculate speed function uses the steering angle as input into a friction model that calculates a speed reference. The speed and steering references are combined and used to control the vehicle.

The learning formulation for the link architecture uses the cross-track and heading

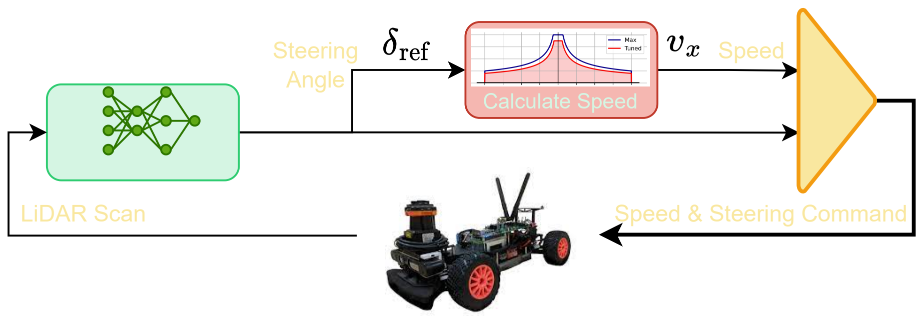


Figure 5.10: Schematic of the linked action architecture showing how the calculate speed function is located after the agent and uses the steering angle to select a speed reference for the vehicle.

reward signal since it demonstrated the best performance in Chapter 4. The state vector uses two consecutive LiDAR scans of twenty beams each, and the agent generates an action that is scaled to be the steering angle. While the same reward signal is used as in Chapter 4, the values received by the agent are now different since the vehicle speed changes depending on the steering angle. The agent is trained using the TD3 algorithm for 100,000 steps so that the results can be compared to the other high-speed approaches.

Friction Model

The main physical limit experienced by racing vehicles is that lateral force F_y must not exceed the tyre friction limits F_{max}. The total tyre friction limit can be calculated using linear friction as $F_{\text{max}} = bmg$, where b is the friction coefficient, and g is the acceleration due to gravity. The lateral force is directly related to the vehicle's mass m and lateral acceleration a_y as $F_y = m \cdot a_y$. The lateral acceleration can be calculated from the kinematic equation for curved motion as $a_y = v_x \cdot \omega$, with ω being the angular speed. The equations for the kinematic bicycle model calculate the yaw rate as $\omega = v_x \tan(\delta)/L$, where L is the vehicle's wheelbase and δ is the steering angle. Therefore, the limit of friction can be described through the inequality,

$$bmg > \frac{v_x^2}{L} \tan(|\delta|)m. \tag{5.6}$$

This equation can be rearranged to calculate the maximum speed for given steering input δ_{ref} as

$$v_x(\delta_{\text{ref}}) = \sqrt{\frac{bg}{\tan(|\delta_{\text{ref}}|)/L}}. \tag{5.7}$$

This equation is subject to the velocity being smaller than the maximum linear velocity. The speed that is calculated, v_x, is used as the speed reference that controls the vehicle. The vehicle parameters used are $b = 0.523$ and $L = 0.33$ m. Using this equation to calculate the vehicle's speed is called the *calculate speed* function.

Tuning

Racing at a vehicle's performance limits inherently requires planning because the vehicle has momentum and takes time to change its speed. This is illustrated by a vehicle slowing down before starting to turn a corner. A shortcoming of using a friction model to calculate speed is that it is reactive and does not plan for the future. While the model presented above represents the friction limit, the vehicle must remain inside the limit, even when changing state. The steering angle can change much faster than the vehicle's speed, and this can cause the vehicle to exceed the friction limit and drift. For example, using the friction model limit with the pure pursuit planner results in the planner drifting and crashing since when the vehicle starts to turn a corner, the vehicle has not yet had time to slow down enough.

To overcome this problem, a reduced friction limit is used that incorporates a buffer region so that the vehicle does not breach the friction limit. This approximation uses a safety factor to reduce the calculated speed and a cap to clip the maximum allowed speed. The function is implemented by reducing the maximum vehicle speed and using a safety factory, $f_s \in [0, 1]$. The speed is multiplied by the safety factor and clipped according to the maximum as $v = \min(v_{\text{tuned\_max}}, f_s \cdot v_x)$. While this reduces the performance from the true friction limit, the approximation is essential for this method to be feasible.

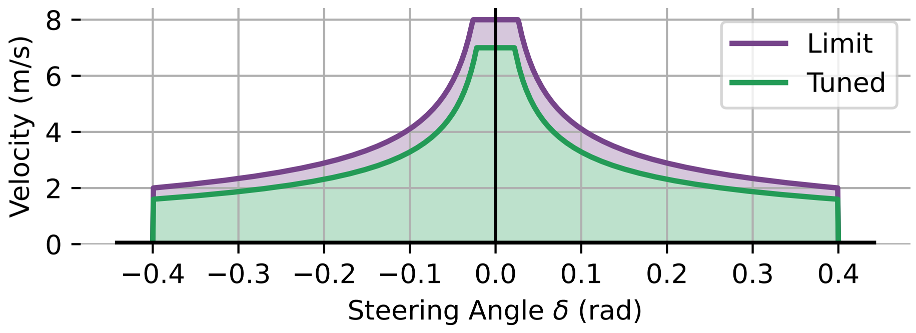


Figure 5.11: Speed at the friction limit (Limit) and at the tuned friction limit (Tuned) for an F1/10th vehicle. The shaded areas represent the velocity-steering space that keeps the vehicle from drifting.

The safety factor and the maximum speed allowed were tuned using the calculate speed method with a pure pursuit controller. The pure pursuit planner was set to follow the racing line and the parameters that allowed the vehicle to not crash with the fastest lap time were $f_s = 0.8$ and to cap the speed at $v_{\max} = 7$ m/s. Figure 5.11 shows the speed at the friction limit (labelled Limit) and the speed at the tuned friction limit (labelled Tuned). The figure shows the steering-velocity graph with the shaded area indicating the regions which keep the vehicle within the friction limit.

5.3.2. Architecture Evaluation

The linked action architecture, abbreviated as the *link agent*, is evaluated for its ability to train agents for F1/10th racing. The learning is evaluated by measuring the training performance on four maps. The performance is evaluated by comparing the trained agents to a pure pursuit planner that follows the racing line and uses the tuned calculate speed function to select a speed.

Training

The link architecture training performance is evaluated by training ten agents to race on each of the MCO, GBR, ESP and AUT maps. The agents all use the cross-track and heading reward and the tuned friction model to calculate a speed reference. The agents are trained for 100,000 steps on each map and the average track progress during training is recorded.

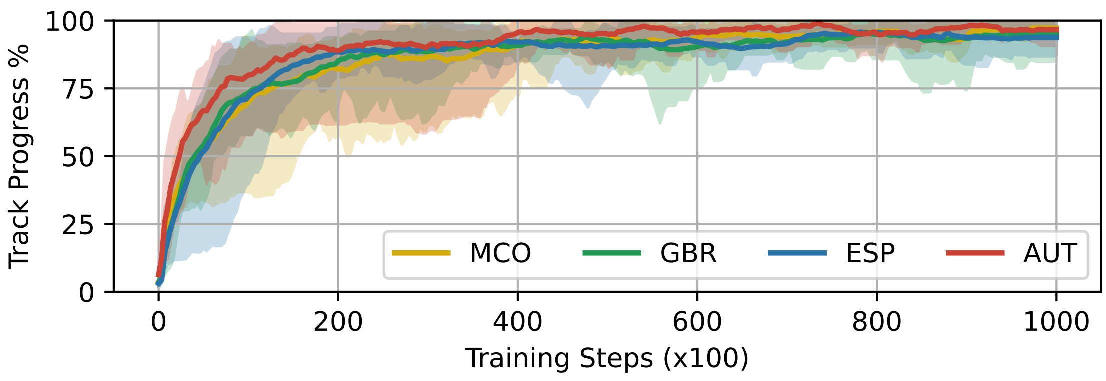


Figure 5.12: The average progress made by link agent on the MCO, GBR, ESP and AUT maps during training for 100,000 steps. The shaded regions show the minimums and maximums from 10 repetitions.

Figure 5.12 presents the progress during training for the link agent on four test maps. The graph shows that the average progress rises quickly on all of the maps, starting near zero and reaching around 95% by around 50,000 steps. The shaded regions are relatively small for each map, indicating that the training is stable and the random seed does not have a big effect. All of the maps show similar behaviour, with the agent trained on AUT (red line) slightly outperforming the other maps. This result indicates that the learning formulation is not dependent on the map used. While the agents were trained for 100,000 steps, the lack of change in the graph after 50,000 indicates that the link agent may require less training. Therefore, it is concluded that training agents using the link architecture is highly stable, and independent of the map used or random seed, with all training runs achieving a high average progress of over 90%.

Performance Evaluation

The behaviour of the link agent is now compared to the pure pursuit planner that follows the racing line and uses the same tuned friction model to calculate the speed reference. Figure 5.13 shows an example trajectory taken by the pure pursuit planner (left) compared to the link agent (right).

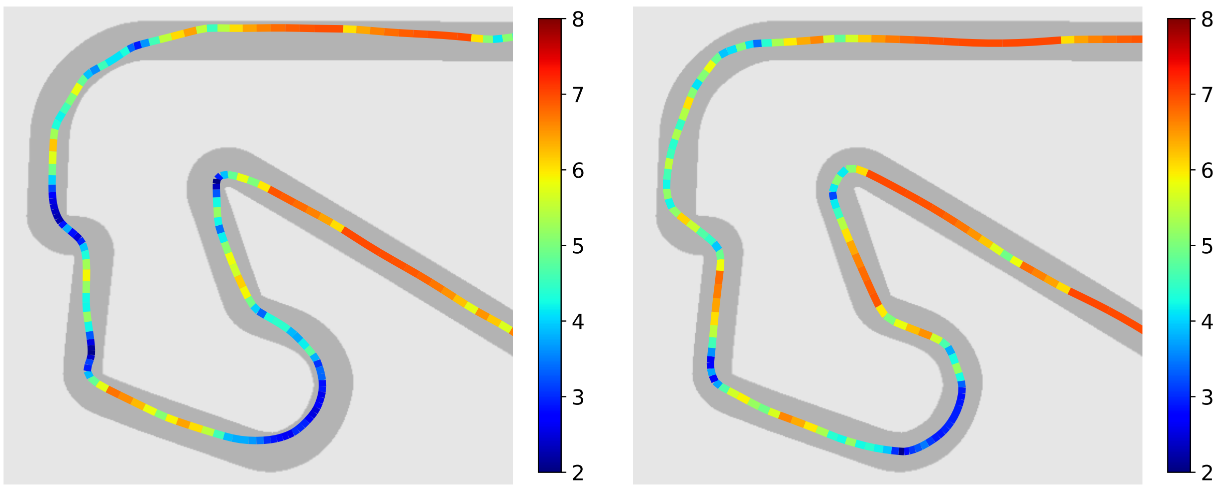


Figure 5.13: Example trajectories taken by the pure pursuit planner (left) and link agent (right) on the MCO track, both using the same method to calculate speed.

In Figure 5.13, the link agent trajectory (right) shows the agent selects faster speeds (orange and red) for the straights and slower speeds (green and blue) around the corners. Comparing the link agent performance with the pure pursuit planner shows that the agents select similar paths with similar speed profiles. Both trajectories have sudden changes in the line colour, showing that the vehicles change speed quickly. Visually it appears that the pure pursuit planner has a slightly smoother speed profile, and selects slightly lower speeds due to the increased dark blue regions. The link agent trajectory demonstrates that the design is successful because the agent slows down around the corners and speeds up for the straight sections, in a similar manner to the pure pursuit agent.

Further analysis is done by plotting the velocity profile of the two agents for the laps shown above. Figure 5.14 presents the velocity profiles with the link agent shown in red and the pure pursuit planner in blue.

The steering profile in Figure 5.14 shows that the link agent selects similar steering actions to the pure pursuit planner. The link agent occasionally shows spikes of large steering angles in the graph. This means that for most of the lap, the link agent follows a similar pattern to the pure pursuit planner. The speed profiles also show that the link agent selects similar speeds to the pure pursuit planner. This confirms that the link agent can select an appropriate speed profile, slowing down in similar places to the pure pursuit planner. The evaluation has demonstrated that the link architecture can be trained stably to achieve a high average progress and can select an appropriate speed profile of slowing

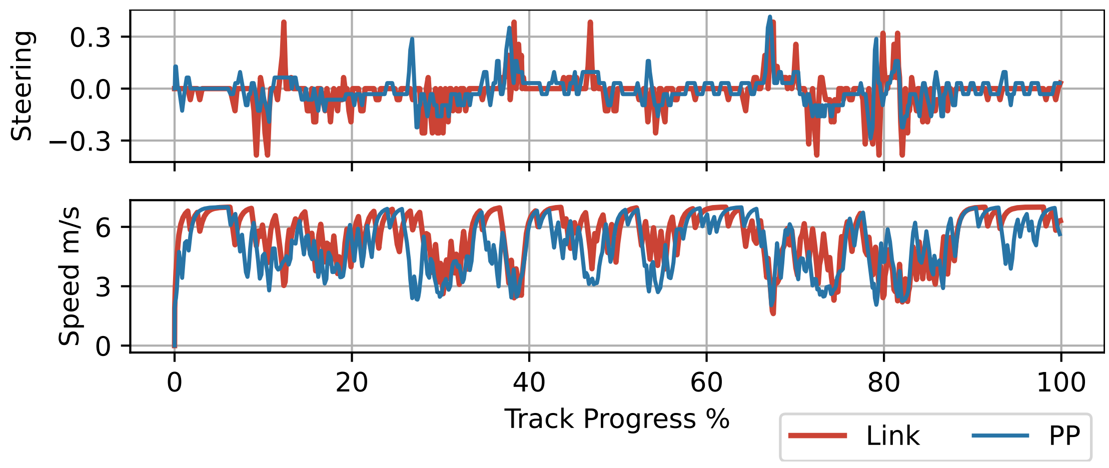


Figure 5.14: The steering and speed profiles selected by the pure pursuit (PP) planner and the link agent on the ESP track.

down around the corners and speeding up in the straights.

5.4. Comparative Evaluation

This section compares conventional end-to-end training (called the conventional agent), agents trained with the pure pursuit policy search (PPPS) reward and the linked action architecture. The conventional agent uses the cross-track and heading reward because it showed better results. The pure pursuit planner is used as a reference to compare the trajectories. The evaluation in §4.5 showed that conventional learning formulations were highly dependent on the vehicle's maximum speed. Since agents using maximum speeds of 7 or 8 m/s could not complete any test laps, conventional agents with a maximum speed of 6 m/s are used as the baseline for comparison.

5.4.1. Training Comparison

The ability of the different learning formulations to train the agents for the task of autonomous racing is considered by training agents on the ESP map. The training of the different agents is considered by training 10 agents from each group for 100,000 steps. The average progress during training is used to evaluate their performance.

Figure 5.15 shows the average progress during training for the conventional end-to-end agent with a maximum speed of 6 m/s (E2e6), the PPPS agent and the link agent on the ESP map. The link agent trains the fastest and most stably, as shown by the red line that quickly rises to achieve near 100% average progress. The PPPS agent reaches an average progress of around 70%, outperforming the conventional agent with a maximum speed of 6 m/s, which only reaches around 60%. Both the link and PPPS agents train to a higher average progress level than the conventional agent with a maximum speed of 6 m/s. The link agent has a maximum speed of 7 m/s and the PPPS agent has a maximum

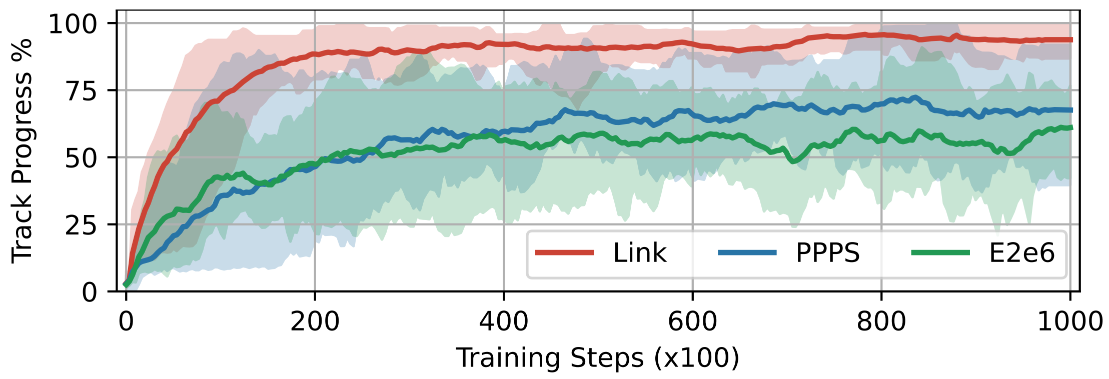


Figure 5.15: Training graph for the conventional end-to-end with a maximum speed of 6 m/s (labelled E2e6), PPPS and Link agents on the ESP map.

speed of 8 m/s. Therefore, both the link architecture and agents trained with the PPPS reward signal train to a higher level of average progress, using higher maximum speeds than conventional learning formulations.

5.4.2. Performance Comparison

The performance of the conventional agent, link architecture and agents trained with the PPPS reward is evaluated and compared to a pure pursuit planner following the racing line. The pure pursuit planner in this evaluation uses the speed references from the optimal trajectory. The metrics of lap time, total curvature, average velocity, success rate and average progress are used to represent the results. The results are the average from 20 test laps.

Metric	Link	PPPS	E2e6	PP
Lap Time (s)	44.62	42.53	42.03	46.70
Curvature (m^{-1})	70.23	72.43	86.21	70.94
Avg. Velocity (m/s)	5.35	5.54	5.80	4.92
Success Rate (%)	100	70	30	100
Avg. Progress (%)	100	90	75	100

Table 5.2: The lap-time, total curvature, average velocity, success rate and average progress for the link architecture, PPPS agent, a conventional agent with a maximum speed of 6 m/s (E2e6) and the pure pursuit (PP) planners on the ESP track.

Table 5.2 presents a comparison of how well each of the methods performs for the task of high-speed autonomous racing. All the learning agents achieved faster lap times than the pure pursuit planner of 46.7 s. While the conventional agent performed the fastest with a time of 42.03 s, only 30% of the laps were completed with an average progress of 75%. This low completion rate indicates that the trajectories being taken are near to infeasible, since the lap time is not repeatable.

The PPPS agent has a lower lap time of 42.53 seconds compared to the link agent of

44.62 seconds. This is explained by the higher average velocity of 5.54 m/s for the PPPS agent, compared to 5.35 m/s for the link agent. The PPPS uses the full speed range of up to 8 m/s, compared to the link agent, which only reaches 7 m/s. While the link agent completed all the test laps with a 100% success rate, the PPPS agent only completed 70% of the laps without crashing. The high average progress of 90% indicates that the PPPS agent only crashes towards the end of the lap, still achieving good performance. The link and PPPS agents both have significantly lower curvatures of 70.23 and 72.43, respectively, compared to the conventional agent of 86.21. The link agent achieves a comparable curvature to the pure pursuit planner.

These results demonstrate that the link architecture and PPPS agent outperform the conventional planner by achieving an improved success rate. The link agent completed all test laps while achieving a lower curvature than the pure pursuit planner. Critically, the link and PPPS agents could outperform conventional learning while using higher maximum speeds of 7 m/s and 8 m/s, respectively. The ability of the link and PPPS agents to outperform conventional learning by using higher maximum speeds is further explored by analysing the trajectories taken, actions selection, speed profiles and slip angles.

Trajectory Comparison

The trajectories are analysed to compare the different planner's abilities to select appropriate speed profiles of speeding up in the straights and slowing down in the corners. Baseline agents using the conventional learning formulation with the cross-track and heading reward are used for comparison. Trajectory segments from conventional agents using a maximum speed of 6 m/s and 8 m/s are used.

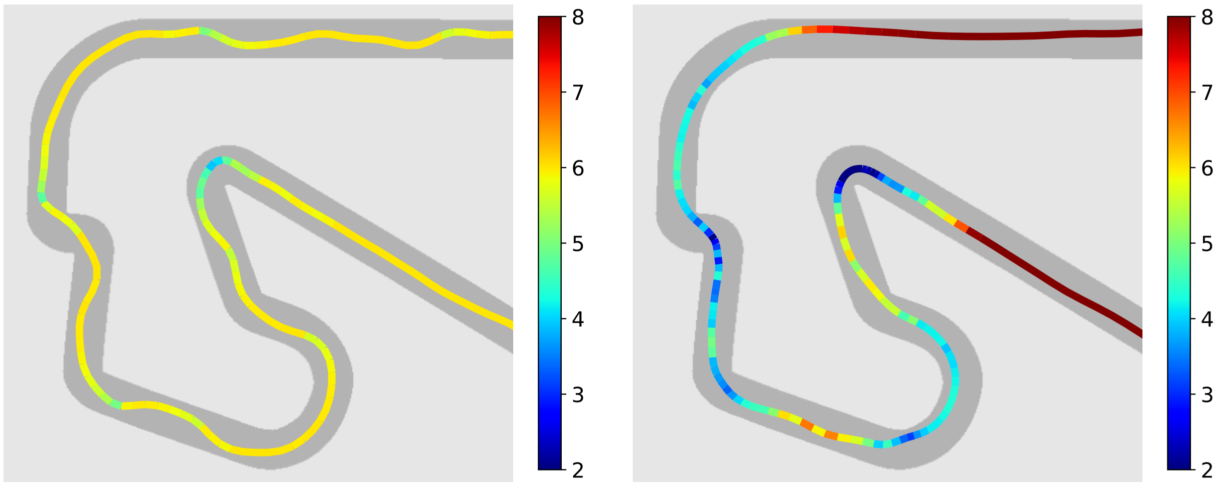


Figure 5.16: Comparing example trajectories from the conventional agent with a maximum speed of 6 m/s (left) and the PPPS agent (right) on a section of the ESP track.

Figure 5.16 compares the trajectories taken by the conventional agent with a maximum speed of 6 m/s and the PPPS agent on the ESP track. The conventional agent trajectory is mainly yellow, showing that the vehicle almost always travels at 6 m/s, slowing down slightly in the hairpin corner, as indicated by the blue region. The path taken by the conventional agent is squiggly due to the vehicle drifting with high-slip angles. In contrast, the PPPS agent speeds up and slows down, as shown by the dark red lines in the straight sections and dark blue sections around the corners. The PPPS agent travels smoothly around the track; it cuts the corners and takes a low curvature route, allowing for high-speed selection without losing traction. This trajectory shows that the PPPS agent outperforms the conventional agent by speeding up and slowing down and selecting a smoother path. Critically, the PPPS agent is able to use the full speed range of up to 8 m/s, while the conventional agent is limited to 6 m/s.

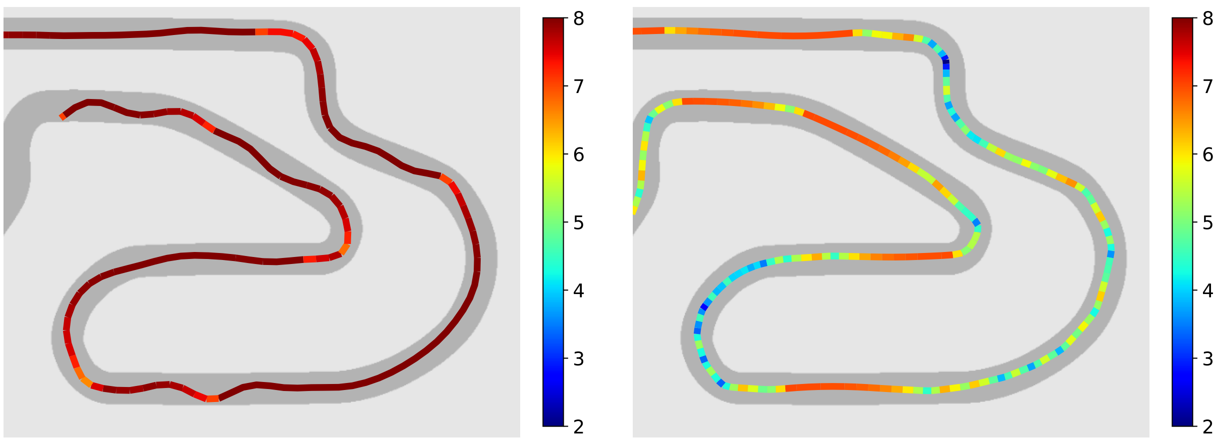


Figure 5.17: Comparing example trajectories from the conventional agent with a maximum speed of 8 m/s (left) and the link agent (right) on a section of the ESP track.

Figure 5.17 shows example trajectories from the conventional agent with a maximum speed of 8 m/s (left) and the link agent (right) on a section of the ESP track. The conventional agent selects almost always the highest speed, as shown by the dark red line. This results in the vehicle swerving around the track and ultimately crashing, as shown by the line stopping. The link agent drives mainly in the centre of the track while speeding up (shown by the orange line) and slowing down (shown by the green and blue lines). This change in the colour profile demonstrates that the link agent has good racing behaviour of selecting a smooth path, speeding up and slowing down. The link agent's ability to speed up and slow down without crashing demonstrates an improvement over conventional learning.

The trajectory analysis concludes that the PPPS and link agents overcome the problem of poor speed selection and high curvature that the conventional agents experienced. The ability to select higher speeds in the straights and lower speeds in the corners enables them to use higher maximum speeds of 7 m/s with the link agent and 8 m/s with the PPPS agent.

Action Analysis

The actions selected by the trained conventional and PPPS agents are compared to demonstrate the difference caused by the change in learning formulation. Figure 5.18 shows the comparison of the actions selected by the conventional and PPPS agents. The steering graph shows that the conventional agent selects many extreme steering actions, a problem noted in the literature [22]. The PPPS agent selects many smaller steering actions showing that it has learned a more moderate racing behaviour. Selecting smaller steering actions causes the forces on the tyres to be smaller. While the PPPS agent selects more moderate steering actions on average, there are still regions, such as between 15% and 20% progress, where the PPPS agent's steering angle still fluctuates.

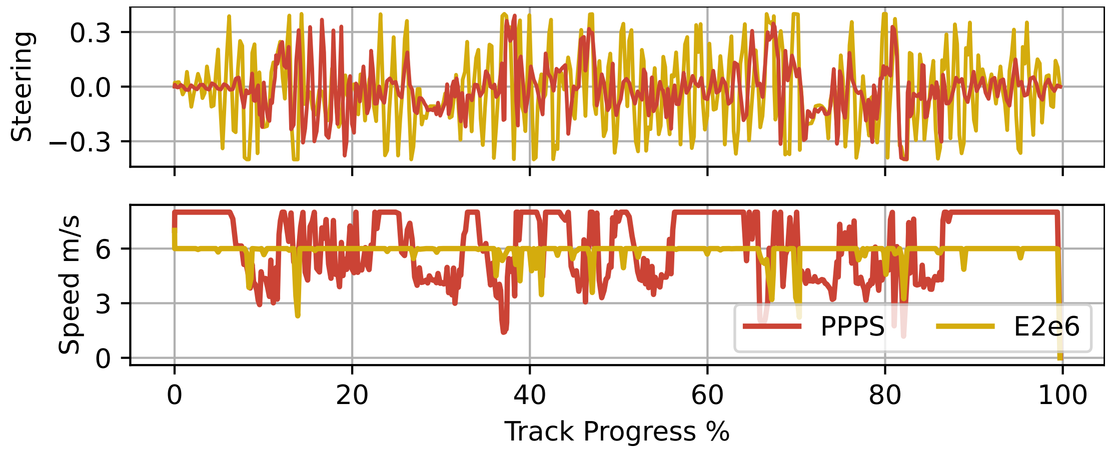


Figure 5.18: Comparison of the steering and speed actions selected by the conventional agent with a maximum speed of 6 m/s (E2e6) and PPPS agent on the ESP map.

The speed graph confirms what was seen in the trajectories in Figure 5.16; the conventional planner almost always selects the same speed (the maximum), and the PPPS agent learns an appropriate speed profile of speeding up and slowing down. The PPPS agent's speed increases to the maximum in some regions and slows down to between 3 m/s and 4 m/s in some regions. While both use the same architecture of the agent selecting two actions, the PPPS agent learns to select much more moderate actions, with fewer spikes in the steering angle and more variation in the speed profile selection. This result demonstrates that the PPPS reward results in more moderate action selection while using a higher maximum speed.

Speed Profile Analysis

The speed profiles selected by the PPPS and link agents are now compared in detail to understand how each planner speeds up and slows down. The speed profiles of the PPPS and link agents are plotted for a section of the ESP track so that the agents' behaviour can be analysed. The conventional planner is neglected from this study since it has already been seen that it is unable to select a feasible speed profile.

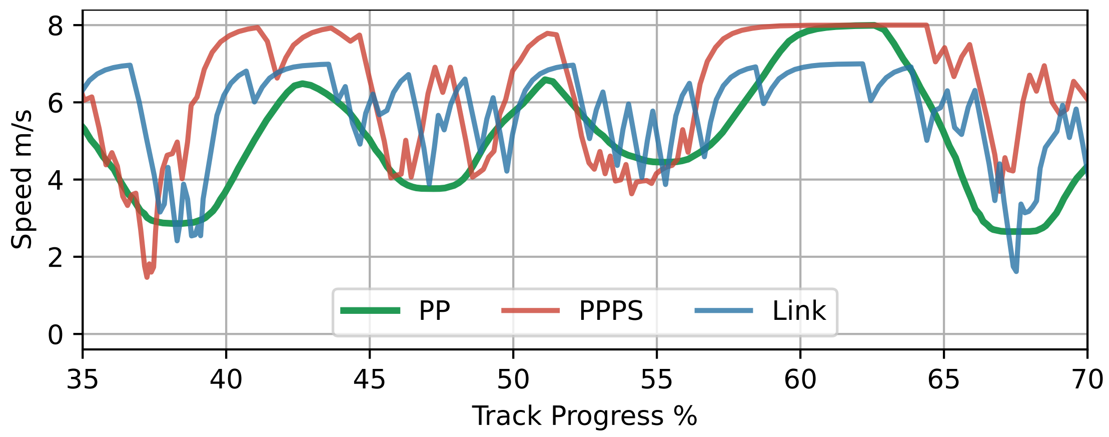


Figure 5.19: The speed profile of vehicles controlled with the pure pursuit (PP), PPPS and link planners on a portion of the ESP track.

The graph in Figure 5.18 shows the speed profiles of vehicles controlled with the pure pursuit, PPPS and link planners on a portion of the ESP track. The pure pursuit planner (green line) selects a speed smoothly moving through the track, and the agents both have less smooth speed profiles. The PPPS and link agents generally select speeds slightly higher than the pure pursuit planner; for example, around the 40% progress mark, the pure pursuit planner has a speed of 4 m/s, the link agent 6 m/s and the PPPS agent 7.5 m/s. The link agent shows a jerky speed selection, with many sharp corners in the speed profile; for example between 50 % and 55%, the link agent fluctuates below and above the pure pursuit speed four times. The PPPS agent selects a smoother speed profile but often neglects to slow down enough for the corners. This is seen at the 65% mark, where the pure pursuit planner slows down and the PPPS agent tracks the gradient of the line while being on average 1.5 m/s faster. The fluctuations from the PPPS agent are smaller than the link agent with their magnitude mainly being within 1 m/s.

The investigation into the speed profiles concludes that the link agent selects more conservative speed profiles than the PPPS agent. The PPPS agent is more aggressive, often selecting a higher speed than the pure pursuit planner. Both agents show some fluctuations with the link agent having bigger fluctuations than the PPPS agent. In general, both agents show a similar pattern in speed profile to the pure pursuit planner.

Slip Angle Analysis

The slip angles of the pure pursuit, conventional, PPPS and link agents are studied to understand the frictional forces on the tyres caused by each agent. The slip angle is recorded directly from the simulator and plotted for a lap of the ESP track. The absolute slip angle is used since the magnitude is significant and the sign is irrelevant.

Figure 5.20 shows the slip angles for the pure pursuit planner, conventional 6 m/s, PPPS and link planners on the ESP track. The pure pursuit planner has the lowest slip angle with the highest value of 0.2 radians, and an average of less than 0.1 rad.

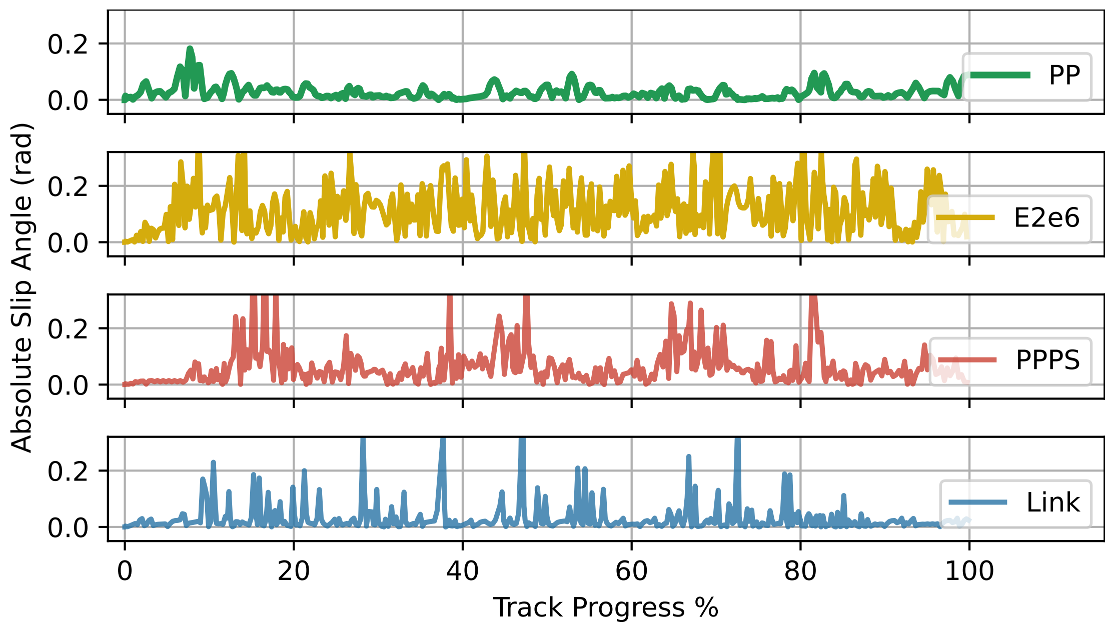


Figure 5.20: The absolute slip angle from the pure pursuit (PP) planner, conventional 6 m/s (E2e6), PPPS and link planners on the ESP track.

The conventional planner with a maximum speed of 6 m/s (E2e6) has a large slip angle, frequently beaching the 0.2-radian line and having an average above 0.1 radians. The PPPS improves on the conventional planner, having a smaller slip angle of around 0.05 radians, with occasional spikes above 0.2 radians. The link planner generally has a small slip angle below 0.05 radians with occasional large spikes.

The first comparative observation is that both the PPPS and link agents improve on the conventional planner by having lower slip angles on the ESP track. Critically, the PPPS and link agents have smaller slip angles while using higher maximum speeds of 7 m/s and 8 m/s, compared to the conventional planner with a top speed of 6 m/s. A limitation of the link and PPPS agents is that they both show occasional spikes in the slip angle which should be addressed in future work. The conservative nature of the link agent, compared to the PPPS agent is reiterated here with the link agent having a generally smaller slip angle.

5.4.3. Literature Comparison

The methods presented here are compared with the results from the literature. Many methods use uncommon maps that cannot always be easily obtained. Bosello et al. [60] and Brunnbauer et al. [22] both use the same simulator and the maps are available in the associated GitHub repository[1]. They use a simulator that employs PyBullet dynamics engine to represent the vehicle dynamics. While the simulator used by these authors differs from the simulator used in this work, it aims to model the same vehicles.

[1]https://github.com/MichaelBosello/f1tenth-RL

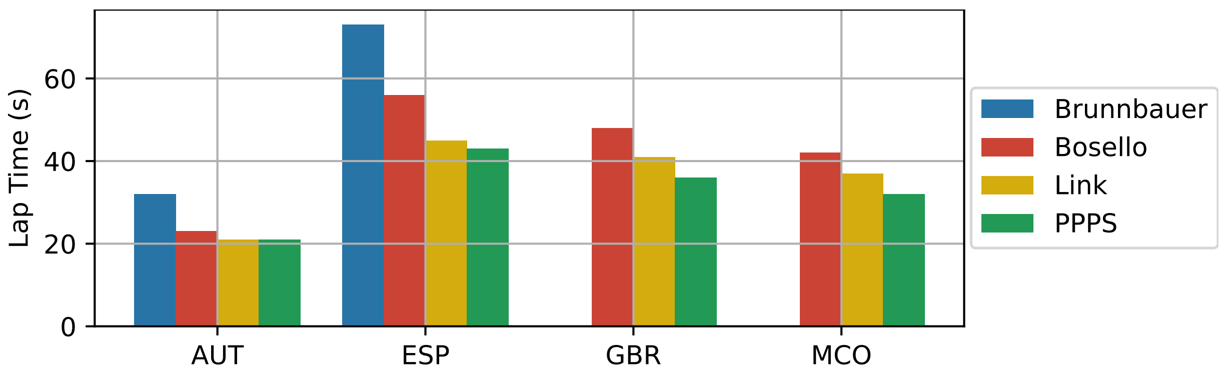


Figure 5.21: Lap time comparison of the PPPS and Link planners to Bosselo et al. [60] and Brunnbauer et al. [22].

Figure 5.21 shows the lap times of the PPPS and link agents compared to Bosello and Brunnbauer. The graph shows that the link architecture and agents trained with the PPPS reward outperform the results presented by Bosello and Brunnbauer in terms of lap time. For example, on the ESP map, Brunnbauer achieved a time of 73 seconds, which Bosello beat with a time of 56 seconds. The link agent has a time of 45 seconds, and the PPPS agent performs the fastest with a time of 43 seconds. On all the maps, the PPPS agent performs the fastest of all the methods considered.

Therefore, this study concludes that the novel link architecture and PPPS reward are effective for training agents to select appropriate speed profiles, outperforming similar studies in the literature in terms of lap time. It is proposed that the key reason that the methods presented in this dissertation achieve faster lap times is that they use the full speed range of up to 8 m/s. While the other authors do not directly report the speeds used, the GitHub repositories indicate that the maximum speed used was 5 m/s. While further analysis of the speed profile selection would be indicative of the reasons for the lower lap times, neither work provided such information.

5.4.4. Discussion

Chapter 4 showed that a key problem in using DRL agents for high-speed racing is poor speed selection, specifically, not slowing down around corners and thus often crashing. In response, this chapter presents improved learning formulations for high-speed F1/10th racing. The key design methodology uses analytical vehicle models in the learning formulation design to aid the learning. New reward signals using the speed profile from the racing line were used to guide the learning, and the link architecture that used a friction model to select a speed based on the agent's steering angle was presented.

The results showed that using the waypoints from the racing line decreased the distance travelled, total curvature and mean steering angle compared to using the centre line. Of the four reward candidates presented, the only one to achieve good racing performance was the pure pursuit policy search reward, which could train an agent to select a speed profile similar to the optimal trajectory. The PPPS agent still selected slightly higher speeds than the racing line, especially on the exit of corners.

The link architecture achieved its goal of simplifying the learning formulation by producing a policy that trains repeatably and converges in a smaller number of training steps. The link agent selects a similar profile to the pure pursuit planner, with the difference of not slowing down as far before corners. Critically, the link agent achieved a success rate of 100%, confirming that it selects safe, high-performance trajectories. A slight limitation of the link architecture is that it only selects speeds up to 7 m/s, thus not using the complete speed profile of the vehicle.

The comparative evaluation showed that both methods improved the speed profile selection of the baseline greatly, speeding up and slowing down appropriately. The link agent selects slightly lower speeds than the PPPS and, being more conservative, achieves a 100% completion rate, which is higher than the PPPS of 70%. The slip angles of the PPPS and link agents are significantly better than the end-to-end agent, making these solutions physically feasible. The comparison with lap times from the literature showed that both the link and PPPS agents achieved lower lap times than Bosello et al. [60] and Brunnbauer et al. [22].

A key part of the design and evaluation process was measuring a large set of variables, such as speed profile through the track, slip angle, completion rate and average velocity, mean steering and speed, curvature and action profile analysis. The in-depth evaluation aided in identifying the problem and validating the effectiveness of the proposed approach.

This chapter concludes that the PPPS and link architectures outperform previous methods regarding lap times, completion rates and slip angles. These improvements are due to incorporating knowledge from vehicle models into the learning formulation. Future methods should continue to consider ways to improve DRL formulations by using analytical models.

Limitations

A significant limitation in all of the methods presented is the lack of ability for the agent to plan. This is because the agent does not know where it is on the map and, therefore, cannot harness the power of the map to consider what is coming in the future. Ultimately, the agent requires a method of perceiving the future like a human who has driven on a track does. This could be addressed in the future by training agents with localisation or perception networks [113] that can learn the track's structure so that the agent can plan for the future. An alternative approach is to learn a dynamics model that could be used for planning [22].

While the learning formulations presented in this chapter greatly improved the problem of jerky steering actions (and thus high-curvature paths), the paths selected by the agent can still be improved. The link agent has occasional spikes in the steering angle, and the

PPPS agent selects a jerky speed profile. Future work could consider action-conditioned learning, where additional reward penalties are used to encourage agents to select spatially and temporally smooth actions [114].

5.5. Summary

This chapter addressed the problem of speed selection in DRL agents, which caused poor completion rates and high-slip trajectories. Analytical vehicle models were employed to improve the learning formulations. Racing line reward signals used the optimal trajectory to provide more specific rewards to the agent, training it to copy the behaviour of a pure pursuit planner. This approach has been shown to produce fast lap times due to appropriate speed selection but suffered from only achieving a 90% average progress. The link architecture uses a DRL agent to select a steering angle and a friction model to select a speed reference. The link architecture trains repeatably and quickly, and the trained agents achieve a 100% completion rate. A comparative evaluation demonstrated that the PPPS and link agents could achieve faster lap times of four popular tracks than other methods in the literature due to driving at faster top speeds. An in-depth analysis of the action's selection, speed profiles and slip angles were used to better understand the results and validate that the proposed approaches select an appropriate speed profile.

Chapter 6

Supervisory Safety System

It is difficult to evaluate DRL algorithms at high speeds, on physical vehicles. Three reasons that contribute to this are (1) the sim-to-real problem caused by the difference between simulation and real-world dynamics, (2) the black-box nature of DNNs, and (3) the safety challenges present in high-performance control tasks, demonstrated by poor completion rates, even in simulation. In response to these problems, this chapter presents an supervisory safety system (SSS) that enables unsafe planners to be safely tested. The SSS uses a viability kernel (list of safe states) and a vehicle dynamics model to ensure that the actions selected by the agent are safe. The general racing vehicle kernel formulation is explained, and the specifics of how the kernel is implemented for F1/10th racing are described. The evaluation measures the kernels for different maps, validates the kernel by using a worst-case scenario planner, tests the kernel's robustness to noise in localisation, and analyses how the kernels can be adapted for physical implementation. The results demonstrate that the supervisor is effective in ensuring safety while having a small impact on the vehicle's performance due to over-conservative safety guarantees. The SSS that is presented here is used in Chapter 7 to train agents online.

6.1. Introduction

Machine learning (ML) offers many advantages to autonomous racing, such as high-performance control without localisation and flexibility in new environments. However, it is difficult to harness these advantages due to current challenges in applying ML algorithms to real-world platforms. The current method for applying many ML algorithms to physical robots is to train the agent in simulation and then transfer the trained policy to the physical robots [24]. Transferring trained policies struggle to overcome the sim-to-real problem, which is the performance drop caused by the difference between simulation and real dynamics.

The sim-to-real problem is further complicated by the black-box nature of DNNs. Xiao et al. [99] point out that interpretability and explainability are major challenges in using machine learning algorithms on real-world systems. The black-box nature of DNNs is that it is a collection of numbers (weights and biases) that can be trained to represent a complex mapping of input to output vectors but have no understandable meaning. The problem arising from the lack of interpretability is that the network's behaviour cannot be known or guaranteed to meet any requirements apart from experimentation.

If a system is known to work and can be easily evaluated for safety, then there is no need to understand what the numbers inside the black box mean. However, in the case of sim-to-real transfer, it is essential to know that the vehicle will not crash. High-performance racing is a difficult problem, as demonstrated by the fact that some simulation-based solutions still crash [19, 52], and it has even been claimed that model-free algorithms are incapable of high-speed F1/10th racing [22].

For as long as DRL agents do not have safety guarantees, they are not a practical solution to any real-world robotics problem. While it might be tolerable if a small robot crashes during an experiment, it is unacceptable for any large robot to fail during operation, possibly causing damage to itself, its environment or humans. For DRL algorithms to be scalable, methods for ensuring their safety must be evaluated.

This chapter addresses the problem of safety in autonomous racing by designing an SSS that ensures that a vehicle does not crash. The supervisory architecture and process of ensuring that only safe actions are implemented is described in §6.2 In §6.3, an overview of Viability Theory is presented and used to derive the viability kernel for a racing car. The viability kernel is evaluated in §6.5 to show that it can keep vehicles safe, and an ablation study evaluates how the system responds to noise in the state and action space.

6.2. Safety Supervisor Design

A safety system is proposed that uses a supervisor to monitor the actions selected by an experimental planner that may be unsafe. Figure 6.1 shows a schematic of the architecture with the supervisor using the vehicle pose to guarantee that the action that is implemented is safe. The supervisor is placed in series after the planner and uses the vehicle pose $\mathbf{x}_t$ to evaluate if the planner's action $\mathbf{u}_0$ is safe. If the planner's action is unsafe, then a safe action must be selected.

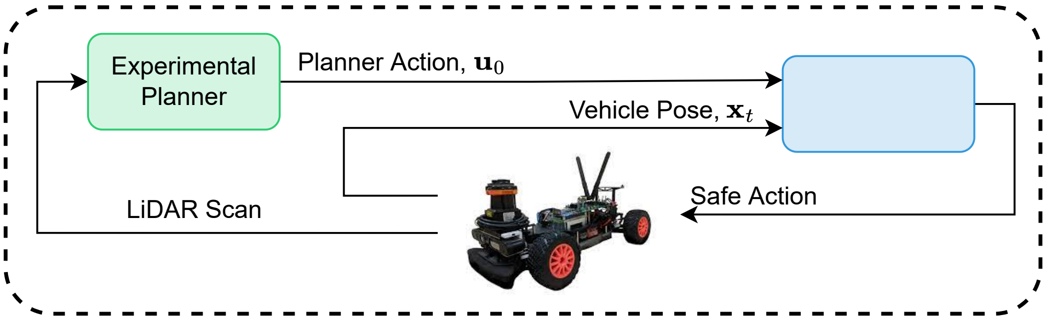


Figure 6.1: The supervisor is placed after an experimental planner with learning components to ensure that only safe actions are implemented on the vehicle.

The supervisor ensures safety through a three-step process of (1) propagating the next state, (2) checking if the next state is safe, and (3) if unsafe, selecting a safe action. Figure 6.2 shows a schematic of the safety process. The process starts with calculating where the vehicle will be after a planning timestep. The single track dynamics model (explained in §3.2.2) is used to calculate the next vehicle pose $\mathbf{x}_{t+1} = f_{\text{ST}}(\mathbf{x}_t, \mathbf{u}_0)$, using the planner's action.

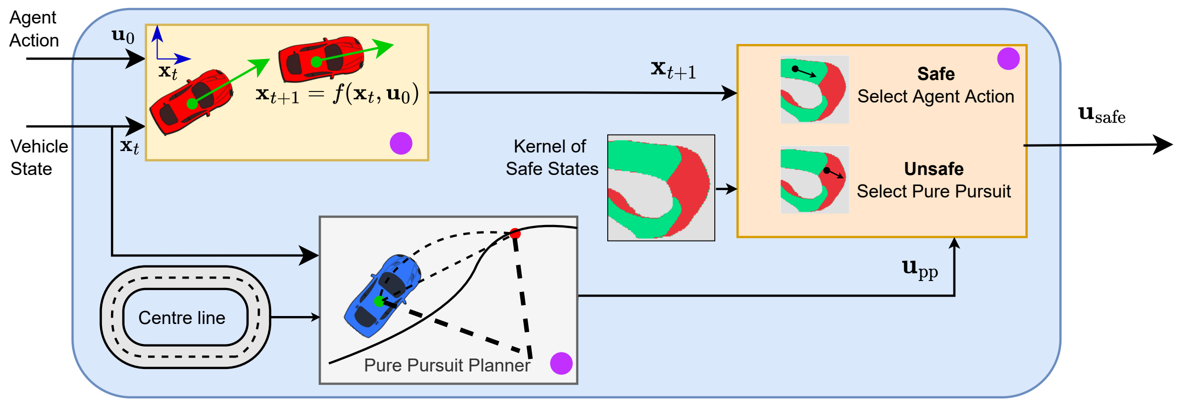


Figure 6.2: The supervisor checks if an action is safe by seeing if the next state is in the kernel of safe states. If the action results in an unsafe state, then a pure pursuit planner is used to select a safe action.

The decision module receives the next vehicle state $\mathbf{x}_{t+1}$ and must return a binary answer for if the agent's action is safe or unsafe. Safe means that after taking an action, the vehicle will not crash, be within the friction limit, and be able to select another safe action. Viability Theory is used to create a list of states, called a kernel, that meets the definition of safe (explained in §6.3).

The next state is converted into a discrete state. The decision module checks to see if this discrete state is in the kernel of safe states or not. The kernel is a list of discrete states made up of position, orientation, speed and steering angle. The kernel can be implemented using a hash table that can be efficiently queried to check if a state is safe. The decision module then uses this binary value (safe or unsafe) to either return the planner's original action or select a safe action.

If the state is unsafe, a pure pursuit controller, following the centre line, is used to select an action. The speed is selected using the formula in §5.3 to calculate a speed relative to the current steering angle. This ensures that the supervisor always moves the vehicle towards the middle of the track at a slow speed, which is the safest that it can be.

6.3. Viability Kernel Definition

The supervisor uses the viability kernel to check if the vehicle state is within a set of safe states. It is recalled that safety means that a state does not lead to a crash, remains within the friction limit and is recursively feasible. Viability Theory has been previously used in racing problems to improve the efficiency of MPC algorithms by generating a set of recursively feasible states for autonomous race cars [32, 115]. Viability Theory is focused on calculating viable solutions that remain within a complex constraint set and thus is a good fit for this problem.

This section provides an overview of Viability Theory and then explains how the theory is applied to the context of racing. A general planning model is considered and transformed into a discrete difference inclusion before the viability algorithm is explained. This general formulation is then applied to the problem of a racing car by presenting the model, transforming the model and formulating the kernel. The work in this section originates in the mathematical theories presented by Saint-Pierre [116], and the applications to racing developed by Liniger et al. [2, 32, 115]. Since Liniger's aim was to speed up the search for MPC solutions, and the focus of this work is to ensure safety, Liniger's application is modified to be more conservative.

6.3.1. Viability Theory

Viability Theory is concerned with generating a set of states that remain within a constraint set forever while evolving according to a set of dynamics. The basic problem of Viability Theory is to find the viability kernel, which is the set of states that remain within the constraints. In presenting the method of calculating the kernel, the general planning model is described, then the discretisation of the model is explained, and finally, the viability kernel algorithm is presented.

General Planning Model

We define a general control problem used for discrete-time planning in a robotic system. Consider a system that has a state, $x \in \mathbb{R}^n$, and control inputs, $u \in U \subset \mathbb{R}^m$. Transitions between states are given by the continuous function $f : \mathbb{R}^n \times U \mapsto \mathbb{R}^n$, such that $x_{k+1} = f(x_k, u_k, T_\mathrm{p})$, where the subscript k is the timestep. The discrete transitions occur with the planning timestep T_p.

To calculate the viability kernel, it is useful to write the dynamic system as a differential inclusion. A differential inclusion is a generalised form of writing ordinary differential equations as a set-valued map. In the case of the general planning model, the differential dynamics equations $f(x, u, T_\mathrm{p})$ are converted to a set-valued map $F(x)$, which is the set of all possible next states for a given initial state. The general model can be formulated as a difference inclusion as,

$$\begin{aligned} x_{k+1} \in & F(x_k), \quad \text{with} \\ & F(x_k) = \{f(x_k, u, T_\mathrm{p}) \mid u \in U\}. \end{aligned} \tag{6.1}$$

In Equation 6.1, $F(x_k)$ is a set of the possible states that the system could be in after any action in the control space is selected and implemented for the planning timestep, T_p. For this problem to be tractable, the set of states and control actions must be a finite number. For there to be a finite number of states and actions, the state and control spaces must be discretised.

Model Discretisation

The state space is discretised along each variable by applying each variable to a uniform grid. This transforms the space from $x \in \mathbb{R}^n$ to $\mathbf{x}_h \in \mathcal{X}_h$, where $\mathcal{X}_h$ is a countable list of discrete states parameterised by $h \in \mathbb{R}$.

The control space is also discretised by using a uniform grid across all the control dimensions. Further, the control space is split into a finite number of countable modes n_q. Each mode, q_j with $j \in [0, n_q]$, represents a control action, such that $q : \mathbb{R} \mapsto \mathbb{R}^m$, where m is the number of dimensions in the control space. Splitting the control space into a countable number of modes allows for all the dimensions of the control space to be represented using a single real number.

Viability Kernel Algorithm

Formally, the viability kernel is the set of states for which the system can remain within a constraint set forever while evolving according to a set of dynamics. Given a constraint set $K \subset \mathbb{R}^n$, solutions to the difference inclusion in Equation 6.1, which stay in K forever, are known as viable solutions. The task of generating the viability kernel is to find the set of all viable solutions.

Definition 1 [116]: A set $D \subset \mathbb{R}^n$ is a discrete viability domain of F, if for all $x \in D$, we have that $F(x) \cap D \neq \varnothing$. The discrete viability kernel of a set $K \subset \mathbb{R}^n$ under F, denoted by $D_F(K)$, is the largest closed discrete viability domain contained in K.

The definition implies that a discrete viability domain is defined by the states for which the intersection of the differential inclusion and the domain is not equal to the empty set. The viability kernel is defined as the largest discrete viability domain.

The viability kernel is built by recursively calculating the set of states that fit within the constraint set, K. This process is known as the viability kernel algorithm [116],

$$\begin{aligned} K^0 &= K \\ K^{i+1} &= \{x \in K^i \mid \forall\ F(x) \cap K^i \neq \varnothing\}. \end{aligned} \tag{6.2}$$

The viability kernel algorithm is initialised with a given constraint set, which is the zeroth iteration viability kernel. The algorithm recursively builds further iterations of the viability kernel, by selecting all the states within the previous kernel where the difference inclusion for that state intersects with the previous iteration of the kernel. For the difference inclusion of a state $F(x)$ to intersect with the kernel means that at least one of the control actions results in a next state that remains within the kernel. This process is repeated until all the states within the kernel are viable.

6.3.2. Racing Kernel Formulation

A vehicle model is used to formulate a viability kernel for racing applications. The model is transformed to formulate the system as a differential inclusion so that Viability Theory can be used. The transformation consists of discretising the control and state spaces and converting the dynamics into a set-valued map.

Kernel State Formulation

The first step in formulating the kernel is to select the variables that comprise a kernel state. The size of the kernel expands exponentially with each additional dimension, and therefore, the fewest number of state variables should be used that can accurately represent the state. The kernel is formulated using the state variables for the position, the vehicle orientation, and the dynamics mode. Therefore, the kernel is a 4-dimensional vector, which keeps the size reasonable while capturing the vehicle state.

The state is written as $\mathbf{x}_h = [X, Y, \theta, q]$, which includes the position, orientation and mode number. The mode is the two control quantities of steering and speed, represented as $q = [\delta, v_x]$. The position is discretised by splitting the track into uniform blocks with a resolution of n_{dx} blocks per metre. The orientation angle is discretised by dividing a revolution into n_θ angle segments.

Mode Definition

In racing, the control modes consist of steering and speed. Recalling the requirement of a safe system, the vehicle must remain within the friction limit. Therefore, the friction limit must be modelled and only control action combinations that are within the limit can be used.

The maximum lateral friction is reached when the lateral force is equal to the vehicle's total frictional force. Using this definition, for the vehicle to be within the friction limit, the following inequality must hold,

$$bmg > \frac{v_x^2}{L}\tan(|\delta|)m. \tag{6.3}$$

This inequality can be rewritten to find the maximum velocity within the friction limit for a given steering angle as,

$$v_x(\delta) < \sqrt{\frac{bg}{\tan(|\delta|)/L}}. \tag{6.4}$$

Generally speaking, this limits the speed for extreme steering actions. This makes intuitive sense that a vehicle travelling fast cannot turn sharply. Now that the friction limit has been established, the modes can be selected by gridding the speed and steering dimensions and numbering the modes.

State Update Model

A model is required to calculate the next state, given a current kernel state and mode action. The vehicle model comparison in §3.2.3 showed that at high speed, the single-track model deviates significantly from the kinematic model. Therefore, the single-track model is used to calculate the transitions between states with several modifications.

The two additional variables that form part of the single-track model (yaw rate and slip) are set to 0 for each next state calculation. The model takes velocity and steering angle as inputs to a simple proportional control system (as used on the vehicle) to calculate acceleration and steering velocity. The calculated acceleration and velocity are used as inputs in the single-track bicycle model. The model uses a small timestep of 0.01 seconds and is repeatedly updated until the simulation time is reached.

Mode Transition Calculation

Racing vehicles have dynamic limits that limit the rate at which the velocity and steering can change. The steering angle is typically able to change faster than the speed. This means that to keep within the friction limits, only certain mode transitions are allowed. For example, a vehicle must slow down before turning sharply and may not start to turn while still moving at high speed.

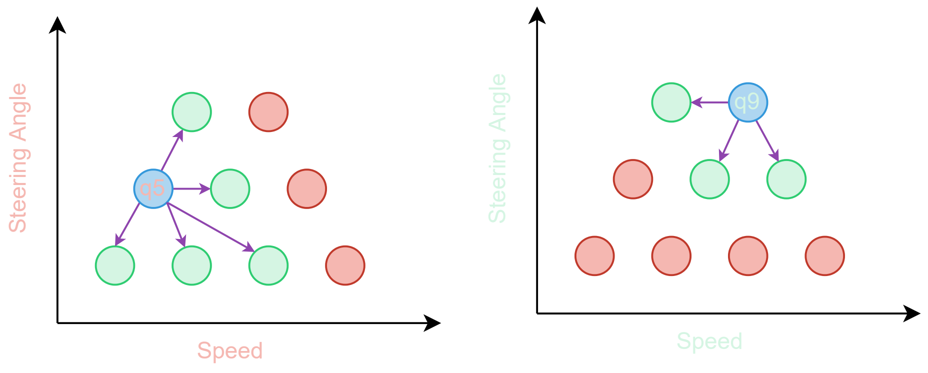


Figure 6.3: A schematic demonstrating valid mode transitions. For initial modes 5 (left) and 9 (right), arrows show the allowed transitions to valid modes of the state at the next timestep (shown by purple arrows). The red nodes are not reachable within the current planning timestep.

Figure 6.3 schematically shows the allowed transitions between modes. The allowed transitions are calculated by evaluating how much the velocity and speed can change within the planning step and then considering which other nodes are reachable. It is important when considering the mode and timestep discretization to ensure that all modes are reachable. For example, if the timestep is very small and there are few speed modes, then the vehicle speed will not be able to change enough during the timestep to reach the next mode.

Formulation Summary

The viability kernel has been formulated for car racing, and the states, modes, dynamics equations, and mode transitions have been defined. The definitions given above are now summarised as

$$\begin{aligned} x_{k+1} \in F(x_k), &\quad \text{with} \\ &F(x_k) = \{f_{\mathrm{ST}}(x_k, q_i, T_p) \mid i \in [0, 1, ..., q_n]\} \\ &x_k = [X_k, Y_k, \theta_k, q_k] \\ &q_i = [\delta, v_x]. \end{aligned} \tag{6.5}$$

Recalling the viability kernel algorithm in Equation 6.2, the only remaining definition required is K^0, the initial constraint set. For racing on a track, K^0 is defined as all the states where the vehicle is on the map and not in contact with the boundaries.

6.4. F1/10th Kernel Generation

This section shows how the kernel that was formulated in §6.3 is now implemented for F1/10th racing. After describing the implementation and generation process, an evaluation of the kernels for F1/10th racing is conducted, including an ablation study to show how the kernels react to noise in the vehicle location.

6.4.1. Implementation

Mode Selection

The modes for the task of F1/10th race are identified using the friction formula in Equation 6.4. Figure 6.4 shows how the friction limit is determined for F1/10th vehicles and how the modes are positioned within the limit. The speed is discretised by splitting the speed range into a fixed number of modes. For each speed level, three steering modes are selected by calculating the steering angle on the friction limit on each side and a steering angle of zero.

The decision to use only three steering modes on the edge of the limits is because the kernel gets exponentially larger with each additional mode, so the list should be as small as possible while still enabling fine control of the vehicle.

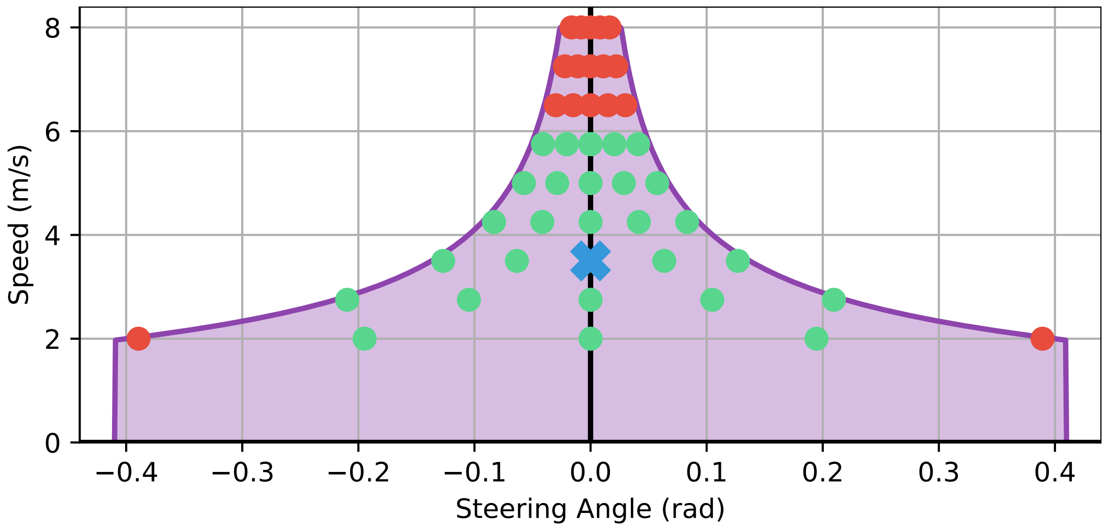


Figure 6.4: Illustration of the modes used with evenly spaced velocity points and steering modes on the friction limits. For the initial mode shown in blue, the green modes represent the allowed transitions, and the red modes are not reachable within the current planning timestep.

Discretisation

The states and actions must be discretised along each dimension for the kernel to be generated. Table 6.1 shows the parameters used in the kernel generation process. The race track is gridded into n_{dx} points per metre using a uniform grid. The angle range of $[-\pi, \pi]$ is gridded into n_θ equal angle segments.

Parameter	**Value**
Number of X, Y discretisation points, n_{dx}	40 per metre
Number of θ segments, n_θ	41
Number of speed modes, n_q	9
Number of steering modes per speed, n_q	5
Kernel discretisation timestep, T_p	0.2 s
Number of edge shrink pixels	8

Table 6.1: Standard discretisation parameters used in the kernel generation process.

A discretisation timestep of 0.2 seconds allows enough time for the vehicle to move into a new state. Therefore, the look-ahead timestep used by the supervisor is set to the same value. While this is longer than the actual planning timestep of 0.1 seconds, using this value results in the kernels being too big to compute and use. This approximation results in the supervisor being more conservative, which is useful to absorb model uncertainties in the model and discretisation.

Before being used for kernel generation, the race track boundary is shrunk by several pixels to take the size of the vehicle into account. A buffer of 20 cm (slightly bigger than the centre to the rear measurement of 17 cm) shrinks the track by 8 pixels ($0.2 \times 40 = 8$) from each boundary. The shrinking is done using a flood-fill algorithm.

6.4.2. Generation Process

After the dynamics table has been calculated, the kernel is recursively generated according to Equation 6.2. The kernel is an array that has the shape $[n_x, n_y, n_\theta, n_q]$ with $n_x = n_{dx} \times W_{\text{track}}$, and $n_y = n_{dy} \times L_{\text{track}}$, where $W_{\text{track}}, L_{\text{track}}$ are the width and length of the track in metres. The result of the kernel generation is a 4D array with a Boolean value for each state, indicating if the state is safe or unsafe.

There are many graphics presented to help the reader visualise the viability kernel. For the visualisation, a modified version of the AUT map is used called WAUT. The AUT map is scaled to be smaller and the track is widened so that the vehicle can turn the corners.

Generation Visualisation

The track image with the boundaries is used as the input to the kernel generation process as the first iteration of constraints, K^0. All orientation angles and modes that are in contact with the boundary are marked unsafe. The kernel algorithm then runs recursively, updating the kernel by looping through the states and for each state checking to see if there is a valid action. If a safe action exists, then the state is marked safe, and if no safe action is found, then it is marked unsafe. The algorithm terminates when the kernel has not changed while looping over all the states.

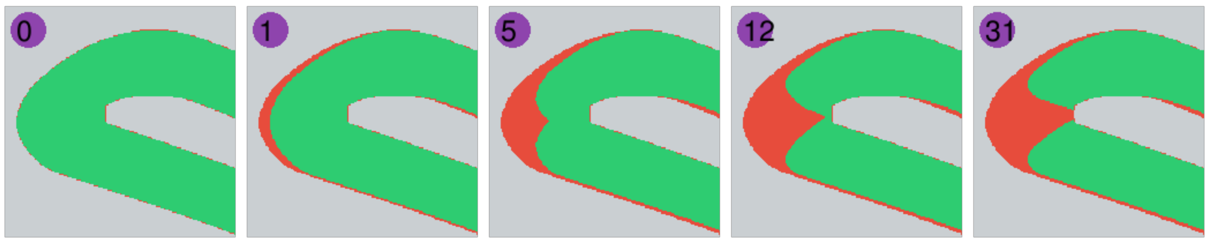


Figure 6.5: AUT map section used as input to the kernel algorithm, and the kernel after 1, 5 and 12, and 31 steps. Grey is the track boundary, green is the viable region, and red is the unsafe region.

Figure 6.5 shows the process of a kernel being generated at different steps for a corner of a racetrack for a specific orientation and mode. The image shows how, as the kernel generates, more and more states are declared to be unsafe until all the remaining states are safe. In the kernel images, the grey represents the shrunk racetrack boundaries which are the input to the kernel generator. The green represents the viable area, and the red represents the unsafe region.

The kernel is visualised for a specific angle meaning that the shape changes according to the vehicle orientation. While the final kernel (right-most image labelled 31), appears to have no safe path around the corner, this is not the case. The path around the corner exists for a different orientation angle, i.e. as the vehicle starts to turn the corner, the path for that orientation angle will appear. According to the Viability Kernel definition, there must always exist a safe path around the entire track.

Slow Kernel Visualisation

We present two sets of images of the kernels to communicate what the 4D vectors look like. Figure 6.6 shows an image of the kernel that was built using the standard parameters listed in Table 6.1. The kernel is sliced to show four different angles and indicate how the kernel shape depends on the vehicle orientation. One can see how the safe and unsafe regions are related to the orientation angle of the vehicle and rotate as the vehicle rotates.

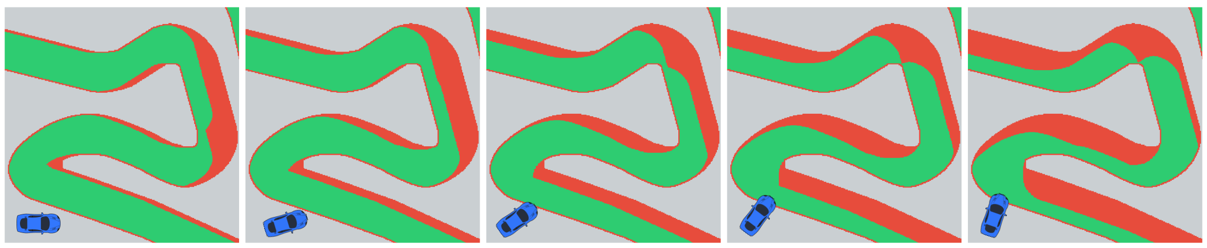


Figure 6.6: Illustration of viability kernel viewed at different angles (105, 11, 132 degrees) with the steering angle being 0 rad (straight).

Figure 6.7 shows the kernel for a fixed orientation but different modes. In the left image, the mode represents steering the maximum amount to the left and in the right image, steering the maximum amount to the right. One can see how the safe and unsafe region shifts slightly up as the mode changes.

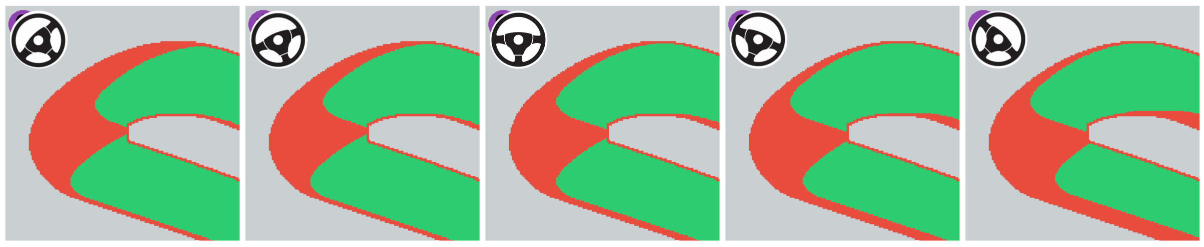


Figure 6.7: Illustration of viability kernel viewed at different modes with steering angles, -0.4, 0.2, 0, 0.2, 0.4 (from left to right) radians. The vehicle is facing to the left.

Fast Kernel Visualisation

Now the effect of speed is added and the kernel is shown with different colours representing different speeds permissible in different regions.

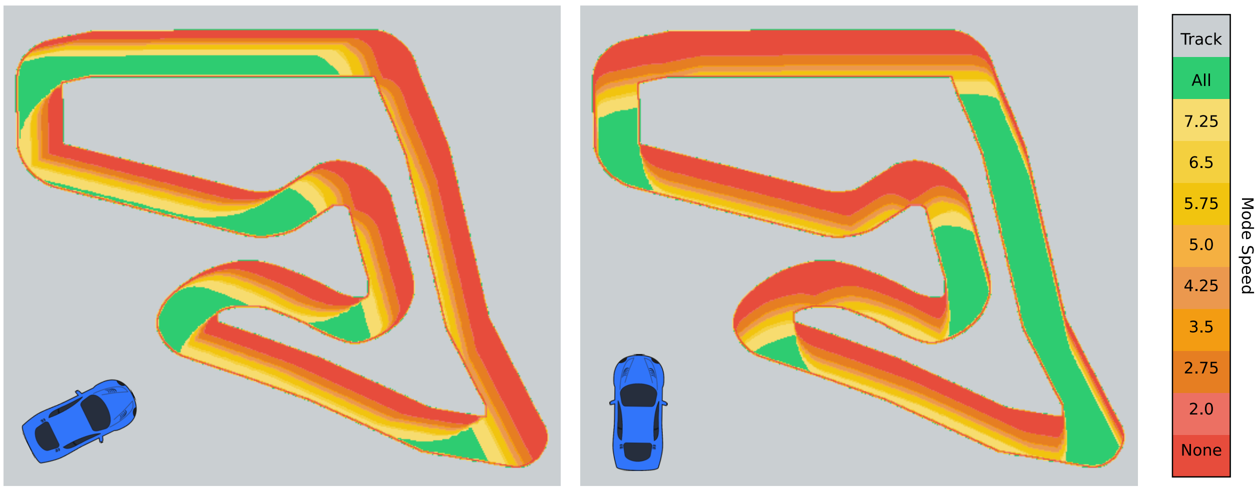


Figure 6.8: The kernel is shown with different speeds representing the speeds that are allowed in different regions.

In Figure 6.8, it is clear that the slower the vehicle is moving, the more of the track is safe. The red areas are unsafe for vehicles of any speed and as the colour fades to yellow, the area is safe for higher speeds. The green areas are safe for all speeds. The image shows that the lower the vehicle speed, the larger the size of the safe region. There are few regions where the vehicle is safe if it is moving at high speed.

6.4.3. Algorithm Implementation

A brief description of the algorithmic implementation is presented. The algorithms are programmed in Python 3.8 and extensively use the NumPy and Numba libraries. The Numba library enables code to be compiled to machine language and cached, which greatly speeds up the execution time, especially of loops and mathematical operations.

Kernel Storage

In Liniger et al.'s implementation [2], the kernels were around 4.5 GB each and included all of the states on a map. Racing track maps contain mainly the boundaries and only 15-30% of the map contains positions on the track. The kernel is a 4D vector, meaning that an increase in each dimension results in an exponential growth in kernel size. For example, using the parameters in Table 6.1, the MCO map of 54 m by 57 m, discretised with 40 points per metre, with 25 modes and 41 angle segments, results in 5.05 billion states. However, considering that only 15% of these are on the track, this can be reduced to 760 million states.

This reduction is done by using two tables, a *position table* and a *kernel list*. The position table is a grid of all the map positions that is stored with the integer data type. For each position, if the location is not on the track, the value -1 is stored. If the position is on the track, then the integer to locate that state in the kernel list is stored. The kernel list is a list of all the states on the track. For each location state, the number of angle segments and the number of modes is stored. Therefore, the kernel list has the shape $(n_{\text{track}}, n_\theta, n_q)$ and Boolean data values. The n_{track} is the number of states with a position on the driveable area of the track. The process for checking if a state is safe requires checking its position in the position table, finding the kernel state number and then identifying the angle and mode. Using this method allows for a reduction in kernel size to around 100 MB for single-speed kernels and 500 MB for high-speed kernels (see §6.5.1).

Dynamics Table Calculation

A dynamics table is built that represents the difference inclusion in Equation 6.5. The change in X and Y position ($\dot{X}$ and $\dot{Y}$) is independent of the current position and dependent only on the orientation, θ, and mode, q. Therefore, we build a table that references each state by the discrete orientation and mode, and then for each state, evaluates the next state for each possible mode action. The next state is discretised and the 4 state dimensions $(\dot{X}, \dot{Y}, \theta_{k+1}, q_{k+1})$ are stored in the table. For each planning timestep, T_{p}, there are n_s points that are calculated at equal time intervals. These points are later used to ensure that there is a clear path between the state and the next state. This results in a dynamics table with the shape $(n_\theta, n_q, n_q, n_s, 4)$.

Kernel Generation

The kernel is generated by looping through all the states in the kernel list and for each state using the dynamics table to calculate the next state for each action. If a next state is found that is within the safe set, then the state remains safe. If no safe next state is found, then that state is marked unsafe. This process of cycling through all the states and checking them is performed iteratively until no more changes are made to the kernel.

6.5. Kernel Safety Evaluation

The safety of our supervisory system is evaluated by using a random planner in conjunction with the SSS. The random planner selects random steering actions within the vehicle limits, which are checked by the SSS and modified to be safe. The safety system then outputs safe actions to the vehicle.

The kernels generated are analysed and validated using a random planner at low and high speeds. The kernel generation process shows how the kernels for the different maps are generated, noting important facts about the generation. The random planner validation demonstrates that for a worst-case scenario planner that selects completely random actions, the SSS can prevent the vehicle from crashing. Finally, a study is done where noise is added to the measurements, and the optimal planner and random planner results are shown as an ablation study.

6.5.1. Kernel Generation

Kernels are generated for the MCO and WAUT maps for vehicles for low and high-speed vehicles. The low-speed, *slow*, kernels are for vehicles travelling at a constant speed of 2 m/s. The high-speed, *fast*, kernels are for vehicles within the speed range of 2-8 m/s, with modes distributed as shown in Figure 6.7.

The number of positions refers to the map's total number of discretised x, y locations. The number of track states is the number of states (position, orientation, mode) that lie on the track and are considered for calculation. The iterations required are the number of loops of the viability kernel algorithm (Equation 6.2) required for the kernel to converge. The size of the kernel is the size of the NumPy array that is stored as a .npy file. The percentage of the track that is safe is the percentage of states that are safe as a fraction of the total number of track states.

Map	No. of Positions	No. Track States	Iterations Required	Kernel Size (MB)	% Track Safe
WAUT - slow	1,160,000	70,223,160	31	70.22	72.14
ESP - slow	3,465,600	93,195,460	32	93.2	52.75
MCO - slow	4,968,000	70,087,040	35	70.09	51.19
WAUT - fast	1,160,000	632,008,440	29	632.01	58.18
ESP - fast	3,465,600	838,759,140	29	838.76	39.03
MCO - fast	4,968,000	630,783,360	33	630.78	37.64

Table 6.2: Number of discrete positions, states on the track, iterations required for generation, the kernel size and the % of the track that is safe for the WAUT, MCO, and ESP maps.

Table 6.2 shows the metrics for the generation of the kernels for the WAUT, ESP and MCO maps. The WAUT map has a much higher percentage of the square map size that is part of the race track. Despite having less than a quarter of the number of positions than the MCO map (1e6 vs 5e6), it has almost the same number of states on the track of around 70e6. The high percentage of the safe kernel is because wider tracks have more area relative to the boundaries and, therefore, more safe space.

The slow kernels take slightly more iterations to converge. For the ESP map, the slow kernel requires 35 iterations, while the fast kernel requires 29. While the reason for this is not certain, it is suggested that the fast kernels may require fewer iterations due to more states being marked unsafe earlier in the process, due to the further distance moved by a vehicle travelling at a higher speed.

Using the implementation that only stores states for positions on the track, the kernel size is kept to a reasonable size. The slow kernels are all smaller than 100 MB, with the ESP kernel being the smallest at 70.09 MB. Of the fast kernels, the WAUT kernel is the largest at 838.76 MB, with the other two being around 630 MB. These efficient kernels are small enough to be loaded into the computer's RAM and used for real-time look-up.

6.5.2. Planner Validation

The supervisor should display two kinds of behaviour, letting safe actions be executed while ensuring that unsafe actions are identified and modified. The safety of the system is evaluated using a worst-case scenario *random* planner and observing if the vehicle crashes. The safety system's ability to allow safe actions is evaluated by using a pure pursuit planner following the racing line. The effect of the SSS is measured by comparing the lap times with and without the supervisor.

Random Planner Safety

The kernels are now validated for safety by running 50 test laps using a random planner. For the constant speed tests, the random planner samples a steering angle from the steering range using a uniform distribution. For the high-speed tests, the random planner samples a steering angle from the steering range and a speed from the speed range using a uniform distribution.

	Slow		*Fast*	
Metric	**No. of Interventions per Lap**	**Intervention Rate (%)**	**No. of Interventions per Lap**	**Intervention Rate (%)**
WAUT	78.5 ± 9.4	13.8 ± 1.4	227.5 ± 7.9	79.0 ± 2.7
MCO	186.2 ± 13.9	19.8 ± 1.4	414.7 ± 10.7	80.9 ± 1.6
ESP	221.2 ± 14.4	17.7 ± 1.1	518.1 ± 11.2	81.2 ± 1.7

Table 6.3: The number of interventions per lap and the intervention rate for a worst-case-scenario, random planner being used with the safety system on the WAUT, MCO and ESP maps. Slow kernels use a fixed speed of 2 m/s and fast kernels have a speed range from 2 m/s to 8 m/s.

Table 6.3 records the average times and interventions per lap. The results show that the vehicle's number of interventions varies significantly between laps which is what is expected for a random planner. The supervisor intervenes a lot more for the fast tests than the slow tests. For the slow tests, the intervention rate is between 10-20%, but for the fast tests, it is between 75-85%. This significant increase is due to many of the actions selected by the fast random planner breaching the friction limit and are thus marked as unsafe. For all of the test laps, the vehicle does not crash once, demonstrating that the SSS can guarantee the vehicle's safety.

Figure 6.9 shows an example trajectory from the random planner. The green lines are where the original action (selected by the random action generator) is implemented. The red lines are where the original action was modified to prevent the vehicle from crashing.

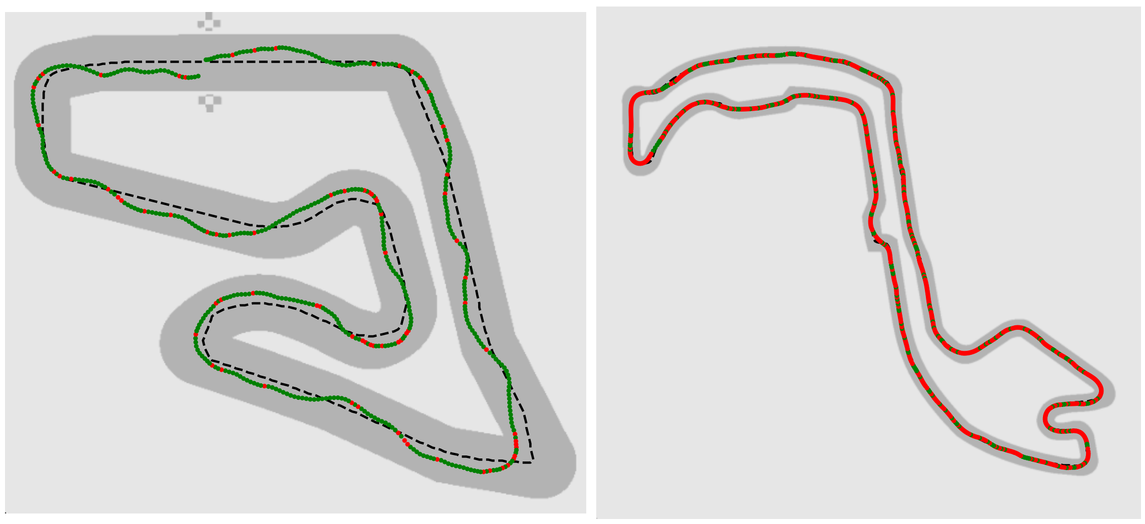


Figure 6.9: Random planner safely driving around WAUT - slow (left) and MCO - fast (right) racetrack. Green is where the random action was safe, and red is where the supervisor intervened.

The trajectories shown in Figure 6.9 show that the supervisor regularly intervenes (shown by the red marks) to prevent the agent from crashing. When driving slowly, the supervisor intervenes less than might be expected due to implementing a pure pursuit action when intervening and thus moving the vehicle towards a safe region. When driving at higher speeds, the supervisor intervenes a lot more, as seen in the image on the right, where most of the path is red, indicating intervention. This is to be expected because actions are higher speeds can more easily exceed the friction limits and also lead to the boundaries faster.

Optimal Behaviour Impact

The kernel's ability to allow safe actions to be executed is now evaluated using a pure pursuit planner. The planner is set to follow the racing line and tested with and without the SSS. Note that the SSS internally uses the pure pursuit planner following the centre line (see §6.2), which is in contrast to the pure pursuit planner following the racing line. Additionally, the SSS uses a friction model to select a conservative speed while the pure pursuit planner following the racing line uses the speed from the optimal trajectory. Table 6.4 shows the metrics of the number of interventions, lap time, average velocity, and distance travelled for the planner with different maximum speeds. The speed selection of the pure pursuit planner is limited at the maximum speeds of 4, 6 and 8 m/s to show how the effect of the safety system varies for different speeds.

	Pure Pursuit		*Pure Pursuit with SSS*		
Max Speed	**Lap Time (s)**	**Avg. Velocity (m/s)**	**No. Interventions**	**Lap Time (s)**	**Avg. Velocity (m/s)**
4	46.4	3.8	183.0	54.4	3.3
6	37.7	4.6	337.0	52.1	3.5
8	35.7	4.9	355.0	50.0	3.6

Table 6.4: The lap times and average velocities for vehicles using the pure pursuit planner following the racing line with maximum speeds of 4, 6 and 8 m/s on the MCO map.

In Table 6.4, the clearest result is that the lap times for the pure pursuit planner using the safety system are longer than those without the safety system. For the planner with the full maximum speed of 8 m/s, the lap time without the supervisor is 35.7 seconds, and with the supervisor, it is 50.0 seconds, 14.3 seconds slower. This is confirmed by the average velocity of 3.6 m/s, 1.3 m/s slower than the original pure pursuit planner. When using speed limits, the performance difference becomes smaller. The vehicle with a 4 m/s speed limit is only 8.0 seconds slower with the safety system. This tendency is explained by the decreasing number of interventions by the supervisor. The supervisor intervened 355 times for the vehicle travelling at 8 m/s while only intervening 183 times for the vehicle with a speed limit of 4.

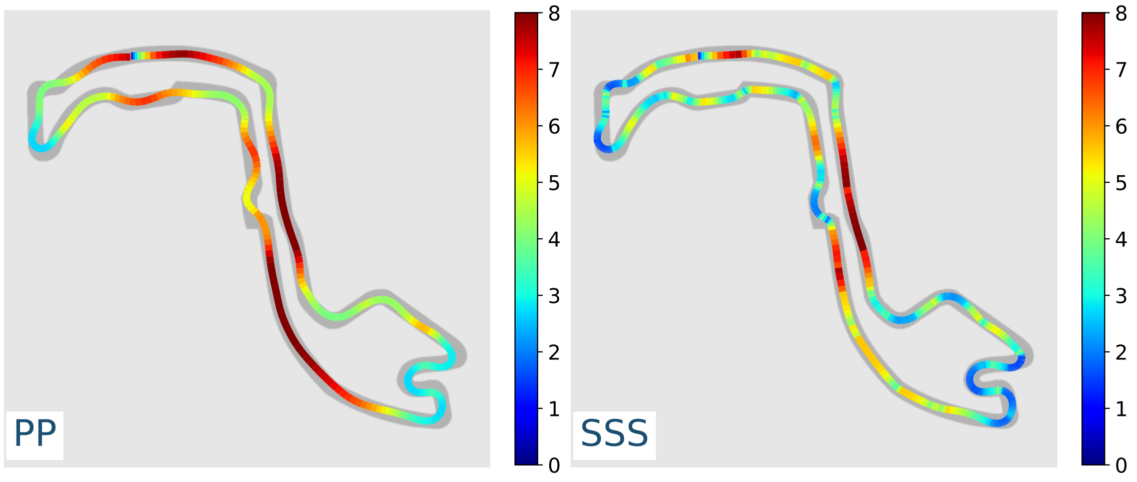


Figure 6.10: Trajectories with the velocity profiles of the pure pursuit planner driving on the MCO map without (left) and with (right) the safety system.

Figure 6.10 shows the trajectories of the pure pursuit planner with and without the SSS. The trajectory on the right is evidently slower, as shown by the lighter (more blue) colours. The SSS specifically slows the vehicle down around the corners, shown by the darker blue on the right-hand-side trajectory.

This test shows that using the supervisor comes at the cost of performance, as the lap times using the safety system are slower. While this is a limitation, it still enables fast driving throughout the track. It is suggested that this problem could be improved by using a variable lookahead timestep depending on the vehicle speed.

6.5.3. Ablation Study

In real life, noise is constantly present, arising from many factors, including errors in the particle filter used for localisation, the friction coefficients being slightly different, any aerodynamic forces, delay in the system or modelling errors. For the WAUT kernel, noise is now added to the measurement to evaluate its sensitivity to noise in the observed vehicle position, which is the largest contributor of uncertainty in the system. For the noise in the position, after each update, noise sampled from a normal distribution is added to the x, y variables that are given to the safety system. Random and pure pursuit planners are used for the experiments, and the lap success rate is recorded out of 20 test laps.

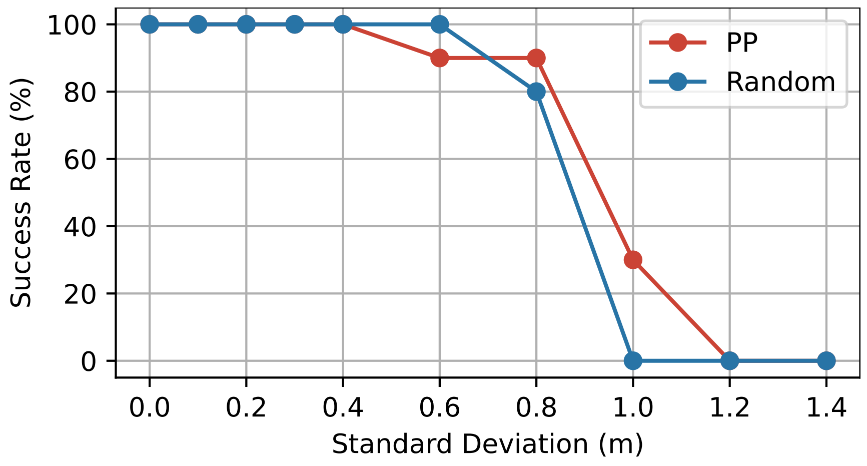


Figure 6.11: The success rate for the pure pursuit and random planners on the WAUT map for position noise with increasing standard deviations of noise added to the location used by the safety sysem.

The graph in Figure 6.11 shows that the safety system can prevent the vehicle from crashing with noise sampled from a normal distribution with a standard deviation of up to 0.4 m. As is expected, the random planner shows slightly worse performance than the pure pursuit planner. This result demonstrates that the safety system is robust to noise in the localisation with a standard deviation of up to 0.4 m.

6.5.4. Filtered Kernels

In reality, it is often impossible to accurately estimate the vehicle's steering angle. This is due to the servo not providing feedback, noisy IMU data and the particle filter data being too slow and inaccurate to estimate the steering angle accurately. Therefore, the kernels are filtered to include only states that are safe for any initial steering angle. The results of filtering the slow kernel are presented below. This method is physically validated by its use in the online training of a DRL agent in §7.4.

Filtered Kernel Generation

At low speeds, the filtered kernels appear similar to the full kernels. Table 6.5 shows the metrics of the number of states and the percentage of safe states in the kernel before and after the filtering. The results show that the size of the kernel is decreased by a factor of five, since the mode dimension of five is reduced to a single unit. The percentage of the states that are safe drops from 72.14% to 57.08% for the WAUT map. This is to be expected since for a state to be safe, it must now be safe for all steering angles.

	Original		*Filtered*	
Map	**No. Track States**	**% Track Safe**	**No. Track States**	**% Track Safe**
WAUT - slow	70,223,160	72.14	14,044,632	57.08
ESP - slow	93,195,460	52.75	18,639,092	30.31

Table 6.5: The number of states and percentage filled for the WAUT, and ESP kernels at a constant speed of 2 m/s.

Figure 6.12 shows the WAUT with the original and filtered perspectives. The difference graph shows that there is relatively little difference between the two. It is clear that the filtered kernel is more restrictive. The right hand image shows the additional states that were permitted in the original kernel but declared unsafe in the filtered kernel. However, despite being more restrictive, the filtered kernel does provide a path around the entire track.

Filtered Kernel Vehicle Performance

The random and pure pursuit planners are evaluated on the filtered kernels and compared to the performance with the original kernels.

The results in Table 6.6 show the effect of the filtering on performance of the random planner. The intervention rate for the MCO map increases from 23.0 to 54.1%. This increase is seen across all the maps, indicating that fewer states are safe. It is concluded that the filtered kernels can be used to guarantee a vehicle's safety at low speed, even if the steering angle is unattainable.

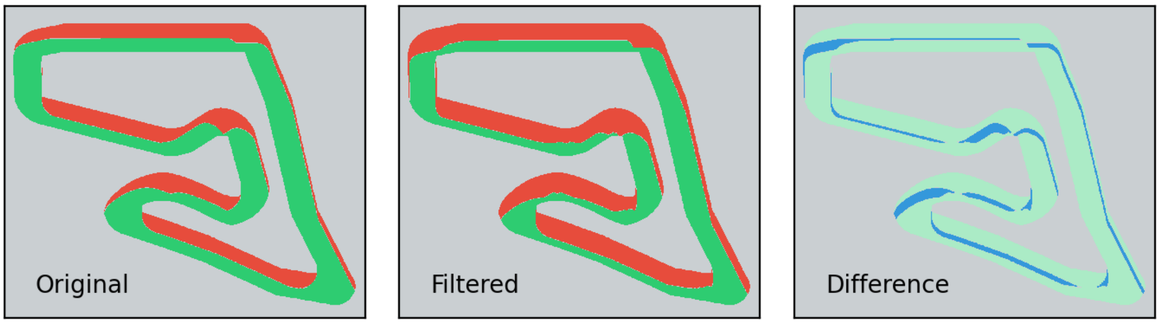


Figure 6.12: The original (left) and filtered (right) kernels. The difference between the kernels (right) is shown in blue.

	Original		*Filtered*	
Metric	**No. of Interventions**	**Intervention Rate (%)**	**No. of Interventions**	**Intervention Rate (%)**
WAUT - slow	72.7	13.4	189.2	41.7
MCO - slow	225.9	23.0	392.2	54.1
ESP - slow	270.3	20.7	486.6	50.8

Table 6.6: The number of interventions and intervention rate using the original and filtered kernels on the WAUT, MCO and ESO maps.

6.5.5. Discussion

Other supervisory systems have used humans to take over if the operator deemed it unsafe [49, 117], using Reachability Theory [87], or reversing if near a wall [60]. Using a human to intervene is inherently a poor solution to algorithms aiming to promote autonomy since they inherently require intervention. Calculating the time-to-collision (TTC) and reversing if below a threshold enables autonomous safety yet has only been implemented at low speeds. This method will not scale well to higher speeds since it takes a significant amount of time for the vehicle to stop and reverse before it keeps driving. Using Reachability Theory involves calculating the future states the vehicle will enter due to its current pose and ensuring that these states are always inside the track region. While reachability concerns important control aspects, such as recursive feasibility and the vehicle's speed, these methods do not scale well to higher speeds due to the heavy real-time requirement of calculating many states into the future. Additionally, reachability methods always have a finite lookahead, meaning they are not guaranteed to be recursively feasible.

In contrast to previous methods that were limited by low speeds, or not requiring intervention, this chapter presented a supervisory safety system that can ensure the safety of a planner at high speeds, while having minimal effect of the vehicle performance. To the author's best knowledge, this work is the first safety system capable of ensuring the safety of high-speed agents.

The SSS uses a viability kernel of all the safe states that exist on the map that can be computed before the race begins. This has the advantage over methods using Reachability Theory, that no further computation is required during the race. The kernel is accessed by conducting a single-step lookahead and then using the kernel as a look-up table. Additionally, the system does not require the system to stop if it is in a dangerous place, but rather includes a pure pursuit planner that steers the vehicle towards the centre line.

The SSS presented in this chapter enables the use and evaluation of ML planners on physical robots with the assurance of safety. It is expected that technologies like this will accelerate the evaluation of planners onboard physical vehicles. The sim-to-real gap, which previously was a hindrance to evaluating ML planners on physical vehicles, is no longer a problem. Another exciting avenue that is explored in Chapter 7 is to build integrated online learning formulations that use the SSS to train agents onboard physical vehicles.

Limitations

A limitation of this work is that only the single-track bicycle model was used to build the kernel. In the future, this should be addressed by comparing different models, specifically the full kinematic model with the Pajecka model to represent the forces.

A second limitation is the modelling of the error due to discretisation. Liniger et al. [2] use the discriminating kernel to model the error due to discretisation. This approach would lead to the system being able to identify the boundary of safety better.

A third limitation of this work is the requirement for localisation to ensure safety. Future work could approach this by using the full 1080-beam LiDAR scan to analytically calculate what actions the planner can select.

6.6. Summary

This chapter presented an SSS that can guarantee the safety of an unsafe planner. The system uses a three-step process of (1) simulating the planner's action, (2) checking to see if the next state is in the viability kernel of safe states, and (3) if the next state is unsafe, implementing a safe (pure pursuit) action. The viability kernel for autonomous racing was presented, and the formulation for the safety system was explained. The implementation for F1/10$^{\text{th}}$ racing was described in detail with notes about the mode selection, discretisation, generation process and algorithmic implementation. The evaluation analysed the kernel generation process, validated the system's safety, measured its sensitivity to noise and tested a filtered kernel that doesn't require the steering action. The results show that the system can ensure the safety of a planner while allowing safe actions to be implemented. At high speeds, the planner has a negative impact on the performance due to the conservative safety guarantees used. The discussion concluded that this kernel is superior to previous approaches due to its ability to ensure safety at high speed while not requiring human intervention.

Chapter 7

Online Learning using a Supervisor

The SSS that was developed in Chapter 6 is now used to train DRL agents online, while never crashing. The learning formulation is modified to take the new supervisor into account by redefining episodes and penalising the agent if the supervisor intervenes. The new learning formulation allows for novel reward signals to be considered since the agent is guaranteed to complete laps while never crashing. The evaluation considers constant and variable speed racing, analyses the supervisor's role in learning, investigates different reward signals, and provides a quantitative and qualitative comparison with conventional learning. The evaluation shows that online training can train agents to race with a 10$\times$ improvement in sample efficiency and that the trained agents drive conservatively within the performance limits. The SSS is validated on a physical vehicle, demonstrating that the SSS can train an agent that takes random actions to drive around a race track without ever crashing. The main advantage of the SSS is that it enables DRL agents to be trained onboard, bypassing the sim-to-real gap.

7.1. Introduction

The sim-to-real problem, the difference in the performance of DNN controllers in simulation and reality, is an open research challenge requiring further research [24]. For as long as DRL agents are trained in simulation, the sim-to-real problem will always exist. Additionally, due to the lack of interpretability of DNNs, it is impossible to know how controllers will perform with small dynamics changes.

Online DRL has been presented as a solution to the safety problems in training a policy in simulation and then transferring it to a physical robot [118]. Training an agent online involves building a framework that allows the agent to explore the state-action space while ensuring that the vehicle will never crash. Previous methods have studied how their supervisors are used to train vehicles [86].

This chapter presents the online learning formulation using the supervisor described in Chapter 6, explains how the architecture is set up in §7.2.1, and how the learning is formulated in §7.2.2. A variable speed evaluation in simulation is presented in §7.3, showing how online training with the supervisor compares to conventional training. Finally, the SSS is validated on a physical vehicle and shown to train an agent from random to drive around a track in §7.4.

7.2. Online Reinforcement Learning

In response to the sim-to-real problem, it is proposed to train the DRL agent online on the vehicle. This is done using the supervisory safety system to ensure that the vehicle does not crash during training.

7.2.1. Online Training Architecture

The SSS that was developed in Chapter 6 is positioned after the agent to ensure that the vehicle does not crash. The supervisor is specifically needed during the initial stages of training when the agent is randomly initialised and thus selects random actions. Figure 7.1 shows a schematic of the architecture with the supervisor using the vehicle pose to guarantee that the action that is implemented is safe.

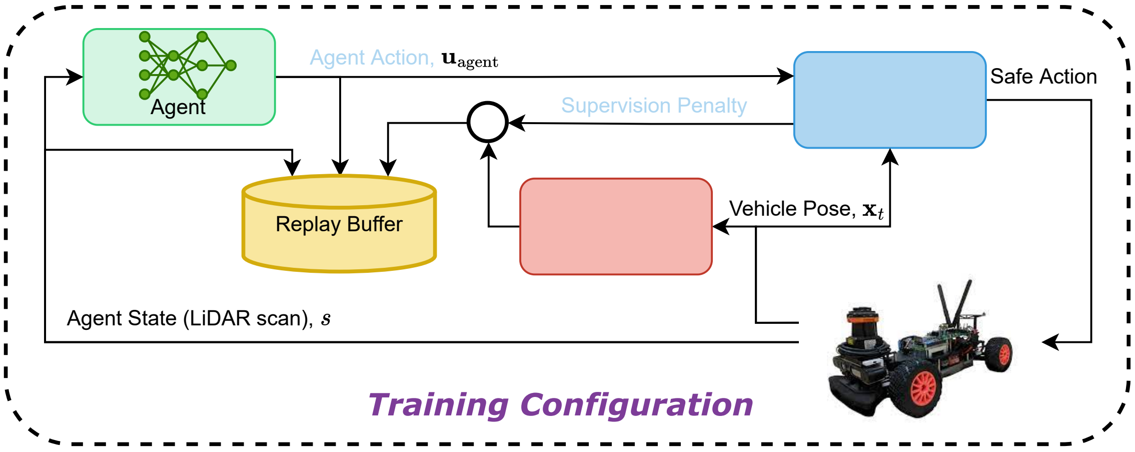

Agent
Agent Action, $\mathbf{u}_{\text{agent}}$
Supervision Penalty
Safe Action
Replay Buffer
Vehicle Pose, $\mathbf{x}_t$
Agent State (LiDAR scan), s
Training Configuration

Figure 7.1: The supervisor is placed after the agent, which ensures that only safe actions are implemented and penalises the agent for unsafe actions.

In Figure 7.1, the supervisor provides a penalty that is added to the reward signal if the supervisor intervenes. This encourages the agent to select safe actions without the supervisor intervening. The agent state consists of the LiDAR scan, and the agent's actions are speed and steering. After each step, the agent state, action and summed reward are stored in the replay buffer and used to train the agent.

7.2.2. Supervisory Learning Formulation

Reinforcement learning trains agents to perform a task by building up experience that is used to improve their policy. Agents build up experience by receiving a state, selecting an action and then being given a reward.

In Chapters 4 and 5, conventional end-to-end DRL agents that receive a LiDAR scan as a state and return an action consisting of a steering angle and velocity were considered. Conventionally, episodes run from the vehicle starting at an initial position and driving until it reaches a terminal state of crashing or completing a lap. At the start of training, the agents crash quickly, and as the training progresses, they learn to select more appropriate actions, resulting in them crashing less and completing more laps.

Using a supervisor results in the agent never crashing and always completing laps. Therefore, the learning formulation can be modified to take this into account. This section explains how the aspects of the learning are reformulated to enable online training.

Episode Configuration

In conventional RL, episodes are terminated when the agent achieves the goal, fails catastrophically or reaches a maximum number of steps. Introducing a supervisor results in the agent never failing catastrophically (crashing) and always completing the goal. Additionally, the action selected by the agent is not implemented, so the next state does not correspond to the result of the state-action pair. Therefore, the concept of an episode needs to be modified.

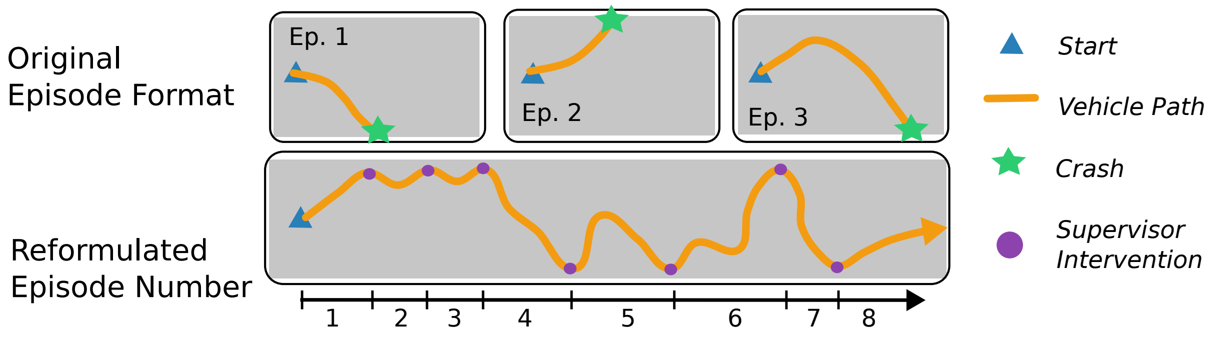


Figure 7.2: The top line shows the conventional learning formulation crashing and requiring a reset, compared to the bottom line of the reformulated learning with the supervisor intervening.

We reformulate the definition of an episode to run from an initial state until the supervisor intervenes, as shown in Figure 7.2. When intervention occurs, this is treated by the agent as a terminal state in which a terminal penalty is given. The agent does not use the transition between the two states, since it would not be correct. Instead, the next state is treated as an initial state in a new episode.

A significant advantage of this method is that an agent can experience many episodes (with terminal rewards) within a single lap of safe driving. This increases sample efficiency since the agent can collect many terminal samples in relatively few steps.

Supervisory Penalty

In previous end-to-end approaches in this dissertation, the reward signal has been in the form of sparse terminal rewards and dense smaller rewards that encourage fast racing behaviour. In changing the terminal conditions (definition of an episode), the rewards that are given must also be updated. The reward signal must train the agent to take safe actions without relying on the supervisor.

The main change is that the punishment for crashing is replaced with punishment if the supervisor intervenes. The new base reward signal is now written as

$$r = \begin{cases} r_{\text{complete}} = 1 & \text{if complete} \\ r_{\text{penalty}} = -1 & \text{if intervention} \\ r_{\text{racing}} & \text{otherwise,} \end{cases} \tag{7.1}$$

where r_{racing} is a reward that encourages fast racing behaviour.

Shaped Reward Signal Candidates

Since the agent is guaranteed to complete laps, different reward signals that were previously impractical may now be considered. The baseline reward signal has no racing reward to evaluate how well the agent can learn to select safe actions, free from any performance requirements. This baseline is labelled as the *zero* reward, since there is zero racing reward after each step. The previously presented progress, cross-track and heading error rewards from §4.3.2 are also considered.

For the variable speed racing, a new reward is introduced called the *velocity* reward. The velocity reward simply uses the scaled velocity as the reward. Since the SSS prevents the vehicle from going too fast, the vehicle should learn to travel at the highest safe speed. The reward is calculated as

$$r_{\text{velocity}} = \frac{v_{\text{agent}}}{v_{\text{max}}}. \tag{7.2}$$

Since the supervisor takes the vehicle's velocity into account when determining the safe speed, this reward automatically trains the vehicle to select paths where the vehicle can drive the fastest.

7.3. Evaluation of Online Learning

The supervisory safety system is evaluated in simulation to demonstrate the ability to train agents without crashing. The focus of the simulation evaluation is the measurement of the performance since all of the vehicle state variables are available for measurement. The evaluation consists of four sections wherein we test constant speed online learning in

§7.3.1, variable speed online learning in §7.3.2, compare reward signals for variable speed online learning in §7.3.3, and compare conventional and online learning in §7.3.4.

7.3.1. Constant Speed Evaluation

The online learning formulation is evaluated at constant speed to understand how the learning formulation performs on different maps, with different random seeds and how the reward signal affects the vehicle performance.

Since the agents complete all the laps, the average progress during training can no longer be used as a performance metric. Therefore, the metrics of reward, interventions by the supervisor and lap time are used to assess the training. The agents should aim to maximise reward, minimise interventions and achieve the lowest lap times.

Training Investigation

The training is investigated to evaluate how the agent learns using the supervisor. The training performance is evaluated by training five agents with different random seeds on the AUT, ESP, MCO and GBR maps for 6,000 steps. The zero reward is used during the investigation to highlight the effect of the supervision penalty, apart from the shaped reward.

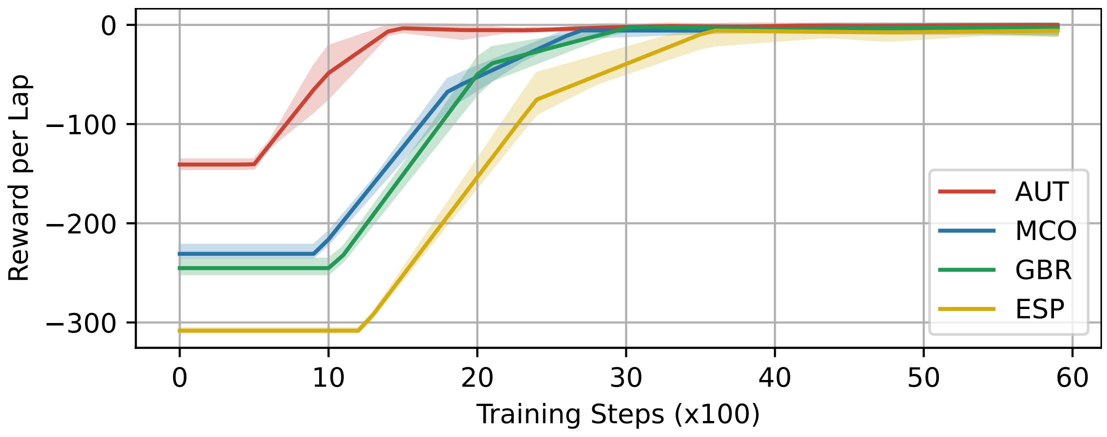


Figure 7.3: The average reward per lap earned by the agents trained using the supervisor for the AUT, MOC, GBR and ESP maps. The shaded regions are the minimum and maximum for five repetitions.

Figure 7.3 shows the reward earned by agents trained onling on the AUT, ESP, MCO and GBR maps. At the beginning of training, the agents all receive large negative rewards between −300 and −140 depending on the length of the tracks. At the end of the training, all the agents receive zero rewards per lap, indicating that the supervisor no longer has to intervene and the agent does not receive any penalties. The agents on the AUT track, the shortest track, receive a reward of around −150. The agents on the ESP track, the longest track, receive an initial reward of around −300. The agents take a varying number of steps to train, with the AUT agents converging in less than 2,000 steps and the ESP agents requiring up to 4,000.

The main observation from Figure 7.3 is that the agents train quickly in less than 4,000 steps. While the conventional agents require 30,000 steps to train, the online agents require only 6,000 steps. The second takeaway is that the training is highly repeatable. The shaded regions are small, indicating that this result is achieved if the tests are run using any of five random seeds. The supervisors role is further analysed by plotting the interventions by the supervisor on different laps of the training.

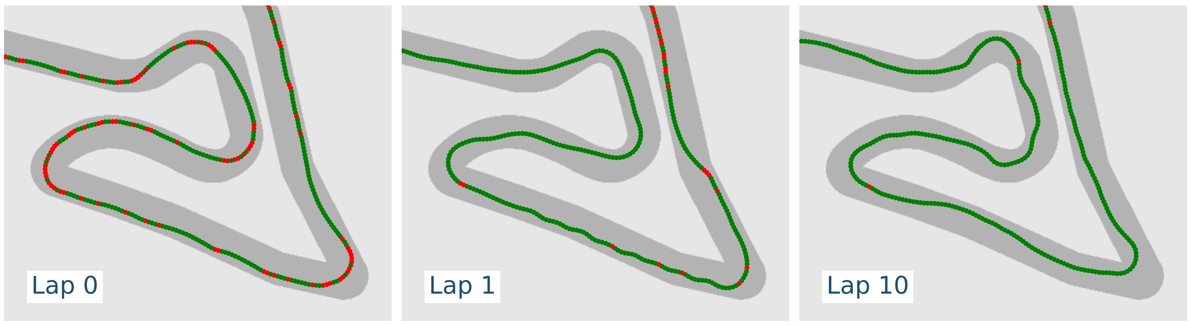


Figure 7.4: The online learning progression on the AUT map: at the beginning (left), the agent intervenes regularly, and as training progresses (moving right), the supervisor intervenes less and less. The red is where the supervisor intervenes, and the green is where the agent's action is implemented.

In Figure 7.4, the progression of the agent training can be seen. At the beginning (left), the agent intervenes regularly, as shown by the many red dots. The specific places where the agent intervenes are around the corners, especially the bottom right corner, which is very sharp. As training progresses from lap 0 to lap 1, there are fewer red dots, indicating that the supervisor intervenes less. By lap 10, near the end of the training, there are very few interventions. The occasional dots of intervention in the 3rd image (right) are always present due to the policy noise added during training.

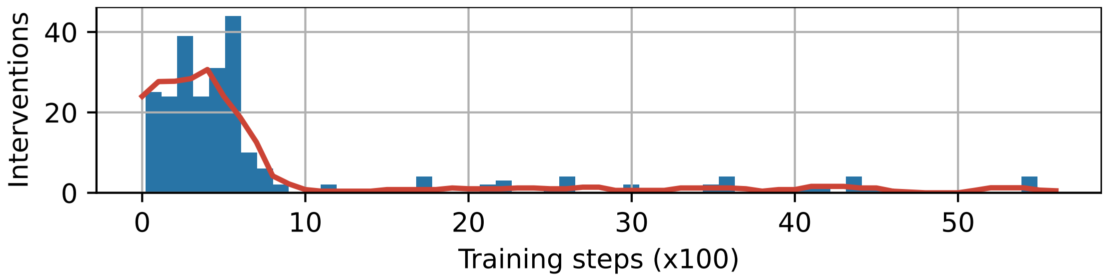


Figure 7.5: Graph of the interventions made by the supervisor per 100 training steps on the AUT map.

Figure 7.5 shows a histogram of the interventions made by the supervisor as the training progresses. In the first 1000 steps, the supervisor intervenes more than 20 times per 100 steps. Between 1000 and 3000 steps, the supervisor intervenes less and less. The times when the supervisor intervenes are usually when going around a corner. The interventions during the first 1,500 training steps are further studied by considering the steering angles where the supervisor intervenes and the deviation of the vehicle from the centre line.

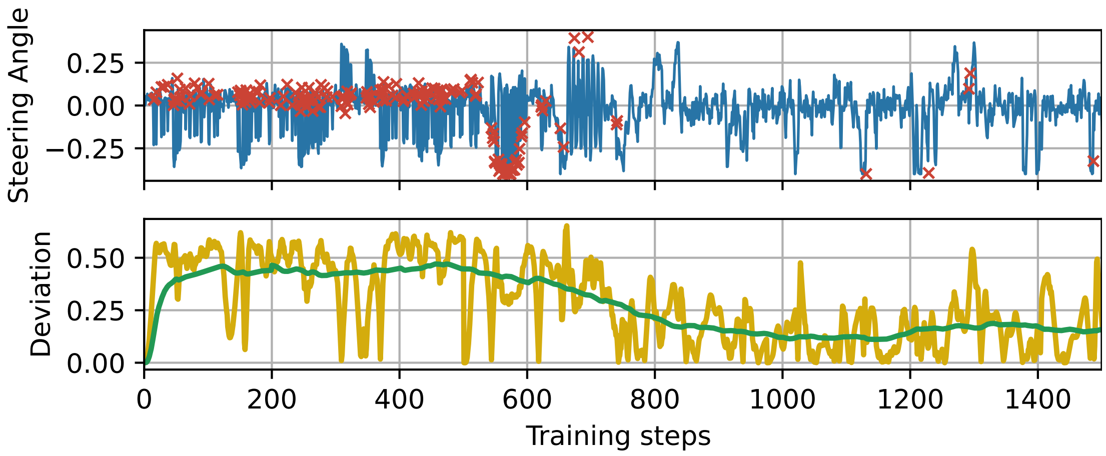


Figure 7.6: Top: A plot of the steering actions selected by the agent (blue) with crosses (red) showing the actions where the supervisor intervenes during training on the AUT map. Bottom: The deviation from the centre line (yellow) and moving average (green) for the agent during training on the AUT map.

The top graph in Figure 7.6 shows the steering actions selected by the agent plotted against those implemented by the safety system. At the beginning of training, there are a lot of red x's on the graph, indicating that the agent's actions are extreme and seldom implemented. As the training progresses, the actions become more moderate, and the supervisor intervenes less.

The bottom graph in Figure 7.6 shows the deviation during the training of the agent on the AUT map. In the beginning, the vehicle is far from the centre of the track, with an average deviation of up to 0.55 m. As the training progresses, the average moves down to around 0.15 m. This demonstrates that the agent learns to stay in the middle of the track, even without any explicit reward.

Reward Signal Performance Study

Different reward signals are now considered for their effectiveness in autonomous racing. Figure 7.7 shows a graph of the lap times of agents trained with progress, cross-track and heading error, and PPPS rewards compared with no racing reward (zero).

Figure 7.7 shows all of the agents start with lap times of around 122 seconds, and during the training, the lap times improve to around 116 seconds. There is little difference between the reward signals, except for the PPPS and cross-track and heading error reward signals performing around 1 second faster than the zero reward. Therefore, it is concluded that this new learning formulation does not require a shaped reward signal to train the agent to race at a constant speed.

The numerical results of distance travelled, deviation from the centre line, total curvature and mean steering angle of the zero reward, progress and PPPS reward signals

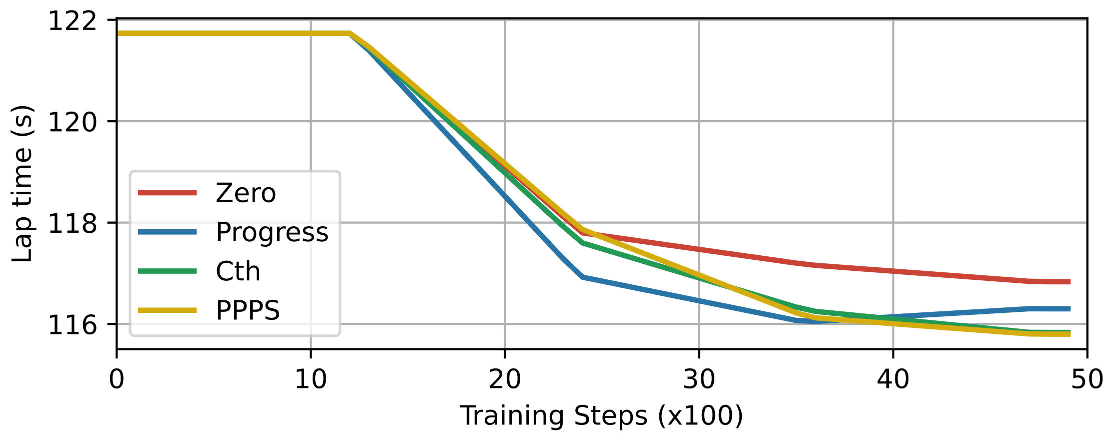


Figure 7.7: The lap times during training of constant speed agents trained with the zero, the progress, the cross-track and heading error (Cth), and PPPS rewards. The lines are the average of 3 repetitions.

are recorded.

Metric	Zero	Progress	PPPS
Distance (m)	232.50	233.20	231.75
Deviation (m)	311.75	245.52	166.52
Curvature (m^{-1})	154.49	187.29	141.62
Mean Steering (rad)	0.049	0.062	0.040

Table 7.1: The performance metrics of distance travelled, deviation from the centre line, total curvature and mean steering for the zero, progress and PPPS rewards on the ESP map.

Table 7.1 shows the performance results for the agents trained online. The agents all travel similar distances around the map of around 232 m. Using zero reward results in the agent having a large total deviation from the centre line of 311.75 m. The progress reward has a lower deviation of 245.52 m and the PPPS reward has the lowest deviation of 166.52 m. Considering that the zero reward has such a large deviation, it is surprising that the zero reward has a low curvature of 154 m compared to the progress reward of 187 m. The PPPS reward has the lowest curvature of 141 m, which is expected since it follows the racing line. Using zero racing reward leads to a mean steering angle of 0.049 radians, which is larger than the PPPS reward and smaller than the progress reward.

Figure 7.8 shows example paths of agents trained with different reward signals on the ESP track. The path from zero reward is squiggly, sometimes tracking the centre line and sometimes not. The progress reward demonstrates similar behaviour to the conventional training, with the agent cutting the corners to take the shortest path. The PPPS reward tracks the racing line, cutting the corners and hitting the corner apexes.

This result shows that the zero, progress, cross-track and heading and PPPS rewards

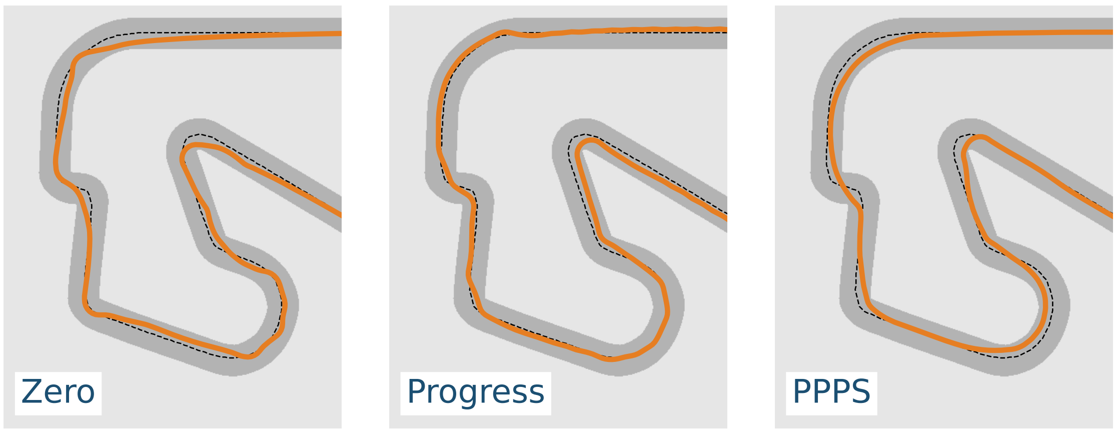


Figure 7.8: Segments of the path selected by the zero, progress and PPPS reward signals on the ESP map.

are all effective for training agents to drive at a constant speed. This result demonstrates that no shaped reward is required to train an agent to drive around the track. Using the racing line in the reward signal leads to improved results for online learning, with the vehicle selecting smoother paths.

7.3.2. Variable Speed Evaluation

The online learning formulation is now evaluated for the task of variable-speed autonomous racing. The evaluation uses DRL agents that select speed and steering actions. The evaluation starts by analysing the agent's ability not to require the supervisor and then considers the effect of maximum speed.

Training to Safety

The first investigation into online training is to evaluate the agent's ability to no longer require supervision. Agents are trained on the AUT, ESP and MCO maps for 5,000 steps, each using the velocity reward signal with a maximum speed of 5 m/s. During all these training runs, the vehicle never crashes once. Figure 7.9 shows the average interventions by the supervisor per 100 training steps for agents trained on the AUT, ESP and MCO maps.

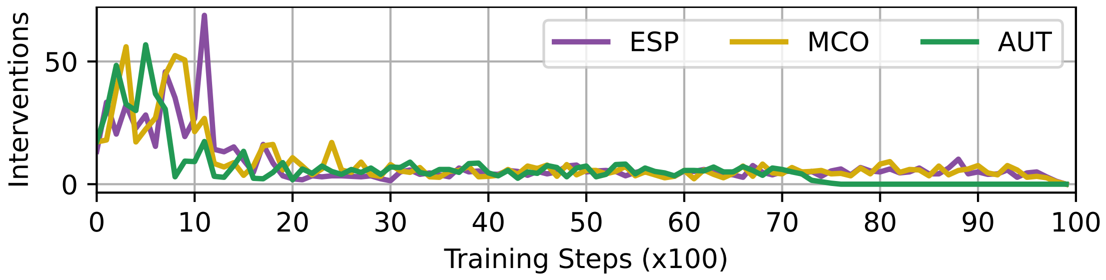


Figure 7.9: Average interventions per 100 steps during online training on the ESP, MCO and AUT maps. Average from repeating the experiment five times.

Figure 7.9 shows the training for a full 5,000 steps for the ESP, MCO and AUT maps. The graph shows that in the beginning, there are many interventions, but these reduce to fewer than 5, around 3,000 steps. For the remainder of the training, there are still several regular interventions. This result shows that the agents learn to not require supervision within 3,000 steps. Since most of the changes in the graphs appear in the first 3,000 steps, a zoomed-in graph of the first 3,000 steps with each of the individual repetitions is shown in Figure 7.10.

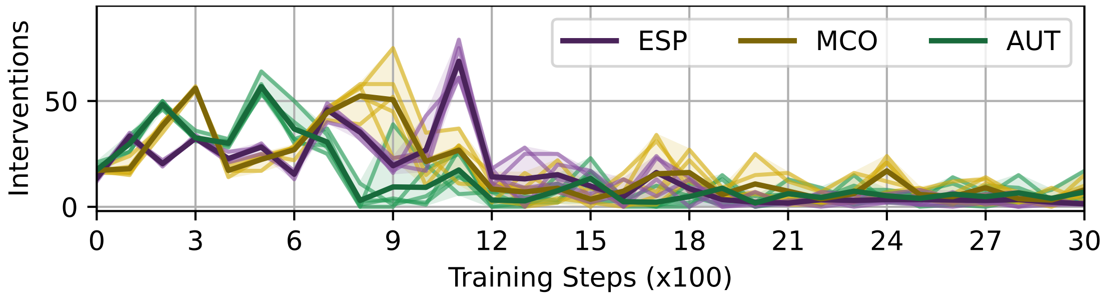


Figure 7.10: The interventions from the safety system per 100 steps for the velocity reward on the AUT, ESP and MCO maps.

Figure 7.10 shows that the agents on each map follow a similar pattern with spikes in similar places. This indicates that the interventions in the initial stages of training are dominated by the shape of the map since agents with different random seeds require interventions in similar places. As the training progresses past 1,200 steps, the interventions become fewer and more random, and the individual runs diverge. It is suggested that this is due to the vehicle being able to drive around the track and the interventions being due to the random policy noise added during training.

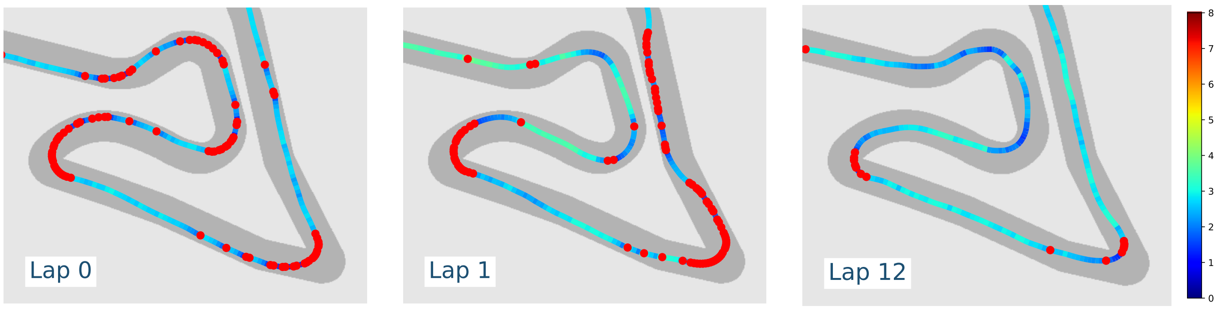


Figure 7.11: The online learning progression of the high-speed agent on the AUT map: at the beginning (left), the agent intervenes regularly, and as training progresses (moving right), the supervisor intervenes less and less. The red dots show where the superivsor intervenes.

The trajectories selected by the vehicles during training are now considered. The 0th, 1st and 12th trajectories during training on the AUT are shown in Figure 7.11 with red dots to indicate where the supervisor intervened. The trajectories show how at the beginning of training, the agent drives at slow speeds and the supervisor intervenes a lot. As the training progresses, the supervisor intervenes less, and the agent learns to select a feasible speed profile, speeding up in the straights and slowing down for the corners. The trajectories explain the shape in the graphs in Figure 7.9 since it is clear that from the beginning of training, the supervisor mainly intervenes around the corners.

The training investigation concludes that the agents can be trained to race around a map in 10,000 steps. At the beginning of training, there are many interventions, especially around the corners, but as training progresses past the 1,500 steps, the intervention rate decreases rapidly.

Maximum Vehicle Speed Training

The speeds considered in the evaluation thus far have been up to 5 m/s. Now agents with maximum speeds of 5, 6, 7 and 8 m/s are considered. The analysis starts with the lap times and rewards earned during training and then presents a numerical analysis of the performance and success rates. Figure 7.12 shows the training graph of the agents with maximum speeds ranging from 4-8 m/s trained using the velocity reward signal.

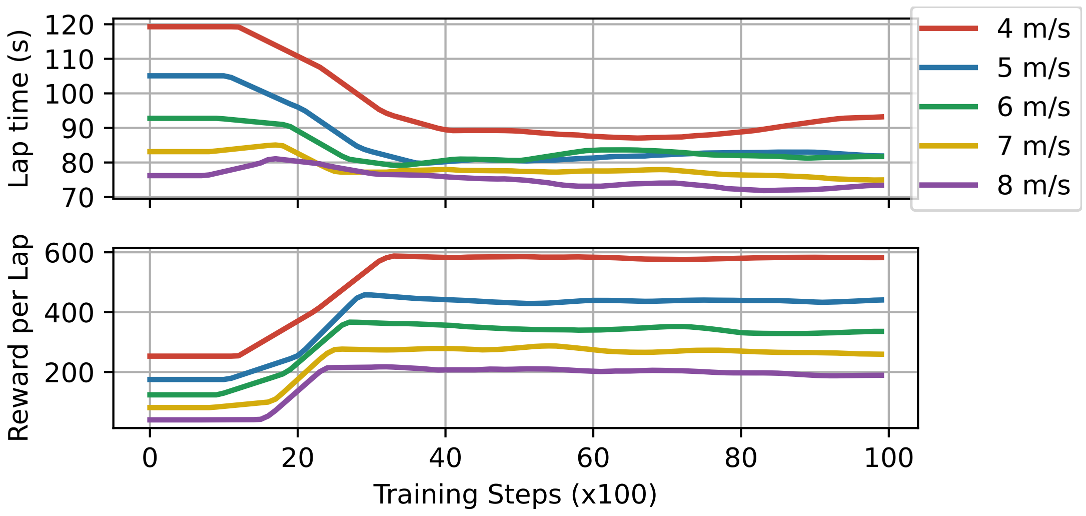


Figure 7.12: Graph showing the rewards earned per lap and lap times from agents trained with maximum speeds of 4, 5, 6, 7, and 8 m/s on the ESP map. Results are from an average of five repetitions, trained for 10,000 steps.

The graph in Figure 7.12 shows a clear pattern that as the maximum speed increases, the lap time decreases. The 4 m/s agent starts with lap times of around 120 seconds and learns to race with lap times of under 90 seconds. The 8 m/s agent maintains near-constant lap times of around 75 seconds throughout training. While the 8 m/s agent's lap times do not change during training, the reward graph shows that the agent earns significantly more reward, indicating that as the training progresses, it learns to select actions that maximise the reward. The 4 m/s agent earns the most reward because the reward is scaled according to the maximum speed. For the 4 m/s agents, the maximum speed is 4, so the agents will receive more reward for a speed than if the maximum is 8 m/s. This result shows that by the time training is complete, the lap times range from 75 seconds to 95 seconds. It is surprising to see a small difference in lap time, considering the large difference in maximum speed from 4 m/s to 8 m/s. The training is further investigated by plotting the interventions per 100 training steps for the 4, 6 and 8 m/s agents.

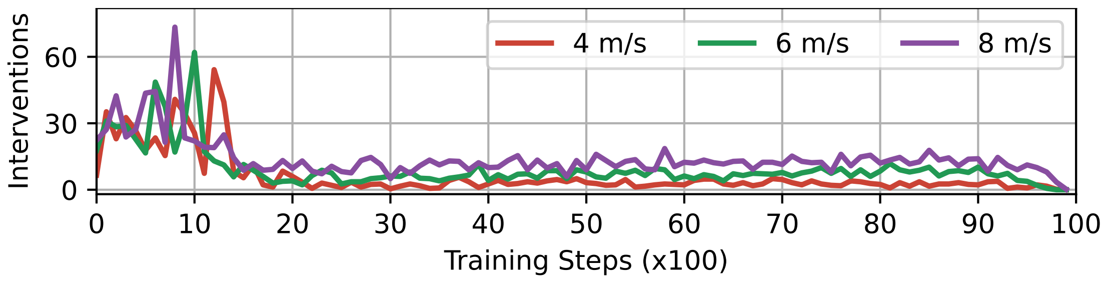


Figure 7.13: The interventions per 100 steps during training agents with maximum speeds of 4, 6 and 8 m/s on the ESP map.

Figure 7.13 shows the graph for the number of interventions by the supervisor for the 4, 6 and 8 m/s agents. At the beginning of training, all the agents require a lot of interventions, roughly 0 per 100 steps, with all the agents having a spike of around 70 interventions in the first 1,500 steps. By 2,000 steps, all the agents have roughly converged, requiring only a few interventions. The 4 m/s (red line) agent converges, requiring the least interventions. The 8 m/s agent requires many more interventions, maintaining an average of around 5 per 50 steps. The 6 m/s agent is in the middle, requiring fewer interventions than the 8 m/s agent. This result indicates that the 8 m/s converges to requiring some interventions during training, meaning that it does not learn to select only safe actions.

The training study concludes that agents with maximum speeds ranging from 4 m/s to 8 m/s can be trained to converge in 10,000 steps using the velocity reward signal. Agents with higher maximum speeds achieve similar lap times from the beginning of training, while agents with lower maximum speeds show a significant improvement in lap time during training. While all the agents tested converge to requiring few interventions, agents with higher maximum speeds require more interventions during training.

Maximum Vehicle Speed Performance

The performance of the trained agents is measured by running 20 test laps with the 4 m/s, 6 m/s and 8 m/s agents on the ESP track. Table 7.2 shows the performance metrics of lap time, average speed, and distance travelled for the agents using maximum speeds of 4 m/s, 6 m/s and 8 m/s. The results show that the distance travelled by the agents is similar, around 235 m, indicating that the distance travelled by the agents is not related to the maximum speed.

In Table 7.2, the 8 m/s agent achieves a much faster lap time of 63.89 seconds compared to the 4 m/s agents of 99.62 seconds. This is explained by the higher average velocity of 3.7 m/s compared to the 4 m/s agent's average velocity of 2.38 m/s. The average velocity is surprisingly low compared to the maximum speed, being less than half of the maximum

Metric	**4 m/s**	**6 m/s**	**8 m/s**
Lap time (s)	99.62	71.92	63.89
Avg. Speed (m/s)	2.38	3.30	3.70
Distance (m)	235.48	236.88	235.70

Table 7.2: The performance metrics of lap time, average speed, distance and for agents with maximum speeds of 4 m/s, 6 m/s, and 8 m/s tested on the ESP track.

for each agent. For example, the agent with a maximum speed of 6 m/s has an average speed of 3.3 m/s, which is just 55% of the maximum. The trajectories selected by the agents are further analysed by plotting trajectory segments and speed profiles from the tests on the ESP map.

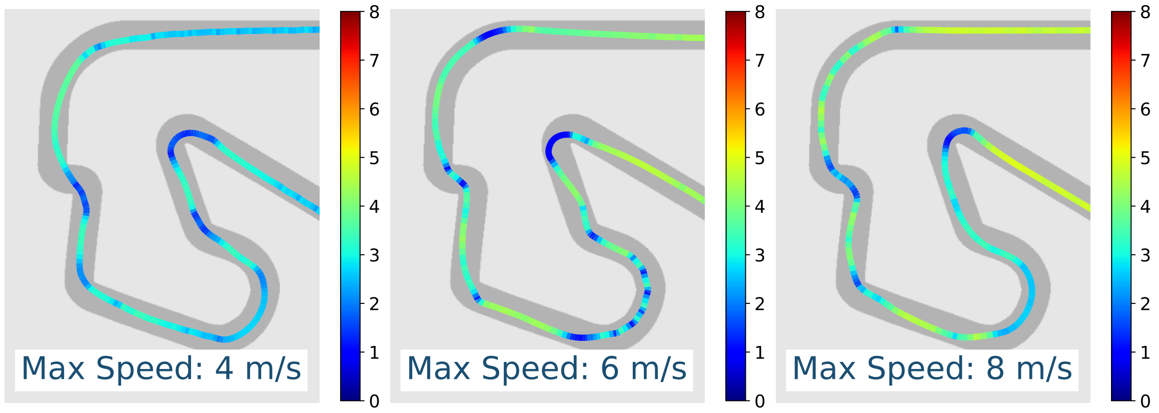


Figure 7.14: Trajectories segments on the ESP map from agents trained with maximum speeds of 4 m/s, 6 m/s and 8 m/s.

Figure 7.14 shows trajectory segments from agents trained with maximum speeds of 4 m/s, 6 m/s and 8 m/s on the ESP map. The paths taken in all the segments are similar, confirming that the maximum speed does not influence the path selected by the agent. The 4 m/s trajectory is mainly dark blue with lighter blue sections on the straights. The 6 m/s trajectory starts to have some yellow regions in the straights, indicating that it selects higher speeds. The 8 m/s trajectory has many yellow regions, with dark blue only on the hairpin corner. While these trajectories show that increasing the maximum speed leads to the agent selecting higher speeds, the agents still select relatively low speeds, as shown by the lack of red on the graph. The speed profiles of the three different agents are plotted for the first half of the ESP track.

Figure 7.15 shows the speed profiles for the 4, 6 and 8 m/s agents on the ESP track. The 4 m/s agent (red line) selects average speeds of 3 m/s, and the 6 m/s agent selects slightly faster speeds, occasionally above 4 m/s. The 8 m/s agent selects speeds up to around 5.5 m/s, but not above that. Surprisingly, the agents do not learn to select speeds up to the limits of what they can.

This result shows that while the agents learn valid speed profiles of speeding up and

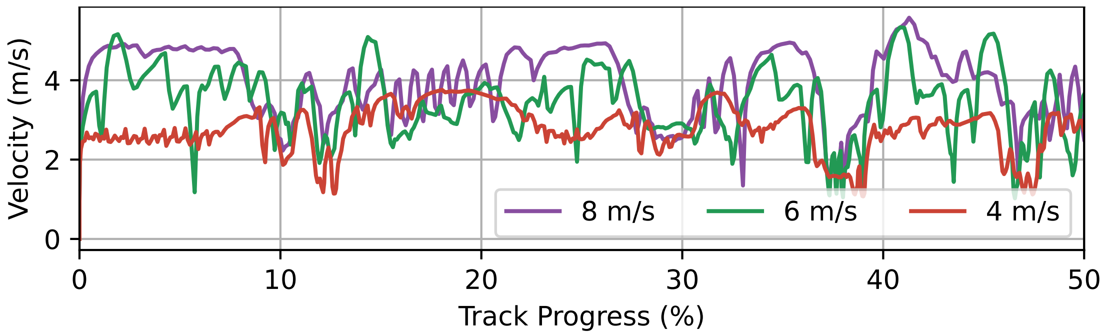


Figure 7.15: Speed profiles for the 4, 6 and 8 m/s agents on the ESP track.

slowing down, they are conservative and select low speeds. This conservatism explains why the average speeds in Table 7.2 are so low. The reason for this behaviour is unknown and should be further investigated in future work. A possible explanation is that the supervisor is inherently over-conservative to ensure safety and thus would penalise the agent for exceeding a safe speed at any point on the track.

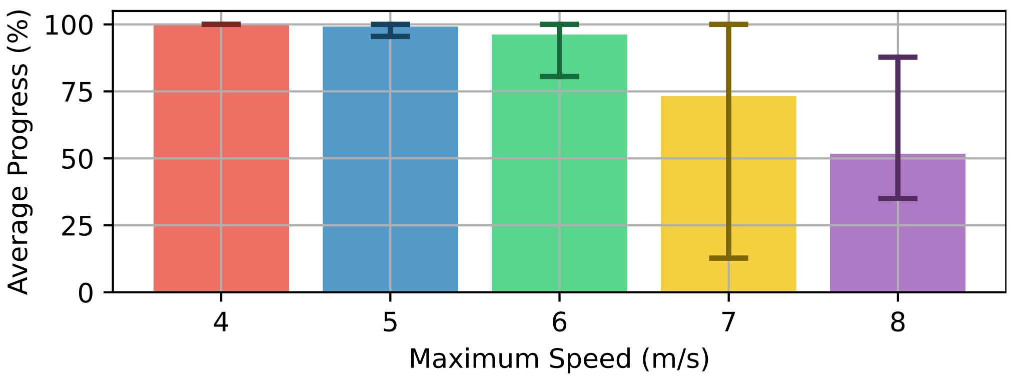


Figure 7.16: The average progress made by each of five repetitions from training agents with maximum speeds ranging from 4-8 m/s on the ESP map.

Figure 7.16 presents the average progresses of the five agents trained at each speed. The coloured bar represents the mean of the five training runs, and the error bar represents the minimum and maximum between the five runs. The results show that the 4 m/s, 5 m/s and 6 m/s agents all achieve high average progresses, with means above 95%. The 7 m/s agent has a mean of around 75%, and the 8 m/s agent has a mean of 50%. The 7 m/s and 8 m/s agents have large error bars indicating that some of the training runs performed poorly. Therefore, it is concluded that the training is repeatable up to a maximum speed of 6 m/s. While higher maximum speeds lead to better lap times, the training becomes less repeatable, with some random seeds producing poor performance with low completion.

It is concluded that agents can be trained online to learn a conservative speed profile of speeding up and slowing down. The online agents complete most laps when the maximum speed is 6 m/s. While online training can sometimes train agents up to a high speed, the training becomes less stable, and sometimes the agents achieve low average progress. The advantages of online training are that the agent never crashes during training and the training requires only 10,000 steps.

7.3.3. Reward Signal Comparison

The supervisor's ability to train an agent to select actions to race the vehicle around the track has been demonstrated using the velocity reward signal. The previous evaluations used the velocity reward signal so that the effect of the supervisor could be studied. This evaluation considers using different reward signals to encourage high-performance racing behaviour. The zero, progress and cross-track and heading rewards are considered and compared to the velocity reward and the PPPS reward from §5.2.2. For this study, the maximum vehicle speed is set to 5 m/s.

Training Comparison

The training performance generated by each reward signal is investigated by training five agents using each reward for 10,000 steps, recording the lap times, the reward earned per lap and interventions during training. The ESP map is used for the training investigation.

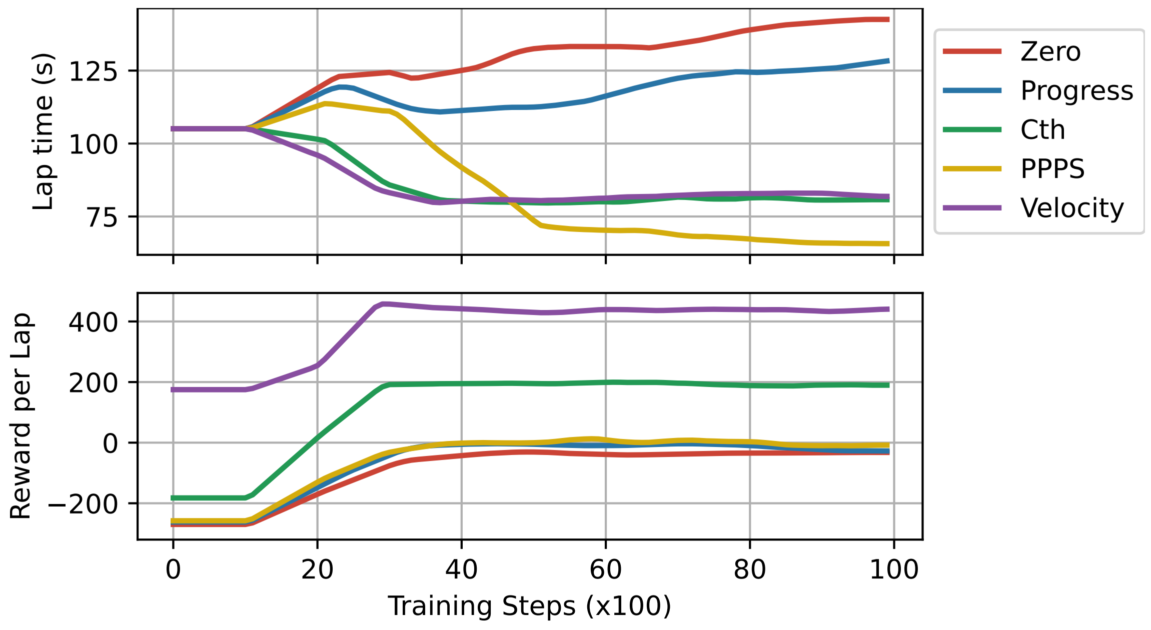


Figure 7.17: The lap times and reward earned per lap during training of the zero, progress, cross-track and heading error (Cth), PPPS, and velocity rewards on the ESP map. The average of 5 repeats.

Figure 7.17 presents the lap times during training using the zero, progress, cross-track and heading error, PPPS, and velocity rewards on the ESP map. The results in the figure are averaged over five runs, but the minimums and maximums are not shown. The graph shows a significant difference between the reward signals. The zero progress rewards produce the worst results in terms of lap times, with the lap times getting worse as training progresses. This is due to the lack of direct motivation for the vehicles to select high-speed actions.

All the lap times start at around 105 seconds and diverge from around 1,000 training steps. The cross-track and heading error, velocity and PPPS rewards improve the vehicle's lap times. The velocity and cross-track and heading reward demonstrate similar performance with the lap times moving to around 80 seconds within 4,000 steps. The improvement in these rewards is due to them both explicitly rewarding velocity. The PPPS reward outperforms the conventional reward signals by reaching lap times of less than 70 seconds. This indicates that the advantage of using the racing line in the reward also improves the performance of online learning.

While the lap time graphs vary greatly between rewards, the reward graph shows a similar pattern for all the rewards. All the rewards start at a value and, within 4,000 steps, rise about 200 and then settle there for the rest of the training. Online learning using the SSS enables the agents to complete laps from the beginning of training, meaning that the lap times can be measured and evaluated. The lap times of the velocity and PPPS rewards are further investigated by plotting all the individual runs from the five repetitions.

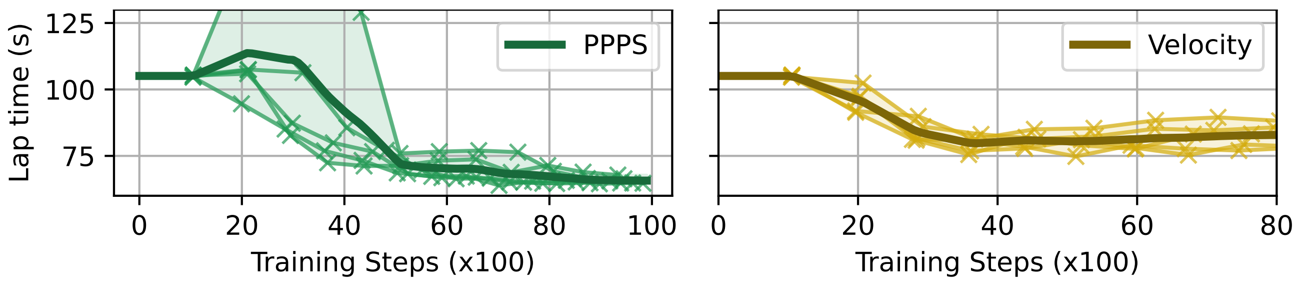


Figure 7.18: The lap times for the pure pursuit policy search (PPPS), and velocity reward signals with each repetition of five experiments plotted with the average. The graphs use the same y-axis for comparison.

Figure 7.18 shows the lap times during training for agents trained with the PPPS and velocity rewards. While there is a significant variation between different random seeds using the PPPS reward, by around 5,000 steps, all the repetitions have converged to within 75 seconds per lap. The velocity reward repetitions are more closely grouped together, with them all following the same pattern of moving from 105 seconds at the start to around the average of 80 seconds.

The performance of the rewards is further investigated by considering the supervisor's interventions during training. The focus of the rest of the evaluation is on the velocity, cross-track and heading and PPPS rewards since the improve they demonstrate improved lap times during training.

Figure 7.19 shows the interventions during the first 2,000 training steps for the cross-track and heading error, PPPS and velocity reward functions. The graphs show that for the first 500 steps, the pattern of interventions looks similar for all the graphs. As the

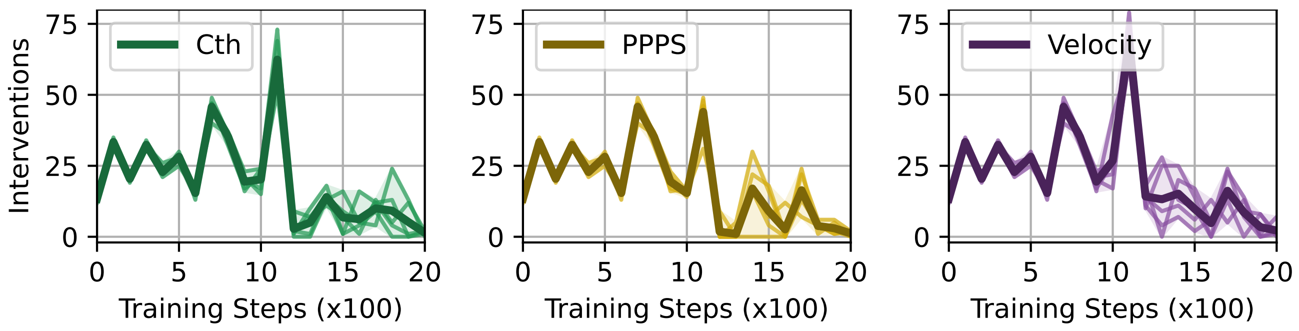


Figure 7.19: The interventions per 100 training steps during the first 2,000 steps of training for the cross-track and heading error (Cth), PPPS and velocity rewards.

training progresses, the patterns diverge according to the reward signal. By 2,000 training steps, the number of interventions has significantly decreased to around 10 per 100 training steps.

The training comparison concludes that the velocity, cross-track and heading and PPPS rewards all effective for the online training of DRL agents. The velocity and cross-track and heading reward show similar behaviour in terms of training speed and lap times. The PPPS reward using the racing line, produces the lowest lap times during training. Within 2,000 steps, the agents learn not to require interventions from the supervisor.

Performance Comparison

The performances of the different reward signals are compared with each other. The ESP track is used for comparison, and the metrics of lap time, average speed, distance travelled, and curvature are recorded. The zero agent is excluded due to generating poor performance.

Metric	Progress	Cth	PPPS	Velocity
Lap time (s)	137.37	80.26	58.88	74.31
Avg. Speed (m/s)	1.75	2.90	3.98	3.18
Distance (m)	239.17	232.43	233.18	235.48
Curvature (m^{-1})	295.66	137.32	85.52	129.18

Table 7.3: The metrics of lap time, average speed, distance and curvature for the progress, cross-track and heading error (Cth), PPPS and velocity reward signals on the ESP track.

The results in Table 7.3 show a similar pattern of lap times to the training graph in Figure 7.17. Reading the table left to right, using the same ordering as the training graphs, the lap times steadily decrease. The progress reward achieves the slowest average lap time of 137.37 seconds, with the lowest average speed of 1.75 m/s. The cross-track and heading error agent achieves a lap time of 80.26 seconds. The velocity reward achieves a faster lap time of 74.31 seconds, and the PPPS agent achieves the fastest lap time of 58.88 seconds.

The PPPS agent has the lowest curvature of 85.52 m^{-1}. The curvatures increase in the same order as the lap times of PPPS, velocity, cross-track and heading error and progress, with the highest curvature of 239.17 m^{-1}. In contrast to end-to-end learning, where the progress reward results in a shorter distance travelled, the progress reward now results in the longest distance travelled of 239.17 m. The cross-track and heading and PPPS agents take similar distances of around 233 m, and the velocity agent takes a slightly longer path of 235.48 m.

The average progress of the training is investigated by plotting the mean the average progress out of the five runs with the minimum and maximum.

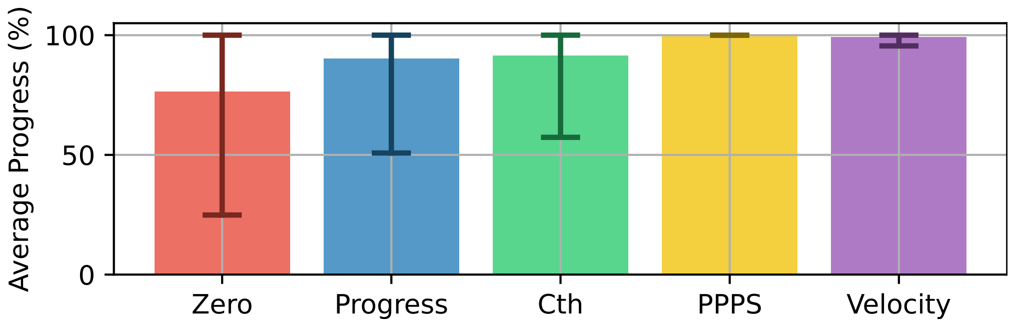


Figure 7.20: Comparison of the average progress achieved by the zero, progress, cross-track and heading error (Cth), PPPS and velocity rewards on the ESP map.

Figure 7.20 shows the average progress made by agents trained with the zero, progress, cross-track and heading error, PPPS and velocity rewards on the ESP map. The mean of the average progresses increases from left to right. The zero reward performs poorly with a mean of around 75%, and the progress and cross-track and heading reward have similar performance of around 90%. The PPPS and velocity reward perform excellently, with average progress of almost 100% in all the repetitions.

The performance study showed that the PPPS reward outperforms the conventional rewards in online learning. This is expected since it uses the racing line to train the agent. Of the conventional reward signals, the velocity reward performs the best with the lowest lap times and highest average progress. The velocity reward is simple, using only the vehicle's speed, but produces good behaviour. The cross-track and heading reward is slightly slower, and some repetitions do not complete all the laps. The progress and zero rewards produce poor results with slow lap times and thus are not suitable for training variable speed agents. These results show similar patterns to conventional learning, with the cross-track and heading reward outperforming the progress reward and the racing line reward outperforming the conventional rewards.

Speed Profile Analysis

The final reward signal analysis compares the speed profiles selected by different reward signals. The speeds selected by the cross-track and heading, velocity and PPPS reward are compared on the ESP track by considering a trajectory segment and the speed profile graph.

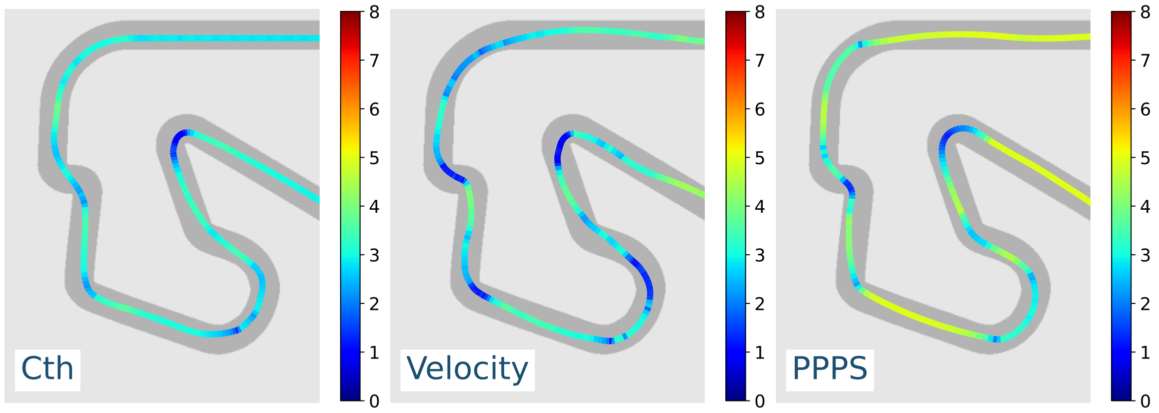


Figure 7.21: Trajectories selected by agents trained with the PPPS, cross-track and heading error (Cth), and velocity reward signals.

Figure 7.21 shows a trajectory segment from the cross-track and heading error, velocity and PPPS agents on the ESP track. The cross-track and heading agent selects an appropriate speed profile of speeding up and slowing down around the corners. The velocity agent slows down more around the corners, as shown by the dark blue, but also appears to speed up more in the straighter sections. The PPPS agent reaches the highest speeds (as shown by the green and yellow) in the straights while slowing down for the corners. The speed profile graphs of the PPPS and velocity agents are compared to the pure pursuit planner.

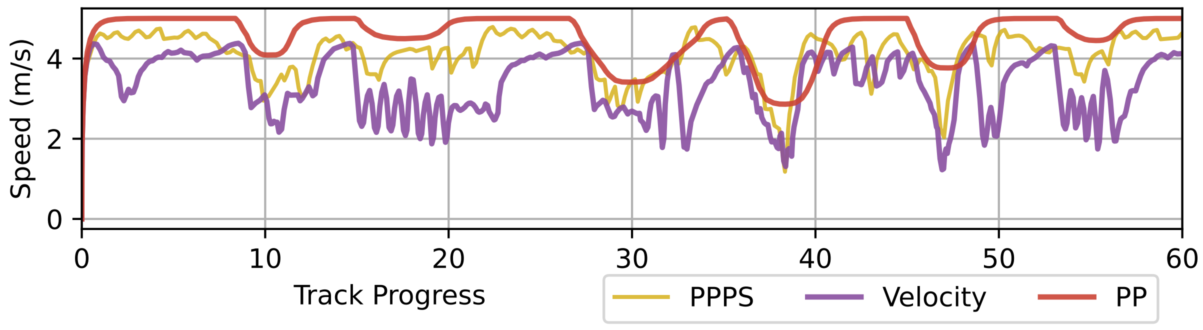


Figure 7.22: Speed profiles comparing velocity and PPPS rewards with the pure pursuit (PP) planner speed profile for a section of the ESP track.

Figure 7.22 compares the speed profiles of the velocity and PPPS agents with the pure pursuit planner. The pure pursuit planner selects a smooth speed profile, often at the vehicle's maximum speed used in the experiments of 5 m/s. The PPPS agent tracks the shape of the pure pursuit speed profile while being around 0.5 m/s slower. The velocity agent is even slower than the PPPS agent, often selecting speeds 1-2 m/s slower than the pure pursuit planner. This result shows that agents trained using online learning with the supervisor select more conservative speed profiles that are lower than the pure pursuit planner.

It is surprising that where the conventional agents always selected speeds that were too high and towards the maximum of the speed range, online learning selects speeds that are lower than the allowed speed range. The conclusion is that the speed profile selection in autonomous racing is by no means a solved problem, and further work is required to understand how to select a more moderate speed profile.

The study on reward signals concludes that the PPPS reward achieves the best racing performance in terms of the lowest lap times, highest average performance, and best speed profile selection. The second best reward is the velocity reward, due to fast lap times and high average progress, followed by the cross-track and heading reward. While the zero and progress rewards occasionally work to train agents, they are not recommended for online training.

7.3.4. End-to-end vs Online Learning

The method of online learning presented here is evaluated against conventional learning, as presented in Chapter 4. The evaluation starts with a constant speed evaluation, followed by a variable speed evaluation and qualitative comparison of the methods.

Constant Speed Performance

The constant speed evaluation compares conventional end-to-end learning with an agent trained online using the safety system. The pure pursuit planner is given for comparison and treated as the optimal racing planner. The conventional agent is trained with the cross-track and heading error reward signal, and the online agent is trained with zero racing reward.

Metric	E2e	Online	PP
Total Distance (m)	236.87	233.86	231.14
Total Curvature (m^{-1})	326.64	193.05	147.65
Mean Steering (rad)	0.13	0.06	0.04
Success Rate (%)	100.00	100.00	100.00

Table 7.4: Performance metrics of distance travelled, curvature, centre line and race line deviation for the conventional end-to-end (E2e), and online agents compared against the pure pursuit (PP) planner following the racing line on the ESP map.

Table 7.4 shows the performance metrics of distance travelled, curvature, centre line and race line deviation for the conventional end-to-end agent (E2e) and online agents compared to the pure pursuit (PP) planner following the racing line on the ESP map. The online agent achieves a shorter average distance of 233.86 m compared to the conventional agent's distance of 236.87 m. The total curvature of the path selected by the online agent of 193.05 m^{-1} is significantly smaller than the conventional agent's total curvature of 326.64 m^{-1}. The mean steering angle of the online planner (0.06 rad) is less than half of the conventional agent (0.13 rad), suggesting that the online planner selects much more moderate steering actions. All of the planners complete all of the test laps, resulting in a 100% success rate. While the online agent improves on the conventional agent with a shorter distance travelled, less curvature and a smaller mean steering angle, the pure pursuit planner still outperforms the online trained agent with a total distance of 231.14 m, a curvature of 147.65 m^{-1} and mean steering of 0.04 radians.

While the online planner significantly improves over the conventional planner, it does not perform as well as the pure pursuit planner in any of the metrics considered. The pure pursuit planner has a shorter distance of 231.14 m, smaller curvature of 147.65 m, and a lower mean steering angle of 0.04 rad. The paths selected by the three planners are compared for a portion of the ESP track in Figure 7.23.

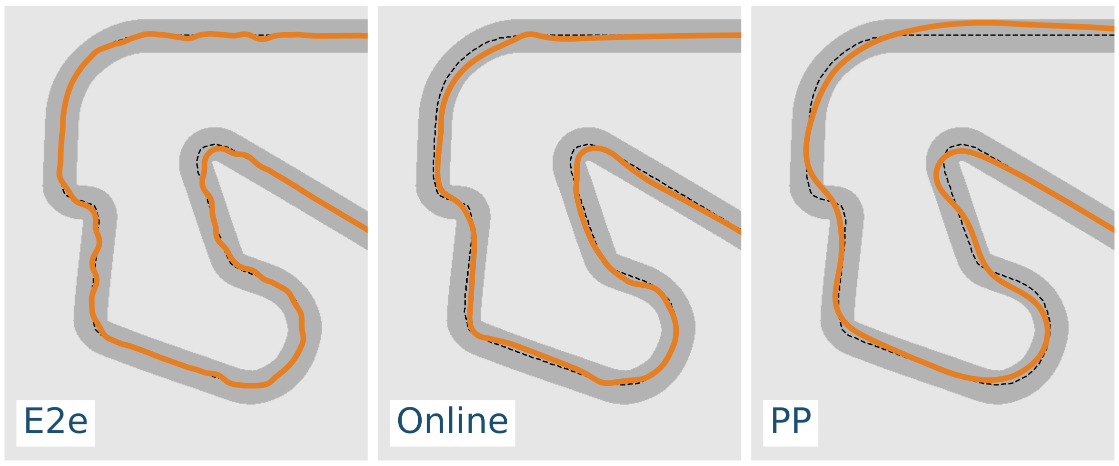


Figure 7.23: Trajectories by the conventional end-to-end (E2e) agent, online agent and pure pursuit (PP) planner on a section of the ESP map.

The paths selected by the conventional end-to-end agent (E2e), online agent and pure pursuit (PP) planner on a section of the ESP map are compared in Figure 7.23. The slaloming problem present in conventional learning is no longer present using the online learning formulation. While online learning does not exactly match the pure pursuit planner, it learns a good racing policy of driving smoothly around the corners, roughly tracking the centre line. The behaviour is now analysed by considering the steering actions selected by the planners.

The steering actions selected by the conventional, online and pure pursuit planners on a portion of the ESP track are plotted in Figure 7.24. The graph shows the fluctuating, extreme steering angles selected by the convention planner. The online agent significantly improves over the conventional planner selecting more moderate steering actions. The online planner still has occasional spikes in the steering angle, compared to the smooth

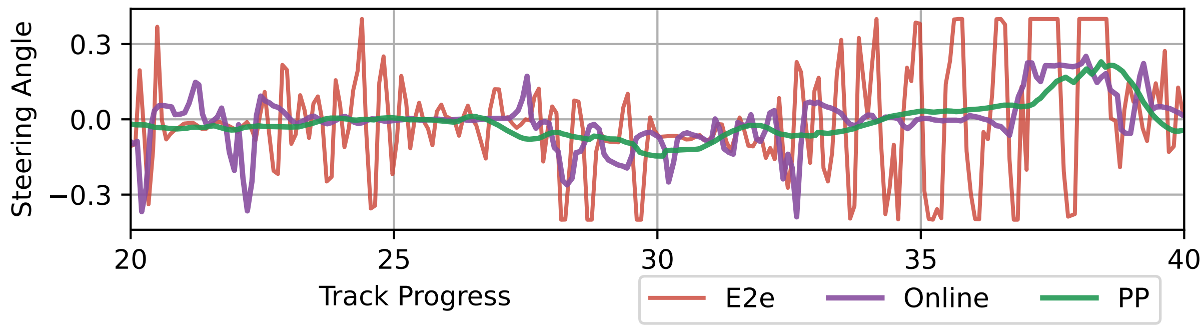


Figure 7.24: Steering actions selected by the conventional (E2e), online and pure pursuit (PP) planners on a portion of the ESP track.

pure pursuit planner.

The constant speed evaluation concludes that online learning using zero reward signal results in smoother, shorter paths than end-to-end learning. A specific performance improvement is that the steering actions that are selected are smoother and do not fluctuate between the extremes.

Variable Speed Performance

The variable speed of the online training is compared between the conventional, online and pure pursuit planners. For all the evaluations in this section, the maximum speed is limited to 5 m/s.

Metric	E2e	Online	PP
Lap time (s)	48.43	74.31	52.30
Avg. Speed (m/s)	4.96	3.18	4.40
Distance (m)	239.62	235.48	229.53
Avg. Progress (%)	91	100	100
Completion Rate (%)	75	100	100

Table 7.5: Performance metrics of lap-time, average speed, the success rate for the conventional (E2e), online and pure pursuit (PP) planners on the ESP map with a maximum speed of 5 m/s.

Table 7.5 presents the metrics of lap time, average speed, distance travelled, average progress and completion rate for the conventional planners on the ESP map. The online agent has the longest lap time of 74.31 seconds, compared to the conventional planner's time of 48.43 seconds and the pure pursuit planner's 52.3 seconds. This difference is explained by the significantly lower average speed of 3.18 m/s compared to the conventional agent's speed of 4.96 m/s. The conventional agent does not complete all the test laps, achieving only a 91% average progress and 75% completion rate. This shortcoming is not experienced by the online planner that completes all the test laps. The online planner takes a slightly shorter path around the track. The trajectories selected by the conventional and online agents are further analysed by comparing them to the pure pursuit planner.

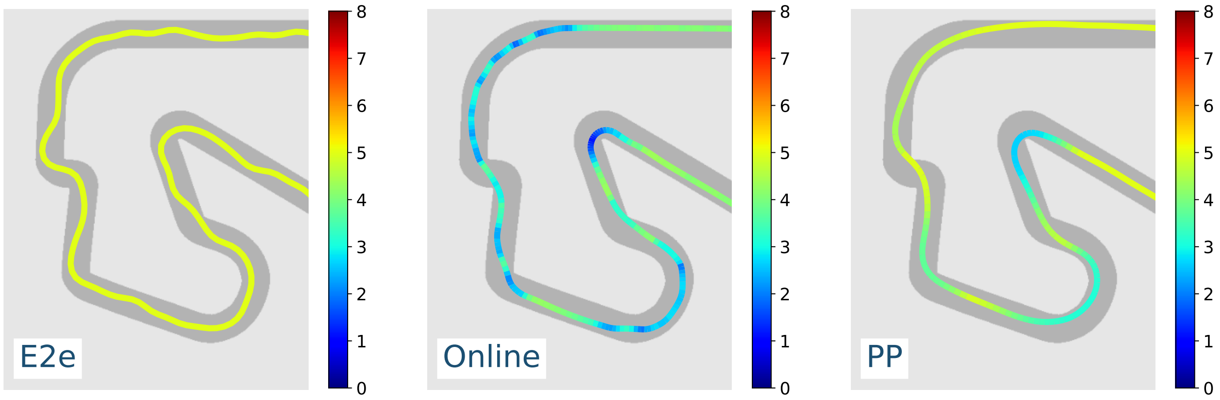


Figure 7.25: Trajectories by the conventional (E2e), online and pure pursuit (PP) planners on a section of the ESP map.

Figure 7.25 shows the trajectories for the conventional, online and pure pursuit planners on a section of the ESP track. The conventional planner selects only the maximum speed for the whole way around the track. The pure pursuit planner selects an optimal speed profile and path. The online planner selects an appropriate speed profile that speeds up on the straights and slows down around corners. The online planner selects much lower speeds than the pure pursuit planner for the entire trajectory, explaining why the lap times in Table 7.5 are longer.

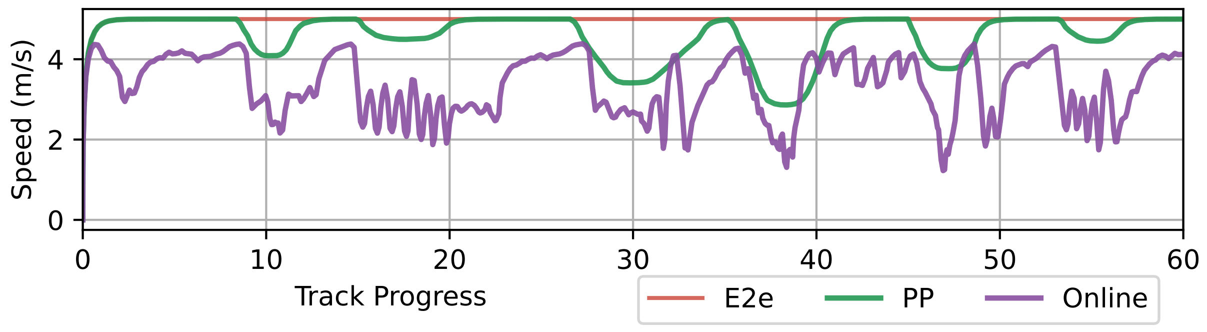


Figure 7.26: Speed profile comparison for the conventional (E2e), pure pursuit (PP) and online planners on a portion of the ESP track.

Figure 7.26 provides further analysis on the speed profiles selected by the different planners. The conventional planner (E2e) selects the maximum speed of 5 m/s for the entire trajectory. The pure pursuit planner (PP) is capped at 5 m/s and selects the maximum speed for a significant part of the lap. The online agent (shown in purple) selects speeds significantly lower than the pure pursuit planner. The lower speeds selected by the online planner make the planner more conservative and safer while achieving slower lap times.

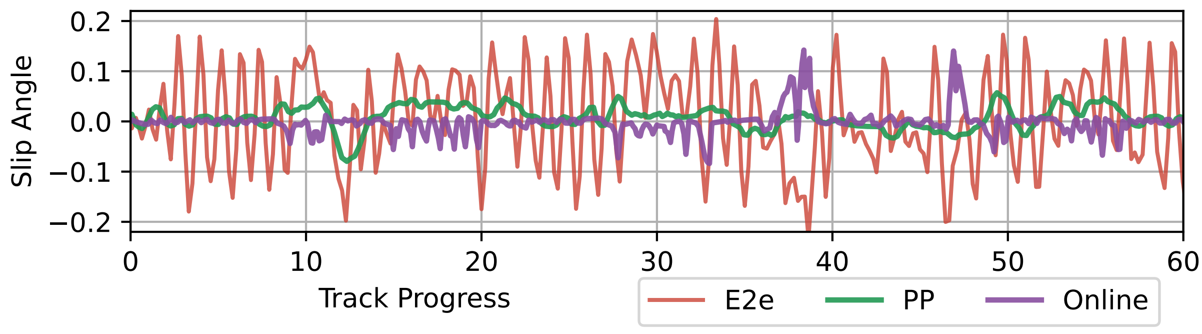


Figure 7.27: Slip angle comparison for the conventional (E2e), pure pursuit (PP) and online planners on a portion of the ESP track.

The slip angles for the conventional, pure pursuit and online planners on a portion of the ESP track are plotted in Figure 7.27. The conventional planner (E2e) has a high slip angle with many spikes above the 0.1-radian line. The online planner has a low slip angle, mainly staying within 0.05 radians and only breaching the 0.1-radian line twice. The pure pursuit planner has a smooth slip profile with no spikes above the 0.1-radian line. This result indicates that the conservative speed selection shown by the online agent results in the vehicle maintaining a small slip angle comparable to the pure pursuit planner.

The variable speed comparison concludes that agents trained online learn to select a speed profile that speeds up in the straights and slows down in the corners. The online agent selects a conservative speed profile resulting in significantly lower lap times than the pure pursuit and conventional planners. The conservative behaviour leads to agents trained online completing all the test laps and racing with a low-slip angle. While in racing, slow speed selection leads to undesirable longer lap times, a significant advantage of over-cautious systems is that they are further from the safety boundaries and thus remain safe.

Qualitative

Conventional and online learning are compared by analysing the quality of training steps required, the average number of crashes during training, and the feasibility of online training.

Table 7.6 shows the differences between end-to-end and online learning across the factors of the number of training steps required, the average crashes during training and the feasibility to train on physical vehicles. The comparison shows a training step reduction of 5× for constant speed and 10× for variable speed. This significant reduction in sample efficiency enables this method to be trained on a physical vehicle in a feasible amount of time. A limitation of online learning is that it requires localisation during training. However, this limitation is not significant since localisation can be provided by a particle filter for a track where the map is available.

	Constant Speed		*Variable Speed*	
Metric	**E2e**	**Online**	**E2e**	**Online**
Training steps required	30,000	6,000	100,000	10,000
Avg. crashes during training	48	0	350	0
Avg. Success Rate	100%	100%	75%	100%

Table 7.6: Qualitative comparison of the training steps required, average crashes during training and average success rate for conventional (E2e) and online learning.

The major advantage of online learning is safety during training. While conventional training requires crashing to learn safe behaviour, online learning can learn not to crash without ever crashing. For training on a physical vehicle to be possible, crash-free behaviour must be guaranteed. The safety of the online trained agent is a further improvement on conventional learning. The agent trained online achieves a success rate of 100% compared to the conventional agent, that even after being trained for 100,000 steps still crashes 25% of the time.

7.4. Physical Vehicle Validation

The method of online learning using the supervisory safety system is validated by training an agent to drive onboard a physical vehicle. The focus of the evaluation using the physical vehicle is to compare the simulated performance against that in reality and to demonstrate the advantages of onboard training.

7.4.1. Evaluation Methodology

Online Agent Configuration

The safety system is used to train an agent, with no a priori knowledge or training, onboard a vehicle to drive around a track autonomously. The vehicles drive at a constant speed of 2 m/s in environment 1 and 1.5 m/s in environment 2. The filtered kernels, described in §6.5, are used since the steering angle is not available on the practical vehicle. The online agent is trained with zero reward signal to demonstrate that online training removes the need for reward shaping.

In the simulation, the training can occur in real time, with episode roll-outs happening faster than in real time. However, during training onboard the physical vehicle, there is significantly less computational power due to the smaller computer, and online localisation already uses several resources. The solution was that the vehicle would drive and collect 20 samples (takes 2 seconds) and then stop and train on the data for 20 batches of 40 random samples from memory. The vehicle would then continue driving and collecting more samples before stopping to train. The online agent is trained for two laps (around 800 steps), stopping to train the networks at intervals of 20 steps. Since the vehicle travels at a constant speed, the stopping to train does not affect the action taken. If this experiment were expanded to variable speed racing, the stopping may affect the training and should be studied.

After the online agents are trained, the supervisor is removed and the tests are carried out using only the agent to select actions.

Baseline Agent Configuration

The results are compared to a baseline agent that is trained offline, in simulation and then transferred to the vehicle. The baseline agent is trained in simulation using the progress reward signal for 30,000 steps.

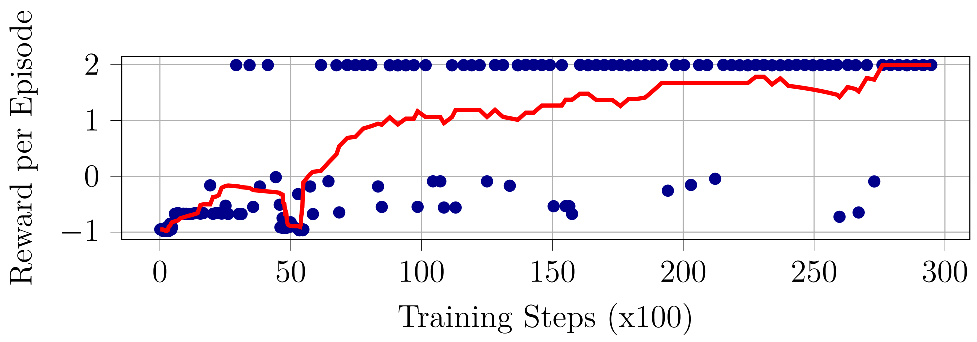


Figure 7.28: Rewards per episode (blue dots) earned by the baseline agent during offline training, with moving average (red line).

Figure 7.28 shows the rewards achieved per episode by the baseline planner, which is trained offline in the simulator, using the progress reward signal. The graph shows that at the beginning of training, the agent receives many negative rewards; as the training progresses, the average reward the agent achieves increases. By the end of the training, the agent can consistently complete laps without crashing.

7.4.2. Onboard Training

The supervisor's ability to train an agent is evaluated by validating that it keeps the vehicle safe, even when random actions are selected, and by measuring the effect of the supervisor on the training.

Supervisory Safety

The safety of the supervisor is shown by initialising a random agent and training it from scratch (random initialisation) on the physical vehicle. Figure 7.29 shows the first training lap of a random agent loaded on the vehicle, run in conjunction with the supervisor.

The lap starts at a red dot before the line break, and the vehicle then drives toward the left. The green points show where the agent stopped to train on the data that had been collected. The trajectory shows the squiggles that the agent takes as it veers to one side and then to the other.

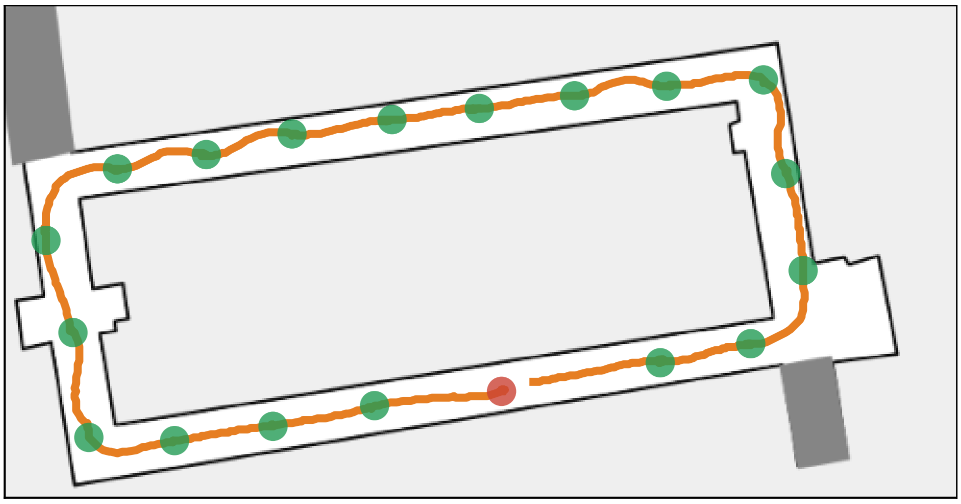


Figure 7.29: A random agent loaded on the vehicle run with the supervisory system. The green dots indicate the locations where the vehicle stops to train the agent.

This trajectory demonstrates that the safety system can keep a randomly initialised vehicle from crashing into the track boundaries.

Learning Analysis

The speed with which the agent learns online, using the supervisor, is measured. Since the episodes have been reformulated (see §7.2.2), they always end with a terminal reward of −1 when the supervisor intervenes. Therefore, the sum of the reward achieved every 20 steps (the interval of data collection between the agent stopping to train) is used as the metric to measure the online training performance. Using this metric, the worst reward is −20 if the supervisor intervenes at every step, and the maximum reward is 0 if the supervisor never intervenes.

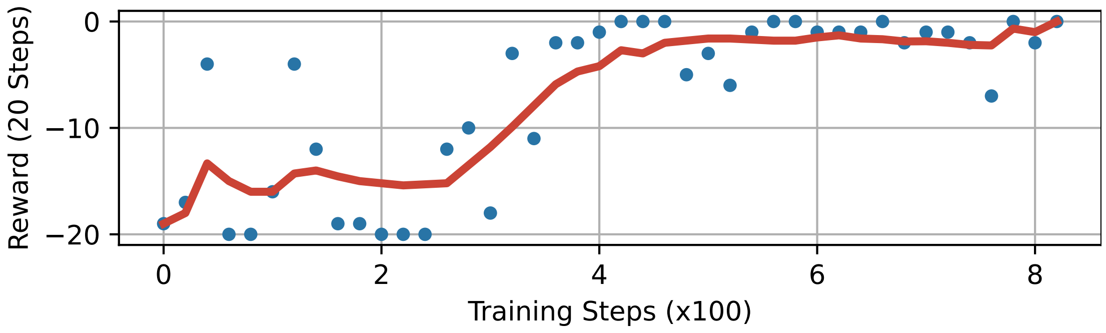


Figure 7.30: Training rewards per 20 steps for safety agent trained on the physical vehicle (blue dots) with moving average (red).

Figure 7.30 shows a graph of the sum of rewards achieved every 20 steps by the agent trained onboard the physical vehicle. The graph shows that in the beginning, the agent receives low rewards; as time progresses, the agent receives higher rewards. After around only 400 steps, the agent displays a significant improvement. The improvement after 400 steps corresponds to the graph of safe steering actions in Figure 7.31, showing that the agent requires less intervention between 300-400 training steps.

This result shows that our method of training a DRL agent onboard a vehicle significantly improved sample efficiency over the baseline method by requiring only 800 training steps, whereas the baseline method required 30,000 to converge.

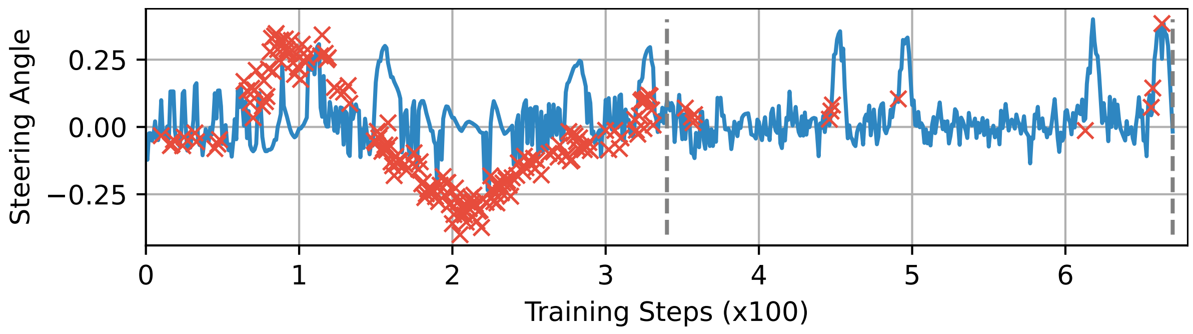


Figure 7.31: Comparison of steering actions implemented on the vehicle (blue) with unsafe actions selected by the agent (red) during training in environment 1.

A graph comparing the steering angles implemented on the vehicle (blue) with unsafe actions selected by the agent (red) during training is shown in Figure 7.31. At the beginning of the training, the agent rarely selects safe actions. As the training progresses, the agent selects more safe actions, and towards the end, the agent rarely selects an unsafe action.

This result shows the essential job of the supervisor to prevent the agent from taking unsafe actions during the initial stages of training, and how, as the agent is trained, it learns to select safe actions without requiring the supervisor. An additional benefit to this training regime is that the agent learns to select moderate actions and does not swerve excessively.

7.4.3. Performance Evaluation

Qualitative Analysis

Trajectories of the trained agents are provided for comparison. Figure 7.32 shows two real-world trajectories of one agent trained in simulation and then transferred to the vehicle and one agent trained online on the vehicle with the SSS.

Figure 7.32 shows that the agent trained in simulation has a extremely wavy path. It regularly comes close to the track boundaries and almost crashes several times. The agent trained on the vehicle using the SSS has a much smoother trajectory. The vehicle drives

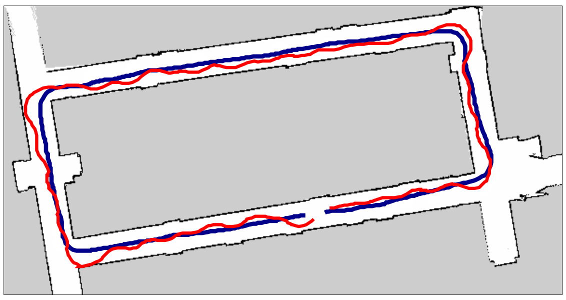


Figure 7.32: A comparison of real-world trajectories driven by the baseline agent trained in simulation (red) and the online trained agent with the SSS (blue) in environment 1.

in a straight line through the environment, smoothly turning the corners and not coming too close to the walls.

Quantitative Analysis

The metrics of distance, lap time, mean steering and total curvature are used to evaluate the agents' performance.

	Simulation		*Reality*	
	Baseline	**Online**	**Baseline**	**Online**
Distance (m)	65.0	**59.8**	65.68	**61.6**
Lap-time (s)	32.8	**31.1**	35.5	**32.5**
Mean Steering (rad)	0.30	**0.03**	0.22	**0.06**
Curvature (m^{-1})	274.6	**34.2**	207.5	**86.3**

Table 7.7: Quantitative comparison of online and offline trained agents in environment 1.

Table 7.7 presents the quantitative results of the offline (baseline), and online (SSS) trained agents in environment 1, with the metrics of distance travelled, lap time, absolute mean steering and curvature. The SSS generally leads to a lower mean steering angle and lower total curvature of the trajectories, resulting in lower distance travelled and lower corresponding lap times than the baseline agents. For example, on the physical vehicle driving in environment 1 (shown in Figure 7.32), the baseline agent travelled 65.0 m, while the SSS agent travelled only 59.8 m, which is 5.2 m shorter. The average steering angle for the SSS agent was 0.03 radians, compared to the mean steering angle for the baseline of 0.3 radians. The baseline total curvature was significantly more (207.5) than the SSS agent's (86.3).

The online trained agent outperforms conventional training with smoother steering actions in both simulation and real-world tests. Although the SSS also performs worse in reality compared to simulation, training the agent on the real car shows a definite improvement in the performance of the physical vehicle compared to the baseline.

Robustness

A crucial aspect of agents is their ability to learn general policies that can be transferred to other environments. Both the agents trained in simulation and on the physical vehicle are tested on the track they were trained on (environment 1) and a different test track (environment 2, shown in Figure 7.33). The first observation is that both the baseline and SSS agents can complete laps on a different track from the one they were trained on, highlighting the advantage of the flexibility and adequate generalization of agents.

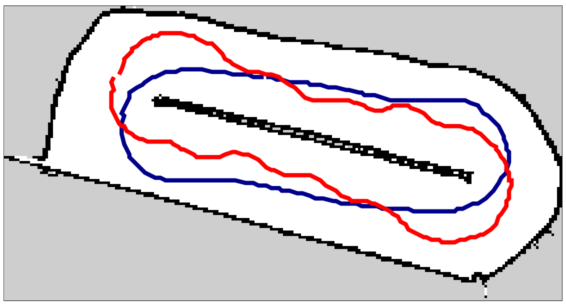


Figure 7.33: A comparison of real-world trajectories driven by the baseline agent trained in simulation (red) and the online trained agent with the SSS (blue) on environment 2.

Figure 7.33 shows that the trajectories followed in environment 2 display a similar pattern to that of environment 1. The SSS agent takes a smoother path and swerves less than the baseline.

	Simulation		*Reality*	
	Baseline	**Online**	**Baseline**	**Online**
Distance (m)	17.0	**15.6**	18.8	**17.3**
Lap-time (s)	12.9	**11.1**	12.8	**10.1**
Mean Steering (rad)	0.36	**0.11**	0.31	**0.09**
Curvature (m^{-1})	119.1	**44.9**	86.6	**49.3**

Table 7.8: Quantitative comparison of online and offline trained agents tested in environment 2.

Table 7.8 reinforces the result that the online trained agent selects a smoother path than the baseline agent. The results show that the SSS achieves a shorter lap time (1.5 s different), with a lower mean steering angle (0.09 versus 0.31) and less total curvature (49.3 versus 86.6) than the baseline planner. Therefore, it is concluded that the SSS agent learns more general behaviour, as demonstrated by improving performance on a different track.

7.4.4. Discussion of Real-world Online Learning

The results of the high-speed online learning evaluation showed that using the supervisor enables agents to be trained without ever crashing. Additionally, the performance of the trained agents is higher than those trained by crashing, providing a 10× improvement in sample efficiency.

As shown in the literature review, there has been a major lack of approaches to using physical robots, mainly due to the lack of safety. The main contribution of this work is to enable the application of methods on physical robotic systems. This ability was studied in demonstration and then validated on a physical racing vehicle. The results showed that training onboard the physical vehicle produced faster lap times and transferred better to unseen tracks.

Previous work in autonomous racing has shown extreme action selection to be a problem that has been overcome by reward hacking [60], and action regularisation through loss function amendments [22]. The proposed method offers the surprising advantage of producing policies that create smooth trajectories. It is suggested this is because extreme steering actions are more likely to be unsafe and thus removed by the supervisor. Additionally, using a supervisor leads to the agent being penalised sooner than it would have, thus improving the action trace.

7.5. Summary

This chapter used the safety system developed in Chapter 6 to train agents online without them ever crashing. §7.2 presented the architecture and learning formulation that incorporates the supervisor to train an agent safely. An in-depth evaluation of this method in simulation was presented in §7.3 and validated on a physical vehicle in §7.4. In the simulation investigation, the vehicle demonstrated the ability to train constant and variable speed agents to race without ever crashing, with a 5× improvement in sample efficiency. The constant speed evaluation did not require any reward shaping to train the agent, and the variable speed evaluation uses a simple reward proportional to the vehicle's speed. A current limitation in variable speed racing is that online learning trains the agent to be overly conservative, selecting speeds below the optimal speed. The comparison with conventional learning showed the main advantages of online learning to be sample efficiency, guaranteed safety during training, and safe racing behaviour after training, all characteristics that conventional learning did not display. The method was validated on a physical F1/10$^{\text{th}}$ vehicle, where it demonstrated improved performance over conventional learning in the metrics of distance travelled, mean steering angle and total curvature. Further, the online agent transferred better to a different race track not seen during training. The major advantage of training agents onboard physical vehicles is that the sim-to-real gap is bypassed since the training and testing happen on the same vehicle with the same dynamics.

Chapter 8

Avoiding Un-mapped Obstacles

This chapter expands on the problem of autonomous racing by adding the challenge of avoiding un-mapped obstacles. Obstacle avoidance is considered an intermediate problem on the path toward developing solutions for competitive head-to-head racing. While classical solutions are good at calculating and tracking high-performance trajectories, methods that rely on maps fail when obstacles are added to the track. End-to-end DRL methods are effective for flexible robot navigation in low-performance contexts but struggle to compete with classical methods. Therefore, hybrid planning architectures that combine path following and DRL components are considered for the task of local planning and obstacle avoidance. The three architectures, called the end-to-end, serial and modification planners, are presented. An evaluation of the three planners concludes that the modification planner can track a reference trajectory the most accurately and with the least curvature while achieving comparable performance with the end-to-end and serial planners on the metrics of completion rate and average performance.

8.1. Introduction

Classical solutions to autonomous racing, namely using a trajectory generator and path following algorithm, have been highly successful at racing around known tracks. However, these solutions rely on computing an optimal trajectory offline before the race begins, and thus they are inflexible to changes on the map - such as the addition of unmapped obstacles. Figure 8.1 shows the average progress of a pure pursuit path follower plotted against the number of randomly located obstacles on the Columbia and AUT tracks. Figure 8.1 shows that as the number of obstacles increases, the pure pursuit planner makes less and less progress. With zero obstacles, the planner achieve 100% completion and when 8 obstacles are added the planner achieves less than 40% completion on both maps.

The problem of autonomous racing with obstacle avoidance is addressed with the aim of designing a planner that can track a reference path while avoiding unmapped obstacles. Obstacle avoidance is a difficult problem because it must be handled in real-time and the only information about the obstacle's location is the LiDAR scan. Additionally, the state space is significantly larger. In racing on a fixed map, there are a fixed number of different

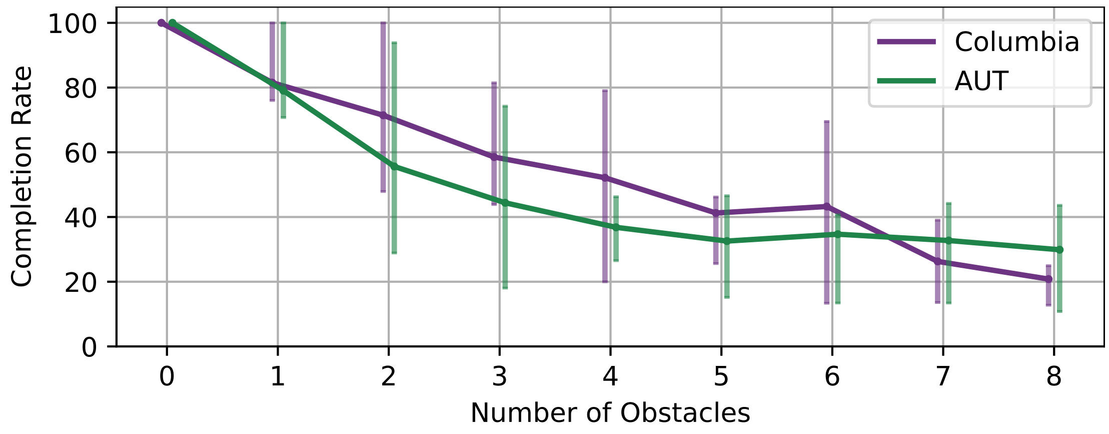


Figure 8.1: The percentage average progress from 100 test laps using the pure pursuit planner, following the centre line on the Columbia and AUT tracks. The vertical error bar shows the 1st and 3rd quartiles.

positions that the vehicle can be in. However, when obstacles are randomly added to the track, the possible situations and required actions grow significantly.

Three approaches are presented in this work, the end-to-end, serial and modification architectures. The end-to-end architecture is the simplest initial design that replaces the planner with a DRL agent. The end-to-end is treated as the baseline DRL method that simply replaces the planning pipeline with a DRL agent. The serial architecture includes the action selected by the pure pursuit planner in the agent's state vector to encourage the agent to follow the optimal path if no obstacles are present. The final design is the modification architecture that uses the DRL agent to learn only the subsystem of obstacle avoidance by adding the pure pursuit action and the DRL action together.

8.2. Candidate Architectures

A planning architecture is a method for connecting different planning components to enable the system to perform a task. This section focuses on how to connect the classical components of a path follower with the DRL components of a DRL agent for the task of obstacle avoidance. The aim of designing these architectures is to enable a vehicle to track a reference path while avoiding unmapped obstacles. The advantage of doing this is to retain the benefit brought by optimal trajectories, generated with classical systems.

8.2.1. End-to-end Learning

The first proposed DRL solution is to evaluate how well an agent trained end-to-end, similar to Chapter 4, can avoid obstacles. The end-to-end agent receives the state vector of beams sliced from the LiDAR scan. The agent calculates a steering action for the vehicle to follow. Figure 8.2 presents the architecture with the state vectors as input and the “calculate speed” function being used to calculate the vehicle's velocity.

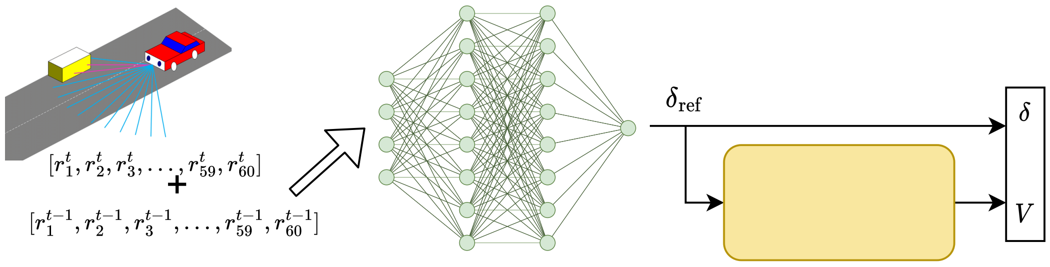

$[r_1^t, r_2^t, r_3^t, \ldots, r_{59}^t, r_{60}^t]$
+
$[r_1^{t-1}, r_2^{t-1}, r_3^{t-1}, \ldots, r_{59}^{t-1}, r_{60}^{t-1}]$
δ_{ref}
δ
V

Figure 8.2: The end-to-end (E2e) architecture uses two stacked LiDAR scans as input to the neural network to generate a steering angle and the calculate speed function to calculate a velocity reference.

State Vector

The state vector is what provides the agent with the relevant information for selecting an action. For the task of obstacle avoidance with small obstacles, the density of the LiDAR scan is increased to 60 beams. This was after preliminary tests showed that fewer beams led the agents to miss obstacles. To communicate the speed and direction of the vehicle's velocity, two consecutive state vectors are stacked together and used as input into the agent.

Calculate Speed

For consistency between methods, the speed is calculated using the formula for the maximum speed to keep the vehicle inside the friction circle as

$$v = f_{\text{s}} \sqrt{\frac{bg}{\tan(|\delta_{\text{ref}}|)/L}}. \tag{8.1}$$

In the equation, f_{s} is a safety factor, b and g are the coefficients of friction and acceleration due to gravity, respectively, and L is the vehicle's wheelbase. This is the same formula as was used by the link architecture in Chapter 5.

The safety factor and the maximum speed to allow were tuned by using the pure pursuit controller. The pure pursuit planner was set to follow the raceline and the parameters that allowed the vehicle to not crash with the fastest lap time were $f_{\text{s}} = 0.8$ and to limit the speed at $v_{\text{max}} = 7$ m/s. The vehicle parameters used are $b = 0.523$ and $L = 0.33$ m.

Reward Signal

The reward signal that is used to train the agents is designed to encourage the agents to follow the centre line. This incorporates the findings from Chapter 5, that rewarding the agent's actions produces good performance. The reward is a positive constant β_c with a negative gradient slope that is proportional to the difference between the pure pursuit action and the agent's action. The pure pursuit planner is set to follow the centre line.

The positive reward is used because otherwise, the sum of the rewards for more track progress is a larger negative value which results in the agent crashing too soon. It is important that the reward is a measure of the agent's progress through the track.

The reward is written as

$$r = \beta_c - \frac{|\delta_{\text{pp}} - \delta_{\text{nn}}|}{\delta_{\text{max}}} \times \beta_{\text{weight}}, \tag{8.2}$$

with the hyperparameter β_{weight} representing the effect of the steering difference. In the experiments presented here, a β_c of 0.2 and β_{weight} of 0.5 are used.

8.2.2. Serial Planner

One of the aims of the planning architecture is to track the reference path. The serial planner aims to help the DRL agent to do this by including the action that the PP agent would have taken in the state vector. The aim of doing this is that the agent can track the pure pursuit trajectory when no obstacles are present. Figure 8.3 shows the serial architecture with the DRL agent located after the pure pursuit path follower. The serial architecture uses the DRL agent's action directly as the steering command and calculates the speed using the calculate speed function.

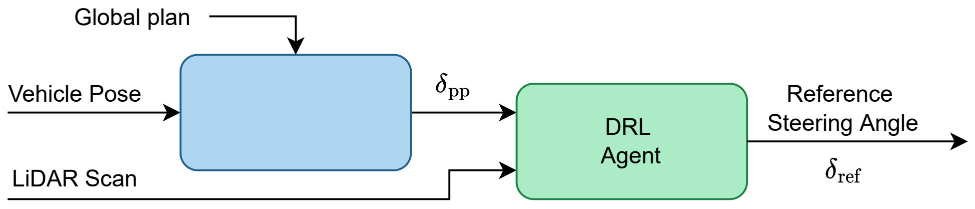


Figure 8.3: The serial architecture using the steering angle from the path follower is as input to the learning agent.

8.2.3. Modification Planner

The modification planner is a hybrid local planner that features a pure pursuit path follower in parallel with a DRL agent. Figure 8.4 shows how the path DRL agent is located in parallel with the path follower such that the agent can modify the path follower. The motivation for the design is for the path follower to track the reference path when there are no upcoming obstacles and for the agent to add the ability to avoid obstacles as they arise.

We train the agent to modify the pure pursuit steering reference δ_{pp} to prevent collisions. The state that the network receives consists of a vector containing the steering angle calculated by the path follower δ_{pp} and the current range finder scan. The network outputs an action quantity used to modify the steering reference calculated by the path follower.

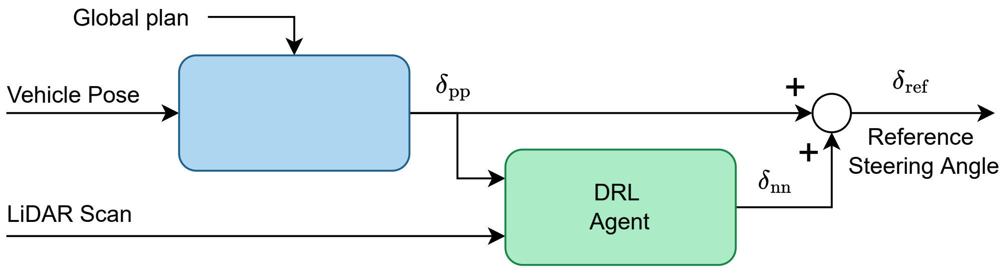


Figure 8.4: The modification planner uses a path follower to track the reference path combined with a DRL agent in parallel that modifies the path follower references.

The path follower (pp) and neural network (nn) steering commands are combined using simple addition as, $\delta_{\text{ref}} = \delta_{\text{pp}} + \delta_{\text{nn}}$. The reference steering value is limited to being within the steering range before calculating the speed.

8.3. Evaluation Methodology

8.3.1. Experiment Description

The focus of the evaluation is to assess the suitability of the architectures to track a reference trajectory while avoiding obstacles. Therefore, a single evaluation is performed with agents that select a steering angle and all use the same method of calculating a speed based on the friction circle. The Columbia map is used since there is enough space to place obstacles while ensuring that a path exists and the wide track is useful for demonstrating the performance.

The experiments consist of training agents in the three architectures (end-to-end, serial and modification) 10 times using different seeds each time. For each training repetition, the agent is tested by running 100 test laps. The averages from the 100 test laps are recorded and then used to represent the training repetition. The repeatability study, in §8.4.3, analyses how the data is spread between training with different seeds and testing with different seeds. Of the 10 experiments, the worst one is removed from each data set because the end-to-end and serial planners had a single run that did not converge.

8.3.2. Obstacle Generation

In this section, an important consideration is the location of the obstacles on the map. Square obstacles with a distance of 0.3 m are randomly spawned on the track. The method of random spawning is to use the centre line points and randomly select one of the points. Then a radius of 1 m is used around each point for where the obstacle can be placed. All the random numbers are generated using the NumPy random number generated, and the experiments are seeded so that they can be reproduced.

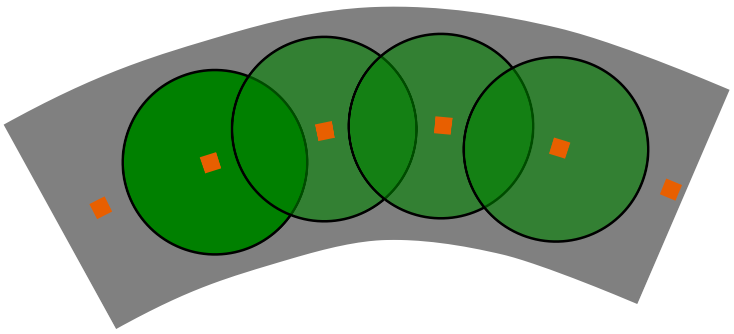


Figure 8.5: Road segment with centre line points in orange and green circles to show where an obstacle may be placed.

Figure 8.5 shows how a random centre line point is used as an anchor with all the possible obstacle placement locations around the centre line considered. The point selected on the map is used for the centre of the box around which the obstacle is created. Additionally, the first and last 10% of the track are obstacle-free zones. This was done to ensure that the vehicle always has a fair chance to avoid obstacles and that the obstacles do not interfere with the start and finish lines. In all the experiments, both training and testing, 8 obstacles are placed on the track.

8.3.3. Random Number Seeding

In the experiments concerning obstacles, the random number seeding is critical. All the experiments are seeded for repeatability. Three sources of random numbers are used and individually seeded. The simulator uses a random number generator for the obstacle placement, which is the most critical generator. The simulator uses a separate random number generator for the noise that is added to the range finders. The LiDAR noise generator is seeded to the same value every time, as per in the original simulator implementation.

The learning algorithms use the random number generator in the PyTorch library to initialise the weights in the neural network. The NumPy built-in random number generator is used to select the samples used to train the networks.

For each training episode, all three of the architectures use the same seeds so that the comparison is against how they perform when training with the same obstacles and the same noise on the LiDAR scans. Arbitrary seeds are calculated using the formula, seed $= 10000 + 10 \times n$, where n is the repetition number of the experiment.

8.4. Results

This section uses simpler maps to analyse the architecture's behaviour without being concerned about the map. Agents are trained for 50,000 steps.

8.4.1. Training

The agents are trained with randomly spawning obstacles in each episode. Figure 8.6 shows the training graphs for the end-to-end, serial and modification planners on the Columbia map.

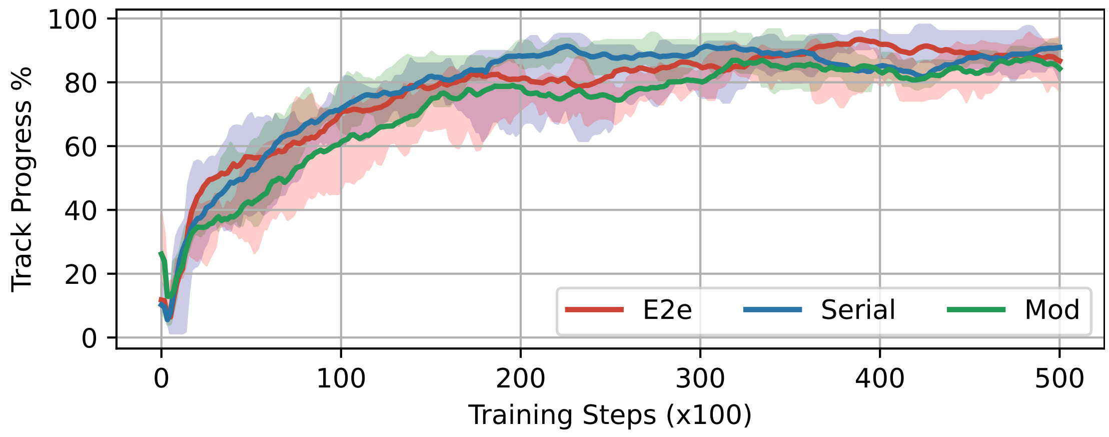


Figure 8.6: Progress during training of the end-to-end (E2e), serial and modification (Mod) planners on the Columbia map.

In Figure 8.6, the progress during the training of the end-to-end, serial and modification planners on the Columbia map is shown. All the planners train to a similar level of success between 80-85%. This means that the planners can all avoid a significantly large number of obstacles, and thus the learning formulation is suitable. However, it also shows that the problem of obstacle avoidance is significantly more difficult than racing on a fixed track.

The graph shows that the modification planner (green line) starts at around 20% when the learning commences. This represents the intended behaviour of not having to start with no knowledge of how to drive but rather improving on what is already possible. This advantage is due to the structure of the modification planner

8.4.2. Performance

Table 8.1 presents the mean numerical results from the planners performance. Since the results vary over training runs, the experiments are all averaged ten times, and the mean and standard deviation are presented. As previously mentioned, the worst result is removed for each agent due to the end-to-end and serial agents not converging for one of the experiments.

The results in Table 8.1 show that the distance travelled by agents trained with the different architectures varies within 0.2 m. The modification planner has the shortest distance of 77.77 m and the serial planner the longest distance of 77.94 m. The end-to-end planner has a significantly larger standard deviation of the distance travelled than the serial and modification planners, suggesting that the path taken is less consistent between different obstacle locations. While all the planners travel a similar distance, the

Metric	**E2e**	**Serial**	**Mod**
Time s	23.62 ± 1.19	24.28 ± 1.17	22.48 ± 1.20
Total Distance m	77.87 ± 1.35	77.94 ± 0.75	77.77 ± 0.81
Total Curvature m^{-1}	67.54 ± 6.62	76.27 ± 9.27	59.13 ± 8.75
Total Deviation m	64.25 ± 6.88	64.06 ± 7.80	56.28 ± 8.26
Avg. Progress %	88.45 ± 5.38	87.27 ± 5.29	87.99 ± 3.54
Completion Rate %	78.89 ± 8.23	76.44 ± 9.15	76.33 ± 6.39

Table 8.1: Mean and standard deviation of the distance, curvature, deviation, average progress and completion rate for tests on Columbia map for the end-to-end (E2e), serial and modification (Mod) architectures for tests with eight obstacles.

modification planner has the fastest time of 22.48 s. This is 1.24 and 1.8 s faster than the end-to-end and serial planners.

The end-to-end planner has a total curvature of 67.54. The serial planner has worse curvature of 76.27, with a larger standard deviation of 9.27. The modification planner has the lowest curvature of 59.13, suggesting that the modification planner takes the smoothest path. The end-to-end and serial planners achieve similar scores of total deviation around 64 m. The modification planner outperforms the other planners by 8 m with a total of 56.28.

All the planners have similar average progresses (88.45, 87.27, 87.99) and completion rates (78.89, 76.44, 76.33). On both of these metrics, the modification planner has smaller standard deviations, suggesting that the results are more repeatable. The serial planner has the largest standard deviation for both rates, indicating that the behaviour of the serial planner is more dependent on the random number seeding.

To study the behaviour of each planner in avoiding obstacles, trajectories from each planner avoiding an obstacle are shown in Figures 8.7.

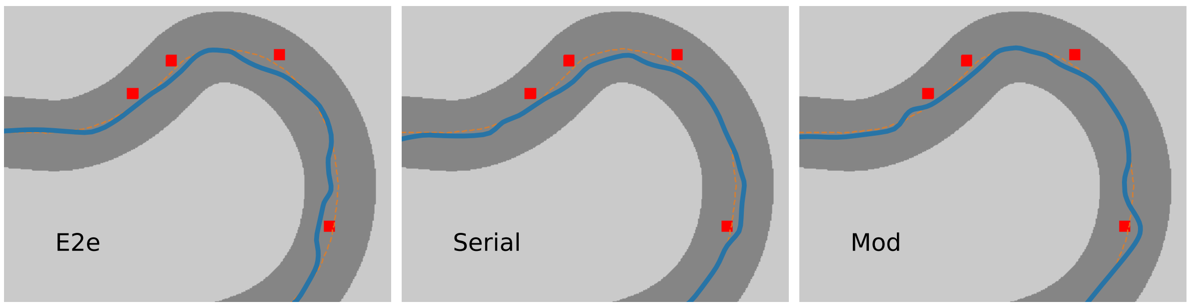


Figure 8.7: The end-to-end (E2e), serial and modification (Mod) planners avoiding obstacles.

Figure 8.7 shows a trajectory from each of the planners where they all successfully avoid the same obstacles. The end-to-end planner (left) and serial planner (middle) take the most wavy path, sometimes turning for no apparent reason. It is also clear that the planners sometimes change the direction of planned avoidance late before missing the obstacle. The modification planner has a smooth path when driving in the sections with no obstacles. Specifically, the modification planner avoids obstacles smoothly before returning to the original path.

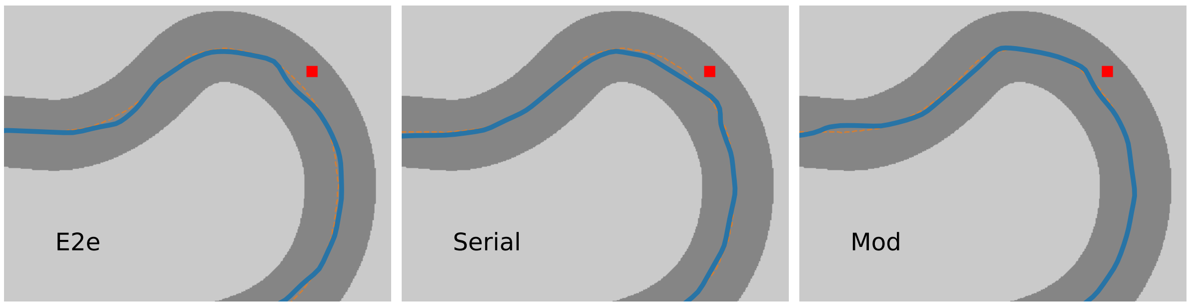


Figure 8.8: The end-to-end (E2e), serial and modification (Mod) planners on a segment with only a single obstacle to show their ability to track the centre line.

Figure 8.8 shows a different test lap where only a single obstacle is placed in that track region. The images show the ability of the end-to-end architecture to follow the centre line (dashed orange line). The end-to-end planner roughly tracks the centre line but is often just inside or outside it. The serial planner takes a smoother path, tracking the centerline more closely. The modification planner tracks the centerline the best, swerving slightly to avoid the obstacle.

The trajectory results in Figure 8.7 and Figure 8.8 confirm what was seen in the quantitative results in Table 8.1. The curvature and deviation from the end-to-end planner are clearly higher than the serial and modification planner. Additionally, the modification planner tracks the centre line closely.

Modification Planner

An in-depth study of the performance of the modification planner is performed to better understand how the modification planner components interact with each other. The actions generated by the agent and the pure pursuit controller work together to formulate obstacle avoidance.

Figure 8.9 shows the output from the modification planner during a test lap. The top graph compares the steering angles from the pure pursuit planner and the neural network. The graph shows that for much of the trajectory, the agent has a small output around zero, indicating that the planner follows the steering angle selected by the pure pursuit planner. At sporadic intervals, the neural network intervenes greatly and dominates the action. The bottom graph shows the steering angle that is implemented on the vehicle (the sum of the two lines in the top graph). The green segments show where the neural network dominates the steering angle by highlighting it in green.

Further investigation into where the neural network changes the pure pursuit action

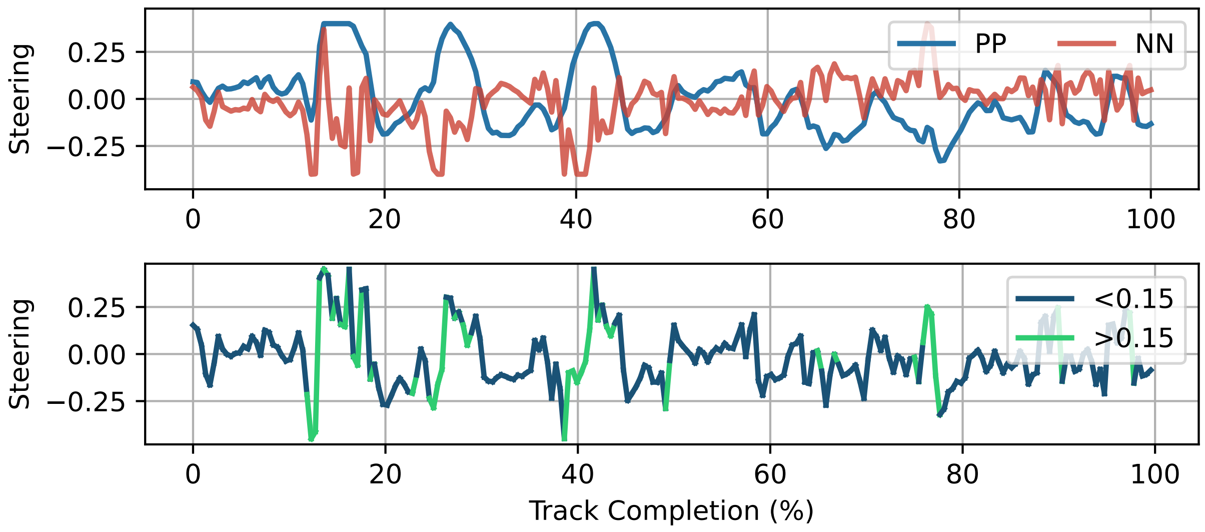


Figure 8.9: The pure pursuit and agent steering angles (top) and the steering angle implemented (bottom) for the modification planner during a test lap of avoiding eight obstacles. The green segments represent the agent dominating the steering angle ($\delta_{\mathrm{nn}} > 0.15$), and the blue segments are where the pure pursuit planner is dominant.

is done by plotting an example trajectory taken by the modification planner in Figure 8.10. The green and blue segments correspond to the graphs in Figure 8.9. The trajectory shows that around obstacles, the planner deviates from following the centre line and avoids the obstacles. For much of the trajectory where no obstacles are present, the planner follows the centre line. This shows that the modification planner performs as designed, in following a reference path while avoiding obstacles.

8.4.3. Repeatability

For every lap, random obstacle locations are generated. Therefore, the effect of the random numbers that are used in the obstacle placement are important. The effects are studied individually in the categories of training and testing.

Training

The training repeatability is measured by training ten agents using the same parameters. Box plots for the metrics of distance travelled, deviation from centre line, average progress and completion rate are shown in Figure 8.11.

In Figure 8.11, box plots are used to show how the data for each metric is distributed. The top left plot shows the distance travelled by each of the agents. The serial planner has a close grouping of distances across different training seeds, followed by the modification planner. The end-to-end planner has a larger difference between the shortest and longest average lap distance.

The total curvature and total deviation measurements show that the modification planner outperforms the serial and end-to-end planners. Interestingly, the end-to-end

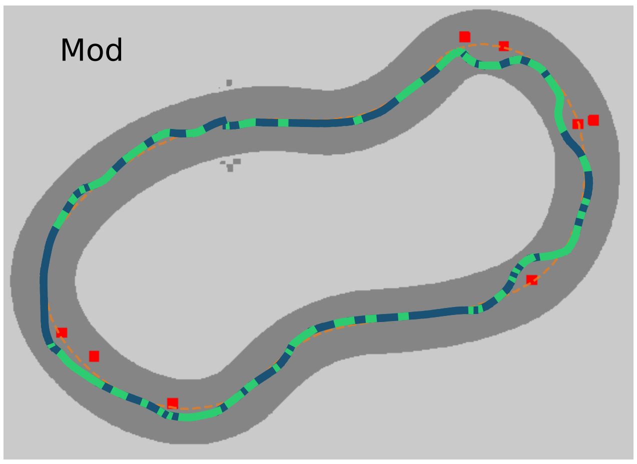


Figure 8.10: Trajectory taken by the modification planner corresponding to the graphs in Figure 8.9. The orange dashed line is the centre line, the red blocks are the obstacles, the green segments are where the neural network steering angle δ_{nn} is greater than 0.15, and the blue lines are the rest.

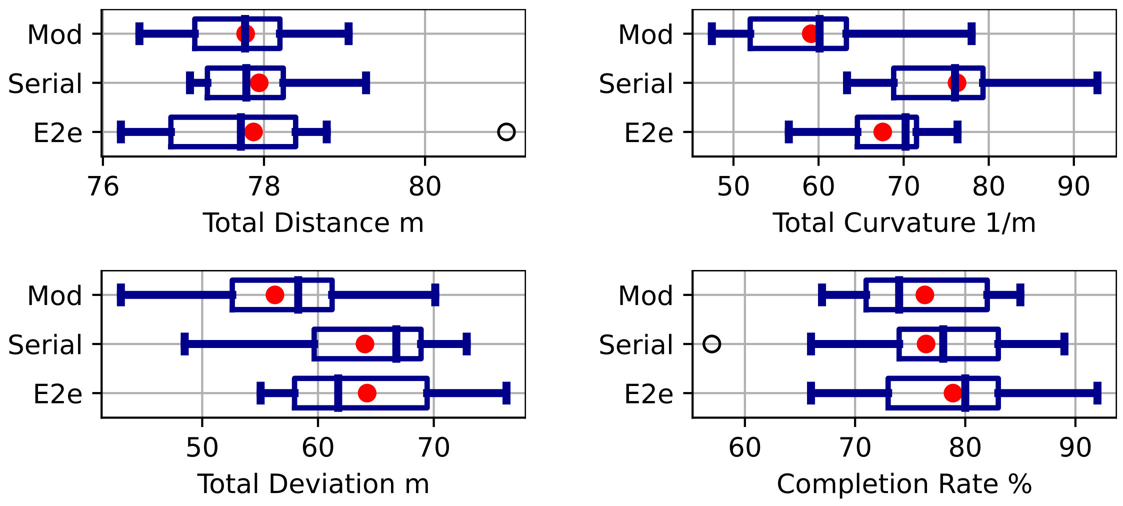


Figure 8.11: Box and whisker plots of the distance travelled, total curvature, total deviation and completion rate with the mean (red dot) for the end-to-end (E2e), serial and modification (Mod) planners.

planner has a small distribution of results for the total curvature. The serial planner has a long upper tail for the total curvature, which means that while for some test runs the planner performed poorly, this is not the general behaviour.

The planners all achieve similar mean completion rates. The mean of the serial planner is lowered, by an outlier (just below 60%). The modification planner has a smaller spread than the end-to-end planner on both the upper and lower ends showing that the behaviour is more constant between runs.

Testing

The repeatability of the testing is now evaluated to understand the effect of the random seed. An agent is trained with each of the architectures presented, and then 100 test laps are run. The distributions of lap time are plotted for each of the architectures. Histograms of 100 test laps for each of the three architectures are shown in Figure 8.12.

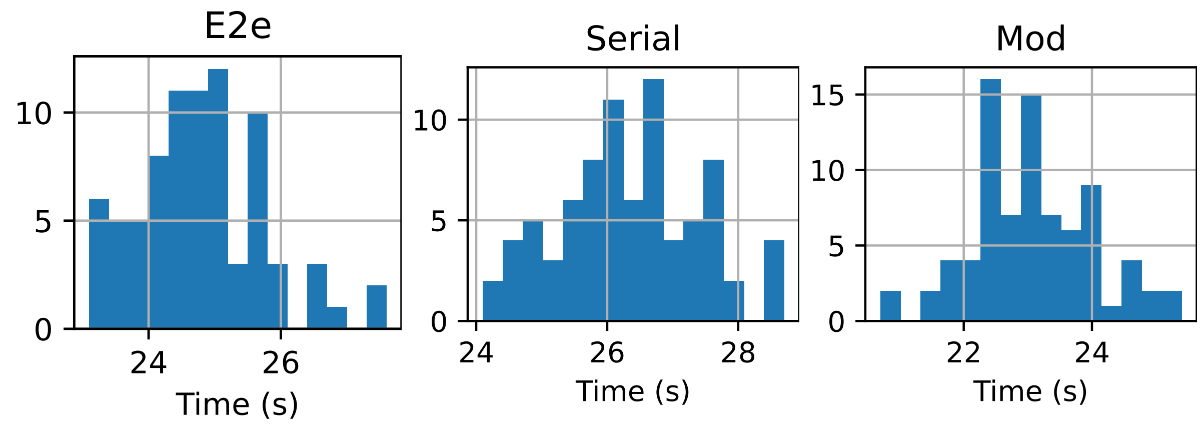


Figure 8.12: Histogram for the lap times achieved by the end-to-end (E2e), serial and modification (Mod) planners on the Columbia map with eight obstacles.

Figure 8.12 shows that the modification planner has the most consistent test result, with over 100 results falling into the central bin. The modification planner achieves repeatable lap times, as shown by the large central columns (note the y-scale). The serial and end-to-end planners have a larger distribution and shorter central column, indicating that the time to complete a lap is more dependent on the location of the obstacles.

8.4.4. Discussion

The results show that the modification planner can learn a policy that produces faster performance than the end-to-end planner. Specifically, the modification planner avoids obstacles more smoothly, which results in the vehicle being able to travel at higher speeds. This shows that the modification planner achieves the behaviour that it was designed for, allowing non-holonomic vehicles to avoid obstacles smoothly at high speeds. The success rate which the planners achieve shows that all the planners perform similarly, achieving around a 76% completion rate.

This problem of guaranteeing vehicle safety is a current research topic and could be solved in future work by exploring different training methods such as imitation learning. Another possible solution is to combine the RL agent with a safety system to stop the vehicle before it crashes.

The advantage of end-to-end solutions is that the agent has a lot of freedom and thus can achieve a high completion rate. This is further improved because the end-to-end agent does not require localisation, which the serial and modification planners do. That is not a big restriction since localisation is possible; however, it remains a requirement.

8.5. Summary

This chapter looked at the problem of obstacle avoidance. DRL methods for autonomous racing were extended to avoid unmapped obstacles placed on the race track. Three different architectures, called the end-to-end, serial and modification planners, were presented. The end-to-end planner used two stacked state vectors of 60 beams as the input to a neural network that generated a steering command, and a calculate speed method to calculate a speed based on the friction circle model. The serial planner included the pure pursuit steering angle in the state vector. The modification architecture uses the DRL agent to modify the pure pursuit action. The evaluation found that the modification architecture selects the fastest, smoothest paths that deviate the least from the centre line. All three of the planners achieved a similar success rate of around 76%, indicating that further work should address the problem of safety in learned obstacle avoidance. Future work should look at methods of guaranteeing safety in obstacle avoidance that ensure that all the laps are completed.

Chapter 9

Conclusion

9.1. Dissertation Summary

This dissertation worked to accelerate the use of DRL methods for F1/10$^{\text{th}}$ autonomous racing by studying current methods with their limitations and presenting novel approaches to improve agent speed selection, bypass the sim-to-real gap and avoid obstacles.

Literature Study: Chapter 2 explored current techniques for classical racing in §2.2, DRL advances in racing games in §2.3 and DRL methods for real-world racing in §2.4. While classical methods have demonstrated excellent high-speed racing performance, learning-based approaches have been severely limited to low speeds. The study on racing games showed that while games have shown the potential performance advantages of DRL for racing, current methods are infeasible due to the lack of safety considerations and the requirement of real-time, accurate, explicit state representation. Limitations in current real-world racing methods were identified as the inadequate evaluation of current learning methods, the poor safety of solutions exacerbated by sim-to-real transfer, and the lack of consideration of the obstacle avoidance problem.

Evaluation of DRL for F1/10$^{\text{th}}$ Racing: Chapter 4 provided an extensive evaluation of current methods of DRL for F1/10$^{\text{th}}$ autonomous racing. The baseline F1/10$^{\text{th}}$ racing formulation presented in §4.3 was evaluated at constant speed in §4.4 where the cross-track and heading reward showed good performance of converging consistently and driving safely, tracking the centre line. The variable speed evaluation in §4.5 demonstrated poor results at high speeds of over 5 m/s, resulting in low completion rates and high slip paths. The key cause of the flawed high-speed performance was identified as poor speed selection, with the agent attempting to select the maximum speed for the entire lap.

High-speed Learning Formulations using Vehicle Models: Chapter 5 addressed the problem of poor speed selection in DRL agents for autonomous racing by using analytical vehicle models to aid the learning. Racing reward signals presented in §5.2 used the optimal trajectory to guide the learning, and the link architecture presented in §5.3 used the agent to select a steering angle and a friction model to calculate a speed reference. The evaluation demonstrated that racing rewards and the link architecture enabled DRL agents to learn an appropriate speed profile of speeding up and slowing down, resulting in trajectories with smaller slip angles. In the comparative evaluation in §5.4, both methods achieved higher success rates than conventional formulations and outperformed similar studies in the literature in terms of lap times [22, 60]. Critically the learning formulations presented enabled agents to use the vehicle's full speed range of up to 8 m/s.

Supervisory Safety System: Chapter 6 addressed the problem of ML safety on physical vehicles by developing a vehicle-model-based supervisor that ensures that only safe actions are implemented on vehicles. The system described in §6.2 uses Viability Theory to generate a list of safe states that are used to keep the vehicle from crashing. The evaluation in §6.5 showed that the system could keep a worst-case-scenario (random) planner from crashing using the vehicle's full speed range while only marginally impacting the performance of a classical planner.

Online Learning using a Supervisor: Chapter 7 uses the SSS that was developed in Chapter 6 to train agents online without them ever crashing during the training process. The modifications to the learning formulation were explained in §7.2, evaluated in simulation in §7.3 and validated at constant speed on a physical vehicle in §7.4. The results from the variable-speed evaluation showed that the system could train an agent to select speed and steering actions, without ever crashing, in just 10,000 steps, demonstrating a 10× improvement in sample efficiency. The comparison of reward signals showed that the PPPS reward achieves the fastest lap times, followed by the velocity reward. The investigation into the maximum speed showed that the agents trained with the velocity reward achieve faster lap times with increased maximum speeds, however, the average progress decreases significantly for maximum speeds higher than 6 m/s. The agents trained online are more conservative than conventionally trained agents, selecting lower, more moderate speeds and thus achieving slower lap times. The physical validation demonstrated that the method could train an agent to drive around a track on a physical vehicle, thus bypassing the sim-to-real problem.

Avoiding Un-mapped Obstacles: Chapter 8 approached the problem of racing while avoiding unmapped obstacles. In §8.2 three planners were presented; the end-to-end planner in §8.2.1, the serial planner in §8.2.2 and the modification planner in §8.2.3. The evaluation in §8.4 showed that all the planners avoided over 75% of the obstacles. The modification architecture was shown to select the fastest, smoothest paths that deviate the least from the centre line. This result demonstrates that hybrid architectures are able to add flexibility to classical approaches, and performance to learning approaches.

9.2. Significance of Contribution

While autonomous robotics problems have classically been solved using optimisation and control systems approaches, there has been a rise in using learning components in the planning pipeline. DRL agents have major advantages for racing systems; specifically, they do not require a track map or real-time localisation, are flexible to different tracks, and show promise for head-to-head racing. Using raw LiDAR scans and not requiring explicit state representation is a general advantage in robotics since state estimation is computationally expensive.

High-speed Learning Formulations

The first contribution of this work is to develop learning formulations that enable agents to use LiDAR scans as the only input to select appropriate speed and steering actions for high-performance racing. This accomplishment enables DRL agents to compete with classical solutions in terms of feasibility and performance, which is essential for learning to be a competitive strategy for high-performance tasks. The aim of racing as a testbed for robotics is the development of high-performance algorithms that operate autonomous systems at the limits of handling. For as long as learning agents cannot use the vehicle's full speed range, the aim of racing is defeated, and further work is required. Therefore, this work has enabled DRL agents to be a competitor for high-performance F1/10$^{\text{th}}$ racing.

A key in the problem identification and design of the solutions was incorporating knowledge from classical methods. Evaluation metrics found in the trajectory optimisation literature of plotting the speed profiles, curvature, and slip angle, indicated where the problems with the current learning formulation were. Racing lines, built using vehicle models, demonstrated their use in aiding the learning, indicating that, where possible, optimal trajectories should be used to improve agent performance. The key solution methodology combines the knowledge available through analytical models with black-box learning components to create flexible, high-performance systems.

Safe Online Training

The second contribution of this work is developing a safe learning framework for autonomous racing that can train agents to race without them ever crashing. Safety is a persistent problem in ML approaches to robotics, exacerbated by the sim-to-real gap. In response, supervisory systems enable the advancement of ML on real-world robotic systems due to removing the concern of failure (crashing). The supervisory safety system presented in this work achieves what previous control methods could not by ensuring the safety of a non-linear vehicle on a racing track at high speeds. The SSS is superior to naive approaches of reversing [60] since it can continue driving, is superior to human intervention [119] since it operates autonomously, and is superior to reachability methods [87] since it can guarantee recursive feasibility and does not require online computation.

The safety system was applied to the problem of training a DRL agent online, with the vehicle never crashing. This is a significant advancement in DRL for real-world robotics since training on a physical vehicle allows for the sim-to-real gap to be bypassed. The agent can be trained and tested on the same platform, meaning the safety can be evaluated before removing the supervisor. An additional benefit to this method is a significant increase in sample efficiency.

Obstacle Avoidance

The third contribution of this work is the development and evaluation of hybrid architectures for obstacle avoidance. Being the first study on static obstacle avoidance in F1/10th racing, this work expands the problem from previous formulations. Obstacle avoidance is a key sub-problem on the road to developing solutions for head-to-head racing. The methods presented here lay a basis for approaching the problem of high-performance flexible systems and should be expanded to consider dynamic obstacles and opposing vehicles.

The key question asked in this work is how to combine classical and learning components to enable high-performance systems that are flexible to unmodelled disturbances. The world has many uncertainties that are not mapped, and DRL offers the unique advantage of being highly flexible to new situations. This work demonstrated how hybrid systems could harness the advantages of classical control and machine learning by developing and evaluating different architectures.

9.3. Future Work

Based on the findings of this dissertation, the following recommendations for future work are proposed:

Online Verification: A critical limitation of the supervisory system in this work is that localisation is required to ensure safety. Future work should address this by using the dense LiDAR scan to verify the safety of actions. This work would enable verifiably safe systems that are not dependent on a map of the environment, and thus could be used during training and once retained once training is complete.

Physical Experimentation: A detailed study of how the methods presented transfer from simulation to reality should be performed. The literature study noted the lack of physical experiments, which was supposed to be due to safety concerns. Now that a method for overcoming the safety concern has been developed, algorithms can be evaluated on physical platforms without the risk of crashing. Firstly, the safe online learning formulations from Chapter 7 should be validated on a physical vehicle, and then the high-speed formulations from Chapter 5 should be evaluated.

Head-to-head Racing: The work in obstacle avoidance should be expanded to high-speed head-to-head racing. This will involve developing custom learning formulations, incorporating strategy and safety. DRL is an ideal method for head-to-head racing since it can learn large general state spaces, doesn't require localisation and can react quickly to raw inputs. A promising direction is to develop a head-to-head racing safety supervisor that could be used to train DRL agents to race online while racing against opponents.

Expansion of Domains: The design methodology of combining classical and learning components for high-performance, flexible systems should be expanded to other domains. Supervisory learning architectures for other safety-critical domains, such as drones, could enable the use of learning agents to perform tasks that are currently not feasible. Using analytical system models could improve the use of DRL agents in the control of any process, from navigation to autonomous control systems.

Bibliography

[1] M. Ben-Ari and F. Mondada, "Robots and their applications," in *Elements of robotics.* Springer, 2018, pp. 1–20.

[2] A. Liniger, "Path Planning and Control for Autonomous Racing," Ph.D. dissertation, ETH Zurich, 2018.

[3] J. Betz, H. Zheng, A. Liniger, U. Rosolia, P. Karle, M. Behl, V. Krovi, and R. Mangharam, "Autonomous vehicles on the edge: A survey on autonomous vehicle racing," *IEEE Open Journal of Intelligent Transportation Systems*, 2022.

[4] M. O'Kelly, H. Zheng, D. Karthik, and R. Mangharam, "F1tenth: An open-source evaluation environment for continuous control and reinforcement learning," *Proceedings of Machine Learning Research*, vol. 123, 2020.

[5] B. Balaji, S. Mallya, S. Genc, S. Gupta, L. Dirac, V. Khare, G. Roy, T. Sun, Y. Tao, B. Townsend *et al.*, "Deepracer: Autonomous racing platform for experimentation with sim2real reinforcement learning," in *2020 IEEE International Conference on Robotics and Automation (ICRA).* IEEE, 2020, pp. 2746–2754.

[6] P. Cai, H. Wang, H. Huang, Y. Liu, and M. Liu, "Vision-based autonomous car racing using deep imitative reinforcement learning," *IEEE Robotics and Automation Letters*, vol. 6, no. 4, pp. 7262–7269, 2021.

[7] R. Bautista-Montesano, R. Galluzzi, V. Gomez-Aladro, R. Bustamante-Bello, and R. Ramirez-Mendoza, "Autonomous vehicles as a development platform: From high school to faculty," in *2021 IEEE Global Engineering Education Conference (EDUCON).* IEEE, 2021, pp. 43–49.

[8] V. S. Babu and M. Behl, "F1tenth.dev-An Open-source ROS based F1/10 Autonomous Racing Simulator," in *IEEE International Conference on Automation Science and Engineering*, vol. 2020-Augus. IEEE Computer Society, 8 2020, pp. 1614–1620.

[9] N. Hamilton, P. Musau, D. M. Lopez, and T. T. Johnson, "Zero-shot policy transfer in autonomous racing: Reinforcement learning vs imitation learning," in *2022 IEEE International Conference on Assured Autonomy (ICAA).* IEEE, 2022, pp. 11–20.

[10] C. H. Walsh and S. Karaman, “Cddt: Fast approximate 2d ray casting for accelerated localization,” in *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2018, pp. 3677–3684.

[11] A. Liniger and J. Lygeros, “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.

[12] J. Betz, T. Betz, F. Fent, M. Geisslinger, A. Heilmeier, L. Hermansdorfer, T. Herrmann, S. Huch, P. Karle, M. Lienkamp *et al.*, “Tum autonomous motorsport: An autonomous racing software for the indy autonomous challenge,” *arXiv preprint arXiv:2205.15979*, 2022.

[13] A. Heilmeier, A. Wischnewski, L. Hermansdorfer, J. Betz, M. Lienkamp, and B. Lohmann, “Minimum curvature trajectory planning and control for an autonomous race car,” *Vehicle System Dynamics*, vol. 58, no. 10, pp. 1497–1527, 10 2020. [Online]. Available: https://www.tandfonline.com/doi/full/10.1080/00423114.2019.1631455

[14] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, “Playing atari with deep reinforcement learning,” *arXiv preprint arXiv:1312.5602*, 2013.

[15] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, “Continuous control with deep reinforcement learning,” *arXiv preprint arXiv:1509.02971*, 2015.

[16] J. Schrittwieser, I. Antonoglou, T. Hubert, K. Simonyan, L. Sifre, S. Schmitt, A. Guez, E. Lockhart, D. Hassabis, T. Graepel *et al.*, “Mastering atari, go, chess and shogi by planning with a learned model,” *Nature*, vol. 588, no. 7839, pp. 604–609, 2020.

[17] O. Vinyals, I. Babuschkin, W. M. Czarnecki, M. Mathieu, A. Dudzik, J. Chung, D. H. Choi, R. Powell, T. Ewalds, P. Georgiev *et al.*, “Grandmaster level in starcraft ii using multi-agent reinforcement learning,” *Nature*, vol. 575, no. 7782, pp. 350–354, 2019.

[18] C. Berner, G. Brockman, B. Chan, V. Cheung, P. Dkebiak, C. Dennison, D. Farhi, Q. Fischer, S. Hashme, C. Hesse *et al.*, “Dota 2 with large scale deep reinforcement learning,” *arXiv preprint arXiv:1912.06680*, 2019.

[19] F. Fuchs, Y. Song, E. Kaufmann, D. Scaramuzza, and P. Durr, “Super-Human Performance in Gran Turismo Sport Using Deep Reinforcement Learning,” *IEEE Robotics and Automation Letters*, vol. 6, no. 3, pp. 4257–4264, 2021.

[20] P. R. Wurman, S. Barrett, K. Kawamoto, J. MacGlashan, K. Subramanian, T. J. Walsh, R. Capobianco, A. Devlic, F. Eckert, F. Fuchs, L. Gilpin, P. Khandelwal, V. Kompella, H. C. Lin, P. MacAlpine, D. Oller, T. Seno, C. Sherstan, M. D. Thomure, H. Aghabozorgi, L. Barrett, R. Douglas, D. Whitehead, P. Dürr, P. Stone, M. Spranger, and H. Kitano, "Outracing champion Gran Turismo drivers with deep reinforcement learning," *Nature 2022 602:7896*, vol. 602, no. 7896, pp. 223–228, 2 2022. [Online]. Available: https://www.nature.com/articles/s41586-021-04357-7

[21] J. Ibarz, J. Tan, C. Finn, M. Kalakrishnan, P. Pastor, and S. Levine, "How to train your robot with deep reinforcement learning: lessons we have learned," *International Journal of Robotics Research*, vol. 40, no. 4-5, pp. 698–721, 2021.

[22] A. Brunnbauer, L. Berducci, A. Brandstatter, M. Lechner, R. Hasani, D. Rus, and R. Grosu, "Latent Imagination Facilitates Zero-Shot Transfer in Autonomous Racing," *2022 International Conference on Robotics and Automation (ICRA)*, pp. 7513–7520, 5 2022. [Online]. Available: https://ieeexplore.ieee.org/document/9811650/

[23] X. Sun, M. Zhou, Z. Zhuang, S. Yang, J. Betz, and R. Mangharam, "A benchmark comparison of imitation learning-based control policies for autonomous racing," *arXiv preprint arXiv:2209.15073*, 2022.

[24] W. Zhao, J. P. Queralta, and T. Westerlund, "Sim-to-Real Transfer in Deep Reinforcement Learning for Robotics: A Survey," *2020 IEEE Symposium Series on Computational Intelligence, SSCI 2020*, pp. 737–744, 2020.

[25] B. Evans, H. A. Engelbrecht, and H. W. Jordaan, "Reward signal design for autonomous racing," in *2021 20th International Conference on Advanced Robotics (ICAR)*. IEEE, 2021, pp. 455–460.

[26] ——, "Learning the subsystem of local planning for autonomous racing," in *2021 20th International Conference on Advanced Robotics (ICAR)*. IEEE, 2021, pp. 601–606.

[27] S. Pendleton, H. Andersen, X. Du, X. Shen, M. Meghjani, Y. Eng, D. Rus, and M. Ang, "Perception, Planning, Control, and Coordination for Autonomous Vehicles," *Machines*, vol. 5, no. 1, p. 6, 2 2017. [Online]. Available: http://www.mdpi.com/2075-1702/5/1/6

[28] A. Liniger, A. Domahidi, and M. Morari, "Optimization-based autonomous racing of 1: 43 scale rc cars," *Optimal Control Applications and Methods*, vol. 36, no. 5, pp. 628–647, 2015.

[29] A. Wischnewski, M. Geisslinger, J. Betz, T. Betz, F. Fent, A. Heilmeier, L. Hermansdorfer, T. Herrmann, S. Huch, P. Karle *et al.*, "Indy autonomous challenge-autonomous race cars at the handling limits," in *12th International Munich Chassis Symposium 2021.* Springer, 2022, pp. 163–182.

[30] J. Kabzan, M. I. Valls, V. J. Reijgwart, H. F. Hendrikx, C. Ehmke, M. Prajapat, A. Bühler, N. Gosala, M. Gupta, R. Sivanesan, A. Dhall, E. Chisari, N. Karnchanachari, S. Brits, M. Dangel, I. Sa, R. Dubé, A. Gawel, M. Pfeiffer, A. Liniger, J. Lygeros, and R. Siegwart, "AMZ Driverless: The full autonomous racing system," *Journal of Field Robotics*, vol. 37, no. 7, pp. 1267–1294, 2020.

[31] A. Liniger, A. Domahidi, and M. Morari, "Optimization-based autonomous racing of 1:43 scale RC cars," *Optimal Control Applications and Methods*, vol. 36, no. 5, pp. 628–647, 9 2015. [Online]. Available: https://onlinelibrary.wiley.com/doi/full/10.1002/oca.2123https://onlinelibrary.wiley.com/doi/abs/10.1002/oca.2123https://onlinelibrary.wiley.com/doi/10.1002/oca.2123

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

[33] V. Sukhil and M. Behl, "Adaptive lookahead pure-pursuit for autonomous racing," *arXiv preprint arXiv:2111.08873*, 2021.

[34] M. O'Kelly, H. Zheng, A. Jain, J. Auckley, K. Luong, and R. Mangharam, "Tunercar: A superoptimization toolchain for autonomous racing," in *2020 IEEE International Conference on Robotics and Automation (ICRA).* IEEE, 2020, pp. 5356–5362.

[35] J. Betz, A. Wischnewski, A. Heilmeier, F. Nobis, L. Hermansdorfer, T. Stahl, T. Herrmann, and M. Lienkamp, "A software architecture for the dynamic path planning of an autonomous racecar at the limits of handling," *2019 8th IEEE International Conference on Connected Vehicles and Expo, ICCVE 2019 - Proceedings*, 11 2019.

[36] R. C. Coulter, "Implementation of the pure pursuit path tracking algorithm," Carnegie-Mellon UNIV Pittsburgh PA Robotics INST, Tech. Rep., 1992.

[37] J. Becker, N. Imholz, L. Schwarzenbach, E. Ghignone, N. Baumann, and M. Magno, "Model-and acceleration-based pursuit controller for high-performance autonomous racing," *arXiv preprint arXiv:2209.04346*, 2022.

[38] A. Tătulea-Codrean, T. Mariani, and S. Engell, "Design and simulation of a machine-learning and model predictive control approach to autonomous race driving for the f1/10 platform," *IFAC-PapersOnLine*, vol. 53, no. 2, pp. 6031–6036, 2020.

[39] M. Luiza Costa Vianna, E. Goubault, and S. Putot, "Neural Network Based Model Predictive Control for an Autonomous Vehicle," *arXiv e-prints*, p. arXiv:2107.14573, Jul. 2021.

[40] J. Jönsson and F. Stenbäck, "Monte-carlo tree search in continuous action spaces for autonomous racing: F1-tenth," 2020.

[41] V. Cataffo, G. Silano, L. Iannelli, V. Puig, and L. Glielmo, "A nonlinear model predictive control strategy for autonomous racing of scale vehicles," 2022.

[42] A. Jain, M. O'Kelly, P. Chaudhari, and M. Morari, "Bayesrace: Learning to race autonomously using prior experience," in *Conference on Robot Learning.* PMLR, 2021, pp. 1918–1929.

[43] E. Ghignone, N. Baumann, M. Boss, and M. Magno, "Tc-driver: Trajectory conditioned driving for robust autonomous racing–a reinforcement learning approach," *arXiv preprint arXiv:2205.09370*, 2022.

[44] R. Wang, "Data-driven system identification and optimal control framework for grand-prix style autonomous racing," Ph.D. dissertation, Clemson University, 2021.

[45] R. Wang, Y. Han, and U. Vaidya, "Deep koopman data-driven control framework for autonomous racing," in *Proc. Int. Conf. Robot. Autom.(ICRA) Workshop Opportunities Challenges Auton. Racing*, 2021, pp. 1–6.

[46] T. Zhang, G. Kahn, S. Levine, and P. Abbeel, "Learning deep control policies for autonomous aerial vehicles with mpc-guided policy search," in *2016 IEEE international conference on robotics and automation (ICRA).* IEEE, 2016, pp. 528–535.

[47] J. Hwangbo, I. Sa, R. Siegwart, and M. Hutter, "Control of a quadrotor with reinforcement learning," *IEEE Robotics and Automation Letters*, vol. 2, no. 4, pp. 2096–2103, 2017.

[48] N. Rudin, D. Hoeller, P. Reist, and M. Hutter, "Learning to walk in minutes using massively parallel deep reinforcement learning," in *Conference on Robot Learning.* PMLR, 2022, pp. 91–100.

[49] L. Tai, G. Paolo, and M. Liu, "Virtual-to-real deep reinforcement learning: Continuous control of mobile robots for mapless navigation," in *IEEE International Conference on Intelligent Robots and Systems*, vol. 2017-Septe. Institute of Electrical and Electronics Engineers Inc., 12 2017, pp. 31–36.

[50] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction.* MIT press, 2018.

[51] K. Arulkumaran, M. P. Deisenroth, M. Brundage, and A. A. Bharath, “Deep reinforcement learning: A brief survey,” *IEEE Signal Processing Magazine*, vol. 34, no. 6, pp. 26–38, 2017.

[52] M. Jaritz, R. De Charette, M. Toromanoff, E. Perot, and F. Nashashibi, “End-to-End Race Driving with Deep Reinforcement Learning,” *Proceedings - IEEE International Conference on Robotics and Automation*, pp. 2070–2075, 2018.

[53] E. Perot, M. Jaritz, M. Toromanoff, and R. De Charette, “End-to-end driving in a realistic racing game with deep reinforcement learning,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, 2017, pp. 3–4.

[54] P. Cai, X. Mei, L. Tai, Y. Sun, and M. Liu, “High-Speed Autonomous Drifting with Deep Reinforcement Learning,” *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 1247–1254, 4 2020.

[55] Y. Song, H. Lin, E. Kaufmann, P. Dürr, and D. Scaramuzza, “Autonomous overtaking in gran turismo sport using curriculum reinforcement learning,” in *2021 IEEE international conference on robotics and automation (ICRA)*. IEEE, 2021, pp. 9403–9409.

[56] J. Herman, J. Francis, S. Ganju, B. Chen, A. Koul, A. Gupta, A. Skabelkin, I. Zhukov, M. Kumskoy, and E. Nyberg, “Learn-to-race: A multimodal control environment for autonomous racing,” in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2021, pp. 9793–9802.

[57] Y.-J. R. Chu, T.-H. Wei, J.-B. Huang, Y.-H. Chen, I. Wu *et al.*, “Sim-to-real transfer for miniature autonomous car racing,” *arXiv preprint arXiv:2011.05617*, 2020.

[58] J. Tobin, R. Fong, A. Ray, J. Schneider, W. Zaremba, and P. Abbeel, “Domain randomization for transferring deep neural networks from simulation to the real world,” in *2017 IEEE/RSJ international conference on intelligent robots and systems (IROS)*. IEEE, 2017, pp. 23–30.

[59] V. Sezer and M. Gokasan, “A novel obstacle avoidance algorithm:“follow the gap method”,” *Robotics and Autonomous Systems*, vol. 60, no. 9, pp. 1123–1134, 2012.

[60] M. Bosello, R. Tse, and G. Pau, “Train in austria, race in montecarlo: Generalized rl for cross-track f1 tenth lidar-based races,” in *2022 IEEE 19th Annual Consumer Communications & Networking Conference (CCNC)*. IEEE, 2022, pp. 290–298.

[61] K. Guckiran and B. Bolat, “Autonomous Car Racing in Simulation Environment Using Deep Reinforcement Learning,” *Proceedings - 2019 Innovations in Intelligent Systems and Applications Conference, ASYU 2019*, 2019.

[62] R. Ivanov, T. J. Carpenter, J. Weimer, R. Alur, G. J. Pappas, and I. Lee, "Case study: verifying the safety of an autonomous racing car with a neural network controller," in *Proceedings of the 23rd International Conference on Hybrid Systems: Computation and Control*, 2020, pp. 1–7.

[63] D. Hafner, T. Lillicrap, J. Ba, and M. Norouzi, "Dream to control: Learning behaviors by latent imagination," in *International Conference on Learning Representations*, 2019.

[64] A. Hussein, M. M. Gaber, E. Elyan, and C. Jayne, "Imitation learning: A survey of learning methods," *ACM Computing Surveys (CSUR)*, vol. 50, no. 2, pp. 1–35, 2017.

[65] S. Ross, G. Gordon, and D. Bagnell, "A reduction of imitation learning and structured prediction to no-regret online learning," in *Proceedings of the fourteenth international conference on artificial intelligence and statistics.* JMLR Workshop and Conference Proceedings, 2011, pp. 627–635.

[66] M. Kelly, C. Sidrane, K. Driggs-Campbell, and M. J. Kochenderfer, "Hg-dagger: Interactive imitation learning with human experts," in *2019 International Conference on Robotics and Automation (ICRA).* IEEE, 2019, pp. 8077–8083.

[67] J. Spencer, S. Choudhury, M. Barnes, M. Schmittle, M. Chiang, P. Ramadge, and S. Srinivasa, "Expert intervention learning," *Autonomous Robots*, vol. 46, no. 1, pp. 99–113, 2022.

[68] C. Sammut and G. I. Webb, *Encyclopedia of machine learning.* Springer Science & Business Media, 2011.

[69] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[70] E. Chisari, A. Liniger, A. Rupenyan, L. Van Gool, and J. Lygeros, "Learning from simulation, racing in reality," in *2021 IEEE International Conference on Robotics and Automation (ICRA).* IEEE, 2021, pp. 8046–8052.

[71] R. Zhang, J. Hou, G. Chen, Z. Li, J. Chen, and A. Knoll, "Residual policy learning facilitates efficient model-free autonomous racing," *IEEE Robotics and Automation Letters*, vol. 7, no. 4, pp. 11 625–11 632, 2022.

[72] T. Weiss, V. S. Babu, and M. Behl, "Bezier curve based end-to-end trajectory synthesis for agile autonomous driving," in *NeurIPS 2020 Machine Learning for Autonomous Driving Workshop*, 2020.

[73] T. Weiss and M. Behl, "Deepracing: a framework for autonomous racing," in *2020 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2020, pp. 1163–1168.

[74] P. Drews, G. Williams, B. Goldfain, E. A. Theodorou, and J. M. Rehg, "Vision-based high-speed driving with a deep dynamic observer," *IEEE Robotics and Automation Letters*, vol. 4, no. 2, pp. 1564–1571, 2019.

[75] A. Brunnbauer, L. Berducci, A. Brandstätter, M. Lechner, R. Hasani, D. Rus, and R. Grosu, "Model-based versus model-free deep reinforcement learning for autonomous racing cars," *arXiv preprint arXiv:2103.04909*, 2021.

[76] P. R. Wurman, P. Stone, and M. Spranger, "Challenges and opportunities of applying reinforcement learning to autonomous racing," *IEEE Intelligent Systems*, vol. 37, no. 3, pp. 20–23, 2022.

[77] A. Asperti and M. Del Brutto, "Microracer: a didactic environment for deep reinforcement learning," *arXiv preprint arXiv:2203.10494*, 2022.

[78] M. Vitelli and A. Nayebi, "CARMA: A Deep Reinforcement Learning Approach to Autonomous Driving," Stanford University, Tech. Rep., 2016.

[79] R. Ivanov, J. Weimer, R. Alur, G. J. Pappas, and I. Lee, "Verisig: verifying safety properties of hybrid systems with neural network controllers," in *Proceedings of the 22nd ACM International Conference on Hybrid Systems: Computation and Control*, 2019, pp. 169–178.

[80] B. Thananjeyan, A. Balakrishna, S. Nair, M. Luo, K. Srinivasan, M. Hwang, J. E. Gonzalez, J. Ibarz, C. Finn, and K. Goldberg, "Recovery RL: Safe Reinforcement Learning with Learned Recovery Zones," *IEEE Robotics and Automation Letters*, vol. 6, no. 3, pp. 4915–4922, 2021.

[81] L. Zhang, R. Zhang, T. Wu, R. Weng, M. Han, and Y. Zhao, "Safe reinforcement learning with stability guarantee for motion planning of autonomous vehicles," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 32, no. 12, pp. 5435–5444, 2021.

[82] T. Stahl, M. Eicher, J. Betz, and F. Diermeyer, "Online verification concept for autonomous vehicles–illustrative study for a trajectory planning module," in *2020 IEEE 23rd International Conference on Intelligent Transportation Systems (ITSC)*. IEEE, 2020, pp. 1–7.

[83] Z. Li, U. Kalabić, and T. Chu, "Safe Reinforcement Learning: Learning with Supervision Using a Constraint-Admissible Set," *Proceedings of the American Control Conference*, vol. 2018-June, pp. 6390–6395, 2018.

[84] G. Dalal, K. Dvijotham, M. Vecerik, T. Hester, C. Paduraru, and Y. Tassa, "Safe exploration in continuous action spaces," *arXiv preprint arXiv:1801.08757*, 2018.

[85] R. Cheng, G. Orosz, R. M. Murray, and J. W. Burdick, "End-to-end safe reinforcement learning through barrier functions for safety-critical continuous control tasks," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, no. 01, 2019, pp. 3387–3395.

[86] J. Niu, Y. Hu, B. Jin, Y. Han, and X. Li, "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.

[87] P. Musau, N. Hamilton, D. M. Lopez, P. Robinette, and T. T. Johnson, "On using real-time reachability for the safety assurance of machine learning controllers," in *2022 IEEE International Conference on Assured Autonomy (ICAA)*. IEEE, 2022, pp. 1–10.

[88] J. Nilsson, J. Fredriksson, and A. C. Ödblom, "Verification of collision avoidance systems using reachability analysis," *IFAC Proceedings Volumes*, vol. 47, no. 3, pp. 10 676–10 681, 2014.

[89] D. Seto, B. Krogh, L. Sha, and A. Chutinan, "The simplex architecture for safe on-line control system upgrades," *Proceedings of the American Control Conference*, vol. 6, pp. 3504–3508, 1998.

[90] A. J. Taylor, A. Singletary, Y. Yue, Y. Edu, A. D. Ames, A. Bayen, A. Jadbabaie, G. J. Pappas, P. Parrilo, B. Recht, C. Tomlin, and M. Zeilinger, "Learning for Safety-Critical Control with Control Barrier Functions," *Proceedings of Machine Learning Research*, vol. 120, pp. 1–10, 2020.

[91] M. Alshiekh, R. Bloem, R. Ehlers, B. Könighofer, S. Niekum, and U. Topcu, "Safe Reinforcement Learning via Shielding," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 32, no. 1, pp. 2669–2678, 4 2018. [Online]. Available: https://ojs.aaai.org/index.php/AAAI/article/view/11797

[92] J. F. Fisac, A. K. Akametalu, M. N. Zeilinger, S. Kaynama, J. Gillula, and C. J. Tomlin, "A General Safety Framework for Learning-Based Control in Uncertain Robotic Systems," *IEEE Transactions on Automatic Control*, vol. 64, no. 7, pp. 2737–2752, 2019.

[93] Y. Li, Y. Edu, N. Li, H. E. Tseng, A. Girard, and D. Com, “Safe Reinforcement Learning Using Robust Action Governor Dimitar Filev,” *Proceedings of Machine Learning Research*, vol. 144, pp. 1–12, 2021.

[94] A. D. Ames, X. Xu, J. W. Grizzle, and P. Tabuada, “Control Barrier Function Based Quadratic Programs for Safety Critical Systems,” *IEEE Transactions on Automatic Control*, vol. 62, no. 8, pp. 3861–3876, 8 2017.

[95] S. Bak, J. Betz, A. Chawla, H. Zheng, and R. Mangharam, “Stress testing autonomous racing overtake maneuvers with rrt,” in *2022 IEEE Intelligent Vehicles Symposium (IV)*. IEEE, 2022, pp. 806–812.

[96] H. Zheng, Z. Zhuang, J. Betz, and R. Mangharam, “Game-theoretic objective space planning,” *arXiv preprint arXiv:2209.07758*, 2022.

[97] P. Yue, J. Xin, H. Zhao, D. Liu, M. Shan, and J. Zhang, “Experimental research on deep reinforcement learning in autonomous navigation of mobile robot,” in *2019 14th IEEE Conference on Industrial Electronics and Applications (ICIEA)*. IEEE, 2019, pp. 1612–1616.

[98] F. de Villiers and W. Brink, “Learning fine-grained control for mapless navigation,” in *2020 International SAUPEC/RobMech/PRASA Conference*. IEEE, 2020, pp. 1–6.

[99] X. Xiao, B. Liu, G. Warnell, and P. Stone, “Motion planning and control for mobile robot navigation using machine learning: a survey,” *Autonomous Robots*, pp. 1–29, 2022.

[100] T. Glasmachers, “Limits of end-to-end learning,” in *Asian conference on machine learning*. PMLR, 2017, pp. 17–32.

[101] L. Kästner, T. Buiyan, L. Jiao, T. A. Le, X. Zhao, Z. Shen, and J. Lambrecht, “Arena-rosnav: Towards deployment of deep-reinforcement-learning-based obstacle avoidance into conventional autonomous navigation systems,” in *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2021, pp. 6456–6463.

[102] W. Gao, D. Hsu, W. S. Lee, S. Shen, and K. Subramanian, “Intention-net: Integrating planning and deep learning for goal-directed autonomous navigation,” in *Conference on robot learning*. PMLR, 2017, pp. 185–194.

[103] W. Xue, P. Liu, R. Miao, Z. Gong, F. Wen, and R. Ying, “Navigation system with SLAM-based trajectory topological map and reinforcement learning-

based local planner," *Advanced Robotics*, pp. 1–22, 2021. [Online]. Available: https://doi.org/10.1080/01691864.2021.1938671

[104] A. Faust, O. Ramirez, M. Fiser, K. Oslund, A. Francis, J. Davidson, and L. Tapia, "PRM-RL: Long-range Robotic Navigation Tasks by Combining Reinforcement Learning and Sampling-based Planning," *arXiv*, pp. 5113–5120, 2017.

[105] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, A. Y. Ng *et al.*, "Ros: an open-source robot operating system," in *ICRA workshop on open source software*, vol. 3, no. 3.2. Kobe, Japan, 2009, p. 5.

[106] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym," *arXiv preprint arXiv:1606.01540*, 2016.

[107] M. Althoff, M. Koschi, and S. Manzinger, "Commonroad: Composable benchmarks for motion planning on roads," in *2017 IEEE Intelligent Vehicles Symposium (IV)*. IEEE, 2017, pp. 719–726.

[108] S. Fujimoto, H. van Hoof, and D. Meger, "Addressing Function Approximation Error in Actor-Critic Methods," *35th International Conference on Machine Learning, ICML 2018*, vol. 4, pp. 2587–2601, 2 2018. [Online]. Available: http://arxiv.org/abs/1802.09477

[109] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller, "Deterministic policy gradient algorithms," in *International conference on machine learning*. PMLR, 2014, pp. 387–395.

[110] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *4th International Conference on Learning Representations, ICLR 2016 - Conference Track Proceedings*, 2016.

[111] H. Zheng, J. Betz, and R. Mangharam, "Gradient-free multi-domain optimization for autonomous systems," *arXiv preprint arXiv:2202.13525*, 2022.

[112] J. Tobin, R. Fong, A. Ray, J. Schneider, W. Zaremba, and P. Abbeel, "Domain Randomization for Transferring Deep Neural Networks from Simulation to the Real World," *Iros*, pp. 0–7, 2017.

[113] Z. Zang, H. Zheng, J. Betz, and R. Mangharam, "Local_inn: Implicit map representation and localization with invertible neural networks," *arXiv preprint arXiv:2209.11925*, 2022.

[114] S. Mysore, B. Mabsout, R. Mancuso, and K. Saenko, "Regularizing action policies for smooth control with reinforcement learning," in *2021 IEEE International Conference on Robotics and Automation (ICRA).* IEEE, 2021, pp. 1810–1816.

[115] A. Liniger and J. Lygeros, "A viability approach for fast recursive feasible finite horizon path planning of autonomous rc cars," in *Proceedings of the 18th International Conference on Hybrid Systems: Computation and Control*, 2015, pp. 1–10.

[116] P. Saint-Pierre, "Approximation of the viability kernel," *Applied Mathematics and Optimization*, vol. 29, no. 2, pp. 187–209, 1994.

[117] A. Kendall, J. Hawke, D. Janz, P. Mazur, D. Reda, J.-M. Allen, V.-D. Lam, A. Bewley, and A. Shah, "Learning to drive in a day," in *2019 International Conference on Robotics and Automation (ICRA).* IEEE, 2019, pp. 8248–8254.

[118] J. Garcıa and F. Fernández, "A comprehensive survey on safe reinforcement learning," *Journal of Machine Learning Research*, vol. 16, no. 1, pp. 1437–1480, 2015.

[119] W. Saunders, G. Sastry, A. Stuhlmüller, and O. Evans, "Trial without error: Towards safe reinforcement learning via human intervention," in *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems*, 2018, pp. 2067–2069.