Machine Learning with Applications 14 (2023) 100496

Contents lists available at ScienceDirect

Machine Learning with Applications

journal homepage: www.elsevier.com/locate/mlwa

Comparing deep reinforcement learning architectures for autonomous racing

Benjamin David Evans *, Hendrik Willem Jordaan, Herman Arnold Engelbrecht

Stellenbosch University, Electrical and Electronic Engineering, Banghoek Road, Stellenbosch, South Africa

ARTICLE INFO

Keywords:
Deep reinforcement learning
End-to-end driving
Autonomous racing
Trajectory planning

ABSTRACT

In classical autonomous racing, a perception, planning, and control pipeline is employed to navigate vehicles around a track as quickly as possible. In contrast, neural network controllers have been used to replace either part of or the entire pipeline. This paper compares three deep learning architectures for F1Tenth autonomous racing: full planning, which replaces the global and local planner, trajectory tracking, which replaces the local planner and end-to-end, which replaces the entire pipeline. The evaluation contrasts two reward signals, compares the DDPG, TD3 and SAC algorithms and investigates the generality of the learned policies to different test maps. Training the agents in simulation shows that the full planning agent has the most robust training and testing performance. The trajectory tracking agents achieve fast lap times on the training map but low completion rates on different test maps. Transferring the trained agents to a physical F1Tenth car reveals that the trajectory tracking and full planning agents transfer poorly, displaying rapid side-to-side swerving (slaloming). In contrast, the end-to-end agent, the worst performer in simulation, transfers the best to the physical vehicle and can complete the test track with a maximum speed of 5 m/s. These results show that planning methods outperform end-to-end approaches in simulation performance, but end-to-end approaches transfer better to physical robots.

1. Introduction

Autonomous racing is the task of calculating speed and steering references that move a vehicle around a race track as quickly as possible. Classical approaches to autonomous racing use a perception (calculate the vehicle's track position), planning (generating control references) and control (executing control references) pipeline that use vehicle models to calculate optimal control actions. In contrast, deep learning methods replace part, or the entire pipeline with a neural network that maps an input vector to the control actions. Deep reinforcement learning (DRL) trains neural networks to map state vectors (information describing the environment) to control actions from experience, requiring only a reward signal to indicate how good or bad each action is. Fig. 1 shows how deep learning architectures for autonomous racing can be categorised into full planning, trajectory tracking and end-to-end approaches based on the input vector used by the agent.

Full planning architectures combine the vehicle's sensor readings (such as LiDAR scans, speed, etc.) with information about the upcoming section of the race track, such as centre-line waypoints, to form the state vector (Fuchs, Song, Kaufmann, Scaramuzza, & Durr, 2021). Trajectory tracking designs aim to improve the planner's performance by adding upcoming trajectory waypoints with optimal speed references into the state vector. Full planning and trajectory tracking approaches aim to improve the computation time required by optimisation approaches (Chisari, Liniger, Rupenyan, Van Gool, & Lygeros, 2021), while maintaining high levels of performance (Cai, Mei, Tai, Sun, &

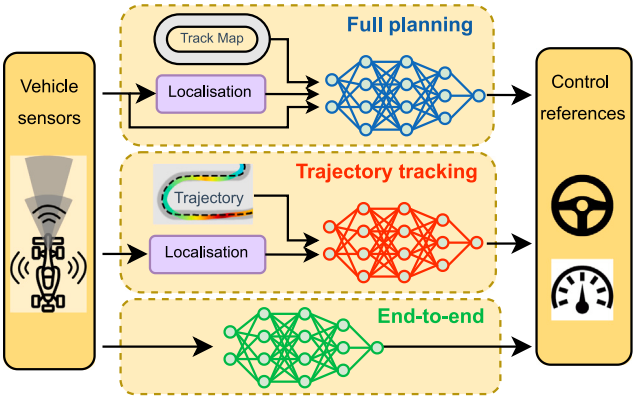


Fig. 1. We compare full planning, trajectory tracking and end-to-end deep learning architectures for controlling autonomous racing cars.

* Corresponding author.
E-mail addresses: bdevans@sun.ac.za (B.D. Evans), wjordaan@sun.ac.za (H.W. Jordaan), hebrecht@sun.ac.za (H.A. Engelbrecht).

https://doi.org/10.1016/j.mlwa.2023.100496
Received 13 June 2023; Received in revised form 4 August 2023; Accepted 30 August 2023
Available online 4 September 2023

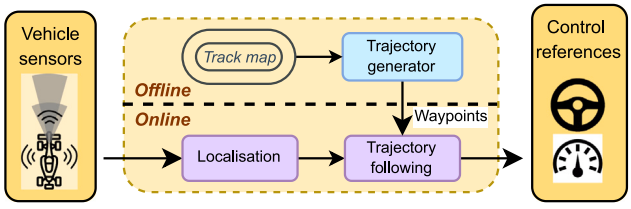


Fig. 2. The classical racing pipeline using a trajectory generator offline and localisation and trajectory following module online.

Liu, 2020). While planning techniques have demonstrated good performance (Wurman et al., 2022), they are still limited by requiring localisation and thus a mapped environment.

End-to-end methods map raw sensor data (LiDAR scans or camera images) directly to control actions (Cai, Wang, Huang, Liu, & Liu, 2021; Hamilton, Musau, Lopez, & Johnson, 2022). Unlike classical approaches, end-to-end methods can plan control actions directly from raw sensor data. They have the advantage of not requiring an explicit vehicle model or real-time processing, and the solutions are flexible to unmapped tracks not seen during training (Bosello, Tse, & Pau, 2022). The limitations of end-to-end approaches are performance, with many solutions limited to low, constant speeds (Evans et al., 2022; Hamilton et al., 2022) and safety with agents achieving low lap completion rates (Brunnbauer et al., 2022).

This paper contributes an extensive examination of full planning, trajectory tracking and end-to-end DRL architectures for autonomous racing through:

1. Comparing the reward, lap progress, crash rate and lap time during training DRL agents for F1Tenth racing.
2. Analysing the trained agent's lap times, success rates, speed profiles, and racing line deviations.
3. Evaluating the simulation-to-reality transfer by contrasting behaviour in simulation and a physical vehicle.

2. Literature study

We provide an overview of classical approaches to autonomous racing that use vehicle models and optimisation to plan and follow a trajectory. Learning techniques for autonomous racing are studied in the categories of planning methods that require the vehicle's location and end-to-end methods that replace the entire pipeline with a neural network. Table 1 provides a summary of the classical, learned planning and end-to-end learning methods referenced.

2.1. Classical racing

The classical racing approach, shown in Fig. 2, calculates an optimal trajectory and then uses a trajectory following algorithm to track it (Betz et al., 2022). During the race, the localisation module uses the LiDAR scan to calculate the vehicle's pose. The path-follower uses the vehicle's pose and optimal trajectory waypoints to calculate the speed and steering angles that control the vehicle.

Trajectory optimisation techniques calculate a set of waypoints (positions with a speed reference) on a track that, when followed, lead the vehicle to complete a lap in the shortest time possible (Christ, Wischnewski, Heilmeier, & Lohmann, 2021). A common approach generates a minimum curvature path and then a minimum time speed profile (Heilmeier et al., 2020).

Localisation approaches for autonomous racing depend on the sensors and computation available. Full-sized racing cars often fuse GNSS and other sensors (Wischnewski et al., 2022), and scaled cars use a LiDAR as input to a particle filter (O'Kelly, et al., 2020; Walsh & Karaman, 2018; Wang, Han, & Vaidya, 2021). Localisation methods enable classical planning since the vehicle can determine its location on a map but are limited by requiring a map of the race track and, thus, are inflexible to unmapped tracks.

Model-predictive controllers (MPC), that calculate receding horizon optimal control commands (Tătulea-Codrean, Mariani, & Engell, 2020; Wang et al., 2021), and pure pursuit path-followers, that geometrically track a path (Becker et al., 2023; O'Kelly, Zheng, Jain, et al., 2020), have been used for trajectory tracking. These methods transfer well to physical vehicles, with a novel pure pursuit algorithm controlling a vehicle at over 8 m/s at the limits of the non-linear tyre dynamics. These classical methods produce high-performance results, accurately tracking the optimal trajectory, but are limited by requiring the vehicle's location on the map.

2.2. Planning architectures

Planning architectures can be split into *full planning* approaches that replace the global (offline) and local (online) planner with a neural network, and *trajectory tracking* methods that replace only the local planner.

2.2.1. Full planning

Full planning methods that use sensor data and track information have demonstrated high performance while maintaining flexibility to different tracks (Fuchs et al., 2021). In Grand Turismo Sport, a hybrid state vector with the vehicle's current speed and acceleration, the curvature of upcoming waypoints and 66 range finder measurements were used (Fuchs et al., 2021). Similarly, Wurman et al. (2022) showed that using a complex state vector leads to outracing world champions. Comparable approaches combining state variables and LiDAR scans or other upcoming track information for autonomous racing have been evaluated in simulated F1Tenth racing (Tătulea-Codrean et al., 2020) and full-size vehicle simulators (Remonda, Krebs, Veas, Luzhnica, & Kern, 2021). While these approaches have demonstrated exceptional performance in racing games and simulators, their performance for physical robots is unknown.

2.2.2. Trajectory tracking

Deep reinforcement learning agents have been combined with an offline trajectory optimiser for high-performance control in autonomous racing (Dwivedi, Betz, Sauerbeck, Manivannan, & Lienkamp, 2022; Ghignone, Baumann, & Magno, 2023). Ghignone et al. (2023) uses a state vector with the vehicle's velocity vector, heading angle a list of upcoming trajectory points. Their results in simulated F1Tenth racing demonstrated improved computational time, better track generalisation and more robustness to modelling mismatch.

DRL agents have demonstrated exceptional performance in autonomous drifting, where agents can track a precomputed trajectory at the non-linear tyre limits (Cai et al., 2020; Orgován, Bécsi, & Aradi, 2021). Cai et al. (2020) used a DRL agent with a state consisting of the upcoming waypoints (including the planned slip angle) and the current vehicle state to control a vehicle in the Speed Dreams racing simulator. Their results showed that the DRL agent could control the vehicle while drifting through corners in many complex environments and generalise to different vehicles.

While full planning and trajectory tracking approaches have demonstrated promising results in simulation, their performance on physical vehicles is unknown due to a lack of investigation into the simulation-to-reality transfer (see Table 1).

Table 1
Classical, planning and end-to-end approaches to autonomous racing with results in simulation and physical vehicles.

Category	Method	Platform	Simulation result	Physical result
Classical	Learning MPC (Wang et al., 2021)	F1Tenth	Optimal racing up to 7 m/s in simulation and reality	
	Model and acceleration pure pursuit (Becker et al., 2023)	F1Tenth	Racing at the non-linear tyre limits with speeds over 8 m/s in simulation and reality	
Planning	Full planning for racing games (Fuchs et al., 2021; Wurman et al., 2022)	Grand Turismo Sport	Outperforms world champion gamers	No physical tests
	Planning for racing (Tătulea-Codrean et al., 2020; Vianna, Goubault, & Putot, 2021)	F1Tenth	5 m/s (Tătulea-Codrean et al., 2020) 3 m/s (Vianna et al., 2021)	No physical tests
	Trajectory tracking for racing (Dwivedi et al., 2022; Ghignone et al., 2023)	F1Tenth	Outperforms path tracking	No physical tests
	Trajectory tracking for drifting (Cai et al., 2020; Orgován et al., 2021)	Speed Dreams & Carla	Able to control the vehicle in non-linear environments	No physical tests
End-to-end	Driving with LiDAR (Evans et al., 2022; Hamilton et al., 2022; Ivanov et al., 2020)	F1Tenth	Constant speeds of 1 m/s (Hamilton et al., 2022), 2 m/s (Evans et al., 2022) and 2.4 m/s (Ivanov et al., 2020) in simulation and reality	
	Racing with LiDAR (Bosello et al., 2022; Brunnbauer et al., 2022)	F1Tenth	Speeds ranging up to 5 m/s	Completes test lap

2.3. End-to-end architectures

End-to-end architectures map raw sensor data directly to control commands. While camera images have been used in end-to-end solutions (Jaritz, De Charette, Toromanoff, Perot, & Nashashibi, 2018), we focus on methods using LiDAR scans (Hamilton et al., 2022). End-to-end architectures have commonly been used for autonomously driving F1Tenth vehicles at constant speeds of 1 m/s (Hamilton et al., 2022), 2 m/s (Evans et al., 2022), and 2.4 m/s (Ivanov et al., 2020). End-to-end methods have the significant advantage of not requiring localisation since they can directly use the LiDAR scan as input to the agent. Not requiring localisation allows them to be transferred to different environments (Bosello et al., 2022; Evans et al., 2022) and achieve zero-shot transfer to physical vehicles (Hamilton et al., 2022).

End-to-end F1Tenth autonomous racing has been approached with model-free (Bosello et al., 2022), and model-based (Brunnbauer et al., 2022) algorithms, reaching up to 5 m/s in simulation. While these approaches have demonstrated the ability to complete a lap on a physical vehicle, several problems, such as constant swerving (Brunnbauer et al., 2022), were observed. It remains to be investigated how high-speed racing transfers to physical cars.

3. Racing architectures

3.1. Problem description

The general racing problem is to use the onboard sensor readings to select control signals that move the vehicle around the track as quickly as possible. Racing cars have a 2D LiDAR scanner that provides geometric information regarding the environment and perception method (such as a particle filter) that can estimate the vehicle's odometry on a map.

Racing vehicles are commonly modelled using the bicycle model. Fig. 3 shows how the model represents the position x, y and orientation θ on a map, speed v, steering angle δ and slip angle β. The control inputs are speed v and steering angle δ references that a low-level controller follows.

Deep Learning Problem: Fig. 4 shows how the deep learning problem for autonomous racing uses a neural network that selects steering and speed actions.

The actions control the vehicle in the racing environment. A crucial part of the deep learning formulation is the state passed from the environment to the agent. This work evaluates different state vectors to investigate the effect of the state vector on agent performance. We now describe the composition of the full planning, trajectory tracking and end-to-end state vectors.

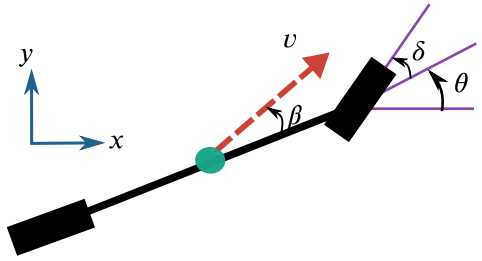


Fig. 3. Bicycle model showing how the position, orientation, speed, steering angle and slip angle are represented.

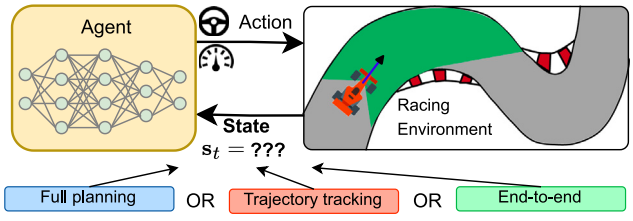


Fig. 4. The deep learning formulation uses a neural network to select actions.

3.2. Planning architectures

Planning architectures replace the classical planning components with a neural network agent that has access to the vehicle's pose and a track map. Some approaches have replaced the complete planning pipeline, while others have retained the optimal trajectory generator and use a neural network for trajectory tracking. The aim of replacing the planner is that optimisation-based planners, such as model predictive control, are computationally expensive and brittle to model mismatch.

3.2.1. Full planning

The full planning architecture replaces the global and local planners with a neural network agent. The complete planning task uses a race track map, the vehicle's odometry and incoming sensor data to plan speed and steering commands. These methods have the advantage of not requiring a precalculated optimal trajectory and, thus, being less dependent on the vehicle model.

Fig. 5 shows the state vector for the full planning architecture, consisting of the vehicle's speed, steering angle, a set of upcoming centre line points, and the current LiDAR scan. The LiDAR scan is

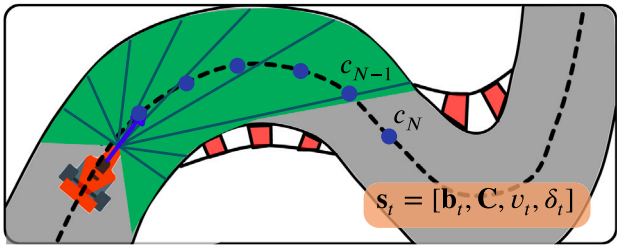


Fig. 5. The full planning architecture uses a state vector consisting of beams from the LiDAR scan, a set of upcoming centreline points, and the linear speed and steering angle.

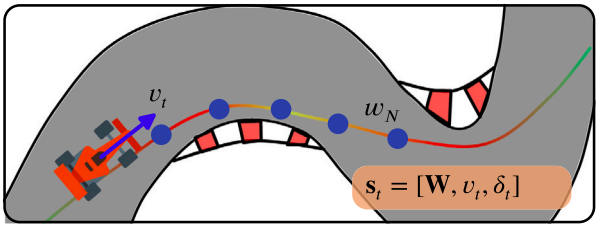


Fig. 6. Trajectory tracking state vector consisting of upcoming waypoints and speed references on the optimal trajectory and the linear and steering angle.

included since the vehicle must know the track geometry to select an optimal racing line. The state is written as $\mathbf{s}_t = [\mathbf{b}_t, \mathbf{C}, v_t, \delta_t]$, where $\mathbf{C}$ is a set of $N_{\text{waypoints}}$ upcoming centre line points transformed into the vehicle's coordinate frame.

3.2.2. Trajectory tracking

The task of the local planner is to follow a predefined set of waypoints called the race line. Each track's race line is generated using a trajectory optimisation algorithm (Heilmeier et al., 2020). The racing line is a set of evenly spaced ordered waypoints containing an x, y position and speed.

The state vector, shown in Fig. 6, comprises the vehicle's current speed, steering angle, and $N_{\text{waypoints}}$ from the racing line. The state is $\mathbf{s}_t = [\mathbf{W}, v_t, \delta_t]$, where $\mathbf{W}$ is the set of upcoming relative waypoints. The waypoints are transformed from the global coordinate frame into the vehicle's reference frame by translation (according to the vehicle's positions) and rotation (according to the vehicle's orientation).

3.3. End-to-end architecture

The end-to-end learning architecture replaces the entire planning pipeline with a neural network meaning that the LiDAR scan is used as the state vector. End-to-end solutions have the advantage of not requiring any preprocessing or localisation on the LiDAR scan. The full LiDAR scan has a field of view (FOV) of $3/2\ \pi$ radians and 1080 beams. A subset of N_{beams} is selected from the LiDAR scan and used as input for the network. The resulting set of beams is written $\mathbf{b} = [b^1, b^2, \dots, b^N]$.

The vehicle's motion is communicated to the agent by stacking the previous and current LiDAR scans and including the linear speed in the state vector. The end-to-end state vector, shown in Fig. 5, is written as, $\mathbf{s}_t = [\mathbf{b}_{t-1}, \mathbf{b}_t, v_t]$.

4. Methodology

We outline the methodology employed for conducting simulation and real-world experiments aimed at comparing three different architectures on the F1Tenth platform. A preliminary overview of reinforcement learning, specifically focusing on the soft actor-critic (SAC), is presented. Subsequently, the F1Tenth racing platform, our experimental setup and the implementation details are described. Finally,

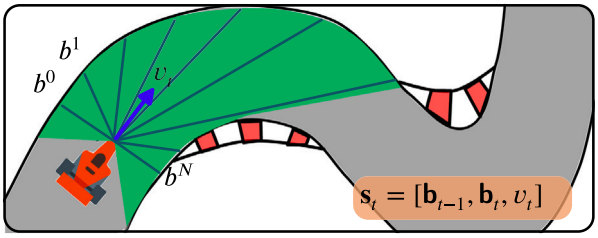


Fig. 7. The end-to-end architecture uses a state of the previous and current LiDAR scan and the vehicle's speed.

we explain the process of hyperparameter selection for the number of beams and the number of waypoints.

4.1. Reinforcement learning preliminary

Reinforcement learning problems are modelled as Markov Decision Processes (MDPs) (Sutton & Barto, 2018). MDPs contain states s in the state space $\mathcal{S}$, actions a in an action space $\mathcal{A}$, a transition probability function of how states change based on the action and a reward function that is used to calculate a reward for a state–action combination. DRL algorithms use the agent's policy of selecting actions based on a state to collect experience to train the agent to maximise a reward signal, thus producing the desired behaviour. We present a preliminary on each of the three algorithms considered in this work.

4.1.1. Deep deterministic policy gradient (DDPG)

Deep deterministic policy gradient (DDPG) methods can select continuous actions for robotic control (Lillicrap et al., 2015). DDPG is an actor-critic algorithm that uses a policy network μ to select actions and a Q-network Q to approximate the expected return for a state action pair (Q-value). The algorithm uses model networks, trained and used to select actions and target networks to calculate the target values to update the networks. DDPG is an off-policy algorithm, meaning that experience tuples of state, action, next state and reward are stored in a replay buffer. After each action has been selected, a set of N transitions is randomly sampled from the replay buffer and used to train the networks.

The model Q-network is trained to learn the expected return for each state–action pair by minimising the loss between the current Q-value estimates $Q(s_j, a_j)$ and a calculated Q-target y_i for each transition j in the sampled batch. The bootstrapped targets are calculated using the Bellman equation by adding the reward earned and the discounted Q-value for the next state if the agent followed its target policy,

$$y_j = r_j + \gamma Q'(s_{j+1}, \mu'(s_{j+1})), \tag{1}$$

where γ is the discount factor.

The model policy network selects continuous deterministic actions based on a state. Noise sampled from a random distribution is added to each action for exploration. The policy network, parameterised by θ, is trained to maximise the objective ($J(\theta)$) of selecting actions with high Q-values. The gradient that maximises the objective $J(\theta)$ is calculated as,

$$\nabla_\theta J(\theta) = \frac{1}{N} \sum_j \nabla_\theta Q(s_j, \mu(s_j)). \tag{2}$$

After each network update, a soft update is applied to adjust the target networks towards the model networks. In our implementation, we used a discount factor of 0.99, learning rates of 0.0005 for the actor and 0.001 for the critic, batch size of 100, and soft update temperature of 0.005.

4.1.2. Twin-delayed DDPG (TD3)

The twin-delayed DDPG (TD3) algorithm features three improvements over the original DDPG algorithm, (1) using a pair of Q-networks, (2) delaying policy updates, and (3) target policy smoothing (Fujimoto, Hoof, & Meger, 2018). Using two Q-networks is done to prevent the agent from overestimating the expected return by using the smallest Q-value to calculate the Q-target. The frequency of the policy updates is slowed down to improve the stability of the updates. In our implementation, after every training step, the Q-networks are updated twice, and the policy once. The target policy that is used to calculate the Q-target values (Eq. (1)) is smoothed by adding random noise to each action to prevent the Q-function learning sharp peaks or troughs. Taking these changes into account, the TD3 Q-targets are calculated as,

$$\begin{aligned} y_j &= r_j + \gamma \min_{i=1,2} Q_{\theta_i'}(s_j', \mu(s_j') + \epsilon) \\ \epsilon &\sim \text{clip}(\mathcal{N}(0,\sigma), -c, c) \end{aligned} \tag{3}$$

In the equation, γ is the discount factor, i is the number of the Q-network (i.e. $Q_{\theta_1'}, Q_{\theta_2'}$), μ is the actor network, ϵ is the clipped noise, and c is the noise clipping constant. In our implementation, we used a batch size of 100, a discount factor of 0.99, a soft update temperature of 0.005, exploration noise with a standard deviation of 0.1, policy smoothing noise with a standard deviation of 0.2 and a noise clip and 0.5.

4.1.3. Soft actor-critic (SAC)

The soft actor-critic (SAC) algorithm improves the original DDPG algorithm by redefining the object to maximise policy entropy (Haarnoja et al., 2018). By doing so, the SAC algorithm inherently promotes exploration by encouraging the agent to select the most unpredictable policy that maximises the reward.

The algorithm uses a single policy network π, two model Q-networks and two target Q-networks. For each state, the policy π outputs a mean μ and standard deviation σ to sample actions using the reparameterisation trick. Actions are selected as $\pi(s) = \tanh(\mu(s) + \sigma(s)\xi)$, where ξ is sampled from a normal distribution with a mean of 0 and standard deviation of 1 as $\xi \sim \mathcal{N}(0,1)$. The policy objective is updated to include entropy, calculated as the negative log probability of selecting an action. The SAC policy gradient is written as

$$\nabla_\theta J(\pi) = \sum_j \nabla_\theta \min_{i=1,2} Q_{\phi_i}(s_j, \tilde{a}_j) - \alpha \nabla_\theta \log \pi_\theta\left(\tilde{a}_j | s_j\right). \tag{4}$$

In Eq. (4), the next actions $\tilde{a}_j$ are sampled from the policy as $\tilde{a}_j \sim \pi_\theta(\cdot|s_j)$. The objective maximises the estimated Q-values using the minimum of the pair of Q-networks (same as TD3). The SAC gradient then maximises the entropy by subtracting the log probabilities of the sampled actions, which are scaled by the parameter α.

The Q-target values are calculated to maximise the entropy for the next actions as,

$$y = r + \gamma \left(\min_{i=1,2} Q_{\phi_i'}(s', \tilde{a}') - \alpha \log \pi_\theta(\tilde{a}'|s') \right). \tag{5}$$

In Eq. (5), the Q-target is the reward plus the discounted Q-value for the next state minus the log probabilities of the next actions $\tilde{a}_j$. The Q-networks are updated towards the targets using the mean squared error (MSE) loss. Finally, the scaling parameter alpha is auto-tuned towards a desired entropy $\tilde{\mathcal{H}}$ by minimising the distance between the log probabilities of the next actions and the desired entropy. The objective is written as,

$$\nabla_\alpha J(\alpha) = \sum_j \nabla_\alpha \left(-\alpha(\log \pi_\theta(\tilde{a}_j|s_j) + \tilde{\mathcal{H}}) \right). \tag{6}$$

In our implementation, we used a batch size of 100, discount factor γ of 0.99, soft update temperature of 0.01, and learning rate of 0.001 with the Adam optimiser.

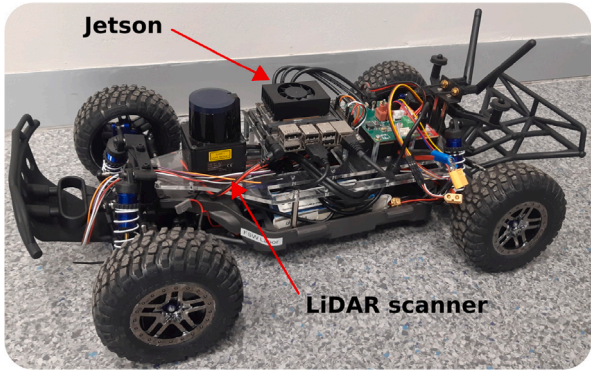


Fig. 8. The F1Tenth vehicle used for the physical experiments.

Fig. 9. Map shapes of the AUT, ESP, GBR and MCO (left to right) tracks.

4.2. F1Tenth platform

F1Tenth racing cars are 1/10th the size of real F1 vehicles and are used as a sandbox for developing autonomous algorithms (Betz et al., 2022). The platform focuses on high-performance, safe algorithms for cyber–physical systems. Fig. 8 shows our F1Tenth vehicle, which is equipped with a LiDAR scanner for sensing the environment, an NVIDIA Jetson NX as the central computation platform, a variable electronic speed controller (VESC) and drive motor to move the vehicle forwards, and a servo motor to steer the front wheels.

A Gym-style Python simulator that uses the single-track bicycle model is available that is used to train the agents (O'Kelly, Zheng, Karthik, & Mangharam, 2020). The simulator provides the planner with a state consisting of the LiDAR scan, location and speed. The simulator and the physical vehicle use a planning frequency of 10 Hz. The planner then calculates an action that is returned to the simulator. This planning and acting loop is repeated until a lap is completed or the vehicle crashes. Fig. 9 shows the shapes of the tracks.

The vehicle uses the ROS2 middleware for the sensors and software components to communicate with each other. ROS2 nodes are written that extract the required data from the sensors and form the state vectors used by the agent. The vehicle's odometry is estimated using a particle filter that implements scan matching (Walsh & Karaman, 2018). The steering angle cannot be directly measured and is recursively estimated as $\delta_t = 0.5\delta_{t-1} + 0.5\delta_a$, where δ_a is the steering command selected by the agent.

All the quantities that are passed into and out of the neural network are scaled into the range $[-1, 1]$ by using the relevant maximum. For the LiDAR beams, the maximum value is 10 m, for the speed 8 m/s, and for the steering angle 0.4 rad. The upcoming waypoints are spaced 20 cm apart and scaled using a maximum distance of 4 m. The output values from the neural network are rescaled to speed and steering references using the same values. The speed is rescaled to the range $[1, v_{\max}]$ to ensure the vehcile is never stationary.

4.2.1. Classical planner

The classic planner uses an optimal trajectory generated using the method from Heilmeier et al. (2020), and the pure pursuit path follower from Coulter (1992). The trajectory optimiser uses the centre line to calculate a minimum curvature path. The vehicle dynamic limits and friction parameters are then used to generate a minimum time speed

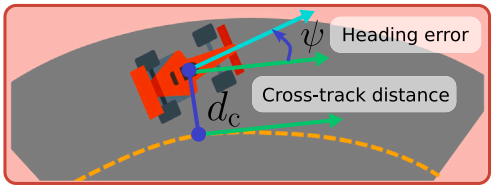


Fig. 10. The cross-track and heading error reward uses the vehicle speed v_t, heading error ψ and cross-track distance d_c.

profile. The pure pursuit path following algorithm tracks a reference path using a lookahead distance to select a point to steer towards. Given a lookahead point that is a lookahead distance l_d away from the vehicle, and at a relative heading angle of α, the steering angle δ is calculated as,

$$\delta = \arctan\left(\frac{L\sin(\alpha)}{l_d}\right), \tag{7}$$

where L is the length of the vehicle's wheelbase.

4.2.2. Reward signals

After each timestep, a reward is calculated and given to the agent. To train the agent to complete laps without crashing, the agent is rewarded with the largest reward of 1 for completing a lap and punished with −1 if the vehicle crashes. Additionally, a shaped reward to encourage fast racing behaviour is given to the agent. The reward is calculated as,

$$r = \begin{cases} 1 & \text{if lap complete} \\ -1 & \text{if collision} \\ r_{\text{shaped}} & \text{otherwise.} \end{cases} \tag{8}$$

We consider two shaped reward signals to train the agents, the *cross-track and heading error* reward and the trajectory-aided learning reward signal. The cross-track and heading error signal rewards the agent for speed in the direction of the centre line and punishes the vehicle proportionally to the lateral deviation (Bosello et al., 2022). The cross-track and heading error reward is calculated as,

$$r_{\text{CTH}} = \frac{v_t}{v_{\max}}\cos\psi - d_c, \tag{9}$$

where v_t is the vehicle's speed at timestep t, $v_{\max}$ is the maximum speed, ψ is the heading error (angle between the vehicle heading and the track centre line), and d_c is the cross-track distance. Fig. 10 shows how the speed, heading error and cross-track distance are measured for a vehicle on a race track segment.

The second reward signal considered is the *trajectory-aided learning* (TAL) reward signal (Evans, Engelbrecht, & Jordaan, 2023), which trains the agent for high-speed racing. The trajectory-aided learning reward encourages agents to follow the racing line by using the difference between the agent's actions and the actions that a classical planner would have selected. The TAL reward is calculated as,

$$r_{\text{TAL}} = 1 - |\mathbf{u}_{\text{agent}} - \mathbf{u}_{\text{classic}}| \tag{10}$$

Fig. 11 graphically shows the comparison between the action selected by the classic planner following the optimal trajectory and the agent action. The shaped reward is scaled by 0.2 to ensure it is much smaller than the rewards for crashing and lap completion.

4.3. Hyperparameter tuning

The hyperparameters are tuned by investigating how well the neural network can fit an expert dataset. The dataset is built using the classic planner to race for a single lap on each of the four test maps. At each timestep, the vehicle's state and LiDAR scan and the planner's actions are saved to form the dataset. For each test, a dataset is constructed by generating the relevant state vector at each timestep in the data batch. The networks are trained for 2000 epochs of sampling 100 state–action pairs and evaluating the mean squared error (MSE) loss between the true and predicted actions. A 90/10 training/testing split is used. Each experiment is seeded and repeated 10 times with different seeds. The training and testing losses are evaluated by averaging the loss from 10 sampled mini-batches. The square root of the loss (RMSE) value is used in the results.

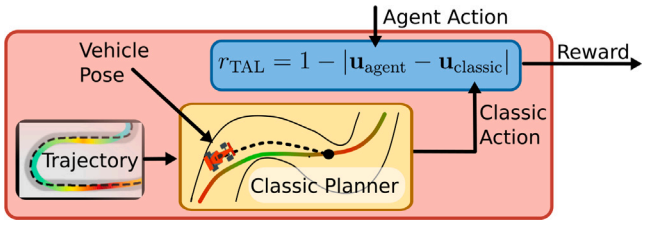


Fig. 11. The trajectory-aided learning reward penalises the difference between the agent's actions and those selected by a classical planner following the optimal trajectory.

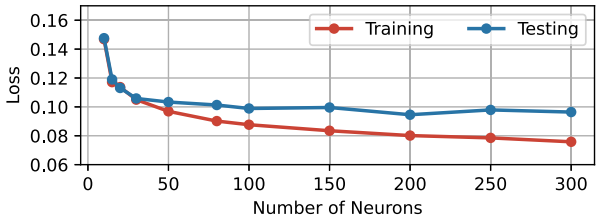


Fig. 12. The training and testing losses from tuning the number of neurons used per layer in the neural network.

4.3.1. Neural network size

We aim to select the smallest neural network that can feasibly represent a policy. smaller networks are faster to train and more computationally efficient. However, if the network is too small, it will not be able to learn the dynamics. Therefore, we evaluate how many neurons per layer are required for the testing loss to stop decreasing. We use the end-to-end state vector with 20 beams for this test, since it is the largest input vector out of the three architectures. Two-layer multi-layer perceptron networks are used, since this is a common architecture in autonomous racing (Chisari et al., 2021; Hamilton et al., 2022). Tests are done with a two-layer network with numbers of neurons ranging from 10 to 300. The hidden layers use the *ReLU* activation function, and the output layer uses the *tanh* function. The Adam optimiser is used with a learning rate of 0.001.

Fig. 12 shows the training and testing losses plotted against the number of neurons used per layer in the neural network. While using only ten neurons gives a large loss above 0.14, the loss decreases with an increasing number of neurons. While the training loss continues to increase, the testing loss levels out when using more than 100 neurons. Therefore, 100 neurons per layer are used for all further tests. We note that we keep the number of neurons the same for each test so that the architectures themselves can be evaluated.

4.3.2. State vector parameters

Fig. 13 shows the training and testing losses for tuning the number of beams and number of waypoints used.

The number of beams required is evaluated using 5, 10, 12, 15, 20, 30 and 60 beams in the end-to-end architecture. The loss graph for the number of beams (left) shows that using fewer beams results in a larger loss than using more beams. Using more than 20 beams does not have a significant impact on the loss, and therefore, N_{beams} is selected as 20.

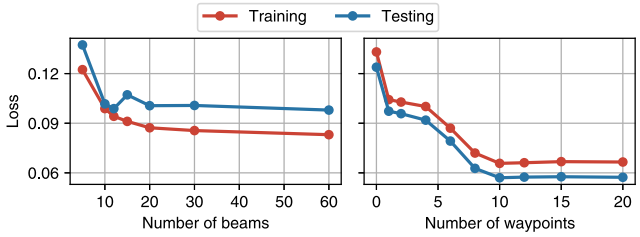


Fig. 13. The losses from tuning the number of beams and the number of waypoints used in the state vectors.

The number of waypoints required is evaluated using 0, 1, 2, 4, 6, 8, 10 and 20 waypoints and the trajectory tracking architecture. The graph shows a similar pattern of large losses when using few waypoints, and smaller losses when using many waypoints. The performance flattens out when using more than 10 waypoints, and therefore, $N_{\text{waypoints}}$ is selected as 10. All further experiments use 20 LiDAR beams and 10 upcoming waypoints.

5. DRL simulation evaluation

The different architectures are evaluated by investigating the: The full planning, trajectory tracking and end-to-end DRL architectures are evaluated using three tests:

1. **Reward signal comparison** of the behaviour generated by the cross-track and heading reward compared to the trajectory-aided learning reward.
2. **Training algorithm comparison** of the DDPG, TD3 and SAC algorithms for their impact on training and performance.
3. **Policy generality investigation** of how well the learned policies transfer to different maps.

In each of the tests, the focus is to show the differences between the different architectures. For all the tests, agents are trained for 60k training steps, and then 20 test laps are run. All the simulation experiments are repeated three times.

5.1. Reward signal comparison

We compare the cross-track and heading reward and the trajectory-aided learning reward by training agents using each reward on the GBR map. The reward signal comparison tests use the soft actor-critic (SAC) algorithm. The main metric used to analyse the learning is the average track progress made by the agent. The average track progress is a moving average of the progress made by the vehicle for each episode before the car crashes or completes a lap. In the reward signal study, the episode reward is also used to explain the learned behaviour.

Fig. 14 shows the average progress and episode reward during training from each of the reward signals. The cross-track and heading (CTH) average progress demonstrates a significant difference between the architectures, with the full planning architecture achieving around 90%, the trajectory tracking agent 60% and the end-to-end agent 50%. The TAL reward trains all the agents to achieve higher average progress, with the full planning and trajectory tracking agents achieving near 100% and the end-to-end agent around 90%.

The rewards earned by the CTH agents follow a similar pattern to the average progress of the full planning earning the most reward, followed by the trajectory tracking and finally, the end-to-end agent. The TAL agents show an interesting pattern of the trajectory tracking agent achieving a higher reward than the full planning agent from around 15k steps. This suggests that the trajectory tracking agent selects actions that are more similar to the classical planner (following the optimal trajectory) than the other agents.

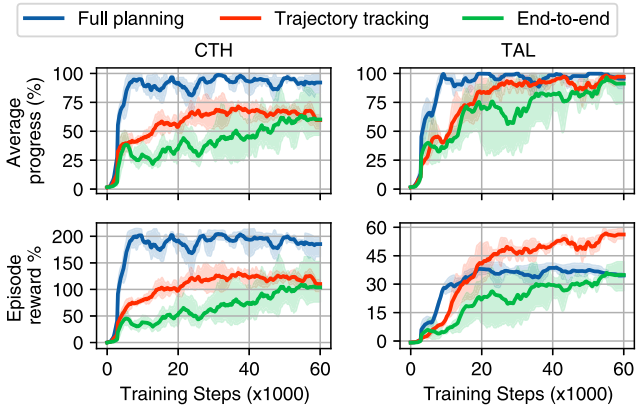


Fig. 14. Average progress and episode reward during training with the cross-track and heading (CTH) and trajectory-aided learning (TAL) reward signals on the GBR map.

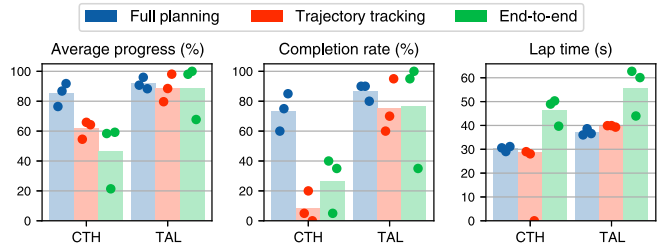


Fig. 15. Average progress, completion rate and lap times achieved by agents trained with the cross-track and heading (CTH) and trajectory-aided learning (TAL) reward signals.

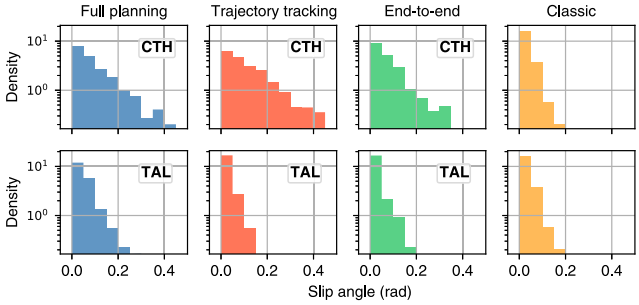


Fig. 16. The distribution of absolute slip angles from tests laps of agents trained with the cross-track and heading (CTH) and trajectory-aided learning (TAL) reward signals.

Fig. 15 shows the average progress, completion rate and lap times achieved by agents trained with the cross-track and heading (CTH) and trajectory-aided learning (TAL) reward signals. The agents were trained and tested on the GBR map. The average progress values correspond well to the training graph, with the CTH agents achieving around 85%, 60% and 50%, respectively. The trajectory-tracking and end-to-end agents trained with the CTH reward have a low completion rate of around 8% and 25%, indicating that very few laps are completed. The CTH full planning agent achieves a significantly higher completion rate indicating that the full planning agent is more robust to different rewards than the other architectures. While the CTH agents have critically low lap completion rates, they have fast lap times, outperforming the TAL agents. This is further investigated by plotting the distribution of slip angles for the agents trained with the two rewards. The slip angle is the difference between the vehicle's velocity (direction of motion) and orientation (which way the front of the vehicle faces).

Fig. 16 shows the slip angles from test laps of agents trained with the cross-track and heading (CTH) and trajectory-aided learning (TAL)

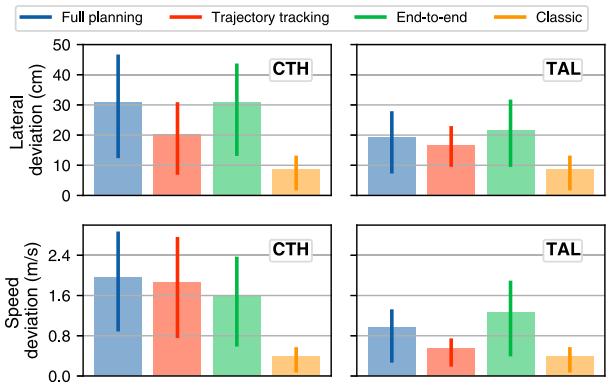


Fig. 17. The lateral and speed deviation for agents trained with the cross-track and heading (CTH) and trajectory-aided learning (TAL) reward signals.

reward signals on the GBR map. The agents trained with the cross-track and heading error reward have a higher number of large slip angles than the TAL agents. The CTH agents have many slip angles above 0.2 radians, while most of the TAL agents do not have any slip angles above 0.2. The high-slip angles resulting from the CTH reward signal indicate that the agent learns to exploit the simulator by causing the vehicle to drift around parts of the track. This drifting behaviour is unstable and not repeatable.

We consider the lateral and speed deviation for the agents trained with each reward signal to the optimal trajectory. The lateral deviation is the perpendicular distance between the optimal trajectory and the vehicle, and the speed deviation is the absolute value of the difference between the vehicle's speed and the speed at the optimal trajectory point. Fig. 17 shows the lateral and speed deviation for agents trained with the cross-track and heading (CTH) and trajectory-aided learning (TAL) reward signals tested on the GBR map. The general pattern is that the TAL agents have lower lateral and speed deviations than the CTH agents. Across both reward signals, the trajectory-tracking agent has the lowest lateral deviation of the three DRL agents. This shows that giving trajectory waypoints to the planner leads to the agent selecting a more similar path to the optimal trajectory. In contrast, the TAL, end-to-end agent, has the largest deviations, indicating that giving only LiDAR scans leads to behaviour that is dissimilar to the optimal trajectory. While the TAL reward improves on the cross-track and heading reward, none of the agents can outperform the classical trajectory following method.

The study on reward signals shows that the trajectory-aided learning reward can train DRL agents to autonomous racing to achieve higher average progresses during training and higher lap completion rates after training. While the full planning architecture achieved around a 75% completion rate with the cross-track and heading reward, the other two architectures achieved below 50% lap completion. This indicates that the full planning architecture is the most robust to different rewards, followed by the end-to-end architecture, and the trajectory tracking architecture is very sensitive to reward. Studying the lateral and speed deviations showed that the trajectory tracking architecture, trained with the TAL reward, has the most similar behaviour to the classical planner following the optimal trajectory. This indicates that while sensitive, the trajectory tracking architecture shows the best ability to replicate a specific behaviour.

5.2. Algorithm comparison

We use the DDPG, SAC and TD3 algorithms to train three agents of each architecture for autonomous racing. These tests use the TAL reward and the GBR map. We consider the differences during training and the differences in the behaviour of the trained agents.

Fig. 18 shows the average progress during training from using each algorithm on the GBR map. All the agents trained with the DDPG algorithm perform poorly with averages remaining around 50%. In contrast, the SAC and TD3 algorithm both train the agents well, with the average progresses reaching over 90%. The TD3 algorithm appears to train the agents slightly faster and to higher average progress. With all three algorithms, the full planning agent's progress rises the fastest, quickly reaching high values. The end-to-end agent trained with the SAC algorithm takes around 50k steps to reach over 90% average progress. The trajectory tracking agent generally remains between the full planning and end-to-end agents. This indicates that the full planning agent is the most robust to different training algorithms, always training quickly, and the end-to-end agent requires the most training data to converge, training more slowly. Since only the SAC and TD3 algorithms can reliably train the agents to complete laps, we study their training behaviour in more detail.

Fig. 19 shows the lap times during training of the agents trained with the SAC and TD3 algorithms. When training with the SAC, all the architectures show a pattern of starting with slower lap times and as the training progresses, achieving faster lap times. In contrast, the TD3 algorithm shows little change in lap times throughout training. It is proposed that this is due to the maximum entropy formulation of the SAC algorithm in contrast to the deterministic nature of the TD3 algorithm. The trajectory-tracking architecture has the least variance between lap times, and the end-to-end architecture has the most. This is possibly due to large differences between LiDAR scans compared to upcoming trajectory points.

Fig. 20 shows the crash frequency per algorithm for each architecture. The agents trained with the SAC algorithm crash for a longer period of time during training, indicating that the algorithm is trying all possible actions before converging. The TD3 agents crash less than the SAC agents. This pattern is similar to the lap times presented in Fig. 19, indicating that the SAC algorithm fails more before learning, while the TD3 algorithm learns faster by exploiting its current knowledge. In comparing the architectures, the full planning architecture crashes the least during training, completing almost all laps after 40k steps. In contrast, the end-to-end agent has a high number of crashes during training, still occasionally crashing at the end of training.

Fig. 21 shows the average progress and lap times from training agents with the DDPG, SAC and TD3 algorithms. The DDPG agents perform poorly, with all the agents having some repetitions below 70% completion. Of the agents trained with the DDPG algorithm, the full planning architecture performs the best, with two of the repetitions reaching near 100% completion rate. The trajectory tracking average is similar, with a large spread between the three repetitions. The end-to-end architecture performs the worst, with all three repetitions below 75%. The inconsistency in the results highlights the shortcoming of the DDPG algorithm of being brittle to random seeding.

The poor results achieved by the DDPG algorithm show that while it has the potential to train agents, the performance is often poor and lacks reliability. In contrast, the TD3 algorithm achieves near 100% completion rate for all of the architectures. The SAC algorithm is slightly behind with around 90%, and the results are more spread out. The agents trained with the TD3 algorithm all have near 100% completion rates and similar lap times of around 40 s. The agents trained with the SAC algorithm vary more between repetitions, with the trajectory tracking agent's completion rate ranging from 80%–100% and the end-to-end agent's lap times ranging from 40–60 s. This indicates that the TD3 algorithm produces more repeatable behaviour with less dependence on the random seeds.

The differences in lap times, specifically between the end-to-end planner, are investigated by plotting the frequency of the vehicle speeds for agents trained with each planner. Fig. 22 shows the frequency of the vehicle speeds. While the full planning and trajectory tracking agents have similar performance, there is a notable difference in the end-to-end agent. The end-to-end agent trained with the SAC algorithm never

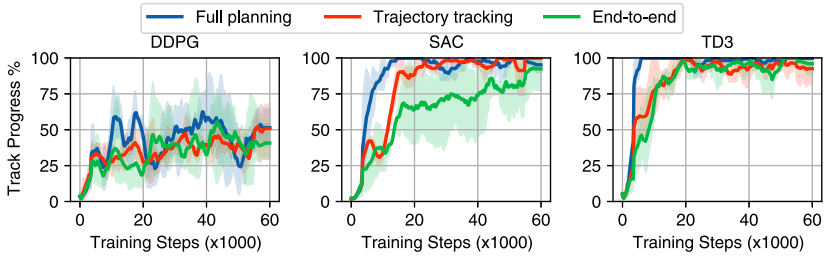


Fig. 18. Average progress during training agents on the GBR map with the DDPG, SAC and TD3 algorithms.

Table 2
Lap times from agents trained on MCO and tested on four test maps compared to the classical planner.

TestMap	AUT	ESP	GBR	MCO
Classic planner	22.11 ± 0.00	47.85 ± 0.00	38.95 ± 0.00	35.51 ± 0.00
Full planning	**21.52** ± 0.15	**48.19** ± 2.38	42.46 ± 1.18	37.85 ± 0.12
Trajectory tracking	22.28 ± 0.81	49.81 ± 0.42	**39.64** ± 1.58	**36.08** ± 1.24
End-to-end	21.90 ± 1.42	49.25 ± 2.90	43.34 ± 3.21	38.14 ± 1.83

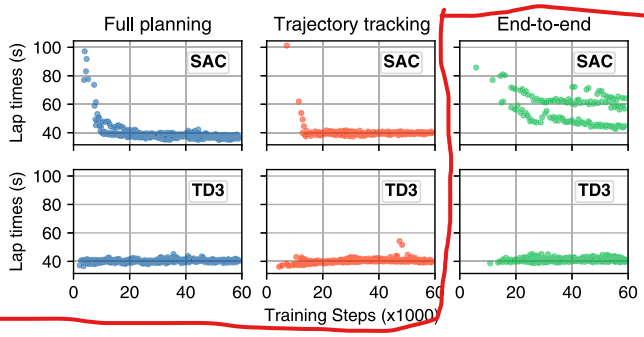


Fig. 19. Lap times during training agents with the SAC and TD3 algorithms on the GBR map.

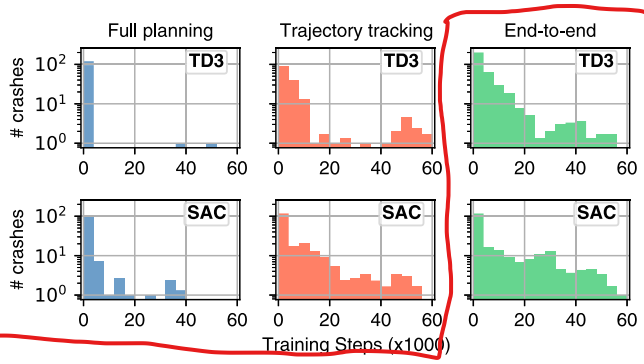


Fig. 20. Crash frequency during training agents with the SAC and TD3 algorithms on the GBR map.

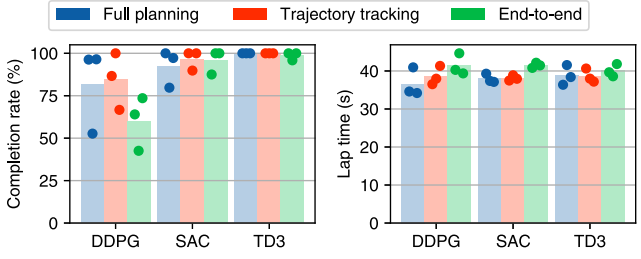


Fig. 21. The average progress and lap times from training agents with the DDPG, SAC and TD3 algorithms.

selects high speeds, while the one trained with the TD3 algorithm does. The result of this is slower speed selection. While it is not clear why the end-to-end agent trained with the SAC algorithm selects lower speeds, it could be related to the difficulty in racing at high speeds using only LiDAR scans. This result indicates that the end-to-end architecture is less robust to changes in algorithm compared to the full planning and trajectory tracking architectures.

The investigation into algorithms showed that while the DDPG algorithm produces inconsistent, poor completion rates, the TD3 and SAC algorithms can train agents to achieve good racing performance. The TD3 algorithm learns a racing policy without first learning slow laps, while the SAC algorithm starts out learning slowly and steadily improves. The TD3 algorithm produces repeatable results, with the three repetitions being closely grouped. The SAC algorithm shows a larger variance between results, indicating a greater dependence on the seed used. The end-to-end architecture is the most sensitive to the training algorithm, completing very few laps with the DDPG and achieving slow lap times with the SAC algorithm.

5.3. Policy generality investigation

The agents are investigated for their ability to generalise and transfer to maps other than the training maps. Agents of each architecture are trained using the TAL reward and the TD3 algorithm on the MCO and GBR maps before being tested on four maps (see Fig. 9). A quantitative comparison of the completion rate and lap time performance is presented, followed by a qualitative assessment of the performance.

5.3.1. Quantitative comparison

For each agent, 20 test laps are run on the AUT, ESP, GRB and MCO maps. Fig. 23 records the average success rates from each train-test combination.

The results in Fig. 23 show that the full planning agents learn the most general behaviour, achieving near 100% success on all the maps considered. The end-to-end agent achieves good performance on the AUT map and over 80% on the ESP map. The trajectory-tracking agent trained on the GBR map achieves below 60% and 40% success on the AUT and ESP maps, respectively. This behaviour indicates that the trajectory tracking architecture generalises less well to other maps and is more dependent on the training map used. The graph also indicates that the agents trained on the MCO map generalise to other tracks better than agents trained on the GBR track. We further study the performance by considering the lap times achieved by the agents trained on the MCO map on the different tracks.

Table 2 shows the lap times for the agents trained on the MCO map and tested on each of the four test maps. The classic planner

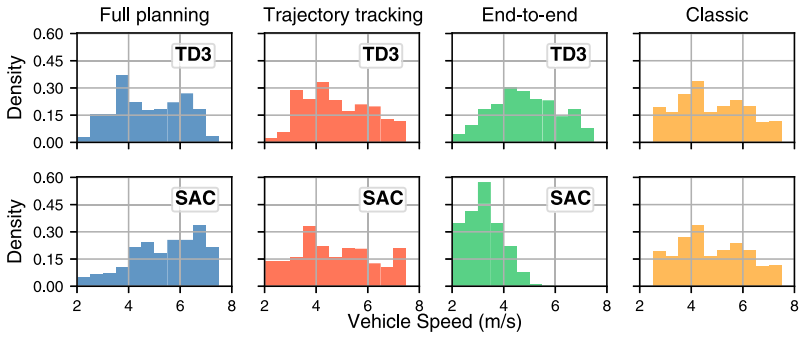


Fig. 22. The frequency of vehicle speeds experienced by agents trained with the SAC and TD3 algorithms.

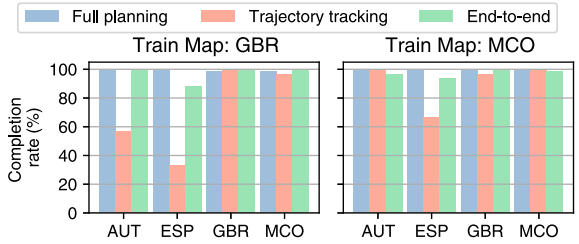


Fig. 23. Completion rates from testing vehicles trained on GBR and MCO maps on four tracks.

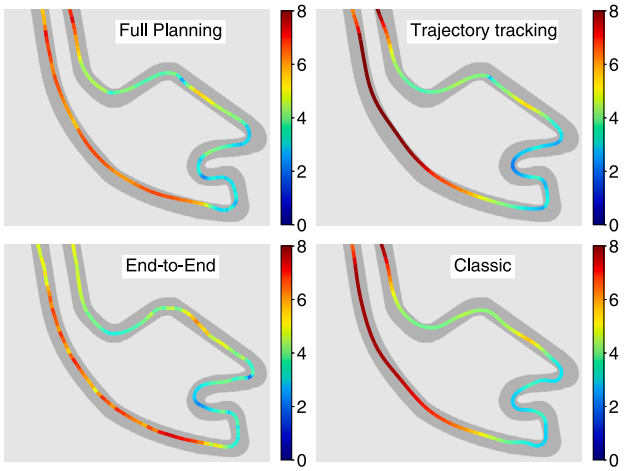


Fig. 24. Trajectories from full planning, trajectory tracking, end-to-end and classical planners. The colours represent the speeds in m/s.

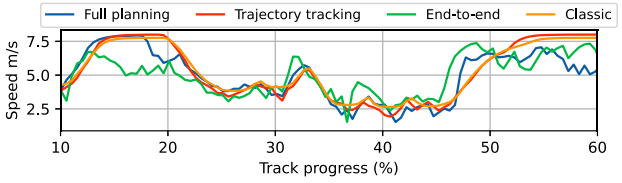


Fig. 25. Comparison of the speed profile on a section of the MCO track.

times are shown for reference and have a standard deviation of 0 since they do not depend on a random seed. On the training map, MCO, the trajectory tracking agent, achieves the fastest average lap time of 36.08 s, followed by the full planning agent with 37.85 s, and the end-to-end agent with 38.14 s. This indicates that the trajectory tracking agent achieves the best racing performance on the training map, probably due to tracking the optimal trajectory well (see Section 5.1). On the AUT and ESP maps, the full planning agent achieves the fastest times of 21.52 s and 48.19 s, respectively. Despite being the slowest on MCO (the training map), the end-to-end agent outperforms the trajectory tracking agent on both the AUT by 0.38 s and the ESP map by 0.56 s. This indicates that the full planning and end-to-end agents learn more general, transferable behaviour than the trajectory tracking agent. This suggests that having LiDAR beams in the state vector leads to policies that are robust to different maps.

5.3.2. Qualitative trajectory analysis

The reasons for results from the quantitative performance comparison are investigated by comparing the behaviour of each agent. The trajectory, speed profile and actions selected are used to represent the behaviour.

Fig. 24 shows the trajectories from the full planning, trajectory tracking, end-to-end and classical planners on a section of the MCO track. The trajectories show that the classic planner selects the smoothest trajectories speeding up in the straights and slowing down in the corners. The end-to-end trajectory contains no red sections indicating the agent never reaches the vehicle's maximum speed. The full planning and trajectory tracking trajectories roughly track the pattern on the classic planner speeding up in the straighter sections and slowing down in the corners.

Fig. 25 shows the speed profiles for a section of the MCO track. The classic planner smoothly speeds up to the maximum speed of 8 m/s, and slows down around the corner sections. The trajectory tracking and full planning agents also speed up to 8 m/s and sometimes have a higher speed than the classic planner. The end-to-end agent maintains a slow speed of around 5 m/s for most of the trajectory while still slowing down in the corner. All the agents select more jerky speed profiles than the classical planner.

We now consider the trajectories selected by the agents trained on the MCO map and tested on the ESP map. Fig. 26 shows trajectories of the agents trained on MCO and tested on ESP. The full planning and end-to-end agents show the reason for them achieving fast lap times being that they easily select high speeds, shown in dark red. The trajectory tracking agent selects a more smooth speed profile, similar to the classic planner. The trajectory tracking trajectory appears to come closer to the boundaries, as does the classic planner, possibly suggesting why the completion rate is lower. In contrast, the full planning and end-to-end agents remain closer to the middle of the track. The end-to-end agent trajectory appears jerky, with the vehicle slowing down around the sharp corner (the dark blue) and quickly speeding up again.

The policy generality study shows that the full planning architecture is the most robust to different maps, achieving nearly 100% completion on all the maps tested. While the trajectory tracking agent achieves the fastest lap times on the training map, it transfers poorly to other maps, achieving low completion rates and poor lap times. The end-to-end agent, which achieved the slowest lap times on the training

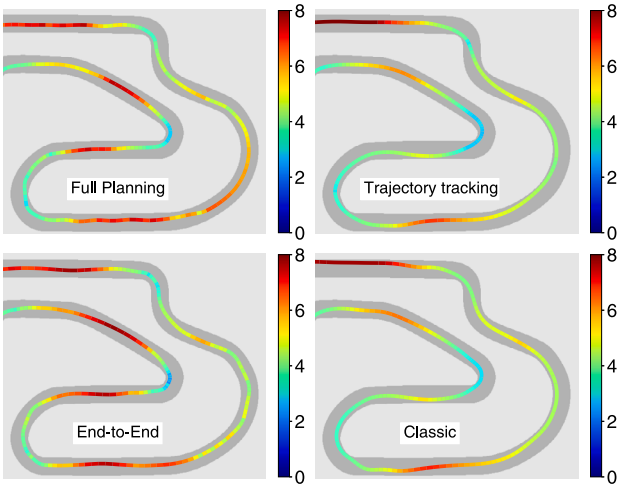


Fig. 26. Trajectories from full planning, trajectory tracking, end-to-end and classical planners tested on the ESP map.

map, outperformed the trajectory tracking agent on some of the test maps. These results demonstrate that including LiDAR beams in the state vector leads to improved generality. A qualitative investigation into the trajectories showed that the trajectory tracking agent tracks the classic planner well. The end-to-end agent selects jerky speed action, often slightly slower than the other planners. The full planning agent balances high performance and safety well, with fast trajectories that remain near the middle of the track.

6. Sim-to-real transfer

Simulation-to-reality transfer is the process of deploying an agent train in simulation on a physical robot. The sim-to-real gap is the difference between the simulation and real-world environments and is caused by a myriad of factors, including dynamics modelling differences, sensor noise, and delays on the physical hardware. Additionally, the real-world tests are normally run in a different domain to the training domain, i.e. in our context a different map. This gap makes it difficult to deploy agents on physical robots, because the agents perform differently, usually worse, to how they did in simulation. As a result, few studies study the difference in performance between simulated and real-world racing.

The agent's sim-to-real transfer is investigated by using the agents trained in simulation to control a physical F1Tenth vehicle. The results are compared to the F1Tenth ROS2 simulator for direct comparison using the same nodes and timing. The agents trained on the MCO map are used. The same scaling quantities are used, and a speed cap that clips speeds above v_{cap}, is added for safety. Tests are performed using a speed cap of 2 m/s, 3 m/s and 4 m/s. A straight corridor with a 90-degree bend is used, as shown in Fig. 27. The starting point is a fixed point with the vehicle's back touching the wall, and the finish line is once the vehicle has travelled 16.5 m in the x direction. The area is mapped using the ROS2 SLAM toolbox, the centre line is extracted and smoothed, and the optimal trajectory is calculated.

The sim-to-real transfer is investigated in the two categories of the steering angle selection and the speed selection. The steering angle investigation uses a low-speed cap of 1.5 m/s and compares the steering angles selected and resulting paths with their distances and curvatures. The speed investigation considers the lap times to complete the track at higher speeds of up to 5 m/s, the speed profiles and the trajectories selected.

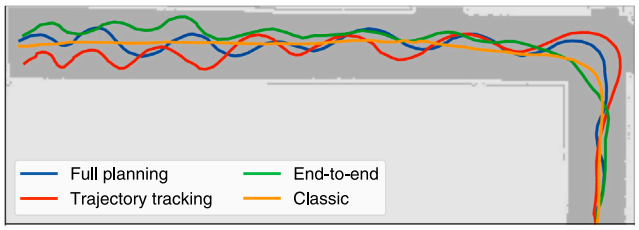


Fig. 27. The paths taken by the different on the physical vehicle with a speed cap of 2 m/s.

6.1. Steering investigation

The full planning, trajectory tracking and end-to-end agents are evaluated by transferring the agents trained in simulation on the MCO map to the physical vehicle. The training uses the SAC algorithm and the TAL reward signal. Using low speeds allows for the steering angles and the resulting paths to be studied in detail. Fig. 27 shows the different paths taken by the agents from the starting point in the bottom left corner to the end of the track.

In Fig. 27, the classic agent selects a smooth path of driving around the corner. The trajectory tracking agent swerves a low, regularly coming close to the boundaries. The end-to-end agent selects the smoothest path of the agents, with only a few small wiggles at the end. The swerving is quantified by the distance travelled and the mean path curvature.

Fig. 28 shows the distances and curvatures of the paths selected in the simulation and on the physical vehicle. The dots show the results from each of the five repetitions, the clear bars show the simulation results and the hatched results show the results from the physical vehicle. The full planning and trajectory tracking agents have longer distances on the physical vehicle compared to the simulation, while the end-to-end agent and classic planner have similar distances. The longer distances are explained by the higher mean curvatures of the full planning and trajectory tracking agents. Across both metrics the classic planner performs the best and out of the learning agents, the end-to-end agent has the most similar performance between simulation and reality.

Fig. 29 shows the steering angles selected by the agents in simulation and reality. In both simulation and reality, the classic path follower selects a smooth steering profile. In simulation, the end-to-end agent slaloms the most, followed by the planning agent. The trajectory tracking agent selects the smoothest steering angle profile of the learning agents. In reality, the opposite is observed, with the trajectory follower oscillating the most and end-to-end the least.

Fig. 30 shows the distribution of steering angles in simulation and on the physical vehicle. In simulation, the trajectory tracking agent has a majority of its actions having small steering angles. The full planning and end-to-end agents have well-distributed steering angles. On the physical vehicle, the trajectory tracking agent demonstrates a large difference with many steering angles being near to the maximum. The full planning and end-to-end agents also show a small increase in the number of large steering angles.

The steering study shows that the end-to-end agent, which has the worst quality paths in simulation, has the smallest sim-to-real gap. It is proposed that the reason for this is that the LiDAR scans are used directly as input to the agent, removing the effect of errors or delays in the localisation. Considering that all the agents perform worse on the real-vehicle indicates that there remains a difference between the simulated vehicle dynamics and the real-world dynamics. Future work could consider using a higher-order model in the simulator or using domain randomisation to ensure that the policies learned are robust.

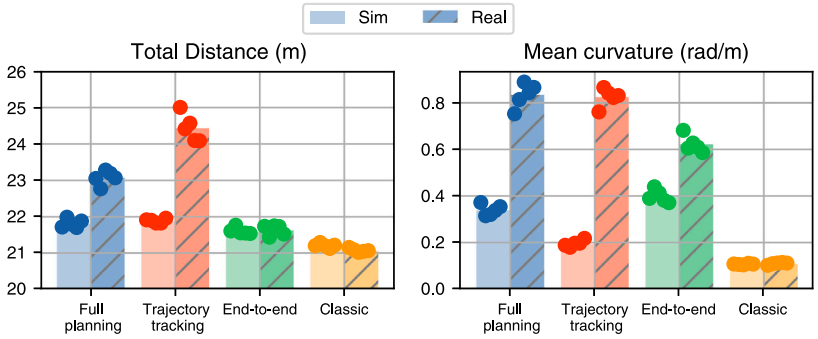


Fig. 28. Distances and mean curvatures from simulated and real experiments.

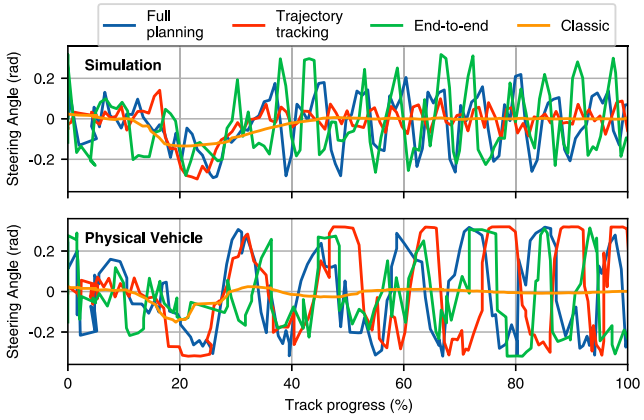


Fig. 29. Steering angles selected by agents in simulation (top) compared to the real vehicle (bottom).

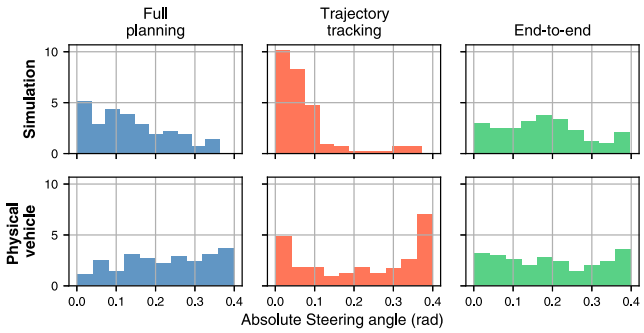


Fig. 30. Distribution of steering angles in simulation and reality for the full planning, trajectory tracking and end-to-end agents.

6.1.1. Delay investigation

Understanding simulation-to-reality transfer is difficult due to the large number of possible differences in the simulator compared to a physical vehicle. Options include delays in receiving perception updates, sensor noise, dynamics model mismatch and delays in executing control commands. To provide a further understanding of the differences, we present a study on the effect of delay on agent performance. We investigate the effect of delay in localisation, which on physical vehicles is a reality. In the real world, the LiDAR scanner runs at around 20 Hz and is then used by a particle filter that estimates the vehicle's position at around 10 Hz. We simulate this effect by giving the agent a location update, delayed by a certain amount of time.

Fig. 31 shows the impact of localisation delay on the total distance and mean curvature of the vehicles in simulation. In the tests, the actions are also delayed by a fixed 100 ms for all tests to represent the

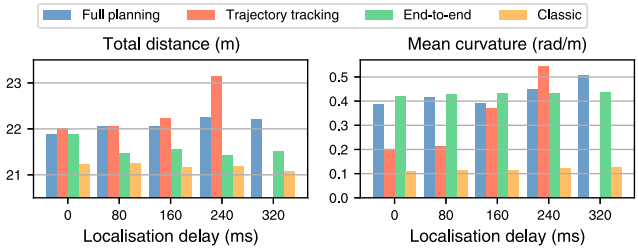


Fig. 31. The effect of localisation delay on the total distance and mean curvature of the vehicles.

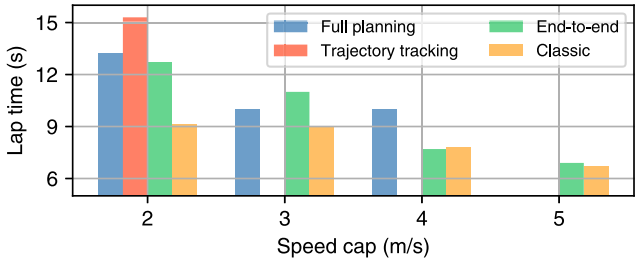


Fig. 32. Lap times using increasing speed caps. Missing bars mean that the agent was unable to complete the lap.

computation delay in executing steering commands. In the graphs, the end-to-end agent achieves similar total distances and mean curvatures for all the localisation delays used. This is expected since the end-to-end agent does not use the vehicle's localisation. The trajectory tracking agent has a low mean curvature of 0.2 rad/m when no localisation delay is present; this grows to 0.38 and then 0.55 when a 240 ms delay is introduced. The missing line at the 320 ms delay indicates that the trajectory tracking agent was no longer able to complete laps. The full planning agent shows a small decrease in performance as the delay increases but maintains a similar result across tests. The classic planner is highly robust to localisation delays, consistently achieving low distances and curvatures.

6.2. Maximum speed investigation

The impact of the speed cap is investigated by conducting tests with increasing speed caps. The lap times, speed profiles and trajectories are used to categorise the behaviour.

Fig. 32 shows that the end-to-end can complete laps faster at higher speeds. All the agents can complete laps at 2 m/s. Using a cap of 3 m/s or higher, the trajectory tracking agent is no longer able to complete

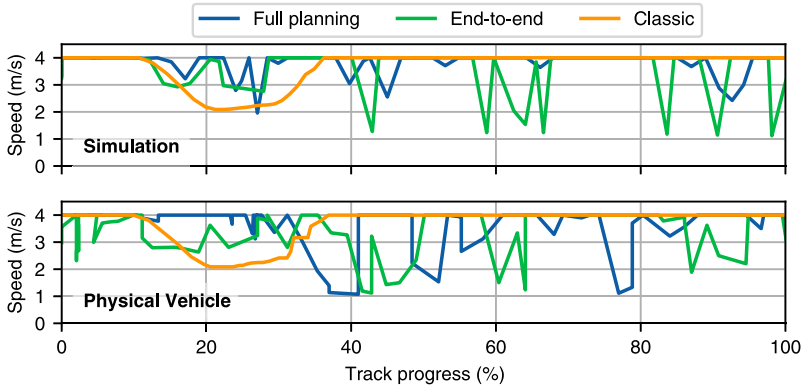


Fig. 33. Speed profiles for the full planning, end-to-end and classic planners in simulation and reality.

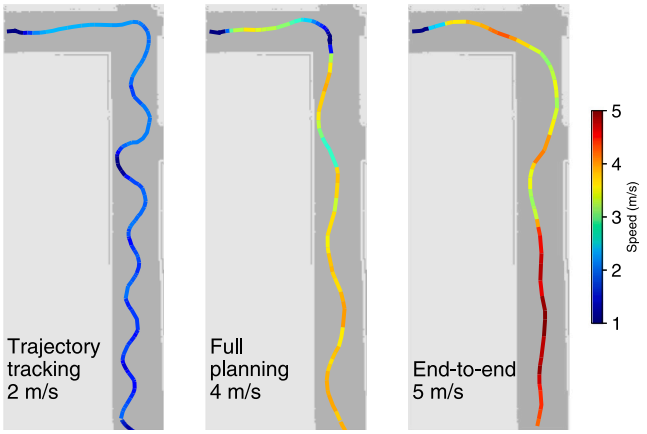


Fig. 34. Fastest trajectories from the trajectory tracking, full planning and end-to-end agents on the physical vehicle.

any laps due to crashing into the walls. The full planning agent can complete laps up to 4 m/s. The end-to-end and classic agents can complete laps using a speed cap of 5 m/s, indicating that they perform the best at high-speed racing.

Fig. 33 shows the speeds selected by the agents using a speed cap of 4 m/s. On the physical vehicle, the full planning agent selects slower speeds more regularly, possibly due to the increase in extreme steering actions. The end-to-end agent displays a similar speed profile selection in both simulation and reality with occasional spikes of low speeds.

Fig. 34 shows the fastest trajectories by the trajectory tracking, full planning and end-to-end agents. The trajectory tracking agent's extreme swerving is a key problem limiting the agent's performance. The full planning agent swerves less, enabling success at 4 m/s. The end-to-end agent can smoothly control the vehicle at high speeds, resulting in a fast trajectory with a speed cap of 5 m/s. The end-to-end trajectory demonstrates that the agent can select a feasible speed profile that transfers to high-speed racing on a physical vehicle.

The sim-to-real investigation studied how the DRL agents trained in simulation transferred to a physical vehicle. In the study on steering, the end-to-end agent had the smallest difference between simulation and reality in terms of the distance travelled and the mean curvature. In contrast, the full planning and, specifically, the trajectory tracking architectures transferred poorly, having high curvature and selecting large steering angles. The study on delay indicated that a key reason for this was that the end-to-end agent did not require localisation while the other architectures did. The study on speed showed the impact of the curvature trajectories was that trajectory tracking architecture could only complete the track with a maximum speed of 2 m/s and the full planning agent with a maximum speed of 4 m/s. In contrast, the end-to-end agent could complete the track with a maximum speed of 5 m/s, achieving a similar result to the classic planner.

7. Conclusion

This paper compared the full planning, trajectory tracking and end-to-end DRL architectures for autonomous racing. End-to-end agents use the raw LiDAR scan and vehicle speed. Full planning agents use the LiDAR scan, upcoming centre line points and the vehicle's state variables. Trajectory tracking agents use upcoming trajectory points and the vehicle's state variables. The simulation results showed that the full planning agent is the most robust to reward signal, training algorithm and test maps. The end-to-end agent generally achieves slower lap times, sometimes caused by not selecting high speeds. The trajectory tracking agent can achieve the fastest lap times on the training map, and tracks the optimal trajectory closely, but is brittle to changes in reward signal and test map.

The policies were tested on a physical vehicle to evaluate the sim-to-real transfer. The study on the steering actions and resulting paths showed that the full planning and trajectory tracking agents swerved excessively. The end-to-end agent had the most similar performance between the simulation and physical vehicle, swerving less, travelling a shorter distance and having a lower curvature. Further investigation showed that the end-to-end agent selected less extreme steering actions compared to the full planning and trajectory tracking agents. The study on speed showed that the trajectory tracking agent could complete laps with a maximum speed of 2 m/s, the full planning agent 4 m/s and the end-to-end agent 5 m/s. The good performance of end-to-end agents demonstrates their advantage of not relying on other systems, limiting the difference between simulation and reality. However, even with end-to-end architectures, simulation-to-reality transfer is still difficult and requires further study.

Future work should further explore sim-to-real transfer for DRL agents to physical vehicles and how all three architectures can be transferred to physical vehicles. Domain randomisation and adaptation (Carr, Chli, & Vogiatzis, 2018) could help train more robust policies or a more complex model could be used in the simulator to minimise the differences. The key limitation to be addressed is the extreme swerving displayed by the trajectory tracking agent and seen in other studies (Brunnbauer et al., 2022). Investigating the sensor noise, latency, and compute requirements will provide explanations, and domain randomisation and adaptation can possibly improve performance.

Acknowledgment

The vehicle that was used within the practical experiments was constructed in conjunction between Stellenbosch University and FH Aachen.

CRediT authorship contribution statement

Benjamin David Evans: Conceptualization, Methodology, Software, Formal analysis, Investigation, Data curation, Writing – original draft, Visualization. **Hendrik Willem Jordaan:** Supervision, Resources, Writing – review & editing, Project administration. **Herman Arnold Engelbrecht:** Supervision, Writing – review & editing, Funding acquisition.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

Data will be made available on request.

References

Becker, J., Imholz, N., Schwarzenbach, L., Ghignone, E., Baumann, N., & Magno, M. (2023). Model-and acceleration-based pursuit controller for high-performance autonomous racing. In *2023 IEEE international conference on robotics and automation* (pp. 5276–5283). IEEE.

Betz, J., Zheng, H., Liniger, A., Rosolia, U., Karle, P., Behl, M., et al. (2022). Autonomous vehicles on the edge: A survey on autonomous vehicle racing. *IEEE Open Journal of Intelligent Transportation Systems*.

Bosello, M., Tse, R., & Pau, G. (2022). 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* (pp. 290–298). IEEE.

Brunnbauer, A., Berducci, L., Brandstatter, A., Lechner, M., Hasani, R., Rus, D., et al. (2022). Latent imagination facilitates zero-shot transfer in autonomous racing. In *2022 International conference on robotics and automation* (pp. 7513–7520).

Cai, P., Mei, X., Tai, L., Sun, Y., & Liu, M. (2020). High-Speed Autonomous Drifting with Deep Reinforcement Learning. *IEEE Robotics and Automation Letters*, *5*(2), 1247–1254.

Cai, P., Wang, H., Huang, H., Liu, Y., & Liu, M. (2021). Vision-based autonomous car racing using deep imitative reinforcement learning. *IEEE Robotics and Automation Letters*, *6*(4), 7262–7269.

Carr, T., Chli, M., & Vogiatzis, G. (2018). Domain adaptation for reinforcement learning on the atari. arXiv preprint arXiv:1812.07452.

Chisari, E., Liniger, A., Rupenyan, A., Van Gool, L., & Lygeros, J. (2021). Learning from simulation, racing in reality. In *2021 IEEE international conference on robotics and automation* ICRA, (pp. 8046–8052). IEEE.

Christ, F., Wischnewski, A., Heilmeier, A., & Lohmann, B. (2021). Time-optimal trajectory planning for a race car considering variable tyre-road friction coefficients. *Vehicle System Dynamics*, *59*(4), 588–612.

Coulter, R. C. (1992). *Implementation of the pure pursuit path tracking algorithm: Technical report*, Carnegie-Mellon UNIV Pittsburgh PA Robotics INST.

Dwivedi, T., Betz, T., Sauerbeck, F., Manivannan, P., & Lienkamp, M. (2022). Continuous control of autonomous vehicles using plan-assisted deep reinforcement learning. In *2022 22nd international conference on control, automation and systems* (pp. 244–250). IEEE.

Evans, B., Betz, J., Zheng, H., Engelbrecht, H. A., Mangharam, R., & Jordaan, H. W. (2022). Accelerating online reinforcement learning via supervisory safety systems. arXiv preprint arXiv:2209.11082.

Evans, B. D., Engelbrecht, H. A., & Jordaan, H. W. (2023). High-speed autonomous racing using trajectory-aided deep reinforcement learning. *IEEE Robotics and Automation Letters*, *8*(9), 5353–5359.

Fuchs, F., Song, Y., Kaufmann, E., Scaramuzza, D., & Durr, P. (2021). Super-human performance in gran turismo sport using deep reinforcement learning. *IEEE Robotics and Automation Letters*, *6*(3), 4257–4264.

Fujimoto, S., Hoof, H., & Meger, D. (2018). Addressing function approximation error in actor-critic methods. In *International conference on machine learning* (pp. 1587–1596). PMLR.

Ghignone, E., Baumann, N., & Magno, M. (2023). Tc-driver: A trajectory conditioned reinforcement learning approach to zero-shot autonomous racing. *Field Robotics*, *3*(1), 637–651.

Haarnoja, T., Zhou, A., Hartikainen, K., Tucker, G., Ha, S., Tan, J., et al. (2018). Soft actor-critic algorithms and applications. arXiv preprint arXiv:1812.05905.

Hamilton, N., Musau, P., Lopez, D. M., & Johnson, T. T. (2022). Zero-shot policy transfer in autonomous racing: Reinforcement learning vs imitation learning. In *2022 IEEE international conference on assured autonomy* ICAA, (pp. 11–20). IEEE.

Heilmeier, A., Wischnewski, A., Hermansdorfer, L., Betz, J., Lienkamp, M., & Lohmann, B. (2020). Minimum curvature trajectory planning and control for an autonomous race car. *Vehicle System Dynamics*, *58*(10), 1497–1527.

Ivanov, R., Carpenter, T. J., Weimer, J., Alur, R., Pappas, G. J., & Lee, I. (2020). 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* (pp. 1–7).

Jaritz, M., De Charette, R., Toromanoff, M., Perot, E., & Nashashibi, F. (2018). End-to-end race driving with deep reinforcement learning. In *2018 IEEE international conference on robotics and automation* ICRA, (pp. 2070–2075). IEEE.

Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., et al. (2015). Continuous control with deep reinforcement learning. arXiv preprint arXiv:1509.02971.

O'Kelly, M., Zheng, H., Jain, A., Auckley, J., Luong, K., & Mangharam, R. (2020). Tunercar: A superoptimization toolchain for autonomous racing. In *2020 IEEE international conference on robotics and automation* (pp. 5356–5362). IEEE.

O'Kelly, M., Zheng, H., Karthik, D., & Mangharam, R. (2020). F1tenth: An open-source evaluation environment for continuous control and reinforcement learning. *Proceedings of Machine Learning Research*, *123*.

Orgován, L., Bécsi, T., & Aradi, S. (2021). Autonomous drifting using reinforcement learning. *Periodica Polytechnica Transportation Engineering*, *49*(3), 292–300.

Remonda, A., Krebs, S., Veas, E., Luzhnica, G., & Kern, R. (2021). Formula RL: Deep reinforcement learning for autonomous racing using telemetry data. arXiv preprint arXiv:2104.11106.

Sutton, R. S., & Barto, A. G. (2018). *Reinforcement learning: An introduction*. MIT Press.

Tătulea-Codrean, A., Mariani, T., & Engell, S. (2020). Design and simulation of a machine-learning and model predictive control approach to autonomous race driving for the F1/10 platform. *IFAC-PapersOnLine*, *53*(2), 6031–6036.

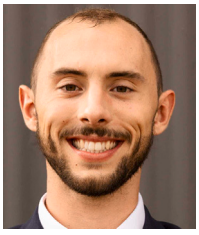
Vianna, L. C. M., Goubault, E., & Putot, S. (2021). Neural network based model predictive control for an autonomous vehicle. arXiv e-prints, arXiv:2107.14573.

Walsh, C. H., & Karaman, S. (2018). Cddt: Fast approximate 2d ray casting for accelerated localization. In *2018 IEEE international conference on robotics and automation* ICRA, (pp. 3677–3684). IEEE.

Wang, R., Han, Y., & Vaidya, U. (2021). Deep koopman data-driven control framework for autonomous racing. In *Proc. int. conf. robot. autom.(icra) workshop opportunities challenges auton. racing* (pp. 1–6).

Wischnewski, A., Geisslinger, M., Betz, J., Betz, T., Fent, F., Heilmeier, A., et al. (2022). Indy autonomous challenge-autonomous race cars at the handling limits. In *12th International Munich Chassis symposium 2021* (pp. 163–182). Springer.

Wurman, P. R., Barrett, S., Kawamoto, K., MacGlashan, J., Subramanian, K., Walsh, T. J., et al. (2022). Outracing champion gran turismo drivers with deep reinforcement learning. *Nature*, *602*(7896), 223–228.



Benjamin David Evans studied for a bachelor's in Mechatronic Engineering at the University of Stellenbosch. He went on to complete his Ph.D. in the field of deep reinforcement learning for autonomous racing. His interests include the intersection of classical control and machine learning for autonomous systems. He is a postdoctoral researcher at Stellenbosch University, focusing on machine learning applications in high-performance control.

Hendrik Willem Jordaan received his bachelor's in Electrical and Electronic Engineering with Computer Science and continued to receive his Ph.D degree in satellite control at Stellenbosch University. He currently acts as a senior lecturer at Stellenbosch University and is involved in several research projects regarding advanced control systems as applied to different autonomous vehicles. His interests include robust and adaptive control systems applied to practical vehicles.

Prof. Herman Engelbrecht received his Ph.D. degree in Electronic Engineering from Stellenbosch University (South Africa) in 2007. He is currently the Chair of the Department of Electrical and Electronic Engineering. His research interests include distributed systems (specifically infrastructure to support massive multi-user virtual environments) and machine learning (specifically deep reinforcement learning). Prof Engelbrecht is a Senior Member of the IEEE and a Member of the ACM.