Contents lists available at ScienceDirect

Cognitive Robotics

journal homepage: http://www.keaipublishing.com/en/journals/cognitive-robotics/

Safe reinforcement learning for high-speed autonomous racing

Benjamin D. Evans*, Hendrik W. Jordaan, Herman A. Engelbrecht

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

ARTICLE INFO

Keywords:
Reinforcement learning
Safe learning
Autonomous racing
Safe autonomous systems

ABSTRACT

The conventional application of deep reinforcement learning (DRL) to autonomous racing requires the agent to crash during training, thus limiting training to simulation environments. Further, many DRL approaches still exhibit high crash rates after training, making them infeasible for real-world use. This paper addresses the problem of safely training DRL agents for autonomous racing. Firstly, we present a Viability Theory-based supervisor that ensures the vehicle does not crash and remains within the friction limit while maintaining recursive feasibility. Secondly, we use the supervisor to ensure the vehicle does not crash during the training of DRL agents for high-speed racing. The evaluation in the open-source F1Tenth simulator demonstrates that our safety system can ensure the safety of a worst-case scenario planner on four test maps up to speeds of 6 m/s. Training agents to race with the supervisor significantly improves sample efficiency, requiring only 10,000 steps. Our learning formulation leads to learning more conservative, safer policies with slower lap times and a higher success rate, resulting in our method being feasible for physical vehicle racing. Enabling DRL agents to learn to race without ever crashing is a step towards using DRL on physical vehicles.

1. Introduction

Deep reinforcement learning (DRL) is a growing method for autonomous racing since agents can be trained to learn end-to-end policies that map raw LiDAR scans directly to control commands [1,2]. DRL algorithms train agents from experience, consisting of positive and negative states and actions with corresponding rewards, thus requiring the agent to crash during training [3,4]. Requiring agents to crash during training limits training to simulation environments where the vehicle can crash without consequences [5]. Training agents in simulation before transferring them to real-world vehicles, called the sim-to-real problem, causes worse performance in reality compared to the simulation due to the difference in dynamics [6].

Safe learning is a developing field in control systems that uses a supervisor to ensure robot safety during training [7]. In control system problems, the constraints are usually static constraints on a state variable or a safe set is available [8]. In racing, it is difficult to build a supervisor since the only input data is an occupancy grid of the track and the safe states must remain within the friction limit and be recursively feasible [9]. Approaches to safety in autonomous driving include human intervention [10,11], time-to-collision (TTC) [12] and reachability theory [13]. These methods have been applied in low-performance contexts and do not scale to high-speed racing.

In this work, we address the problem of training DRL agents to race at high speed while ensuring safety during the training process, i.e. training without crashing. We present a Viability Theory-based supervisor for ensuring high-performance safety, that is used to train agents for simulated F1Tenthracing at speeds of up to 6 m/s. We summarise our contributions as:

[*] Corresponding author.
E-mail address: bdevans@sun.ac.za (B.D. Evans).

https://doi.org/10.1016/j.cogr.2023.04.002
Received 26 January 2023; Received in revised form 23 March 2023; Accepted 11 April 2023
Available online 22 April 2023

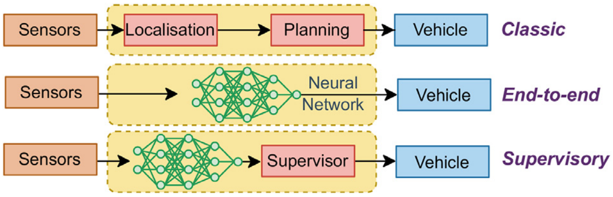


Fig. 1. Classical, end-to-end and supervisory architectures for autonomous racing.

- We expand on previous supervisory safety systems to ensure safe behaviour for high-speed racing, ensuring recursively feasible within the friction limits.
- We demonstrate that training with the supervisor significantly improves sample efficiency, requiring only 10k steps.
- We demonstrate that agents trained with the safety system select more conservative trajectories than the baseline leading to slower lap times but a higher success rate.

This article is organised as follows: §2 studies related work related to autonomous racing and safe learning methods. The supervisory architecture is described in §3, followed by the description of the safe set generation in §4 and supervisor evaluation in §5. §6 describes how the supervisor is incorporated into the learning and presents the results from the safe learning evaluation.

2. Literature study

We study work related to autonomous racing and safe learning methods. Figure 1 shows how classical architectures use a localisation and planning pipeline, end-to-end architectures replace the pipeline with a neural network and supervisory architectures use a supervisor to monitor the network output.

2.1. Autonomous racing

Current approaches to autonomous racing are categorised according to model-based controllers and end-to-end learning methods. Autonomous racing approaches have focused on two main vehicle classes, full-sized vehicles [14], and miniature cars such as 1:10 [15], 1:12 [16], 1:18 [17] and 1:43 [18]. While there is an overlap between methods, we focus on miniature vehicle racing.

2.1.1. Classical racing

The classical method of autonomous racing is to plan an optimal trajectory before the race begins and then during the race to use a perception (localisation), planning and control pipeline to follow the plan [1,19]. Trajectory optimisation uses a vehicle dynamics model to plan an optimal path of coordinates and generate a speed profile for the path [20]. Following the trajectory has been done using geometric controllers such as the pure pursuit algorithm [21] and model-predictive control [22].

Classical racing methods generally produce good results that push the vehicles to their operating limits [18,21]. In F1Tenthracing model predictive control (MPC) [22] and pure pursuit planners [21] have been used to drive vehicles at high-performance levels, reaching speeds of up to 7 m/s [23].

A key limitation in model-based control methods is that they require localisation during the race. The main method of localisation in F1Tenthracing uses the particle filter presented in [24] and is used in [21,23,25]. The other method of localisation is to use a motion capture system (set of cameras that track a marker on the vehicle) such as in [18,26]. Both of these methods are limited to requiring a map of the track or a set of cameras and the hardware requirement for real-time localisation.

2.1.2. End-to-end racing

Deep learning methods have developed solutions for racing that do not require a map of the environment [5]. Deep learning methods use a neural network to map a state vector containing raw data, usually a LiDAR scan, directly to control references, i.e. steering and speed commands. While both imitation learning [11] and reinforcement learning [12] have been used to train agents, reinforcement learning has been shown to outperform imitation learning [2]. While many deep-learning approaches have been used to replace classical components, such as path following [26], we focus on end-to-end approaches that use raw sensor data as input.

The typical method for training deep reinforcement learning (DRL) agents to race is to train them in simulation and then transfer the trained policy to a physical vehicle [27]. Hamilton et al. [2] use a subset of the LiDAR scan as input into the agent and the output is the steering angle. They train the agent in simulation and then deployed it to a physical vehicle. Sim-to-real transfer is a difficult problem [27], that is currently necessitated due to sample inefficiency of DRL algorithms and safety concerns during training [6].

Training agents for autonomous racing is difficult resulting in solutions being simplified or requiring additional processing. Hamilton et al. [2] only used the agent to select steering actions, driving the racing car at a single speed of 1.5 m/s. Brunnbauer et al. [5] claimed that autonomous racing (selecting speed and steering references) was too difficult for model-free learning, and thus resorted to model-based RL. Despite the improvements, including a distance reconstruction as input to the agent, the results showed that their final agents still crashed within around 2 laps. Zhang et al. [28] simplified the problem to only learning a residual policy to complement an artificial potential field planner. These studies show that learning to select speed and steering references is a difficult task.

Table 1
Safe learning methods for robotic control and their limitations.

Method	Advantages	Limmitations
Probabilistic safety [29,30]	Does not require dynamics model	No hard constraint satisfaction
Control barrier functions [31,32]	Hard constraint satisfaction	Needs a recursively feasible safe set
Constraint admissible set [8]	Can generate recursively feasible set	Limited to linear systems
Hamilton-Jacobi reachability [33,35]	Suitable for non-linear dynamics	Requires a static constraint set

2.2. Safe learning methods

Safe learning is concerned with training DRL agents while preventing the agents from failing during training [7]. We study advances in safe learning for control systems, the requirements of safety in racing and current safe learning approaches in autonomous racing.

2.2.1. Safe learning for control systems

Safe learning in robotics is a growing field that is concerned with training agents to select actions while ensuring robot safety.

Probabilistic methods of ensuring safety result in a high probability of not violating the constraints [7]. Probabilistic constraints have been implemented using learning-based predictive models to estimate the probability of the agent crashing [29,30]. While these methods take steps towards improving safety in learning, they are not suitable for racing, since any chance of a vehicle crashing into the barriers renders the method infeasible. Therefore, solutions must be able to guarantee hard constraint satisfaction.

Control barrier functions (CBFs) have been used to enforce safety limits within a safe set, rather than a static limit [31,32]. CBFs are a part of control theory concerned with selecting control inputs that keep a system with a forward-invariant safe set. While CBFs achieve their desired aim, these methods assume that forward-invariant (recursively feasible) safe sets are available. It is difficult to calculate a forward-invariant safe set for domains with complex dynamics.

Constraint-admissible sets have been used to generate safe sets for linear systems [8]. Their method takes a constraint matrix as the input to calculate a set of the state space that is safe. This method was evaluated on a cruise control system and shown to be effective in enforcing state constraints on linear dynamics systems. While this result demonstrates the advantage of safe learning, it is not applicable to non-linear systems.

Safety in non-linear systems has been approached using Hamilton-Jacobi reachability analysis [33,34]. One of the latest papers in this direction presents a general method for ensuring the safety of non-linear systems with uncertain dynamics [35]. These methods propose using reachability theory to calculate the set of safe states that remain recursively feasible and avoid collisions. However, the form of the constraints is static bounds on state variables, such as quad-rotor height [35], or distance between two planes [34]. This is not directly applicable to racing where the constraint is in the form of an occupancy grid.

Table 1 summarises the different methods of safe learning for robotic control. Probabilistic methods do not provide hard constraint guarantees, and control barrier functions require recursively feasible safe sets. Methods to build linear and non-linear safe sets are limited by requiring static constraints on state variables which are not feasible for racing.

2.2.2. Safety constraints in racing

The requirements of a safety system for autonomous racing are now considered. A safety system for racing must ensure hard constraint satisfaction with a guarantee of recursive feasibility for high-performance systems. The only available input to a racing safety system is an occupancy grid of the track and a dynamics model for the vehicle. In addition to not crashing, racing solutions must keep the vehicle within the friction constraints since safely controlling a vehicle that is drifting is difficult due to the tyre’s non-linearities.

A safety approach for classical autonomous driving [36] and racing [37] is to ensure the safety of a trajectory with a finite planning horizon. This method evaluates the safety of the trajectory by checking if it intersects the track boundary at any point. This method is feasible for using an occupancy grid to check the safety of a trajectory. This method does not explicitly consider recursive feasibility because it ensures the safety of a trajectory with a long planning horizon and assumes that the last state (at the end of the horizon) is feasible. This approach is not suitable for safety in learning methods since an entire trajectory is not available, rather only a state and action that must be checked for safety.

Fraichard et al. [38] define an inevitable collision state as a state for which irrespective of the future trajectory followed a collision will occur. For example, a vehicle travelling at high speed with a wall immediately ahead will collide no matter what further actions are implemented. Reachability analysis has been used to calculate the inevitable collision states for robots avoiding obstacles [39] and operating on unknown maps [40]. Reachability Theory is used to generate a set of states that the agent could pass through (reach) in the future. If the set of reachable states includes states that are not inevitable collision states, then the state is safe The major limitation of methods using Reachability Theory is the extensive real-time computation requirement of calculating possible future states, which renders these approaches infeasible for high-speed racing.

Liniger et al. [41] used Viability Theory to improve the performance of model predictive controllers. They present a method to calculate a viability kernel of all the recursively feasible states that the vehicle can be in for a specific track. The safe set contains all the states that are not inevitable collision states, i.e. in every safe state, there exists a recursively feasible action that leads to a safe state. This set is then used to prune unsafe trajectory states in their MPC search. While they do not focus on the safety aspect of this

Table 2
Methods of ensuring machine learning controller safety for autonomous vehicles.

Method	Advantages	Limitations
Human intervention [11,43]	No calculation required	Inherently not autonomous
Time-to-collision and reversing [12]	Easy to calculate	Poor safety approximation
Reachability methods [13]	No precomputation needed	High online computation required
Neural network verification [44]	Formal safety guarantee	Time intensive, no corrective action

set, we propose adapting their work for use in racing vehicle supervisory systems. A major advantage of their method is that they can compute the viability kernel offline before the race and then use a hash table to determine if a state is safe or unsafe during the race.

2.2.3. *Safe learning for autonomous vehicles*

Methods of safe learning for autonomous vehicles address the question of how to ensure that machine learning components do not cause the vehicle to crash. A naive approach to ensuring safety is to use a human to monitor the system and intervene if the system appears to act unsafely [42]. This approach has been applied to trained agents for full-size [43] and miniature vehicle [11] racing. However, using a human in the loop does not achieve the aim of autonomy since a human is inherently required. Additionally, it is difficult for a human to ensure that the vehicle behaves correctly at high speeds.

In order to bypass the sim-to-real gap, Bosello et al. [12] trained an agent on board a physical vehicle. They used a supervisory system that calculated the time-to-collision (TTC) and if it was below a threshold, the vehicle would reverse for a fixed time interval, before the learning continued. They tested their method on a simple circular track where they trained an agent to drive at a fixed speed in 3 hours. While they demonstrate the advantage of onboard training, using the TTC is an over-approximation of vehicle safety. This over-approximation will scale poorly to high-speed racing, where the boundary of safety is needed.

Musau et al. [13] used reachability analysis to ensure the safety of IL and RL controllers for F1Tenthracing. They noted that verification and validation are impossible tasks for “black box” neural networks, and thus a safety system is critical for ML safety. They propose a reachability algorithm that simulates the vehicle’s state and action in the future according to a reach-time parameter. If the simulated state does not collide with the track boundaries, then the action is marked safe. While they demonstrate the success of this approach on a physical vehicle travelling at constant speeds of 0.5, 1 and 1.5 m/s, they note its limitation of requiring intensive real-time computation. This does not scale to higher speeds, where more calculations are required in less time. Additionally, they note that their method does not ensure recursive feasibility, since it only ensures safety for a finite lookahead period.

The authors of [44] proposed using formal methods to ensure that a racing neural network controller performs safely. While they can validate if a controller is safe or not, this approach does not provide any method to correct the problem if the verification fails. This is a fundamental problem for training agents since at the beginning of training, the agent will not select safe actions and will require correction.

The problem of training agents safely for autonomous driving has been noted due to safety concerns, and the desire to bypass the sim-to-real gap. Table 2 contains a summary of safe learning methods for autonomous vehicles. While neural network verification can be a useful tool for validating trained networks, it is not feasible for guaranteeing safety during the training process. Supervisors using human intervention undermines the aim of autonomous systems. Using the time-to-collision and reachability methods are applicable to low, constant speed driving, but these solutions do not scale to fast, high-performance vehicles. Therefore, we approach the problem of designing a safety system for high-performance racing and using it to safely train DRL agents.

3. Supervisory architecture

We present our supervisory safety system for high-speed autonomous racing. The safety system aims to ensure that only safe actions are implemented on the vehicle. We define safety as selecting actions that do not cause the vehicle to crash or breach the friction limit (drift). Safe actions must be recursively feasible, meaning that if a safe action is implemented, a safe action will exist for the resulting state.

Figure 2 shows the role of the supervisor in ensuring that only safe actions are implemented on the vehicle. The supervisor receives the vehicle’s state (pose) $\mathbf{x}_t$, which is available from a particle filter, and the agent’s action $\mathbf{u}_0$ as input. The supervisor outputs a safe action $\mathbf{u}_{\text{safe}}$, that can be implemented on the vehicle.

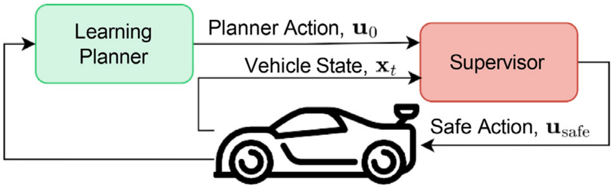

Learning Planner
Planner Action, $\mathbf{u}_0$
Vehicle State, $\mathbf{x}_t$
Supervisor
Safe Action, $\mathbf{u}_{\text{safe}}$

Fig. 2. The supervisor is positioned after the learning agent to ensure that only safe actions are implemented.

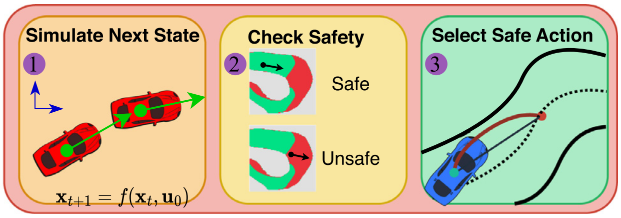


Fig. 3. The supervision process of (1) simulating the next state, (2) checking if the next state is safe, and (3) selecting a safe action.

The supervisor requires a dynamics model to simulate the next vehicle state and a set of safe states that meet the safety criterion and are recursively feasible. The system uses the planning timestep T_p. The dynamics model f is in the form of $\mathbf{x}_{t+1} = f(\mathbf{x}_t, \mathbf{u})$, that takes the current state $\mathbf{x}_t$ and action $\mathbf{u}$, and returns the next state $\mathbf{x}_{t+1}$. The state space is discretised into a finite list of states $\mathcal{X}$. The set of safe states $\mathcal{X}_\mathrm{safe}$ is a subset of the state space $\mathcal{X}_\mathrm{safe} \subset \mathcal{X}$.

Figure 3 shows the three-step process, used by the supervisor, to ensure safety. Firstly, the effect of the action from the planner is simulated on the vehicle pose, using the dynamics model $f(\mathbf{x}_t, \mathbf{u}_0)$, to predict where the vehicle will be at the next planning step $\mathbf{x}_{t+1}$. Secondly, a check is done to see if the next state is in the safe set or not. Thirdly, if the next state is safe, the original action is returned, and if the next state is unsafe, a safe action is returned. Using this process ensures that only safe actions are implemented on the vehicle. We now consider how the safe set for high-speed racing is generated.

4. Safe set generation

The supervisory architecture presented above requires a safe set to be generated. The safe set must contain all the states that do not cause the vehicle to crash and keep the vehicle within the friction limit while remaining recursively feasible. This section explains how Viability Theory is used to calculate this safe set for high-speed racing.

4.1. Viability theory

We consider a system that has a state, $\mathbf{x} \in \mathbb{R}^n$, and control inputs, $\mathbf{u} \in \mathbf{U} \subset \mathbb{R}^m$. Transitions between states are given by the continuous function $f \ : \ \mathbb{R}^n \times \mathbf{U} \mapsto \mathbb{R}^n$, such that $\mathbf{x}_{k+1} = f(\mathbf{x}_k, \mathbf{u}_k, T_\mathrm{p})$, where the subscript k is the time-step, and T_p is the planning timestep. The differential dynamics are converted to a set-valued map (difference inclusion) $F(\mathbf{x})$, which is the set of all possible next states for a given initial state.

$$\begin{aligned} \mathbf{x}_{k+1} \in \quad & F\left(\mathbf{x}_k\right), \quad \text{with} \\ & F\left(\mathbf{x}_k\right) = \left\{ f\left(\mathbf{x}_k, \mathbf{u}, T_\mathrm{p}\right) \, \middle| \, \mathbf{u} \in \mathbf{U} \right\}. \end{aligned} \tag{1}$$

In Equation 1, $F(\mathbf{x}_k)$ 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 time-step, T_p.

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 $K_\mathrm{input} \subset \mathbb{R}^n$, solutions to the difference inclusion in Equation 1, which stay in K forever, are known as viable solutions. The kernel of safe states $\mathcal{X}_\mathrm{safe}$ is a subset of the discrete state space $\mathcal{X}$, for which there exists a safe action, that can be calculated recursively using viability kernel algorithm,

$$\begin{aligned} K^0 \quad &= K_\mathrm{input} \\ K^{i+1} &= \left\{\mathbf{x}_k \in K^i \mid \forall \, F\left(\mathbf{x}_k\right) \cap K^i \neq \phi\right\}. \end{aligned} \tag{2}$$

The viability kernel algorithm generates a set of states for which there recursively exists an action that causes the vehicle to remain within the kernel. The algorithm uses the original constraint set K_input as the initial safe set (kernel) k^0. The algorithm then recursively generates smaller safe sets by looping through each safe state from the previous iteration (denoted by i) and including only states for which there exists an action that leads to another safe state. Formally, this process is defined as selecting states for which the intersection of the next states and the kernel is not equal to the empty set. The kernel generation results in an n-dimensional kernel of recursively feasible safe states $\mathcal{X}_\mathrm{safe}$.

4.2. Vehicle model

Racing vehicles are commonly modelled using the bicycle model to represent the car's motion. Figure 4 shows how the variables of X and Y are measured in the global coordinate frame. The angle θ is the vehicle's orientation relative to the x-axis and the angle δ is the steering angle of the front wheels relative to the vehicle orientation. The vehicles longitudinal speed v is measured in the direction of vehicle orientation.

The state variables are discretised into a finite number of discrete states. For a given racing track, the track is split into a number of uniformly sized blocks in the x and y directions. The orientation angle is split into n_θ equally spaced angle segments. The discretisation of the speed and steering angle is discussed in §4.3. From now on, all variables are considered to be discrete.

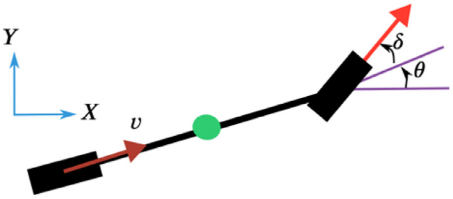


Fig. 4. Bicycle model showing the X and Y position at the centre of gravity, speed v, orientation θ, and steering angle δ.

4.2.1. State update equations

A set of dynamics equations are required to calculate state transitions on the planning time step as $\mathbf{x}_{t+1} = f(\mathbf{x}_t, \mathbf{u}, T_\mathrm{p})$. The vehicle state is recorded as $\mathbf{x}_t = [X, Y, \theta, v, \delta]$. The control action is a speed and steering angle reference, $\mathbf{u} = [v, \delta]$. The car, being a physical system, has dynamic limits on acceleration and steering angle velocity.

While the bicycle model is used to model the state variables that form the kernel, it is unable to model the slip angle, which is an important consideration when racing on the friction limits. Therefore, the single-track model is used to calculate the state updates. An advantage of using the single-track model is that it takes the vehicle's dynamic limits on the acceleration and steering angle velocity into consideration.

In addition to the five state variables of x and y position, orientation, steering and speed, the single-track model also uses the slip-angle β and the yaw rate $\dot{\theta}$. The inputs to the single-track model are the longitudinal acceleration a and the steering angle velocity (steering angle rate of change) $\dot{\delta}$. We refer the reader to [45] for the full update equations.

We apply the state transition model by initialising the slip angle and yaw rate to zero. Then the speed and steering angle control references are converted to acceleration and steering angle velocity references using a proportional control system. The next state can then be calculated using the state vector $\mathbf{x} = [X, Y, \theta, v, \delta, \beta = 0, \dot{\theta} = 0]$. The dynamics model is updated using a timestep of 0.01 seconds.

4.2.2. Friction model

For racing, the vehicle must remain within the friction limit. The friction limit is modelled by ensuring that the lateral force on the vehicle is smaller than the frictional force of the wheels. The frictional force of the wheels on the ground can be written as $F_\text{friction} = bmg$, where b is the coefficient of friction, m is the vehicle's mass, and g is the gravitational constant. The lateral force on the wheels can be written as $F_\text{lateral} = mv\dot{\theta}$. For the kinematic model, the yaw rate can be calculated as $\dot{\theta} = v\tan(\delta)/L$, where L is the vehicle's wheelbase. Therefore, the inequality for the vehicle to remain within the friction limit is,

$$\frac{v^2}{L}\tan(|\delta|)m < bmg. \tag{3}$$

This equation can be rearranged to find the maximum speed within the friction limit for a given steering angle as,

$$v_\text{friction} = \sqrt{\frac{bg}{\tan(|\delta|)/L}}. \tag{4}$$

Equation 4 can be used to calculate the maximum speed for each steering angle, such that the vehicle is inside the friction limit $\forall\ v \mid v < v_\text{friction}$

4.3. Racing kernel formulation

The Viability Theory and vehicle model that have been presented are now combined to formulate the viability kernel for racing. We calculate the safe set for a racing vehicle that can maintain recursive feasibility while keeping the vehicle within the safety limits. We use the discretised model to calculate the subset of states that are safe.

The kernel uses the four state variables of x and y position, vehicle orientation θ and mode number q. The mode number represents the speed and steering combination, or the dynamic state. The reasons for using a single variable are that the kernel increases exponentially with each additional dimension, and only certain speed-steering combinations, remain within the friction limit.

4.3.1. Mode definition

Each mode is a natural number q_i with $i \in \{0, 1, \dots, q_n\}$, where q_n is the number of modes used. Each mode maps to a unique speed and steering angle, such that $q \mapsto [v, \delta]$, Only modes that are within the friction limit are considered.

The vehicle's dynamic limits constrain the system, i.e. the speed can only change within a range between planning timesteps. Therefore, only certain mode transitions are possible within the planning step. Additionally, the vehicle must remain within the friction limit during the transition, despite the steering angle being able to change faster than the speed. The reachable modes are selected by evaluating the effect of selecting every mode action on every mode state and discretising the output. For each mode, all the reachable modes are stored in a list of valid transitions.

Figure 5 shows a graph of the action space (steering angle and speed) with the modes (numbered q1-q9) indicated as circles. The purple region indicates the friction limit of the vehicle, which allows large steering angles for slow speeds and smaller steering

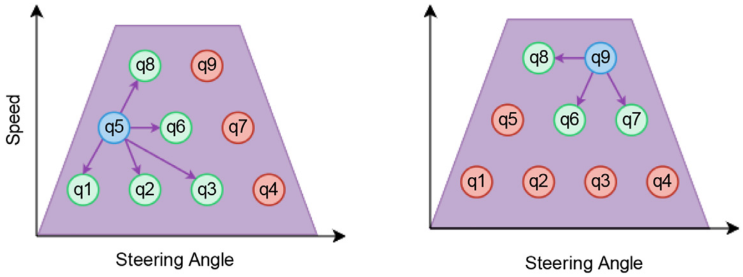


Fig. 5. Example mode placement indicating the friction limit in purple with the modes placed inside the limit. For the blue modes, the arrows represent the valid transitions. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

angles for high speeds, creating a triangular shape. The modes are placed inside the friction limit. For the blue modes (q5 and q9), the dark purple arrows to green modes represent the possible transitions. The red modes represent modes that are not reachable within a single planning timestep from the blue modes.

4.4. Kernel generation

The kernel state, mode definition and dynamics update are now combined to form a difference inclusion for the systems as,

$$\begin{aligned} \mathbf{x}_{k+1} \in F(\mathbf{x}_k), \quad & \text{with} \\ F(\mathbf{x}_k) &= \left\{ f(\mathbf{x}_k, q_i, T_p) \,\middle|\, i \in [0, 1, \ldots, q_n] \right\} \\ \mathbf{x}_k &= [X, Y, \theta, q] \\ q_i &= [\delta, v]. \end{aligned} \tag{5}$$

Recalling the viability kernel algorithm in Equation 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.

5. Safety system evaluation

The safety system is evaluated in the open-source[1] F1Tenthsimulator [15]. The simulator uses the single-track bicycle model to represent the vehicle dynamics and ray-casting to simulate the LiDAR scanner. The planning loop is run at 10 Hz, with the dynamics updates running at 100 Hz. We start by describing the methodology used to validate the safety system, then we present information related to the kernel generation and finally show the results from the validation. All of the experiments are seeded and the code is available online in the accompanying repository at: https://github.com/BDEvan5/SafeRaceLearning.

We use common parameters that have been used in other F1Tenthresearch [15]. For the F1Tenthvehicle, we use a friction coefficient $b = 0.523$ and a wheelbase length $L = 0.33$ m. The maximum speed $v_{\max}$ is 6 m/s and the steering angle range is [-0.4, 0.4] rad.

5.1. Safe sets for F1Tenthracing

The maps are split into blocks with a discretisation of 40 blocks per metre in the x and y directions. The orientation angle is split into 41 equal-angle segments. The kernels are generated with a time step of 0.2 seconds. The reason for choosing a larger time step for the kernel discretisation compared to the planning time step (0.1 seconds) is that smaller time steps require a finer discretisation of the x and y blocks, resulting in the kernels being too large to compute.

The modes are placed on the speed steering graph that enables a reasonable number of transitions. The speeds are split into six speed levels ranging from 2 m/s to 6 m/s, resulting in a resolution of 0.6 m/s between levels. At each speed, the steering range, as defined by the friction limit, is divided into five mode steering angles. Figure 6 shows how the modes are distributed across the action space of speed and steering angle. The purple region with the dark purple border is the friction limit calculated using Equation 4 and the steering range as the input. For each speed level, ranging from 2 m/s to 6 m/s, five modes are equally placed within the friction limit. Similarly to Figure 5, for the blue mode (marked with an x), the green modes are reachable within a single planning timestep, and the red modes are not.

5.1.1. Algorithmic implementation

For any given map, only a small proportion of the positions fall on the map. Only the states on the track map are considered for the kernel generation process.

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

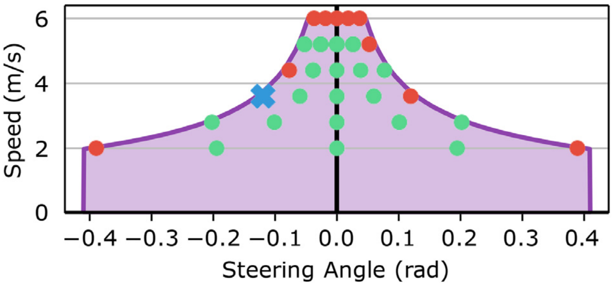
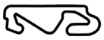


Fig. 6. Speed-steering diagram showing the friction limit (purple) and the modes used in the kernel formulation. For the blue mode, the green modes are valid transitions, and the red modes are not valid transitions. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

Table 3
Track images, distance for the AUT, ESP, GBR and MCO maps.

Map	AUT	ESP	GBR	MCO
Track				
Distance (m)	93.7	236.8	202.2	178.3

This is done 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 position blocks that lie on the driveable area of the track. This method results in a significant reduction in kernel size.

5.2. Methodology

The supervisor aims to display two kinds of behaviour, letting safe actions be executed while ensuring that unsafe actions are identified and modified. The supervisor's ability to produce this behaviour is evaluated through three experiments:

1. A random planner is used to evaluate the supervisor's ability to ensure the safety of a worst-case scenario planner.
2. A pure pursuit planner following the optimal trajectory is used to evaluate the supervisor's effect on high-performance racing.
3. The system's sensitivity to localisation accuracy is evaluated by adding increasing amounts of noise to the location used by the supervisor.

The worst-case scenario random planner uses a random number generator to select random speed and steering actions within the appropriate range. The planner's actions are completely independent of the vehicle's state. The pure pursuit planner follows an optimal trajectory calculated with the method presented by Heilmeir et al. [20], using their online library[2]

We use four maps, called AUT, ESP, GBR and MCO, to test our algorithms. These four maps, listed in Table 3, are scaled versions of the Formula-1 tracks in Austria, Spain, England and Monaco. The AUT map is the shortest, simplest map, and the ESP track is the longest.

5.3. Kernel generation

5.3.1. Generation results

We generate kernels for each of the four maps used during testing. For each kernel, the number states, iterations required for generations and % of the track that is safe is recorded. The number of states on the track is calculated as $n_{\text{track}} \times n_\theta \times n_q$ and is thus the kernel list length. The number of iterations required is the i value from Equation 2, representing the number of recursive function calls needed before every state in the kernel was safe. The percentage of the track that is safe is the number of safe states divided by the total number of states on the track.

Table 4 presents the results from the kernel generation. Each map has a number of states in the range of 2e8 (AUT) to 6e8 (ESP). This large number of states requires between 200 to 600 MB of computer storage. All four maps required between 29 to 31 iterations to converge and were calculated between 43% and 46% of the track to be safe. These similar results, even across different tracks with different shapes and lengths, show that this method is transferable with similar results to different race tracks.

[2] Available at: https://github.com/TUMFTM/trajectory_planning_helpers

Table 4
Number of states on the track, iterations required, and % of the safe states for the AUT, ESP, MCO and GBR maps.

Map	No. Track States	Iterations Required	% Track Safe
AUT	224,681,640	30	44.05
ESP	559,172,760	29	45.22
MCO	420,522,240	31	43.84
GBR	463,872,360	31	43.77

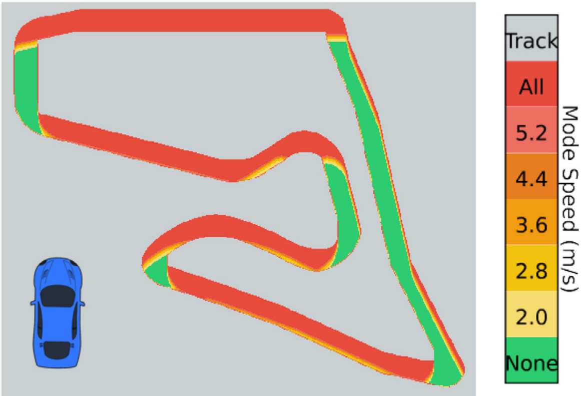


Fig. 7. The kernel for the AUT map for the vehicle facing the top of the map. The colours represent the allowed speed in m/s.

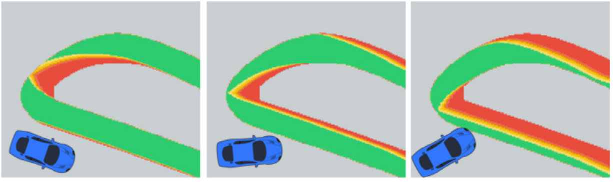


Fig. 8. A zoomed-in view of the AUT kernel for three different orientations denoted by the car's direction. The colour legend is the same as Figure 7.

5.3.2. Kernel visualisation

We present several visuals to communicate the safety kernel performance. Figure 7 shows the kernel for the AUT map for the vehicle facing towards the top of the map. The kernel depends on the vehicle direction and thus is different for different directions.

The different colours in Figure 7 represent the different mode speeds allowed. Red means that none of the modes are safe, and as the colours fade towards orange, lighter yellower and then green, modes of lower speeds become safe. The green regions indicate the area where the vehicle can travel at full speed, and thus all the speeds are permissible. All the modes considered have a steering angle of 0 rad, i.e. the wheels are straightforward.

To give a clearer perspective, Figure 8 shows three zoomed-in views of a single corner on the AUT map with the vehicle facing different directions. The colours represent the maximum allowed speed with the same meaning as in Figure 7. In the images, the safe and unsafe regions change in shape as the vehicle orientation changes. This illustrates the kernel's dependence on the vehicle orientation.

5.4. Validation results

5.4.1. Random planner validation

The kernels are validated for safety by running 50 test laps using a random planner. The random planner samples a steering angle from the steering range and a speed from the speed range using a uniform distribution.

Table 5 records the average times and interventions per lap. The results show that the vehicle's number of interventions varies with a standard deviation of between 7 and 11 interventions. The AUT map has the smallest average number of interventions of 262.2. It is suggested that this is due to the AUT track having the shortest distance (see Table 3). This high variation is what is expected from a random planner that selects completely different actions on each lap. The supervisor has a high intervention rate of between 70%-80% for the four test maps. A significant cause of the high-intervention rate is that many of the actions lead the vehicle to breach the friction limit. The ESP map has the lowest intervention rate of 73.9%, which could be due to the long straight sections in the track. For all of the test laps, the vehicle does not crash once, demonstrating that the supervisor can guarantee the vehicle's safety.

Table 5
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 AUT, MCO, ESP and GBR maps.

Metric	Interventions per Lap	Intervention Rate (%)	Total Crashes
AUT	**262.2** ± **7.7**	77.0 ± 2.3	**0**
MCO	476.1 ± 11.9	75.3 ± 1.8	**0**
ESP	583.5 ± 11.4	**73.9** ± **1.6**	**0**
GBR	527.2 ± 10.5	75.1 ± 2.0	**0**

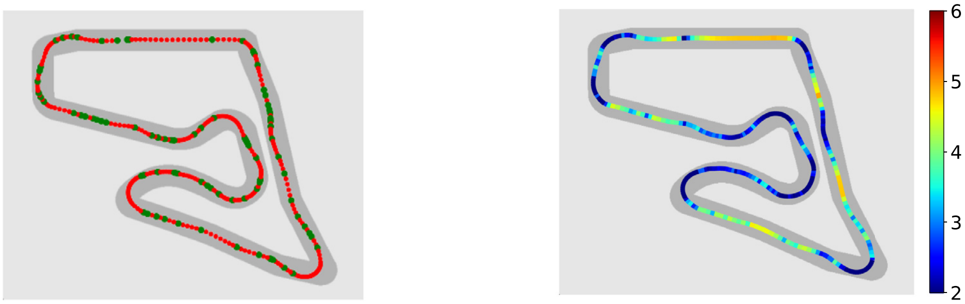


Fig. 9. The random planner path on the AUT map showing the supervisor intervention (left) and the trajectory with speed profile shown by colour bar in m/s (right).

Figure 9 shows the path taken by the random planner used with the supervisory safety system. The left image shows red dots where the supervisor intervened, and green dots where the random action was considered safe. The right image shows the trajectory with the colour bar indicating the speed in m/s. The images show that the supervisor regularly has to intervene to prevent the vehicle from crashing. The supervisor allows the vehicle to come close to the edge but prevents the vehicle from crashing. In the straight section at the top of the map, the supervisor allows the vehicle to drive faster, as shown by the yellow, and in the corners, the vehicle is forced to slow down, as shown by the blue line. This shows that the supervisor takes the location of the vehicle on the map into account in ensuring vehicle safety.

We measure the effect of speed by recording the number of interventions per lap when selecting actions out of different speed ranges. The minimum speed is always 2 m/s, and maximum speeds ranging from 2 m/s to 6 m/s are considered. For each maximum speed, 20 test laps are run and the average number of interventions are recorded.

Figure 10 shows how the number of interventions is dependent on the maximum speed. All of the maps start with few interventions when a speed of 2 m/s is selected. For the MCO map, there are around 250 interventions with a maximum speed of 2 m/s, which increases to 480 interventions for a maximum speed of 6 m/s. As the maximum speed increases, the number of interventions on all the maps also increases.

5.4.2. *Pure pursuit validation*

The pure pursuit validation measures the impact of the kernel on high-performance racing. We consider the difference in lap times between the pure pursuit planner following the racing line and the pure pursuit planner used with the safety system. We evaluate the lap times achieved by both planners with maximum speeds ranging from 3 m/s to 6 m/s.

The left graph in Figure 11 shows the difference between using the pure pursuit planner (PP) and pure pursuit planner with the safety supervisor (Supervised PP) for the ESP map. The lap times are lower for the pure pursuit without the supervisor. For the vehicles with a maximum speed of m/s, the difference between the times is within 3 seconds. As the maximum speed increases, the

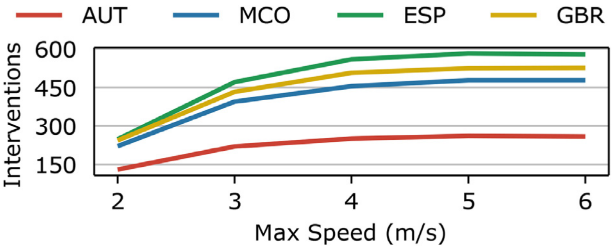


Fig. 10. The mean interventions when using the random planner that sample actions with maximum speeds ranging from 2 m/s to 6 m/s.

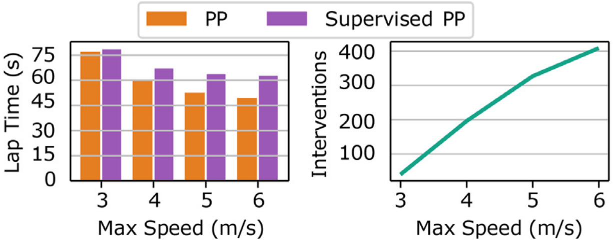


Fig. 11. LEFT: The difference between using the pure pursuit planner (PP) and pure pursuit planner with the safety supervisor (Supervised PP) ESP map. RIGHT: The number of interventions from the supervisor.

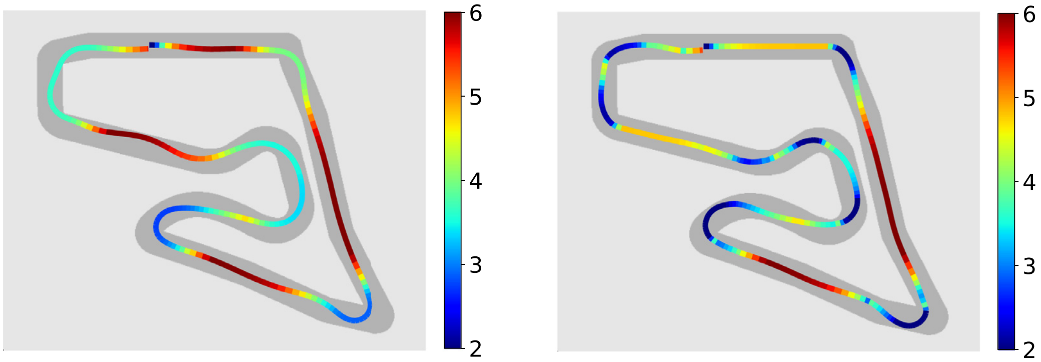


Fig. 12. Trajectories with speed profiles on the AUT map of the pure pursuit planner without the supervisor (left) and with the supervisor (right).

pure pursuit planner gets significantly faster, and the difference between the two increases. This is also shown by the increasing number of interventions made by the supervisor.

Figure 12 shows the trajectories taken by the pure pursuit planner without (left) and with (right) the safety system. The right image (with the safety system) speeds up in the straights, as shown by the dark red, in a similar way to the plain pure pursuit planner. The main difference between the two trajectories is that the safety system causes the vehicle to drive more slowly around the corners as shown by the darker blue in the right-hand image. This is further investigated by plotting the two vehicle's speed profiles.

Figure 13 shows the speed profiles for the trajectories on the AUT map shown in Figure 12. The graph confirms that the supervisor causes the vehicle to select a lower speed in the corners but leaves the rest of the speed profile unchanged. The lower speeds in the corners are the cause of the difference in lap time shown in Figure 11. It is proposed the reason for the lower speed in the corners is that using a large kernel discretisation time step than the planning time step, results in the supervisor being over-conservative in the corners.

5.4.3. Ablation study

We investigate how the supervisor functions in the presence of noise. We add noise sampled from a normal distribution with increasing standard deviations to the x, y location variables used by the supervisor. The random planner with the safety system runs 10 test laps for each test using maximum speeds ranging from 2 m/s to 6 m/s.

Figure 14 shows the success rates from the experiments with noise added to the location. The graph shows that all the speeds except complete all the laps with noise using a standard deviation of 20 cm, with the exception of the 4 m/s vehicle crashing on a single lap. After this, the success rates drop off sharply with most of the laps resulting in a crash when noise with a standard deviation of 60 cm is added. As expected, the vehicles with higher maximum speeds have lower success rates at each noise level. We conclude

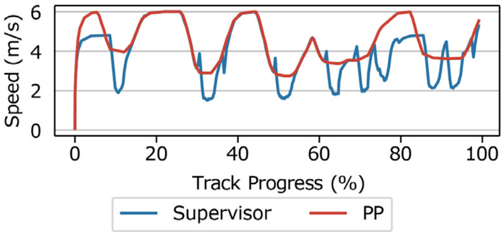


Fig. 13. Comparison of the speed profiles for the pure pursuit (PP) and supervised pure pursuit planner on the AUT map.

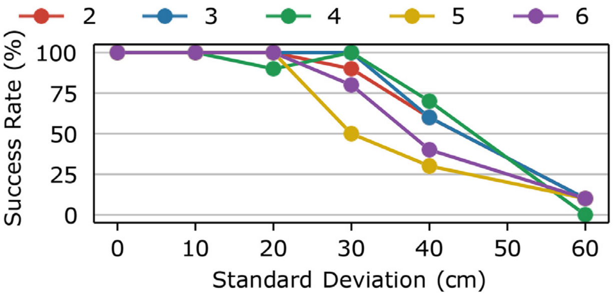


Fig. 14. The success rate for the random planner with the supervisory system, on the AUT map with the noise added to the location used by the supervisory system. The legend entries are the maximum speeds used by the random planner in m/s.

that our safety system is effective when noise with a standard deviation of up to 20 cm is present in the location. It is suggested that a reason the system functions well, even in the presence of noise is that the safety policy of selecting a pure pursuit action to follow the centre line brings the vehicle towards the safest part of the track and away from the boundary.

5.5. Discussion

The safety system evaluation shows that for all four test maps considered, kernels can be created that keep vehicles safe. The random planner validation showed that for a worst-case scenario planner, the supervisor prevents the vehicle from crashing. The pure pursuit validation showed that the supervisor has a minimal effect on the path taken, yet in the corners selects a slightly slower speed due to being over-conservative. The ablation study showed that the supervisor can keep vehicles safe, even when noise is added to the localisation. Therefore, we conclude that our system is effective for ensuring safety and can be used for online training.

6. Safe learning

6.1. Conventional reinforcement learning

6.1.1. Reinforcement learning preliminary

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 [3]. Reinforcement learning problems are modelled as Markov Decision Processes (MDPs). 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. MDPs must satisfy the Markov property, which states that the current state depends only on the previous state and not on the history of states.

Figure 15 shows how the reinforcement learning problem is modelled as having an agent that receives a state s and selects an action a in an environment. After each action has been implemented, the environment returns a reward r indicating how good or bad the action was. RL uses the agent's policy to collect experience to train the agent to maximise a reward signal, thus producing the desired behaviour.

Deep deterministic policy gradient (DDPG) [46] methods can select continuous actions for robotic control. 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, which are trained and used to select actions and target networks which are used 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 Fig. 16.

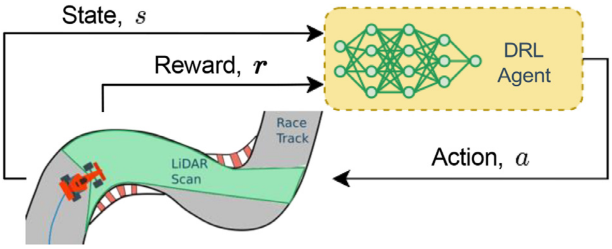


Fig. 15. The DRL agent receives a state s, selects an action a that is implemented, and a reward r based on the vehicle position is given to the agent.

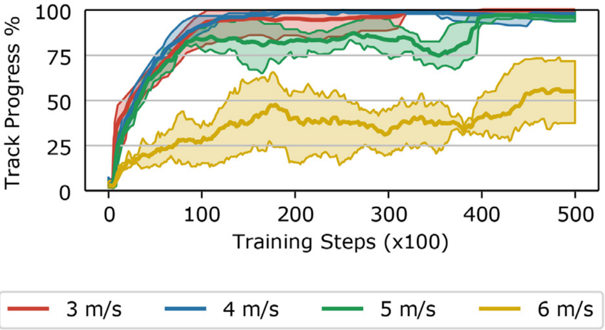


Fig. 16. Average progress during training for conventional agents trained with maximum speeds ranging from 3 m/s to 6 m/s on the ESP map.

The Q-networks are 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 it's target policy,

$$y_j = r_j + \gamma Q'(s_{j+1}, \mu'(s_{j+1})), \tag{6}$$

where γ is the discount factor.

The 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,

$$\nabla_\theta J(\theta) = \frac{1}{N}\sum_j \nabla_\theta Q(s_j, \mu(s_j)). \tag{7}$$

After each update to the networks, a soft update is applied to adjust the target networks towards the model networks.

The twin-delayed-DDPG (TD3) algorithm is used because it is a state-of-the-art algorithm for continuous control [47]. The 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. Using two Q-networks is done to prevent the agent 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 (Equation 6) 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,

$$\begin{aligned} y_j = \quad & r_j + \gamma \min_{i=1,2} Q_{\theta'_i}(s'_j, \mu(s'_j) + \epsilon) \\ & \epsilon \sim \text{clip}(\mathcal{N}(0,\sigma), -c, c) \end{aligned} \tag{8}$$

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

6.1.2. Conventional racing formulation

End-to-end learning replaces the entire processing pipeline with a learning agent. A slice of 20 beams from the LiDAR scan is used as input to the neural network. The beams are equally spaced with a field of view of π radians. The slices from the previous and current LiDAR scans are stacked together so that the agent is aware of its motion. The beams are scaled to the range $[0, 1]$ by dividing by the maximum beam length of 10 m.

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, and the speed is scaled to the range $[1, v_{\max}]$ m/s. The minimum speed of 1 m/s is used to prevent the vehicle from not moving.

The reward signal communicates the desired behaviour of quickly completing laps and not crashing to the agent. This behaviour is encoded as a positive reward of 1 for completing a lap and a punishment of -1 for crashing, written as,

$$r = \begin{cases} -1 & \text{if crash} \\ 1 & \text{if lap complete.} \\ \frac{v_t}{v_{\max}} \cos\psi - d_c & \text{otherwise.} \end{cases} \tag{9}$$

If the agent neither completed a lap or crashed, then it receives a reward based on the cross-track distance and heading error, where v_t is the vehicle's speed, $v_{\max}$ is the maximum speed, ψ is the heading error angle, and d_c is the cross-track distance. The aim of the racing reward is to encourage the agent to select good racing trajectories in addition to learning to complete laps without crashing.

6.2. Safe learning

Conventionally, DRL agents build up experience from selecting actions that are implemented. 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.

6.2.1. Episodes

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.

We reformulate the definition of an episode to run from an initial state until the supervisor intervenes. 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.

6.2.2. Reward signal

Since the vehicle never crashes and the supervisor is now used, the reward signal is modified. The penalty for crashing is replaced by a penalty if the supervisor intervenes. This term aims to encourage the agent to not require intervention. The racing reward is simplified from the cross-track and heading error reward to simply rewarding the agent according to the square of its scaled speed. We write the new reward as,

$$r = \begin{cases} 1 & \text{if lap complete.} \\ -1 & \text{if intervention} \\ \left(\frac{v_{\text{agent}}}{v_{\max}}\right)^2 & \text{otherwise.} \end{cases} \tag{10}$$

6.3. Methodology

We design experiments to investigate safe learning using the supervisory approach proposed. We split our experiments into four sections:

1. Investigate the supervisor's ability to train agents for autonomous racing.
2. Investigate the effect of maximum speed of conventional and safe learning.
3. Compare agents trained with conventional learning and safe learning with a classical planner according to lap times and success rates.
4. Compare the behaviour (speed profiles and trajectories) of conventional and safe agents with a classical planner.

The safe learning methods presented are compared to the classical planner following the racing line and a conventional DRL agent. The conventional and safe agents use exactly the same neural networks, state and action vectors. For all of the experiments, except where speed is the independent variable, we use a maximum speed of 5 m/s. All the learning experiments are repeated five times with different random seeds.

6.3.1. Learning implementation

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]. The TD3 algorithm follows the original implementations as closely as possible with the original hyper-parameters being used.

Training agents for conventional learning involves allowing the agents to select an action that is implemented in the simulator. After each action is implemented, a reward is calculated and stored in memory. The memory tuples of state, action, next state and reward are randomly sampled and used to train the agent. Safe learning involves allowing the agent to select an action that is ensured to be safe by the supervisor before being implemented. The conventional DRL agents are trained for 50,000 steps and the safe agents are trained for 10,000 steps.

6.4. Results

6.4.1. Training comparison

We compare the conventional and safe methods of training DRL agents for autonomous racing. We start by training conventional agents to race with maximum speeds of 3 m/s, 4 m/s, 5 m/s and 6 m/s on the ESP map. In the graph, the lines represent the average of the five repetitions and the shaded regions are the minimums and maximums.

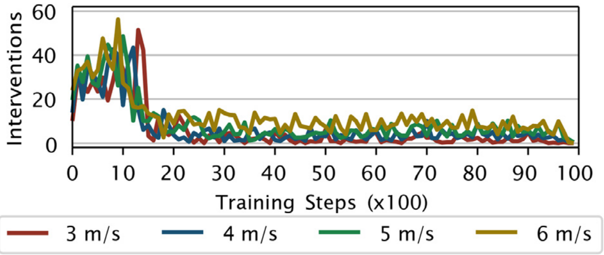


Fig. 17. The interventions per 100 steps for agents trained with the supervisor with maximum speeds of 3 to 6 m/s on the ESP map.

We now consider the training performance of agents trained using the supervisory learning formulation. Since agents trained with the supervisor never crash, the average progress cannot be used as a metric. Therefore the interventions made by the supervisor per 100 training steps is used as the training metric.

Figure 17 show the interventions made by the supervisor during training agents with different speeds. In the beginning, the supervisor intervenes many times, reaching over 60 interventions. After around 2000 steps, the supervisor intervenes much less and the agent requires few interventions. This shows that the supervisor is effective at training agents to no longer require intervention.

The training graphs show that the agents trained with the safety system can be trained in 10k steps to a point where they require little intervention from the supervisor. The conventional agents require 50k training steps to converge. Therefore, we conclude that safe training can train DRL agents for autonomous racing with a 5× improvement in sample efficiency.

6.4.2. Safe learning study

We now investigate the effect of the supervisor during the training in greater detail. We start by considering the number of interventions made by the supervisor during the training.

The graph in Figure 18 shows the number of interventions per 100 training steps and blocks and a line. For the first 3000 steps the agents require around 50 interventions, this is effectively every 2nd training step. After that, the number of interventions required drops significantly to around 5 interventions per 100 steps.

We compare the number of interventions across different maps. Figure 19 shows the number of interventions for five runs on the ESP, AUT and MCO maps. While each run follows its own pattern, it is clear that in the first 3000 training steps the agents require many interventions.

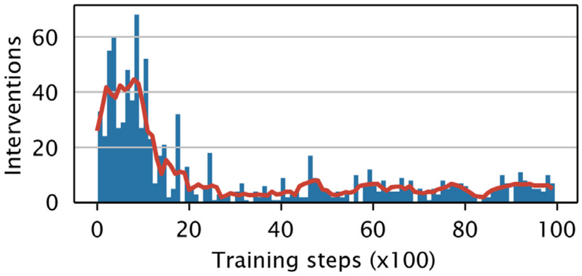


Fig. 18. Histogram of interventions per 100 steps for training on the AUT map for 10,000 steps. The red line shows the moving average. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

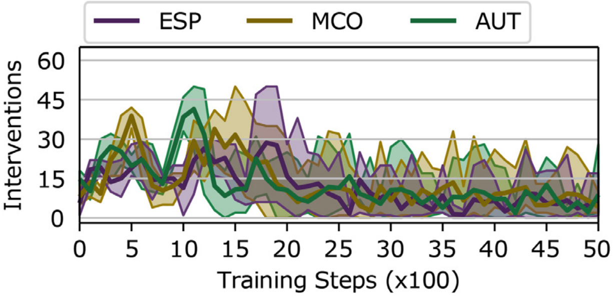


Fig. 19. Average interventions on the ESP, MCO and AUT maps during the first 5000 steps of training. The shaded regions show the minimum and maximum using different random seeds.

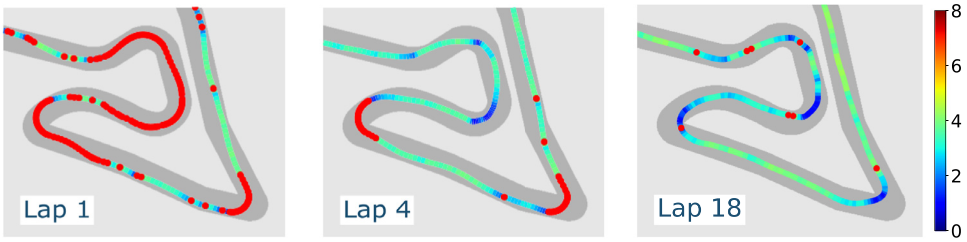


Fig. 20. The 1st, 4th, and 18th trajectories during training an agent on the AUT map with the colour representing the speed profile and the red dots where the supervisor intervened. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

We now consider the trajectories selected by the agents during training and the location of the interventions. Figure 20 shows three trajectories during training with the red dots denoting the location of the supervisor's intervention.

The trajectories in Figure 20 show that in the beginning (left image) the supervisor intervenes most of the time. The specific areas where the supervisor intervenes are in the corners where the vehicle must slow down and turn not to crash. As the training progresses, the supervisor intervenes less until towards the end of training the supervisor only sporadically intervenes.

Figure 21 shows the rewards earned and the lap times for training safe agents on the four test maps. While each track has a different length and thus different lap times and rewards, the patterns in the training are clear. The lap times decrease from the initial lap time as the training progresses. The rewards increase with a specific step between 1000 and 3000 training steps.

This evaluation demonstrates that the learning formulation effectively trains agents to race safely without the supervisor. At the beginning of training, the supervisor regularly intervenes and as training progresses, the supervisor intervenes less and less. As the agent learns to act independently, the lap times decrease, showing that the agent is learning to race.

6.4.3. Performance comparison

We now compare the performance of agents trained with the conventional and safe formulations. We use the variables of lap times and success rates to compare the performance of the trained agent. After the safe learning agents have been trained, the supervisor is removed for testing to investigate the agent's ability to drive safely.

Figure 22 shows the lap times and success rates achieved by the conventional and safe agents. The conventional agents perform faster on all of the test maps. However, the safe agents achieve a significantly higher success rate on all the test maps. Of specific mention is the MCO map, where the conventional planner achieves 45% success, while the safe agent achieves 100%. These results

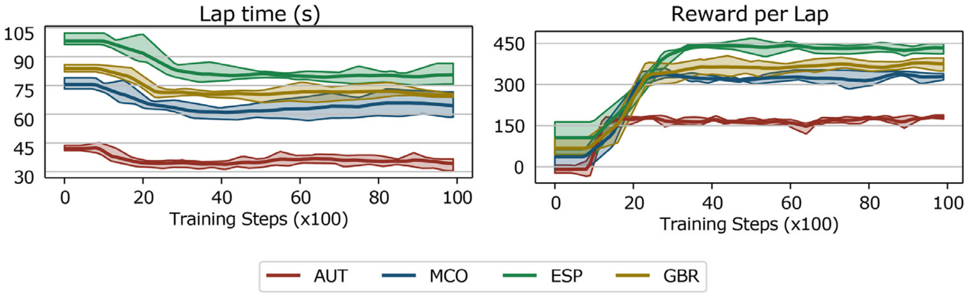


Fig. 21. Rewards earned and lap times achieved by agents trained on the AUT, MCO, ESP and GBR maps.

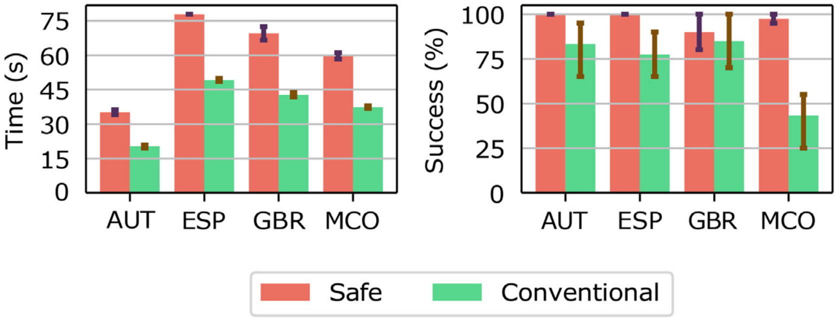


Fig. 22. Lap times and success rate for the conventional and safe agents.

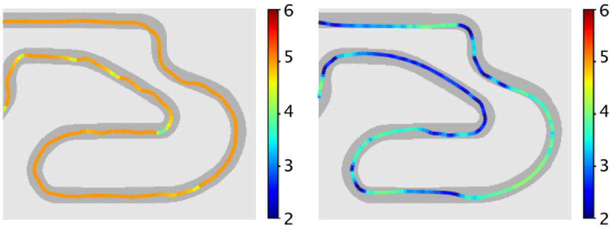


Fig. 23. Trajectories taken by the conventional (left) and safe (right) learning agents on a section of the ESP track. The safe agent speed varies while the conventional agent selects a constant high speed.

indicate the inherent tension in racing that safety and performance are inherently opposed to each other. Therefore, we investigate the behaviour learned by the agents that leads to this result.

6.4.4. Behaviour comparison

The last evaluation investigates the behaviour of the agents. The behaviour is represented by showing example trajectories and speed profiles to explain the results.

Figure 23 shows trajectories taken by the safe and conventional learning agents on a section of the ESP track. The conventional agent selects a near constant speed profile near the maximum speed (shown by the mainly orange line) and swerves left and right a low. In contrast, the safe agent trajectory (right) shows the vehicle speeding up in the straight sections and slowing down in the corners. There is a lot of dark blue, indicating that the vehicle is moving slowly for much of the trajectory. The conventional agent trajectory (right) is orange for most of the trajectory, indicating that the vehicle speed is near constant throughout the section. The vehicle slows down slightly in the hairpin corner, as shown by the green line.

Figure 24 shows the trajectories taken by the classic planner (left) and safe agent (right). Following the optimal trajectory, the classical planner smoothly speeds up and slows down around the corners. The safe agent also speeds up and slows down but generally selects lower speeds than the optimal trajectory. This is seen from the darker blue lines in the safe agent trajectory, compared to the yellow and orange in the classical planner trajectory. We further investigate the speeds by considering the speed profiles selected by the agents.

Figure 25 shows the speed profiles for a section of the ESP track, comparing the conventional and safe agents with the classical planner. The classical planner selects a smooth speed profile of speeding up and slowing down. The conventional agent almost always selects the maximum speed possible. The safe agent selects lower speeds than both the other planners, normally between 3 m/s to 4 m/s. This explains why the safe agents achieve slower lap times than the conventional agents.

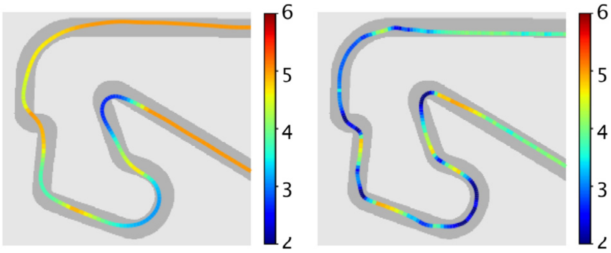


Fig. 24. Trajectories taken by the classic planner (left) and safe agent (right) on a section of the ESP track. While both planners slow down in the corners, the safe agent selects a more conservative speed.

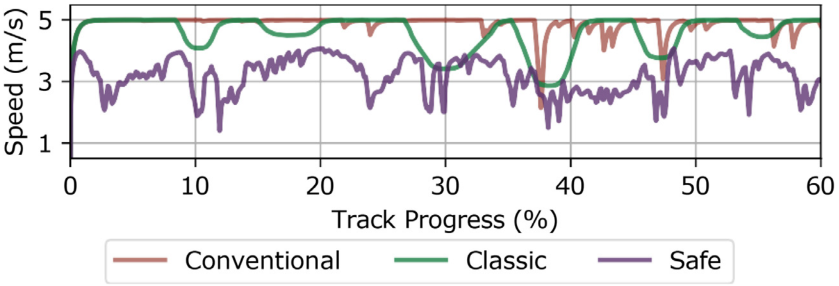


Fig. 25. Speed profiles for the conventional and safe agents compared to the classic planner on the ESP map.

7. Conclusion

While deep reinforcement learning offers many advantages to robotic control and autonomous vehicles by enabling end-to-end solutions, current approaches have been limited by requiring the agent to crash during training. This has limited training to simulation and created the sim-to-real problem of transferring agents trained in simulation to reality. While some approaches have trained low-performance vehicles online using simplistic safety systems, current methods cannot handle high-performance racing. In this work, we addressed the problem of training a DRL agent to race at high-speed without ever crashing.

We presented a supervisory safety system that uses Viability Theory to recursively feasible vehicle safety. The supervisor uses a dynamics model to simulate the agent's action and then checks if the next state is in the kernel of safe states. The validation of the safety system showed that our system can prevent a worst-case scenario planner from crashing on four test maps at speeds of up to 6 m/s. Compared to other safety methods, our approach has the advantages of not requiring human intervention [10], not reversing the vehicle [12], and not requiring online calculations [13]. Additionally, where all previous methods have used low, constant speeds, our method uses variable speeds up to 6 m/s, and ensures the vehicle remains within the friction limit.

We presented a method of safe learning that reformulates deep reinforcement learning to incorporate the supervisor. The safe learning method was evaluated in the F1Tenthsimulator at speeds of up to 6 m/s. The results showed that safe learning presents a 5× improvement in sample efficiency, requiring only 10,000 steps. The supervisor and the learning formulation effectively train the agent to not require supervision. The safe learning agents select lower speed profiles than the conventional learning agents. This results in the safe learning agents achieving slower lap times and higher success rates. The major advantage of our methods is that the vehicles never crash during training. Future work should use this method to train agents onboard physical vehicles.

The ability to train agents for high-performance robotic control while ensuring safety during the training process means that these methods can be used to train DRL agents on real-world robots, thus bypassing the sim-to-real problem. Future work should evaluate how well safe learning using the supervisor performs on real-world, high-performance platforms. Bypassing the sim-to-real gap will mean that there is no difference between the training and testing behaviour since both will be on the same physical vehicle.

The improvement in sample efficiency means that it is easier to use DRL since training time is reduced. Training more conservative policies leads to safer solutions which are essential for safety-critical systems such as autonomous vehicle control. Overall, safe learning adapts conventional DRL to make solutions that are practically feasible and safer. Future work should seek to apply the method of safe learning with a supervisor to other safety-critical domains such as drone control.

A limitation of this approach is that during training, the supervisor requires the vehicle's location to ensure safety. Future work could look to develop methods for online verification using only the LiDAR scan. Being able to ensure safety without localisation would enable the supervisor to ensure safety on unmapped tracks.

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.

References

[1] J. Betz, H. Zheng, A. Liniger, U. Rosolia, P. Karle, M. Behl, V. Krovi, R. Mangharam, Autonomous vehicles on the edge: a survey on autonomous vehicle racing, IEEE Open J. Intell. Transp. Syst. (2022).

[2] N. Hamilton, P. Musau, D.M. Lopez, 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.

[3] R.S. Sutton, A.G. Barto, Reinforcement Learning: An Introduction, MIT press, 2018.

[4] 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, H. Kitano, Outracing champion gran turismo drivers with deep reinforcement learning, Nature 2022 602:7896 602 (7896) (2022) 223–228, doi:10.1038/s41586-021-04357-7.

[5] A. Brunnbauer, L. Berducci, A. Brandstatter, M. Lechner, R. Hasani, D. Rus, R. Grosu, Latent imagination facilitates zero-shot transfer in autonomous racing, 2022 International Conference on Robotics and Automation (ICRA) (2022) 7513–7520, doi:10.1109/ICRA46639.2022.9811650.

[6] W. Zhao, J.P. Queralta, T. Westerlund, Sim-to-real transfer in deep reinforcement learning for robotics: a survey, 2020 IEEE Symposium Series on Computational Intelligence, SSCI 2020 (2020) 737–744, doi:10.1109/SSCI47803.2020.9308468.

[7] L. Brunke, M. Greeff, A.W. Hall, Z. Yuan, S. Zhou, J. Panerati, A.P. Schoellig, Safe learning in robotics: from learning-based control to safe reinforcement learning, Annu. Rev. Control Robot. Autonomous Syst. 5 (2022) 411–444.

[8] Z. Li, U. Kalabić, T. Chu, Safe reinforcement learning: learning with supervision using a constraint-admissible set, in: 2018 Annual American Control Conference (ACC), IEEE, 2018, pp. 6390–6395.

[9] J. Francis, B. Chen, S. Ganju, S. Kathpal, J. Poonganam, A. Shivani, S. Genc, I. Zhukov, M. Kumskoy, A. Koul, et al., Learn-to-race challenge 2022: benchmarking safe learning and cross-domain generalisation in autonomous racing, arXiv preprint arXiv:2205.02953 (2022).

[10] W. Saunders, G. Sastry, A. Stuhlmüller, 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.

[11] X. Sun, M. Zhou, Z. Zhuang, S. Yang, J. Betz, R. Mangharam, A benchmark comparison of imitation learning-based control policies for autonomous racing, arXiv preprint arXiv:2209.15073 (2022).

[12] M. Bosello, R. Tse, 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.

[13] P. Musau, N. Hamilton, D.M. Lopez, P. Robinette, 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.

[14] 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.

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

[16] P. Cai, H. Wang, H. Huang, Y. Liu, M. Liu, Vision-based autonomous car racing using deep imitative reinforcement learning, IEEE Rob. Autom. Lett. 6 (4) (2021) 7262–7269.

[17] 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.

[18] A. Liniger, Path Planning and Control for Autonomous Racing, ETH Zurich, 2018.

[19] J.L. Vázquez, M. Brühlmeier, A. Liniger, A. Rupenyan, J. Lygeros, Optimization-based hierarchical motion planning for autonomous racing, in: 2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), IEEE, 2020, pp. 2397–2403.

[20] A. Heilmeier, A. Wischnewski, L. Hermansdorfer, J. Betz, M. Lienkamp, B. Lohmann, Minimum curvature trajectory planning and control for an autonomous race car, Veh. Syst. Dyn. 58 (10) (2020) 1497–1527, doi:10.1080/00423114.2019.1631455.

[21] M. O'Kelly, H. Zheng, A. Jain, J. Auckley, K. Luong, R. Mangharam, Tunercar: a superoptimization toolchain for autonomous racing, in: 2020 IEEE International Conference on Robotics and Automation (ICRA), IEEE, 2020, pp. 5356–5362.

[22] V. Cataffo, G. Silano, L. Iannelli, V. Puig, L. Glielmo, A nonlinear model predictive control strategy for autonomous racing of scale vehicles, in: 2022 IEEE International Conference on Systems, Man, and Cybernetics (SMC), IEEE, 2022, pp. 100–105.

[23] R. Wang, Y. Han, 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.

[24] C.H. Walsh, 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.

[25] A. Tătulea-Codrean, T. Mariani, 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 53 (2) (2020) 6031–6036.

[26] E. Chisari, A. Liniger, A. Rupenyan, L. Van Gool, J. Lygeros, Learning from simulation, racing in reality, in: 2021 IEEE International Conference on Robotics and Automation (ICRA), IEEE, 2021, pp. 8046–8052.

[27] 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).

[28] R. Zhang, J. Hou, G. Chen, Z. Li, J. Chen, A. Knoll, Residual policy learning facilitates efficient model-free autonomous racing, IEEE Rob. Autom. Lett. 7 (4) (2022) 11625–11632.

[29] B. Thananjeyan, A. Balakrishna, S. Nair, M. Luo, K. Srinivasan, M. Hwang, J.E. Gonzalez, J. Ibarz, C. Finn, K. Goldberg, Recovery rl: safe reinforcement learning with learned recovery zones, IEEE Rob. Autom. Lett. 6 (3) (2021) 4915–4922.

[30] L. Zhang, R. Zhang, T. Wu, R. Weng, M. Han, Y. Zhao, Safe reinforcement learning with stability guarantee for motion planning of autonomous vehicles, IEEE Trans. Neural Netw. Learn. Syst. 32 (12) (2021) 5435–5444.

[31] A. Taylor, A. Singletary, Y. Yue, A. Ames, Learning for safety-critical control with control barrier functions, in: Learning for Dynamics and Control, PMLR, 2020, pp. 708–717.

[32] R. Cheng, G. Orosz, R.M. Murray, 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, volume 33, 2019, pp. 3387–3395.

[33] J.H. Gillula, C.J. Tomlin, Guaranteed safe online learning via reachability: tracking a ground target using a quadrotor, in: 2012 IEEE International Conference on Robotics and Automation, IEEE, 2012, pp. 2723–2730.

[34] I.M. Mitchell, A.M. Bayen, C.J. Tomlin, A time-dependent hamilton-jacobi formulation of reachable sets for continuous dynamic games, IEEE Trans Automat Contr 50 (7) (2005) 947–957.

[35] J.F. Fisac, A.K. Akametalu, M.N. Zeilinger, S. Kaynama, J. Gillula, C.J. Tomlin, A general safety framework for learning-based control in uncertain robotic systems, IEEE Trans Automat Contr 64 (7) (2018) 2737–2752.

[36] H. Krasowski, X. Wang, M. Althoff, Safe reinforcement learning for autonomous lane changing using set-based prediction, in: 2020 IEEE 23rd International Conference on Intelligent Transportation Systems (ITSC), IEEE, 2020, pp. 1–7.

[37] T. Stahl, F. Diermeyer, Online verification enabling approval of driving functionsimplementation for a planner of an autonomous race vehicle, IEEE Open Journal of Intelligent Transportation Systems 2 (2021) 97–110.

[38] T. Fraichard, H. Asama, Inevitable collision statesa step towards safer robots? Adv. Rob. 18 (10) (2004) 1001–1024.

[39] A. Lawitzky, A. Nicklas, D. Wollherr, M. Buss, Determining states of inevitable collision using reachability analysis, in: 2014 IEEE/RSJ International Conference on Intelligent Robots and Systems, IEEE, 2014, pp. 4142–4147.

[40] A. Bajcsy, S. Bansal, E. Bronstein, V. Tolani, C.J. Tomlin, An efficient reachability-based framework for provably safe autonomous navigation in unknown environments, in: 2019 IEEE 58th Conference on Decision and Control (CDC), IEEE, 2019, pp. 1758–1765.

[41] A. Liniger, J. Lygeros, Real-time control for autonomous racing based on viability theory, IEEE Trans. Control Syst. Technol. 27 (2) (2017) 464–478.

[42] L. Tai, G. Paolo, 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, volume 2017-Septe, Institute of Electrical and Electronics Engineers Inc., 2017, pp. 31–36, doi:10.1109/IROS.2017.8202134.

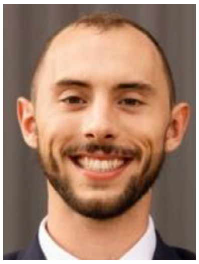
[43] A. Kendall, J. Hawke, D. Janz, P. Mazur, D. Reda, J.-M. Allen, V.-D. Lam, A. Bewley, A. Shah, Learning to drive in a day, in: 2019 International Conference on Robotics and Automation (ICRA), IEEE, 2019, pp. 8248–8254.

[44] R. Ivanov, T.J. Carpenter, J. Weimer, R. Alur, G.J. Pappas, 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.

[45] M. Althoff, M. Koschi, S. Manzinger, CommonRoad: composable benchmarks for motion planning on roads, 2017 IEEE Intelligent Vehicles Symposium (IV), IEEE, 2017, doi:10.1109/ivs.2017.7995802.

[46] T.P. Lillicrap, J.J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, D. Wierstra, Continuous control with deep reinforcement learning, 4th International Conference on Learning Representations, ICLR 2016 - Conference Track Proceedings (2016).

[47] S. Fujimoto, H. Hoof, D. Meger, Addressing function approximation error in actor-critic methods, in: International conference on machine learning, PMLR, 2018, pp. 1587–1596.



Benjamin Evans received a bachelor's in Mechatronic Engineering from Stellenbosch University, graduating in 2019. In 2023 he completed his PhD at Stellenbosch, which focused on deep learning methods for autonomous racing. His interests include the intersection of classical control and machine learning for autonomous systems. He is current projects include the sim-to-real problem for autonomous vehicles, safety in machine learning controllers and learning high-performance control.

Hendrick 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 on practical vehicles.

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.