Accelerating Online Reinforcement Learning via Supervisory Safety Systems

Benjamin Evans\textsuperscript{1}, Johannes Betz\textsuperscript{2}, Hongrui Zheng\textsuperscript{2}, Herman A. Engelbrecht\textsuperscript{1}, Rahul Mangharam\textsuperscript{2}, and Hendrik W. Jordaan\textsuperscript{1}

Abstract—Deep reinforcement learning (DRL) is a promising method to learn control policies for robots only from demonstration and experience. To cover the whole dynamic behaviour of the robot, the DRL training is an active exploration process typically derived in simulation environments. Although this simulation training is cheap and fast, applying DRL algorithms to real-world settings is difficult. If agents are trained until they perform safely in simulation, transferring them to physical systems is difficult due to the sim-to-real gap caused by the difference between the simulation dynamics and the physical robot. In this paper, we present a method of online training a DRL agent to drive autonomously on a physical vehicle by using a model-based safety supervisor. Our solution uses a supervisory system to check if the action selected by the agent is safe or unsafe and ensure that a safe action is always implemented on the vehicle. With this, we can bypass the sim-to-real problem while training the DRL algorithm safely, quickly, and efficiently. We provide a variety of real-world experiments where we train online a small-scale, physical vehicle to drive autonomously with no prior simulation training. The evaluation results show that our method trains agents with improved sample efficiency while never crashing, and the trained agents demonstrate better driving performance than those trained in simulation.

I. INTRODUCTION

A. Motivation

Deep reinforcement learning (DRL) is a growing, popular method in autonomous system control [1]. Like humans that learn from experiences over time, DRL algorithms learn control mappings from sensor readings to planning commands using only observations from the environment and reward signals defined by the engineer. In contrast to humans who learn in the real world, DRL agents are usually trained in simulation. These simulation environments require accurate sensor and dynamics models to represent the robot and its surrounding environment. Unfortunately, the accuracy of simulation environments is limited to maintain good computation time, resulting in the sim-to-real gap when the simulation-trained DRL agent is transferred to a real-world system [2].

It is desirable to train an agent directly on the robot, thus altogether avoiding the sim-to-real gap [3]. An inherent challenge in the online training of DRL algorithms on real-world robots is that DRL algorithms rely on crashing during training, meaning that training on a physical robot is very difficult or nearly impossible [4]. Crashing physical robots is expensive and a safety concern for the surrounding humans [5]. Therefore, being able to train DRL agents safely, crash-free onboard physical robots would result in an interesting technology that enables the application of DRL agents to more physical platforms. Further, it can be expected that bypassing the sim-to-real gap will lead to improved DRL policies.

B. Contributions

In this paper, we provide insights on online DRL training and testing on a real-world vehicle, accelerated by a supervisory system. We present a supervisory safety system (SSS) capable of training a DRL agent onboard a physical autonomous vehicle with no prior simulation training. The supervisory safety system uses a viability kernel (set of safe states) and vehicle model to check if the DRL agent’s action is safe. If the DRL action is unsafe, a safe action from a pure pursuit controller is implemented. This work has three main contributions:

- We combine a supervisory system and a DRL algorithm to achieve safe and efficient policy training.
- We demonstrate that combining the supervisory system and DRL algorithm results in the safe and robust online training of a real-world robot system.
- We demonstrate that an agent trained with the SSS can effectively bypass the sim-to-real gap by outperforming an agent trained in simulation.

II. RELATED WORK

In this section, we discuss works related to DRL for autonomous vehicles, safe DRL training, and online DRL training.

DRL for autonomous vehicles: Many variations of DRL-based methods (model-based, model-free) have been implemented to derive control commands for autonomous vehicles from raw sensor inputs. The authors of [6], [7] used Deep Q-Learning (DQN) to learn steering manoeuvres for autonomous systems and [8], [9] used the soft-actor-critic (SAC) algorithm. Numerous deep learning studies are only evaluated in simulation because they are not practically feasible [10], [11], [12]. Of the DRL algorithms applied to physical systems, the dominant approach in the literature is to train the agents in simulation before transferring them to real vehicles [13], [14], [15]. Evaluations show that DRL is.
that an effective method of autonomous vehicle control, but the sim-to-real gap remains a challenge.

**Safe DRL training:** In [16] and [17], a risk-based approach is used to guarantee safety constraints during DRL training. While [16] uses a Monte Carlo tree search (MCTS) to reduce unsafe behaviours of the agent while training, [17] uses the estimation of trust region constraint to allow large update steps. Temporal logic specifications have also been used to enforce safety constraints during training [18], [19], [20]. A risk-based approach is poorly suited to autonomous vehicles since estimates cannot provide safety guarantees. Wang et al. [21] focuses on ensuring the legal safety of the vehicle by following traffic rules by using a safety layer based on control barrier functions. Control barrier functions and similar set theory techniques have been used in several safety-critical learning problems [22], but have been focused on applications where a safe setting can be assumed [23] or the dynamics can be simplified to linear (affine) equations [24].

**Online DRL training:** Kendall et al. [25] train a DRL agent on a real-world vehicle. Their safety mechanism is a safety driver (human intervention [26]) that intervenes if the vehicle’s behaviour is safety critical or the car comes to a position where it can not proceed. Bosello et al. [27] showed that a DRL algorithm on an autonomous vehicle could be trained by simply reversing the vehicle if it was near to crashing. These approaches demonstrate that online training for autonomous robots is a viable idea but is limited by very simplistic safety systems. Musau et al. [28] used formal reachability theory to enable online training on a small-scale vehicle. Their method depends on estimating future states of the vehicle in real-time resulting in it being very computationally intensive and poorly suited to onboard hardware with limited computation.

In summary, safe online DRL training is a growing field that requires further investigation to explore how DRL agents can be trained onboard real-world robots while guaranteeing safety at the same time.

### III. Methodology

**A. FITenth Platform**

FITenth racing cars are 1/10th the size of real F1 vehicles and are used as a test-bed for autonomous algorithms [29]. The platform focuses on safe algorithms that run autonomously onboard the vehicle. The cars are equipped with a LiDAR scanner for sensing the environment, an NVIDIA Jetson NX as the main 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. The vehicle uses the ROS2 middleware for the sensors, software components and control signals to communicate with each other.

**Problem:** We approach the problem of training a DRL agent to drive a FITenth vehicle autonomously around a provided race track. The task of planning is to use the onboard sensor measurements, LiDAR scan and odometry (estimated using a particle filter [30]) to calculate an optimal steering angle $\delta$ and velocity $v$ that results in the vehicle driving around the track. Training a DRL agent means randomly initializing a policy (neural network), then using the policy to collect experience, and using the collected samples to adjust the policy parameters until the agent can drive around the track.

**Vehicle Model:** The vehicle is a controlled discrete-time system such that $x_{k+1} = f(x_k, u)$. The vehicle state at the current timestep $x_k$, comprises the vehicle location in the $x$ and $y$ directions and the vehicle orientation, such that $x_k = [X, Y, \theta]$. The vehicle control $u$ consists of a steering angle $\delta$, and velocity $v$, such that $u = [\delta, v]$. For our experiments, the vehicle speed is kept constant and thus neglected from the control space.

The safety system, presented in Section III-D, requires that the state space $X$ is discretized into a countable number of states $X_n$. The track map is split into a finite number of blocks by gridding the map with a uniform grid with a resolution of 40 blocks per meter. The orientation angle $\theta$ is split into 41 even angle segments. The control space $U$, consisting of the steering angle range, is split into 9 evenly spaced control modes.

The dynamics are modelled using the single-track vehicle model [31]. The single-track model uses several additional state variables (such as slip angle) that are set to 0 before using the model to update the state. The dynamics are formulated as a difference inclusion such that the next state is always in the set of possible next states, $x_{k+1} \in F(x_k)$. For a given state, the set of all possible next states is written as $F(x_k) = \{ f(x_k, \delta) \mid \delta \in [-\delta_{\text{max}}, \delta_{\text{max}}]\}$.

**B. Supervisory Safety System**

**Supervisory Architecture:** In contrast to solutions that train DRL agents in simulation and transfer them to physical vehicles [13], we present a supervisory safety system that enables the training of DRL agents onboard the physical vehicle.

Figure 1 shows the architecture used during training with the supervisor monitoring the agent and ensuring that only safe actions are implemented on the vehicle. Safe actions are those that do not lead the vehicle to crash into the boundary and are recursively feasible, i.e. after taking a safe action;
another safe action will definitely exist. If the agent’s action is unsafe, the agent receives a penalty (negative reward) from the supervisor. When training is complete, the agent is tested by removing the supervisor to demonstrate that the agent has learned to drive safely around the track.

**Supervisor Operation:** Figure 2 shows how the supervisor fulfills its role of ensuring that only safe actions are implemented on the vehicle through a three-step process of (1) using the current state and action to calculate the next state, (2) checking if the next state is safe, and (3) if unsafe, selecting a safe action.

The supervisor checks if an action is safe by using the current vehicle state and a dynamics model to simulate the next state where the vehicle will be after the planning timestep. A kernel of all the possible states (positions and orientation) of the vehicle on the map is divided into the subsets of safe states \( X_\text{safe} \) and unsafe states through a process described in Section III-D. The next state is evaluated for safety by checking if it is in the subset of safe states. If the next state is safe, then the agent action can be implemented; otherwise, a safe action must be selected.

Selecting a safe action is done using a pure pursuit controller [32] that uses the single-track model of the car to calculate a steering angle that follows the centerline of the race track [33]. Using the pure pursuit controller in this way ensures that the vehicle will always move towards the center of the track where it is safe, away from the potential danger of the track boundaries.

**C. Supervisory Reinforcement Learning**

Deep reinforcement learning uses experience to train an agent to take actions that maximize a reward signal. The TD3 algorithm [34] is used to train the agent since it is a state-of-the-art DRL algorithm for continuous control. The TD3 algorithm is an actor-critic algorithm that uses an actor-network to select actions and a critic network to learn an estimated value of the expected reward. DRL algorithms are trained by allowing the agent to select actions that are implemented and recorded, along with the reward received, in a replay buffer used to train the networks.

**Neural Network Configuration:** The DRL agent uses neural networks with two fully connected hidden layers of 100 neurons each and the ReLu activation function. The agent’s state vector (input) consists of 20 evenly sliced beams from the LiDAR scan scaled from the LiDAR beam range of 10 m to the range [0, 1]. The output is the steering angle, scaled from [-1, 1] (realized using the \( \tanh \) activation function) to the steering angle range of 0.4 rad. The planners operate at a frequency of 10 Hz.

**Episode Reformulation:** In conventional RL, an episode is an ordered set (or trajectory) of state, action, reward, and done tuples, from an initial state to a terminal state (crashing or completing a lap). Using the supervisor, the agent never crashes and always completes laps. Additionally, if the supervisor intervenes, a different action is implemented on the vehicle to that which the agent selected, breaking the link between state, action and next state.

Using the supervisor, we define an episode to run from the initial state until the supervisor intervenes. When the supervisor intervenes, it is recorded as a terminal state, and the supervisor gives the agent a penalty of -1. This penalty for unsafe actions is the only reward used by our method. Figure 3 shows how the definition of an episode has been changed from conventionally requiring many episodes with resetting the vehicle to shorter episodes running while the supervisor does not intervene.

**Online Training:** In the simulation, the neural network could be trained (sample the replay buffer, calculate targets, and update the neural networks) in real-time while samples were being collected. Onboard the vehicle, the requirement for software to run in real-time makes it infeasible to drive and train the network simultaneously. To overcome this problem, the agent collects 20 samples and then stops to train the network before continuing to drive and collecting more data.

**D. Safety Kernel Generation**

The supervisor uses a list (or kernel) of recursively safe states to ensure vehicle safety. As previously mentioned, safe states are defined as states that do not lead to the vehicle crashing into the boundary and are recursively feasible, meaning that every safe state has an action that leads to
another safe state. The formulation and generation of the kernel of safe states are based on the work by Liniger et al. [35].

The kernel of safe states $\mathcal{X}_{\text{safe}}$ is a subset of the discrete state space $\mathcal{X}_h$, for which there exists a safe action. The kernel is calculated using the recursive viability kernel algorithm,

$$K_0 = K_{\text{track}}$$
$$K^{i+1} = \{x_h \in K^i \mid \forall F(x_h) \cap K^i \neq \emptyset\}.$$  

(1)

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’s safe set is initialized ($K^0$) to all the states on the drive-able area of the race track being safe, $K_{\text{track}}$. The algorithm then recursively generates smaller safe sets by looping through each safe state from the previous iteration 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. This process results in a 3-dimensional kernel of recursively feasible safe states, $\mathcal{X}_{\text{safe}}$.

Figure 4 shows how the kernel grows inwards from the track boundaries until all remaining states are safe. Figure 4 shows the kernel with the vehicle orientation pointing towards the right. The kernel shape depends on the vehicle orientation.

IV. EVALUATION

The evaluation compares the performance of a DRL agent trained online a physical vehicle with the safety system against a baseline agent, trained offline in a simulator and then transferred to the vehicle.

The safety system is used to train a DRL agent, with no a priori knowledge or training, onboard a vehicle to drive around a track autonomously. The online agent is trained for two laps (around 800 steps), stopping to train the networks at intervals of 20 steps. The vehicles drive at a constant speed of 2 m/s in environment 1 and 1.5 m/s in environment 2. The open-source F1Tenth simulator [36] is used to simulate the vehicle.

**Baseline Agent:** The baseline agent uses the same neural network configuration as the safety agent but is trained offline in simulation for 30,000 steps and then loaded onto the vehicle. The baseline agent uses a reward signal that rewards the agent for progress made along the track centerline (same as [13]). The reward is the difference between the centerline progresses $p$, scaled according to the length of the track centerline $p_{\text{total}}$, written as

$$r_t = (p_t - p_{t-1})/p_{\text{total}}.$$  

(2)

In addition to the reward for progress, the agent receives a large reward of 1 for completing a lap and a large punishment of -1 for crashing. This means that if the agent completes a lap, it will receive a reward of 2, comprised of 1 representing the sum of intermediate progress rewards, and 1 at the timestep when the lap is completed.

Figure 5 shows the rewards achieved per episode by the baseline planner, which is trained offline in the simulator, using the progress reward signal. The graph shows that at the beginning of training, the agent receives many negative rewards; as the training progresses, the average reward the agent achieves increases. By the end of the training, the agent can consistently complete laps without crashing.

**A. DRL Online Training**

We investigate the supervisor’s ability to train an agent by analyzing the effect of the supervisor in keeping the agent safe and the rewards given by the supervisor to the agent.

Figure 6 shows the first training lap of a random agent being trained online using the supervisor. The green points show where the agent stopped to train the neural networks.
Fig. 7. Comparison of steering actions selected by the agent (green) and those implemented by the safety system (blue) during training of the safety agent in simulation in environment 1.

The trajectory shows the agent’s squiggles as it veers to one side and then to the other. This trajectory demonstrates that the safety system can keep a randomly initialized vehicle from crashing into the track boundaries.

**Supervisor Effect:** A graph comparing the steering angles selected by the agent (green) and the safe actions implemented by the supervisor (blue) is shown in Figure 7. At the beginning of the training, the agent rarely selects safe actions. As the training progresses, the agent selects more safe actions, and towards the end, the agent rarely selects an unsafe action.

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

**Online Training Rewards:** We measure how quickly the agent can learn when being trained online using the supervisory system. Since the episodes have been reformulated (see Section III-C), they always end with a terminal reward of -1 when the supervisor intervenes. Therefore, we use the sum of the reward achieved every 20 steps (the interval of data collection between the agent stopping to train) as the metric to measure the online training performance. Using this metric, the worst reward is -20 if the supervisor intervenes at every step, and the maximum reward is 0 if the supervisor never intervenes.

![Graph showing online training rewards](image)

**Figure 8.** Training rewards per 20 steps for safety agent trained on the physical vehicle (blue dots) with moving average (red).

Figure 8 shows a graph of the sum of rewards achieved every 20 steps by the agent trained online the physical vehicle. The graph shows that in the beginning, the agent receives very low rewards; as time progresses, the agent receives higher rewards. After around only 400 steps, the agent displays a significant improvement. The improvement after 400 steps corresponds to the graph of safe steering actions in Figure 7 showing that the agent requires less intervention between 300-400 training steps.

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

**B. DRL Performance Evaluation**

**Qualitative Analysis:** Trajectories of the trained agents are provided for comparison. Figure 9 shows two real-world trajectories of one agent trained in simulation and then transferred to the vehicle and one agent trained online on the vehicle with the SSS.

![Real-world trajectories](image)

**Figure 9.** A comparison of real-world trajectories driven by the baseline agent trained in simulation (red) and the online trained agent with the SSS (blue) in environment 1.

Figure 9 shows that the agent trained in simulation has a very squiggly, un-smooth path. It regularly comes close to the track boundaries and almost crashes several times. The DRL agent trained on the vehicle using the SSS has a much smoother trajectory. The vehicle drives in a straight line through the environment, smoothly turning the corners and not coming too close to the walls.

**Quantitative Analysis:** Table I presents the quantitative results of the offline (baseline), and online (SSS) trained

![Table I](image)
DRL agents in two different environments with the metrics of distance traveled, lap time, absolute mean steering and curvature. The SSS generally leads to a lower mean steering angle and lower total curvature of the trajectories, resulting in lower distance travelled and lower corresponding lap times than the baseline agents. For example, on the physical vehicle driving in environment 1 (shown in Figure 9), the baseline agent traveled 65.0 m, while the SSS agent traveled only 59.8 m, which is 5.2 m shorter. The average steering angle for the SSS agent was 0.03 radians, compared to the mean steering angle for the baseline of 0.3 radians. The baseline total curvature was significantly more (207.5) than the SSS agent’s (86.3).

We can show that our SSS technique can outperform classically trained DRL methods with much smoother steering actions in both simulation and real-world tests. Although the SSS also performs worse in reality compared to simulation, training the DRL agent on the real car shows a definite improvement in the performance of the physical vehicle compared to the baseline.

Robustness: A crucial aspect of DRL agents is their ability to learn general policies that can be transferred to other environments. Both the agents trained in simulation and on the physical vehicle are tested on the track they were trained on (environment 1) and a different test track (environment 2, shown in Figure 10). The first observation is that both the baseline and SSS agents can complete laps on a different track to the one that they were trained on, highlighting the advantage of the flexibility and adequate generalization of DRL agents. Figure 10 shows that the trajectories followed in environment 2 display a similar pattern to that of environment 1. The SSS agent takes a smoother path and swerves less than the baseline. This outcome is reinforced by the quantitative results in Table I which show that the SSS achieves a shorter lap time (1.5 s different), with a lower mean steering angle (0.09 versus 0.31) and less total curvature (49.3 versus 86.6) than the baseline planner. Therefore, we conclude that the SSS agent learns more general behaviour, as demonstrated by improving performance on a different track.

V. CONCLUSION

This paper presented a supervisory safety system capable of training a DRL agent for autonomous driving onboard a physical car. The supervisory safety system uses a viability kernel to check if the DRL agent’s action is safe. A safe action is implemented if the DRL detects and calculates an unsafe region. Our method was demonstrated to be effective for safely training an agent with no prior training onboard a physical vehicle to drive autonomously around a race track. Our results showed that the agent trained online with the SSS performed better than the agent trained in simulation by driving a shorter path around the track. The safety agent selects a much smoother path (swerving less) than the baseline, without the need for additional reward hacking or action regularisation, which has been identified as a difficulty in other DRL approaches to autonomous driving [27], [13]. Furthermore, we showed that our system could train an agent much faster than currently available baseline DRL algorithms that only train in simulation. By detecting unsafe actions, the safety supervisory prevents the DRL from exploring unsafe and unstable regions, preventing the DRL algorithm from training longer. Once trained, our agent could be transferred to a different environment where it outperformed baseline DRL algorithms and demonstrated that the safety system could effectively reduce the sim-to-real gap.

Future work should address expanding safe learning onboard physical robots to other physical platforms such as UAV control and high-speed autonomous racing. This task requires extending the principle of formulating a supervisor and a safety policy that can operate the robotic system at its performance limits and include more control actions, e.g. velocity, acceleration, and various vehicle dynamics parameters.
