ARIEL UNIVERSITY

Reinforcement Learning Enhanced by Directly-Designed Algorithms for Autonomous Driving

A thesis submitted in partial fulfillment of the requirements for the degree Doctor of Philosophy

By

Gabriel Hartmann

This work was prepared under the supervision of Prof. Zvi Shiller and Prof. Amos Azaria

Submitted to the Senate of Ariel University

December 2022

Abstract

Autonomous driving includes several sub-fields, such as perception, motion planning, and control. There are distinct motion planning objectives, such as increasing energy efficiency and minimizing travel time. The travel time is minimized (i.e., driving at the time-optimal velocity) by maximizing the speed, subject to several constraints, such as traffic laws, sensing limitations, and passenger comfort. However, the most critical constraint that must be respected in every scenario is dynamic stability. By "dynamic stability," we refer to constraints that are functions of vehicle dynamics, such as preventing rollover or sliding. This work focuses on autonomous driving near the time-optimal velocity in several challenges. Two major approaches are used to solve such problems: direct solutions, where the solution is based on an expert's understanding of the problem, and machine learning, where the solution is automatically learned from data. This work proposes several algorithms for autonomous driving that demonstrate the utility of the direct approach and the combination of these two approaches to speed up the learning process and ensure the safety of a learned solution.

Contents

Abstract **i**

1 Introduction **1**
1.1 The Problem 1
1.1.1 Direct vs. Machine Learning Approaches for Autonomous Driving 2
1.2 Review of the Papers 3
1.2.1 Competitive Driving of Autonomous Vehicles 3
1.2.2 Deep Reinforcement Learning for Time Optimal Velocity Control using Prior Knowledge 4
1.2.3 Model-Based Reinforcement Learning for Time-Optimal Velocity Control 4
1.2.4 Meta-Reinforcement Learning Using Model Parameters 5
1.3 Connection Between the Papers 5

2 The Papers **7**

3 Discussion and Conclusions **42**

Bibliography **46**

Chapter 1

Introduction

1.1 The Problem

The operation of autonomous vehicles requires the synergetic application of a few critical technologies, such as sensing, motion planning, and control.

To simplify the autonomous driving task, it is usually divided into hierarchical modules. The first module is responsible for perception; it receives raw data from sensors as input and returns information about the environment and surrounding objects. The output from this module is used by the motion planning module that includes high-level navigation (e.g., whether to turn at an intersection) and low-level planning that defines the steering and throttle commands, while considering the vehicle's capabilities and the near environment. The last module performs low-level steering, throttle, and braking control.

A significant part of autonomous driving research is focused on perception. It includes estimating the vehicle's state and determining the vehicle's surroundings, such as the location of other vehicles. In this work, we use existing perception techniques and focus on motion planning.

Common motion planning objectives include energy efficiency and travel time. The travel time is minimized by maximizing the vehicle's speed, subject to several constraints, such as traffic laws, sensing limitations [1], and passenger comfort [2]. However, the most critical constraint that must be respected in every scenario is dynamic stability. By "dynamic stability," we refer to constraints that are functions of vehicle dynamics, such as preventing rolling over or sliding [3, 4, 5, 6]. The vehicle's dynamic capabilities, such as its maximum and minimum acceleration, ground/wheels interaction, terrain topography, and path geometry, map to an upper-velocity limit, above which one of the dynamic constraints is violated. It follows, that the time-optimal velocity should not cross the velocity limit.

We can identify two main challenges in autonomous driving: Driving a single vehicle safely under challenging conditions [7, 8, 9], and navigating in crowded dynamic environments [10, 11, 12]. These two challenges are combined in an extreme

way in autonomous racing [13, 8, 14, 15], where all vehicles drive very close to their performance envelope and all try to reach the same goal, which is winning the race.

1.1.1 Direct vs. Machine Learning Approaches for Autonomous Driving

Two major approaches are used to solve problems, such as developing an autonomous car: directly designing the solutions and machine learning. In the direct approach, an expert directly develops an algorithm based on general information about the world and the problem, such as mathematics and physics tools, measurements, and previous experience. In contrast to such a direct approach, machine learning automatically extracts knowledge from data and gets solutions that were not directly programmed. For control problems, such as driving a car or controlling a robot, Reinforcement Learning (RL) reduces the development effort by automatically learning from interaction with the environment. The RL agent is usually general and does not include specific information about the target environment; this allows it to adapt to different environments without the need to develop a new controller.

An extreme learning approach is end-to-end learning, which learns a direct mapping between the raw data from the sensors to the final actions. This approach is used, for example, for playing Atari games [16], steering of a real car [17] given only camera data, and simulated car racing [18]. Because the problem is not modularized, higher performance can be potentially achieved because the entire system is optimized as a whole [17]. However, this approach is usually inefficient because the solution needs to be learned from scratch since data from other problems are not available. For this reason, a large amount of data is required to solve one problem [19].

Each approach has advantages and disadvantages. With the direct approach, the developer can take into consideration a broad understanding of the problem without the need to test the system in the real world. RL, on the other hand, needs to extract this knowledge from the interaction with the environment, which requires a high amount of data and leads to long training times. This is especially challenging in safety-critically problems such as driving, where safety needs to be ensured during the whole training phase, including at the beginning, when the learned model may still be highly inaccurate [20]. Another challenge is how to ensure the safety of a learned solution. The internal decision process of the learned solution is usually nontransparent because the learned solution relies on complex models (e.g., neural networks). Therefore, the solution can only be guaranteed statistically and not by a mathematical analysis as in a directly designed solution.

Despite the advantages of direct solutions, they are limited by the human understanding of the problem and the optional solutions. To solve complex problems, usually, a simplified model is developed. This model should capture the most important properties of the problem but should be simple enough to be practical. For example in autonomous driving, the real dynamics of a vehicle and the interaction with the environment are complex, but they may be simplified by modeling the vehicle as a point mass [21], or a bicycle model [22, 8]. Indeed, RL agents may achieve higher performance than human-crafted solutions by training a highly expressive model such as a deep neural network [23, 24, 25].

Research Goals The research goals of this work are:

- Developing learning-based and directly-designed algorithms for time-optimal velocity control and competitive driving
- Speeding up the learning process by combining learning-based and a direct approach
- Ensuring the safety of a learned solution using a directly-designed dynamic model
- Developing a new concept for meta-reinforcement learning that allows quick adaptation to different vehicles

1.2 Review of the Papers

The next sections review the four papers that constitute this thesis's main part.

1.2.1 Competitive Driving of Autonomous Vehicles

This paper [26] addresses the issue of autonomous competitive yet safe driving in the context of the Indy Autonomous Challenge (IAC) simulation race. The IAC is the first multi-vehicle autonomous head-to-head international competition, reaching speeds of 300 km/h along an oval track modeled after the Indianapolis Motor Speedway (IMS). Autonomous racing has unique challenges emanating from the unique properties of the race vehicle, its extreme speeds, and the competitive nature of the driving.

The autonomous racing controller was developed based on the underlying principle that emphasizes safety over performance. To this end, our controller attempts to avoid collisions, including those that the race rules placed the responsibility to

avoid on the opponent's vehicle. Despite this conservative approach, it did not diminish the racing controller's competitiveness.

The racing controller is based on a repeated search for the locally best maneuver that avoids collisions with opponent vehicles, attempts to follow the globally optimal race line, and obeys the race rules. The best maneuver is selected from a tree of local maneuvers, generated to a set of goals across the track by applying time optimal control to a point mass model. The selected maneuver is then tracked using the pure-pursuit controller [27].

Our controller was tested and analyzed in simulations with 6 competing vehicles, all running the same algorithm. It was also tested in the IAC simulation race, where it competed in races with up to 7 vehicles.

1.2.2 Deep Reinforcement Learning for Time Optimal Velocity Control using Prior Knowledge

This paper [28] proposes a reinforcement learning method for driving a vehicle at the time optimal speed along a known arbitrary path. It learns the acceleration (and deceleration) that maximizes vehicle speeds along the path without losing its dynamic stability.

Additionally, we propose to combine prior knowledge from a direct solution with the RL agent to allow the RL agent to begin the training with a relatively good policy, thus speeding up the training process. Instead of learning the actions using RL directly, only the variation from a nominal time-optimal controller is learned by the RL agent. For this purpose, we use a directly-designed controller that controls a vehicle along a path while avoiding rollover, slipping, and losing contact with the ground [21]. This direct method computes the solution efficiently, making it suitable for real-time use. Another variant, proposed in this paper, combines the direct and learning approaches by adding the directly computed action as an additional feature in the RL agent's state.

1.2.3 Model-Based Reinforcement Learning for Time-Optimal Velocity Control

This paper [29] presents a model-based reinforcement learning algorithm for driving a vehicle at near-time-optimal speeds along any arbitrary path. Model-based reinforcement learning is an effective way to learn the complex dynamic model of the vehicle from its actual responses, thus bridging the gap that separates the real vehicle dynamics from its directly-designed dynamic model. The dynamic model of

the vehicle is learned and used for planning the acceleration actions that maximize vehicle speeds along the path. It is shown that the vehicle achieves higher speed compared to the direct approach in just one minute of training.

However, to ensure the safety of the learned controller and avoid failure during training, a dual-model approach is proposed that protects the vehicle from reaching dynamically unstable states. A simplified and conservative directly-designed model that accounts for the vehicle's safety is used to predict if every command proposed by the RL agent is dynamically safe. An alternative safe local maneuver is executed if an unsafe maneuver is attempted. This approach results in safe driving from the beginning of the training process.

1.2.4 Meta-Reinforcement Learning Using Model Parameters

In meta-reinforcement learning, an agent is trained in multiple different environments and attempts to learn a meta-policy that can efficiently adapt to a new environment. This paper [30] presents a *Reinforcement learning Agent using Model Parameters* (RAMP) that utilizes the idea that a neural network trained to predict environment dynamics encapsulates the environment information. Therefore, its parameters can be used as the context for the agent's policy.

RAMP is constructed in two phases: in the first phase, a multi-environment parameterized dynamic model is learned. In the second phase, the model parameters of the dynamic model are used as context for the multi-environment policy of the model-free reinforcement learning agent.

During the first phase, RAMP learns a neural network that predicts the environment dynamics for each environment. However, since the number of the neural network's parameters is high, a multi-environment neural network that shares the majority of the parameters is trained. As a result, only a small set of parameters encapsulate the environment properties, and these parameters are used as a compact context for the meta-policy.

1.3 Connection Between the Papers

All four papers address the problem of autonomous driving near the performance envelope. Each paper demonstrates a different perspective of direct-design compared to machine learning approaches for developing the solutions.

The first paper, *Competitive Driving of Autonomous Vehicles*, relies solely on a directly-designed algorithm for high-performance autonomous racing. Relying on a direct

approach is called for when an immediate solution is required, and a reliable learning-based solution still does not yet exist, like in this domain.

However, directly defining an accurate model that considers all the vehicle and environmental constraints is impossible. Therefore, RL is used to learn the time-optimal velocity of a vehicle to achieve higher speeds compared to a directly-designed controller, as demonstrated in the second paper, *Deep Reinforcement Learning for Time Optimal Velocity Control using Prior Knowledge*. Furthermore, the paper demonstrates that combining the direct solution with a reinforcement learning approach speeds up the learning process.

The second paper's approach still requires relatively long training times, and failures can still occur (i.e., losing dynamic stability). The third paper, *Model-based Reinforcement Learning for Time-optimal Velocity Control*, proposes a model-based reinforcement learning algorithm that decreases training time to under one minute. Furthermore, the direct solution is used to ensure the safety of the actions selected by the reinforcement learning agent.

Direct and hybrid approaches require developing an environment-specific solution and relying on expert knowledge. The fourth paper, *Meta-Reinforcement Learning Using Model Parameters*, presents a meta-reinforcement learning approach that does not require an explicit direct solution but instead extracts information from multiple environments to allow efficient adaptation to a new environment. Even though a direct solution is not explicitly used, the knowledge from direct solutions assists in constructing a better reinforcement learning agent.

Chapter 2

The Papers

IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received 14 September 2022, accepted 10 October 2022, date of publication 19 October 2022, date of current version 27 October 2022.

Digital Object Identifier 10.1109/ACCESS.2022.3215984

RESEARCH ARTICLE

Competitive Driving of Autonomous Vehicles

GABRIEL HARTMANN[1,2], ZVI SHILLER[1], (Life Senior Member, IEEE), AND AMOS AZARIA[2]
[1]Department of Mechanical Engineering and Mechatronics, Ariel University, Ariel 4070000, Israel
[2]Department of Computer Science, Ariel University, Ariel 4070000, Israel

Corresponding author: Gabriel Hartmann (gabrielh@ariel.ac.il)

This work was supported in part by the Ministry of Science and Technology, Israel.

ABSTRACT This paper addresses the issue of autonomous competitive yet safe driving in the context of the Indy Autonomous Challenge (IAC) simulation race. The IAC is the first multi-vehicle autonomous head-to-head competition, reaching speeds of 300 km/h along an oval track modeled after the Indianapolis Motor Speedway (IMS). We present a racing controller that attempts to maximize progress along the track while avoiding collisions with opponent vehicles and obeying the race rules. To this end, the racing controller first computes a race line offline. During the race, it repeatedly computes a small set of dynamically feasible maneuver candidates, each tested for collision with the opponent vehicles. It then selects a collision-free maneuver that maximizes the progress along the track and obeys the race rules. Our controller was tested in a 6-vehicle simulation, managing to run competitively with no collision over 30 laps. In addition, it managed to drive within a close range of the leading vehicle during most of the IAC final simulation race.

INDEX TERMS Autonomous vehicles, collision avoidance, motion planning, multi-robot systems.

I. INTRODUCTION

Autonomous racing has gained great interest in recent years [1], resulting in a number of racing competitions [2], [3], [4]. In competitive driving, the challenge is to maintain safety and minimize motion time while competing against other vehicles that attempt to achieve the same goal. We address the issue of competitive driving in the context of our participation in the recent Indy Autonomous Challenge (IAC) [4]. The IAC is an international competition intended to promote the development of algorithms for driving under challenging conditions. Its goal is to demonstrate the world's first multi-vehicle, high-speed, head-to-head autonomous racing.

The IAC, held on 2020-2021, was carried out in two stages: a simulation race and a real race on the Indianapolis Motor Speedway (IMS) with the Dallara AV-21 autonomous race car [5]. Fig. 1 depicts the IMS and the race car. Over 30 teams from universities worldwide participated in this challenge. A prerequisite for entering the competition was to demonstrate autonomous driving of a real vehicle. Our team's entry submission is shown in [6].

The associate editor coordinating the review of this manuscript and approving it for publication was Jjun Cheng.

(a) (b)

FIGURE 1. (a) The Indianapolis Motor Speedway (b) the AV-21 autonomous race car [5].

While the simulation race achieved its stated goals, the real race ended up with solo driving and controlled overtaking between two vehicles [7]. Evidently, competitive driving with real vehicles is not yet ready and will hence rely for the time being on high-fidelity simulations. It is in the context of this reality that we present our work on competitive driving.

A. CHALLENGES OF AUTONOMOUS RACING

Autonomous racing has unique challenges emanating from the unique properties of the race vehicle, its extreme speeds, and the competitive nature of the driving.

1) EXTREME SPEEDS

Racing speeds coupled with limited frequencies of the sensor readings lead to state updates at large distance intervals

compared to the vehicle size and the distance between neighboring vehicles. Furthermore, driving near the vehicle's performance envelope in close proximity to neighboring vehicles leaves little room for correction and hence requires high-fidelity predictions of the behavior of the opponent vehicles.

2) COMPETITIVE DRIVING

Competitive driving forces the competitors to race in close proximity to opponent vehicles. As a result, the time difference between the leading teams is measured in fractions of a second. This, in turn, forces all competitors to drive on the performance envelope of the vehicle and the driver, leaving little room for safety. Although the goal is to win the race, in our opinion, especially for the first time that such a head-to-head race is taking place, safer behavior and larger safety margins should be preferred over pushing the vehicle's performance to its limits.

3) AERODYNAMIC FORCES

The aerodynamics of a race car has two main effects: a down-force that increases the tire grip and consequently lets the car reach a lateral acceleration of over 2.5 g, which in turn significantly increases the vehicle's maximal speeds along an oval track. The second effect is slipstream, or drafting, which reduces the drag on the vehicle that follows behind at close range. Exploiting the slipstream is an important strategy in car racing since it allows vehicles with identical dynamics to overtake each other at high speeds.

4) RACING RULES

The racing rules [8] for the autonomous race were derived from the rules used in human-driven races [9]. Central to these rules is the principle that an overtaking vehicle is fully responsible for avoiding collision with a vehicle that is moving on its race line, without causing it to veer off its race line. If a collision occurs in such a situation, the overtaking vehicle is held responsible for the collision and is removed from the race.

B. THIS PAPER

The autonomous racing controller was developed based on the underlying principle that emphasizes safety over performance. To this end, our controller attempts to avoid collisions, including those that the race rules placed the responsibility to avoid on the opponent vehicle. While this is a conservative approach to competitive driving that attempts to drive safer than what is allowed by the race rules, it is in our opinion the right approach to ''responsible competitive driving.''

The racing controller is based on a repeated search for the locally best maneuver that avoids collisions with opponent vehicles, attempts to follow the globally optimal race line, and obeys the race rules.

The best maneuver is selected from a tree of local maneuvers, generated to a set of goals across the track by applying time optimal control to a point mass model. The selected maneuver is then tracked using the pure-pursuit controller [10].

Our controller was tested and analyzed in simulations with 6 competing vehicles, all running the same algorithm. It was also tested in the IAC simulation race, where it competed in races with up to 7 vehicles. We note that the controller is efficient for racing with even more vehicles since the algorithm's complexity is linear in the number of opponent vehicles.

Despite its simplicity, this controller demonstrated safe and competitive driving, while overtaking other vehicles over the 30 lap run, without causing even a single collision between any of the 6 competing vehicles. In the IAC simulation race, our vehicle managed to avoid collisions while staying within close range of the leading vehicle for the majority of the semi-finals and finals. Our strategy of avoiding collisions at all cost caused our vehicle to spin off the track while attempting to avoid collision with another vehicle that entered our safety bound. This placed our vehicle in the 6^{th} place in the simulation competition.

The contributions of this paper are twofold:

- Presenting a complete controller for a multi-vehicle race that drives competitively, avoids collision, and obeys the race rules.
- Presenting multi-vehicle (6 vehicles) race results and providing metrics to quantify the performance of a multi-vehicle race.

II. RELATED WORK

Most previous work in the field of autonomous racing have focused on solo-racing [11], [12], [13] or racing against a single opponent vehicle [14], [15]. Very few studies have addressed multi-vehicle racing with more than two opponent vehicles [16], [17]. This paper describes our multi-vehicle racing controller and demonstrates it (in simulation) in a 6-vehicle and 7-vehicle races.

Autonomous racing has been tested in competitions, such as the Formula student challenge [2] or Roborace [3], which focus mainly on solo racing. The recent IAC real race demonstrated controlled overtaking between two vehicles [7]. A multi-vehicle race has been so far demonstrated only in the IAC simulation race [4]. For a comprehensive survey on autonomous racing, see [1].

We now survey some of the existing work on autonomous racing regarding each of our racing algorithm modules, namely global race line planning, opponents' trajectories prediction, local planning, and trajectory tracking.

A. GLOBAL RACE LINE PLANNING

A global race line is the time optimal path (the projection of the time optimal trajectory on the track) for a one lap solo race. Computed offline, it serves as a reference input to a trajectory-following controller [11], [18]. It is also used in multi-vehicle racing as a reference for the online planner, as is demonstrated later in this paper.

The race line optimization for autonomous racing usually minimizes lap time, which can be computed by solving a non-linear optimization problem that considers vehicle dynamics and track constraints [11], [19]. An alternative approach is to construct a race line by minimizing curvature along the path, which impacts the vehicle's lateral acceleration [20]. Since vehicle speeds vary little along the oval track, optimizing path curvature offers a good kinematic approximation for the time optimal dynamic optimization. It is computationally efficient and it produces race lines that differ only slightly from the time optimal line. We used open source software [21] for the offline race line optimization.

B. TRAJECTORY PREDICTION

An early approach to trajectory prediction was based on Kalman filter to generate short time horizon predictions while assuming a constant velocity and acceleration along each segment [22]. A more recent approach trains recurrent neural networks to predict the multi-modal distribution of future trajectories [23]. While the learning-based approach, using neural networks for prediction, is promising, it requires a large amount of data, and it does not guarantee the quality of the learned solution [24].

For computational efficiency, we devised a prediction module that accounts for the opponent's current state and track geometry. We used a conservative assumption that while moving across the track, the opponent vehicle will cross the entire track, until reaching either the inner or outer boundary, and will follow that track boundary while maximizing its speed during the set time horizon.

C. LOCAL PLANNING

Model Predictive Control (MPC) was used to optimize a local trajectory for solo driving [12], [25], and for avoiding static [26] and dynamic obstacles [27]. To avoid obstacles, Wischnewski et al. [27] and Liniger et al. [26] first plan a free driving zone using graph search, then generate an optimal trajectory within the selected zone. In [15], MPC was used directly to overtake a single opponent vehicle in a racing scenario.

Random sampling was used to search for a feasible, collision-free solution using Rapid Random Trees (RRT) [28] and RRT* [13] and CLRRT$^{\#}$ [29] to compute an asymptotically optimal solution. It is important to note that random sampling-based methods do not guarantee a solution in a finite computation time. Furthermore, these methods typically require excessive computation time that is detrimental to real-time execution at high-frequency.

A less computationally demanding solution is to select an optimal trajectory from a small set of trajectories, generated, for example, by simulating various constant steering angles and velocities [26], or by generating cubic spirals to a set of target points [30].

The above mentioned methods are either unsuitable for high-speed racing or have not been demonstrated for more than three vehicles. Our online planner selects a trajectory from a small set of maneuvers generated by a point mass model to a number of target points, while minimizing motion time, avoiding collisions with the opponent vehicles, and attempting to reach the optimal race line, as described later in Section V.

D. TRAJECTORY TRACKING

Trajectory tracking can be done using feedback control by minimizing position error, [18], [31], Model Predictive Control (MPC) by generating a sequence of open loop commands that minimize tracking error subject to dynamic constraints [11], [27], and learning-based control that iteratively minimizes lap time and tracking error [32], [33].

We opted for the pure-pursuit algorithm [10], together with low-level linear and angular velocity controllers. It provides satisfactory performance when fine-tuned for driving along the smooth trajectories generated by the local planner. It is easy to tune, owing to the low number of parameters used, compared to other methods.

E. REINFORCEMENT LEARNING APPROACH

Recently, Wurman et al. [17] beat human drivers in a racing video game, using reinforcement learning. Unlike the hierarchical approach of the methods described earlier, the reinforcement learning agent maps a low-dimensional state directly to control commands. Although learning-based approaches are promising, they usually require a high amount of training data, making it challenging to generalize them to training safely in the real world. Such long training also made it impractical for the IAC simulation race because of the slower-than-real-time performance of the simulator. Another challenge is to validate the safety of the learned controller, which is nontransparent (i.e., black box), unlike physics-based controllers.

F. COMPARISON

Most of the current research focuses on single or two-vehicle racing, and there is currently no demonstration of a real race with more than two vehicles. We note that it is impossible to directly compare different methods implemented in different settings in the racing domain. This is because the performance differences between racing competitors are very small, as shown later in Section VII. In addition, in a racing environment, all methods are extensively fine-tuned to a specific setting, and thus, one method cannot be applied, as is, to a different setting. These challenges emphasize the importance of racing competitions such as the Indy Autonomous Challenge that provides an opportunity to compare the performance of various methods on a common ground.

III. SOFTWARE ARCHITECTURE

The software architecture of the racing controller is shown schematically in Fig. 2. An optimal race line is computed offline for a given track, which serves the controller throughout the race. The data from the cameras and radars provide the position and velocity of the surrounding opponent vehicles,

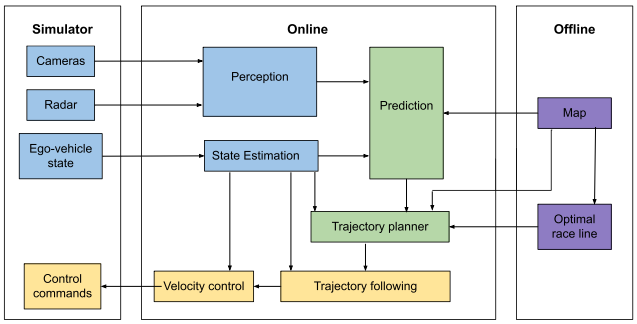


FIGURE 2. The software architecture.

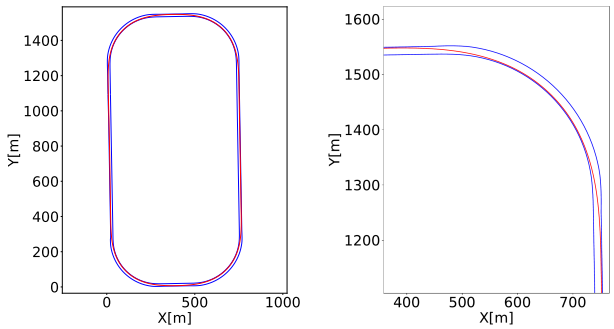


FIGURE 3. (a) The IMS track. (b) The optimal race line around a corner of the track.

and additional simulated sensors provide the ego-vehicle state, e.g., position and velocity. Together with the map and the optimal race line, these data are used by the prediction module to predict the opponent vehicles' future trajectories repeatedly. The trajectory planner uses the same information to plan an optimal local maneuver for the ego-vehicle. This maneuver serves as an input to the trajectory-following controller, which computes the ego-vehicle's desired linear and angular velocities. The linear and angular velocities are controlled by the velocity controller, which outputs the steering, throttle, and brake commands.

IV. COMPUTING THE OPTIMAL RACE LINE

The optimal race line is a time-optimal trajectory when no other vehicles are present on the track. It is computed offline, based on the track geometry and vehicle dynamics.

The IMS is an oval track, 4,023 m (2.5 miles) long, as depicted in Fig. 3a.

We computed the optimal race line using an open-source trajectory optimization software [21]. The optimal race line typically minimizes curvature by entering the corner on the outside boundary of the track, passing through the apex on the inner boundary to the exit point on the outside boundary, as shown in Fig. 3b.

V. ONLINE TRAJECTORY PLANNING

The online trajectory planner computes a collision-free trajectory at 25 Hz. This rate was dictated by the frequency of the simulator sensors updates. This in turn resulted in a travel distance of 3 m between trajectory updates.

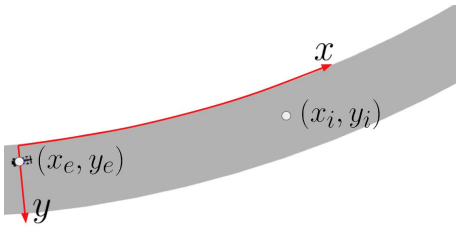


FIGURE 4. Road-aligned coordinate system.

The trajectory computed by the online planner is used as a reference for the trajectory-following controller (see Section VI). The planner first generates a tree of dynamically feasible maneuver candidates to 8 points that span the track width. The best candidate that maximizes progress along the path and avoids collision with the opponent vehicles is then selected.

The online trajectory planner uses a horizon of 200 m, which the stopping distance at 80 m/s. This is also compatible with the sensor range of 200 m. The limited planning horizon decreases the optimality of the local trajectory. However, we mitigate this by attempting to merge with the optimal race line, which is in itself globally optimal. Despite the short time horizon, we demonstrate that the vehicle is able to drive safely and competitively, as described later in Sec. VII.

A. COORDINATE SYSTEM

The coordinate system used in planning the trajectory is attached to the ego-vehicle with the x-axis tangent to the left track boundary and the y-axis normal to track. The position of the ego-vehicle is denoted (x_e, y_e). The ego-vehicle is always located at $x_e = 0$, and y_e represents the ego-vehicle's normal offset from the left boundary (see Fig. 4).

In our coordinate system, for every point (x_i, y_i), $|x_i|$ represents the distance from the ego-vehicle along the track (a negative x_i represents a point behind the ego-vehicle), and y_i represents the offset from the track boundary regardless of the track shape and the ego-vehicle's location.

B. POINT-TO-POINT TRAJECTORY

We now present the computation of a trajectory between two end points, for a point mass model. This is repeatedly used to predict the trajectories of opponent vehicles and for planning maneuver candidates for the ego-vehicle.

Let $p_s = \{x_0, y_0, \dot{x}_0, \dot{y}_0\}$ be the starting point, and $p_g = \{x_g, y_g, \dot{x}_g, \dot{y}_g\}$ be the goal point. We wish to plan a trajectory $C(t) = \{x(t), y(t), \dot{x}(t), \dot{y}(t)\}$, $t \in [0, T]$ so that $C(0) = p_s$ and $C(T) = p_g$. We assume a constant longitudinal speed (along the x axis) so that $\dot{x}_s = \dot{x}_g$. The travel time T to the goal is therefore $T = (x_g - x_0)/\dot{x}_0$. Since the longitudinal distance to the goal is by choice greater than the track width, C is thus generated by using bang-bang control in the lateral direction, while applying the minimum lateral force with a

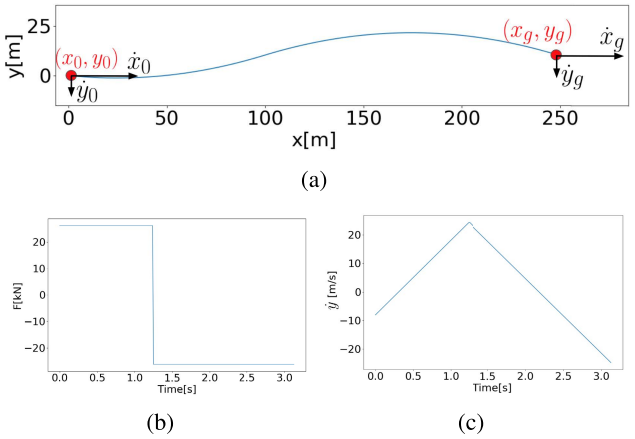


FIGURE 5. A trajectory between given start and goal points. (a) Force F_y is applied on mass m to create a continuous trajectory between the points. (b) The lateral force F_y profile and (c) the lateral velocity $\dot{y}(t)$.

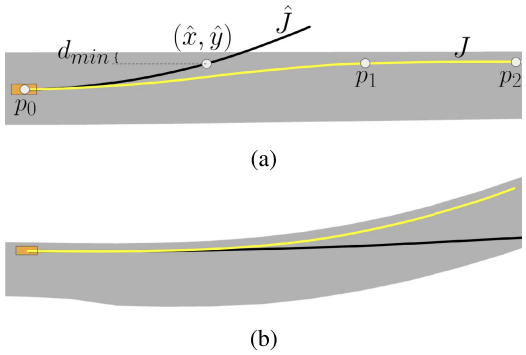


FIGURE 6. Examples of predicted trajectories of an opponent vehicle (marked as an orange bounding box). Black indicates constant curvature trajectory $\hat{J}$, and yellow indicates the predicted trajectory J, which also considers the track boundaries. In (a) $\hat{J}$ exceeds track boundaries; therefore, a lateral-shift trajectory is predicted by J; in (b), although the vehicle drives on a straight line, the predicted trajectory J follows the track geometry.

single switch so that the lateral and the longitudinal end points would be reached both at time T [34].

The lateral force F_y that generates the bang-bang trajectory is:

$$F_y = \frac{-\sqrt{2A} - T(\dot{y}_g + \dot{y}_0) + 2(y_g - y_0)}{T^2 m} \tag{1}$$

where

$$A = (T^2(\dot{y}_0^2 + \dot{y}_g^2) - 2T(y_g - y_0)(\dot{y}_g + \dot{y}_0) + 2(y_g - y_0)^2), \tag{2}$$

and the switching time T_s is

$$T_s = \frac{m(\dot{y}_g - \dot{y}_0) + F_y T}{2F_y(t)}. \tag{3}$$

$C(t)$ is thus computed for $t \in [0, T]$ by:

$$x(t) = x_0 + \dot{x}_0 t \tag{4}$$

$$y(t) = \begin{cases} y_0 + \dot{y}_0 t + \frac{1}{2}\frac{F_y}{m}t^2 & t \le T_s \\ y(T_s) + \dot{y}(T_s)(t - T_s) - \frac{1}{2}\frac{F_y}{m}(t - Ts)^2 & t > T_s \end{cases} \tag{5}$$

$$\dot{x}(t) = \dot{x}_0 \tag{6}$$

$$\dot{y}(t) = \begin{cases} \dot{y}_0 + \frac{F_y}{m}t & t \le T_s \\ \dot{y}_0 + \frac{F_y}{m}(2T_s - t) & t > T_s \end{cases} \tag{7}$$

Figure 5 illustrates an example of a trajectory between two given end states.

C. TRAJECTORY PREDICTION OF OPPONENT VEHICLES

An important part of motion planning in a dynamic environment, especially in racing, is predicting the future positions and velocities of all other vehicles surrounding the ego-vehicle. This is performed by the prediction module, which receives the track boundaries and an opponent vehicle state $s = \{x_0, y_0, \dot{x}_0, \dot{y}_0, \omega_0\}$ as input, where x_0, y_0 is the opponent vehicle's position, $\dot{x}_0, \dot{y}_0$ is its velocity, and ω_0 is its angular velocity. The module predicts the future trajectory $J(t) = \{x(t), y(t), \dot{x}(t), \dot{y}(t)\}$ up to a predefined time horizon $T_{\max}$, which was set to 3 seconds to match the planning horizon.

The prediction module predicts each opponent's trajectory based on its current state under the following two assumptions: the opponent vehicle intends to stay within the track boundaries (as described earlier) and attempts to maximize its velocity (as described in Section V-E). Namely, for a given opponent vehicle, we first predict a future trajectory, $\hat{J}$, that keeps a constant curvature κ, which we approximate by: $\kappa = \frac{\omega_o}{||\dot{x}_o + \dot{y}_o||}$.

It is assumed that the opponent vehicle will stay within the track boundaries. to this end, our prediction module accounts for the track boundaries as follows: Let $(\hat{x}, \hat{y})$ be the first position on $\hat{J}$, at which the opponent vehicle approaches one of the track boundaries at a distance $d_{\min}$. We define three points, p_0, p_1 and p_2, each point consisting of $p_i = \{x_{p_i}, y_{p_i}, \dot{x}_{p_i}, \dot{y}_{p_i}\}$. The prediction module connects these points by a point mass maneuver as explained in Section V-B.

The first point, p_0 is derived from the opponent vehicle's current state s, such that $x_{p_0} = x_o$,$y_{p_0} = y_o$, $\dot{x}_{p_0} = \dot{x}_o$, and $\dot{y}_{p_0} = \dot{y}_o$. The second point, p_1, is based on $(\hat{x}, \hat{y})$, but we assume that the opponent vehicle will not increase its curvature; therefore, we assume that $\hat{y}$ will be reached later on, by a predefined factor, k. That is, $x_{p_1} = k(\hat{x} - x_o)$, $y_{p_1} = \hat{y}$, $\dot{x}_{p_1} = \dot{x}_o$, $\dot{y}_{p_1} = 0\}$.

The third point, p_2 retains a path parallel to the boundary, up to $T_{\max}$, i.e., $x_{p_2} = \dot{x}_o T_{\max}$, $y_{p_2} = \hat{y}$, $\dot{x}_{p_2} = \dot{x}_o$, and $\dot{y}_{p_2} = 0\}$. Examples of predicted trajectories are shown in Fig. 6.

D. CREATING MANEUVER CANDIDATES

The online trajectory planner plans a set of dynamically feasible maneuver candidates and selects one according to multiple criteria, as follows:

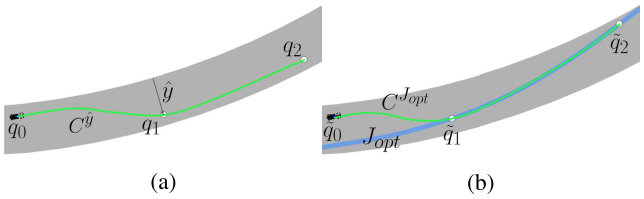


FIGURE 7. Illustration of the maneuver candidates planning: (a) lateral-shift maneuver and (b) a maneuver to the optimal race line.

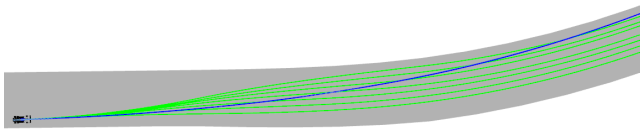


FIGURE 8. Example of 7 lateral-shift maneuvers (green) and a maneuver that merges with the optimal race line (blue).

1) LATERAL-SHIFT MANEUVER

Given the ego-vehicle's state $s_e = \{x_e, y_e, \dot{x}_e, \dot{y}_e\}$ and a lateral-shift target $\hat{y}$, the planner module creates a lateral-shift maneuver, $C^{\hat{y}}$, by connecting three points, $q_i = \{x_{q_i}, y_{q_i}, \dot{x}_{q_i}, \dot{y}_{q_i}\}$, $i \in \{0, 1, 2\}$, consisting of the q_0, the ego vehicle's current state, q_1 being the target point of the lateral-shift, and q_2 being a point down the track that retains the lateral shift of q_1. More formally, $q_0 = s_e$, $q_1 = \{(\hat{y} - y_e)b + c, \hat{y}, \dot{x}_e, 0\}$, $q_2 = \{x_{\max}, \hat{y}, \dot{x}_e, 0\}$, where $\hat{y}$ is the lateral-shift target, b, c are constants and $x_{\max}$ is the planning horizon. Figure 7a illustrates a lateral-shift maneuver.

We define N equally spaced lateral-shifting targets representing potential $\hat{y}$ values. The first and last targets are far enough from the boundaries to allow a vehicle to reach them safely. Our planner generates N lateral-shift maneuvers, one for each defined target. Let $\boldsymbol{M'}$ be the set of all lateral-shift maneuvers:

$$\boldsymbol{M'} = \bigcup_{i=0}^{N-1} C^{\hat{y}_i}, \quad \hat{y}_i = d_{\min} + \frac{w - 2d_{\min}}{N-1} i \tag{8}$$

where $d_{\min}$ is the minimal distance to keep away from track boundaries. The width of the track, w, in most segments along the oval is 14 m; we set the number of lateral-shift maneuvers $N = 7$ to achieve a distance between targets that is close to the vehicle's width (2 m).

2) MANEUVER TO THE OPTIMAL RACE LINE

In addition to these lateral-shift maneuvers, we plan a maneuver $C^{J_{\text{opt}}}$, which smoothly merges with the optimal race line J_{opt}. We define the following three points $\tilde{q}_i = \{x_{\tilde{q}_i}, y_{\tilde{q}_i}, \dot{x}_{\tilde{q}_i}, \dot{y}_{\tilde{q}_i}\}$, where $i \in \{0, 1, 2\}$.

The first point $\tilde{q}_0 = s_e$; the second, $\tilde{q}_1 \in J_{\text{opt}}$ such that $(y_{\tilde{q}_1} - y_e)\tilde{b} + \tilde{c} = x_{\tilde{q}_1}$, where $\tilde{b}$ and $\tilde{c}$ are predefined constants. Finally, $\tilde{q}_2 \in J_{\text{opt}}$ such that $x_{\tilde{q}_2} = x_{\max}$. See Fig. 7b).

The full set of the maneuver candidates is $\boldsymbol{M} = \boldsymbol{M'} \cup C^{J_{\text{opt}}}$, as shown in Fig. 8.

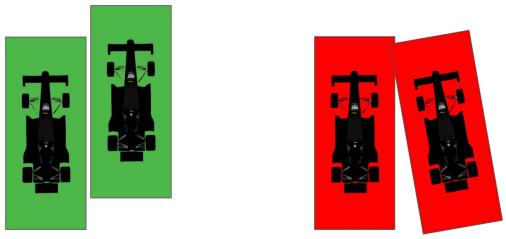


FIGURE 9. Rectangular safety bound. In (a) No overlap between safety bounds; (b) the safety bounds overlap.

E. VELOCITY RE-PLANNING

Although the maneuver candidates $\boldsymbol{M}$ and the predicted trajectories of the opponent vehicle include velocities in addition to positions, these velocities were only used to define the direction of the paths and to estimate the lateral forces on the vehicle. Therefore, the velocity profiles are re-planned to represent the future motion more accurately by assuming that the vehicles accelerate along the trajectories until they reach the maximal velocity. This is possible because our planned trajectories approximate the vehicle dynamics and thus allow maintaining maximal velocity—without losing control—when following them.

F. COLLISION

To avoid collisions, we define a safety bound around the vehicle owing to the uncertainty inherent in our problem. We attempt not only to avoid a collision with another vehicle but also to avoid any overlap between the safety bounds around both vehicles. We use a rectangular safety bound, which best matches the vehicle's shape (as shown in Fig. 9). The vehicle's length is 5 m, and its width is 2 m. We defined the longitudinal safety bound as 0.3 of the vehicle length, both front and rear, and the lateral safety distance as 0.5 of the vehicle width, right and left. This creates a safety longitudinal distance of 6 m between vehicles (3 m from each safety bound) and a lateral distance of 4 m between vehicles. We note that the simpler circular safety bound is less appropriate for this case because of the length of the race car is more than twice its width.

Two trajectories C_1 and C_2 collide if there exists some t such that the safety bounds of both associated vehicles at time t overlap. See Fig. 10 for an illustration.

A maneuver candidate is considered *free* if it does not collide with a predicted trajectory of any opponent vehicle. However, if an opponent vehicle is directly behind the ego-vehicle, and the ego-vehicle completely blocks it, our controller ignores it because changing the path to allow the opponent vehicle to overtake is clearly an uncompetitive behavior.

G. MANEUVER SELECTION

Our planner considers all free maneuver candidates. Colliding candidates that assume maximal velocity are updated by

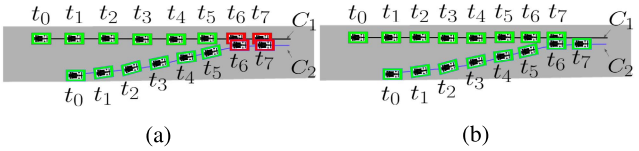


FIGURE 10. Two trajectories describing the location of each vehicle and its safety bounds. In (a), the vehicles collide at time t_6; in (b), the vehicles do not collide since the overlap of their safety bounds is at different times.

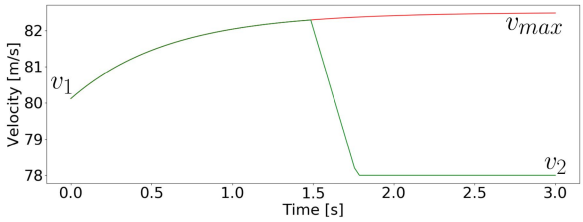


FIGURE 11. Updating a maneuver candidate to become free by decelerating towards a blocking vehicle that drives at a velocity of v_2 (green), instead of accelerating to the maximal velocity, $v_{\max}$ (red), and causing a collision.

reducing the speed along them to avoid collision, as shown in Fig. 11.

If more than one maneuver is free, each maneuver is given a cost that is the sum of three criteria: travel time, denoted $\mathcal{T}(C)$, nearness to the optimal race line, denoted $\mathcal{N}(C)$, and continuity, denoted $\mathcal{K}(C)$, as detailed below. This produces the cost function Cost(C) for each free maneuver:

$$\text{Cost}(C) = \mathcal{T}(C) - \mathcal{N}(C) - \mathcal{K}(C); \quad C \in \boldsymbol{M}. \tag{9}$$

1) TRAVEL TIME

$\mathcal{T}(C)$ represents the estimated travel time for maneuver C. It is computed by integrating the maneuver's velocities along C.

2) NEARNESS TO THE OPTIMAL RACE LINE

The search for the fastest maneuver with a limited time horizon is local by nature. A global search is ineffective because of the unpredictable behavior of the opponent vehicles. A sensible compromise is to attempt to merge with the precomputed race line, which is globally optimal, wherever possible. The function $\mathcal{N}(C)$ thus equals R_{opt} if C is the closest to the optimal race line among the free candidates and 0 otherwise.

3) CONTINUITY

To avoid frequent oscillations between maneuver candidates of similar optimality, it is preferred, when possible, to maintain the same maneuver unless another maneuver is conspicuously better. This is done by rewarding the maneuver that is similar to the current maneuver. For maneuver C, $\mathcal{K}(C)$ equals R_k if C is the same maneuver as the maneuver that the vehicle currently drives on and 0 otherwise. Clearly, switching to a new maneuver is less desired if a switch has just occurred, but once some time has elapsed since the last switch, the controller should be more lenient towards another switch. This is accomplished by decaying R_k linearly at the rate R_d, as long as the same maneuver has been followed.

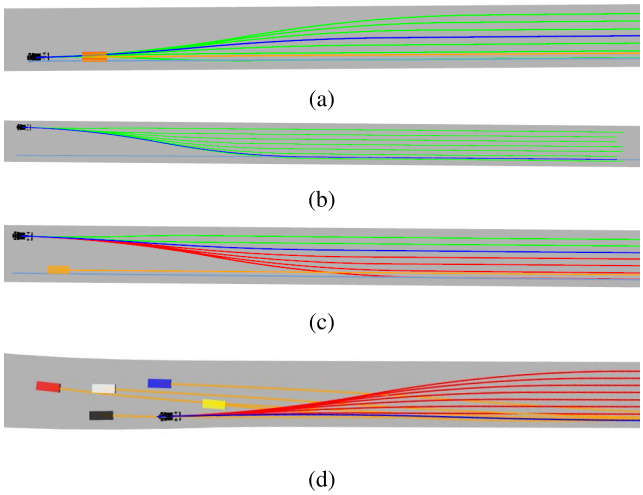


FIGURE 12. Maneuver selection examples, green: free maneuvers, red: blocked maneuvers, blue: chosen maneuver. (a) The planner selects a minimum-time lateral-shift maneuver to avoid slowing down because of the blocking vehicle. (b) The planner prefers the longer maneuver because it returns to the optimal race line (shown in light blue). (c) The planner selects a free maneuver that is the closest to the optimal race line. (d) None of the maneuver candidates are free; therefore, the planner selects the safest maneuver.

H. BEHAVIOR WHEN NO MANEUVER IS FREE

In dynamic environments, such as a multi-vehicle race, there might occur situations when no free maneuver exists. In such sitatuations, the optimality criteria are irrelevant, and then the only selection criterion is safety. The planner then determines the imminent collision and selects the maneuver that is as far as possible from that collision (see an example in Fig. 12d).

VI. CONTROL

The control module outputs throttle, brake, and steering commands that drive the vehicle as close as possible to the selected maneuver.

A. LATERAL CONTROL

The pure pursuit algorithm [10] is used to compute the desired angular velocity based on the current ego-vehicle's state and the selected maneuver, which is mapped to the Cartesian coordinates. The pure pursuit algorithm pursues a target on the selected maneuver C. Let $\boldsymbol{v}$ be the vector representing the ego-vehicle's velocity. The distance to the target is defined to be proportional to the vehicle's speed $v = |\boldsymbol{v}|$, that is, $l_d = k_t v$, where k_t is a predefined constant. The angle between the velocity vector $\boldsymbol{v}$ to the vehicle-target vector is denoted as α (see Fig. 13). The desired angular velocity of the vehicle, ω_d is $2v \sin\alpha / l_d$. The desired angular velocity ω_d is used as a reference for a proportional angular velocity controller that computes the steering command: $\delta = k_\omega(\omega_d - \omega)$ where

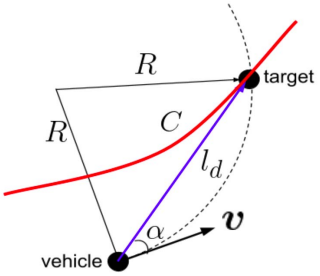


FIGURE 13. Pure pursuit geometry. The red line represents the reference trajectory *C*; the algorithm finds a curve with a constant curvature $1/R$ that will lead the vehicle to a target on *C* that is at a distance l_d from the vehicle.

ω is the current angular velocity of the vehicle and k_ω is a proportional gain.

B. LONGITUDINAL CONTROL

The desired speed v_d is provided by the selected maneuver, which is typically used as a reference for the longitudinal controller. However, when closely following a vehicle, we modify the desired speed to: $v_d = v_f - k_f(L_d - L)$, where v_f is the speed of the leading vehicle, L is the distance to the leading vehicle, L_d is the desired distance to maintain, and k_f is a proportional gain. This modification allows for smooth driving and maintaining a constant distance from the leading vehicle. Figure 14 illustrates the vehicle-following scenario. Finally, the throttle and brake command u is computed by a proportional speed controller: $u = k_v(v_d - v)$ where k_v is a proportional gain.

We note that the vehicle was operated most of the time at high speeds where the vehicle has low acceleration capabilities because of the high aerodynamic drag at these speeds. Therefore, we could assume that the longitudinal dynamics are linear and tune the longitudinal controller for the best performance at those speeds range.

VII. EXPERIMENTS

A. SIMULATION ENVIRONMENT

The simulation platform used by all teams was the Ansys VRXPERIENCE simulator [35], which simulates the AV-21 dynamics and the IMS track (see Fig. 15). The VRXPERIENCE simulator enables multi-vehicle head-to-head racing, in which every vehicle is controlled by a separate controller. The sensors are simulated at 25 Hz and the ego-vehicle's state at 100 Hz, in simulator time. Each controller receives the ego-vehicle's state and the sensor data from the simulator and sends back the throttle, brake, and steering commands.

We use the pre-processed cameras and radars data from the simulator to obtain the position and linear and angular velocity of all vehicles in the sensors range. Our controller is based on Autoware.auto [36], which is an open-source autonomous driving framework that uses ROS2 as middleware. The computation time of each cycle at our algorithm is 20 milliseconds, on average, on an Intel Core i7 2.90 GHz CPU, which enables our algorithm to run in real time. A video demonstrating our controller is available in [37].

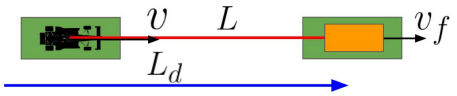


FIGURE 14. The ego-vehicle follows another vehicle. The distance between the ego-vehicle and the vehicle ahead, *L*, is less than the predefined following distance, L_d. Therefore, the ego-vehicle will slow down to increase *L*.

FIGURE 15. The VRXPERIENCE simulator used for the simulation race.

B. SIMULATION RESULTS

We first tested the solo lap performance of our controller, that is, driving along the track without other vehicles. The lap time was 50.0 seconds, and the vehicle had an average speed of 80.83 m/s. The vehicle ran at full throttle along the entire lap. The speed and the lateral accelerations during a single lap are shown in Fig. 16. As depicted by the figure, the vehicle's speed dropped slightly at the corners from a top speed of 82.72 m/s to 78.79 m/s. That is owing to the increased tire slip, which detracts from the longitudinal tire force during the turns. Furthermore, the lateral acceleration during the turns was over 2.5 g, which is way above what is expected of a regular passenger car, reflecting the high down-force generated by the race car due to its high speed and aerodynamic properties.

We then tested our controller in a multi-vehicle scenario, running the same controller on 6 vehicles over 30 laps. Every controller instance operated independently of the other controllers and received information only from its own vehicle sensors. During the entire test, all vehicles drove safely while respecting the race rules; a collision or loss of control never occurred. Figure 17 shows two typical snapshots from the planner of the black vehicle. Notably, all vehicles are very close to the leading vehicle (approximately 40 m) while moving at very high speeds (over 80 m/s), resulting in approximately 0.5 s separating the leading vehicle from the last. A recording of this test, which took 28 minutes, is available at [38] and the raw data at [39].

The lap times in the multi-vehicle test ranged from 49.84 to 51.24 seconds, a difference of less than 3% between the extremes. The average lap time was 50.25 seconds, which is very close to the solo lap time of 50.0 seconds. The lap-time distribution is shown in Fig. 18. Interestingly, some of the individual laps were faster than the solo-lap. This is owing to the higher speeds that individual vehicles can achieve while taking advantage of the slipstream of the leading vehicles.

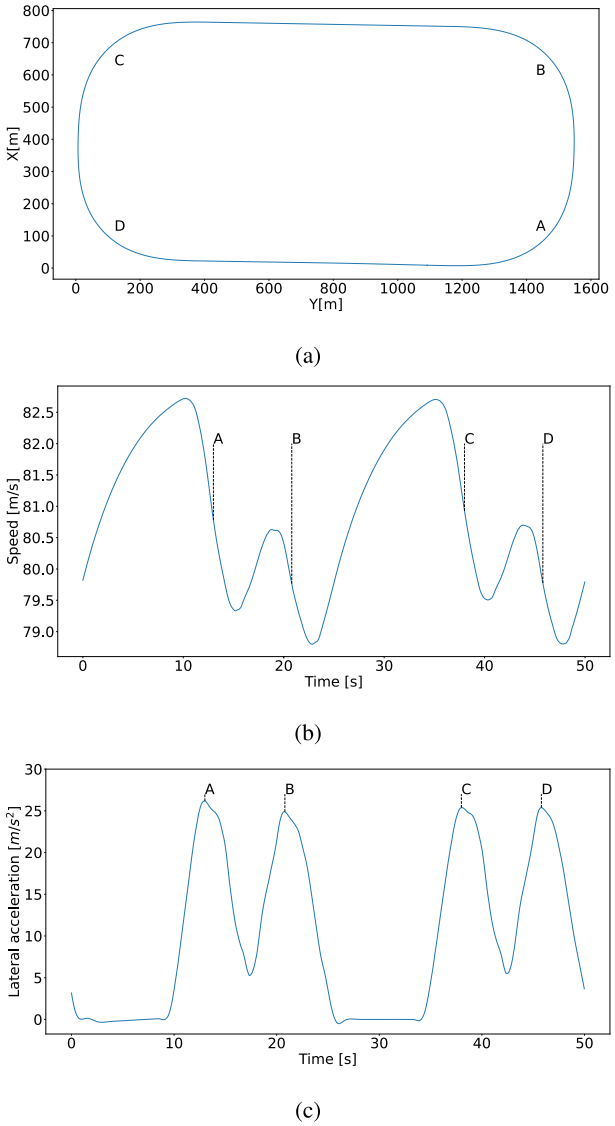


FIGURE 16. (a) Path, (b) velocity profile, and (c) lateral acceleration during the solo lap. The vehicle slows down from a speed of 82.72 m/s to 78.79 m/s in the corners, labeled A-D. The lateral acceleration in the corners exceeds 2.5 g due to the high down-force.

Figure 19 shows the distribution of the longitudinal distance between the first and last vehicles over the 30 laps. The average distance was 64.7 m, which indicates a very tight race.

Another indication of the race tightness and the competitive behavior of all vehicles is shown in Fig. 20. It shows the longitudinal distance from all vehicles to the first vehicle during the last 10 laps. Remarkably, the vehicle starting far behind all other vehicles (represented by the red line) crossed the finish line ahead of the other vehicles. The last vehicle crossed the finish line only 0.32 seconds after the first vehicle. Note that, as mentioned in I-A3, the vehicles can overtake each other, although having identical dynamics, by exploiting the slipstream generated by the leading vehicle.

Figure 21 shows a sequence of snapshots of one overtaking maneuver with three competing vehicles. As shown,

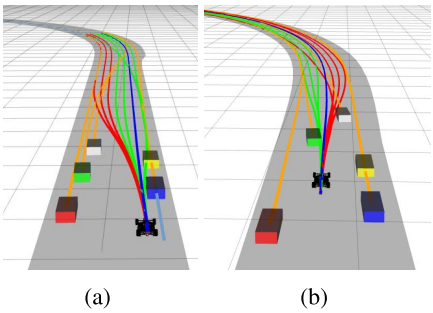


FIGURE 17. Two snapshots of trajectory planning in a scenario with 5 opponent vehicles. Predicted opponent trajectories (orange); free maneuvers (green); colliding maneuvers (red); selected maneuver (blue). (a) The ego vehicle drives on a straight. The colliding maneuvers are close to the inner boundary of the track. The selected maneuver continues on a straight line. (b) The vehicles are approaching a corner. The colliding maneuvers are close to the outer side boundary of the track. The selected maneuver overtakes the green vehicle from the right.

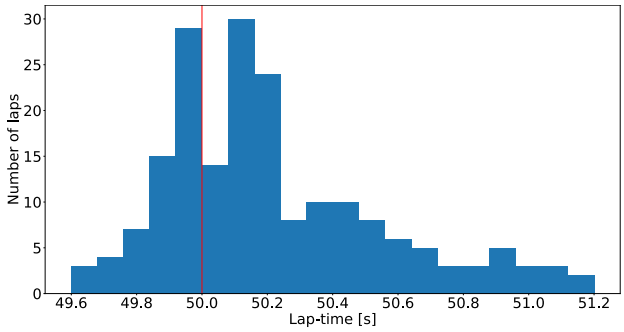


FIGURE 18. Lap times distribution of all 6 vehicles over 30 laps. The red line represents the solo-lap time.

the black vehicle, starting second in Fig. 21b, gains a slight advantage over the blue vehicle at every corner until it successfully overtakes the blue vehicle, as shown in Fig. 21g. This overtaking maneuver took approximately 40 seconds and 3 km to complete. We note that explicitly planning such a maneuver requires a planning horizon of a few kilometers, which is computationally expensive. Nevertheless, our planner executes such maneuvers by repetitive local planning over a horizon of 3 seconds. Although some of our controller's attempts to overtake other vehicles may fail, it does not compromise the vehicle's safety.

C. THE IAC SIMULATION RACE

Sixteen teams reached the simulation race. All teams first competed on one solo lap, with an initial speed of 100 km/h (rolling start), to determine the vehicles' order for the multi-vehicle race. Our solo lap time of 51.968 seconds placed us in the 7th place; the first place finished at 51.848, only 0.12 seconds (0.23%) ahead of us. The average time of all teams was 53.041 seconds.

Only 10 teams passed the multi-vehicle safety tests and were qualified to proceed to the semi-final, which consisted of a multi-car, 10-lap competition. The 10 teams were split into two heats, 5 vehicles on each. Based on our solo lap time, we were placed in the 3rd place in our heat. We finished the semi-final in the 3rd place, at 505.428 seconds, only

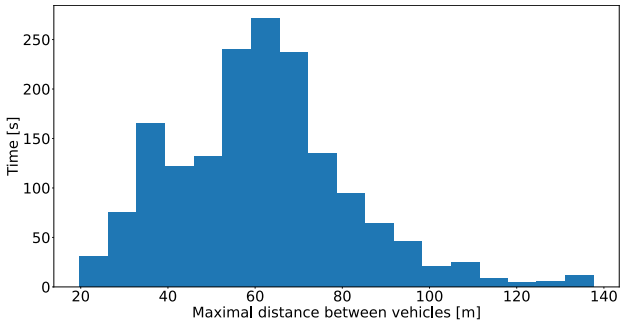


FIGURE 19. Distribution of the longitudinal distance between first and last vehicle.

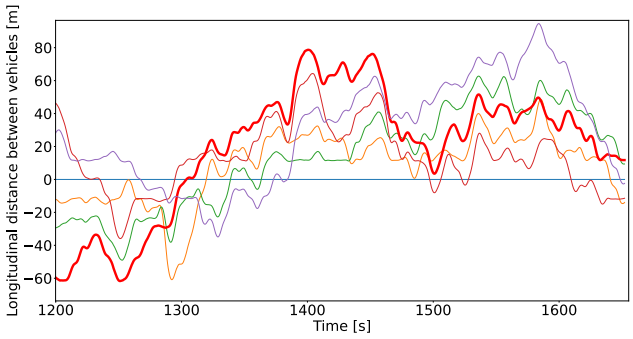


FIGURE 20. The longitudinal distance from each vehicle (each represented by a line with a different color) relative to the first vehicle over time. It shows a time frame of the last 10 laps of the multi-vehicle racing test. Intersections between the lines represent overtakes between vehicles, which indicates that the vehicles repeatedly overtake each other. The marked red line shows that this vehicle overtook all vehicles and finished first.

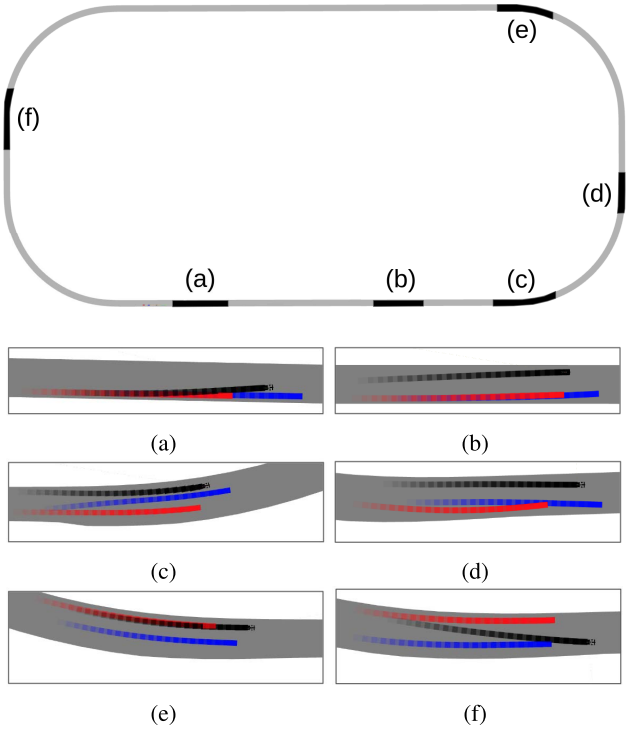


FIGURE 21. A sequence of snapshots of an overtaking maneuver—the black vehicle overtakes the blue vehicle. Each snapshot lasts 0.8 seconds. The track with marked snapshot segments is depicted on the top. (a) The black vehicle initiates an overtake. (b) The black vehicle drives parallel to the blue and red vehicles. (c) The black vehicle prevents the blue vehicle from continuing on the optimal race line, i.e., reaching the apex. (d) The black vehicle gains a slight advantage at the corner. (e) At the next corner, the blue vehicle is overtaken because it is forced to stay on the outside of the corner. (f) The black vehicle returns to the optimal race line.

0.44 seconds (0.0871%) after the winner. The average time for this heat was 506.711 seconds. In the second heat, three of the five teams lost control or crashed. That left seven teams for the final race.

We started the final race in 5th place, and our vehicle overtook two vehicles over the first lap. Throughout the race, our controller demonstrated collision-free and competitive driving capabilities and was able to keep the 3rd place a significant part of the time. With three laps to go, another vehicle entered our safety zone from the right, which triggered a collision avoidance action. Being on the far left track, with no room to maneuver, the collision was avoided by stepping on the brakes. Driving at that moment along the turn caused our vehicle to spin off the track, which placed us at the 6th place in the finals. Four vehicles completed the race without being responsible for a collision. The recording of the simulation race is available in [40].

D. REMARKS

One of the interesting results accomplished by the racing controller presented here is its ability to balance competitiveness and safety. This was demonstrated by driving 6 vehicles for 30 laps each, which accounts for 180 laps over 720 km, without any collision. This resonates with our collision-free run in the IAC simulation race over a total of 17 laps.

This notable result did not diminish the racing controller's competitiveness, as demonstrated by the 6-vehicle race, where the average lap time of all vehicles was only 0.25 seconds greater than the optimal solo-lap time. This implies that the vehicles were running in close proximity to each other, thus demonstrating the great challenge of high-speed racing.

It is interesting to note that despite all vehicles being driven by the same controller, they continuously overtook each other by exploiting the slipstream from the leading vehicle, as was depicted in Fig. 20.

VIII. CONCLUSION

This paper describes a competitive racing controller for an autonomous racing car developed in the context of the Indy Autonomous Challenge simulation race. Its development was guided by an attempt to strike a balance between competitiveness and safety. To this end, the controller attempts to avoid collisions, even those that the race rules placed the responsibility on the opponent vehicle to avoid.

The online planner generates a set of dynamically feasible maneuver candidates using a point mass model. Of those candidates, a maneuver is selected that is collision-free, minimizes travel time along the track, and maximizes proximity to the race line. It is then tracked by a pure-pursuit controller. The speed is controlled by a speed controller that follows the velocity profile along the trajectory and regulates the speed to avoid collision with neighboring vehicles.

Our controller demonstrated competitive and safe driving in a test run with 6 vehicles, all driven by the same controller, and in the IAC simulation race. Our vehicle finished 3rd in the semi-finals with only 0.44 seconds behind the winner, and maintained 3rd place for a significant part of the final race. It demonstrated responsible driving, yet competitive, and was not involved in any collision during the entire race. It is important to note that very few vehicles were not involved in any collision with any other vehicle. Clearly, while the challenge of safe and competitive driving is still unresolved, the Indy Autonomous Challenge competition brought us closer to driving autonomously under extreme conditions.

ACKNOWLEDGMENT

The authors wish to thank the Indy Autonomous Challenge Organizers and the Ansys Team for providing a unique opportunity to test their algorithms in a high fidelity simulation and compare the performance of various methods on a common ground. The author Gabriel Hartmann acknowledges the generous support of Ariel University throughout his graduate studies.

REFERENCES

[1] J. Betz, H. Zheng, A. Liniger, U. Rosolia, P. Karle, M. Behl, V. Krovi, and R. Mangharam, "Autonomous vehicles on the edge: A survey on autonomous vehicle racing," *IEEE Open J. Intell. Transp. Syst.*, vol. 3, pp. 458–488, 2022.

[2] (2021). *Formula Student*. [Online]. Available: https://www.imeche.org/events/formula-student/about-formula-student/the-challenge

[3] (2021). *Roborace*. [Online]. Available: https://roborace.com/

[4] (2021). *Indy Autonomous Challenge*. [Online]. Available: https://www.indyautonomouschallenge.com/

[5] (2021). *The INDY Autonomous Challenge Racecar*. [Online]. Available: https://www.indyautonomouschallenge.com/racecar

[6] (2020). *Ariel Team Passenger Car Autonomous Driving*. [Online]. Available: https://youtu.be/J_26TnDg_sk

[7] (2022). *Polimove Wins the Autonomous Challenge at CES, Making History as the First Head-to-Head Autonomous Racecar Competition Champion*. [Online]. Available: https://www.indyautonomouschallenge.com/polimove-wins-the-autonomous-challenge-at-ces

[8] (2021). *Indy Autonomous Challenge Rules*. [Online]. Available: https://www.indyautonomouschallenge.com/rules

[9] INDYCAR. (2022). *NTT Indycar Series Rulebook*. [Online]. Available: https://paddock.indycar.com/LinkClick.aspx?fileticket=k8xdi-l5zaU%3d&portalid=0

[10] O. Amidi and C. E. Thorpe, "Integrated mobile robot control," *Proc. SPIE*, vol. 1388, pp. 504–523, Mar. 1991.

[11] J. L. Vazquez, M. Bruhlmeier, A. Liniger, A. Rupenyan, and J. Lygeros, "Optimization-based hierarchical motion planning for autonomous racing," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst. (IROS)*, Oct. 2020, pp. 2397–2403.

[12] J. Kabzan, M. I. Valls, V. J. F. Reijgwart, H. F. C. Hendrikx, C. Ehmke, M. Prajapat, and A. Bühler, "AMZ driverless: The full autonomous racing system," *J. Field Robot.*, vol. 37, no. 7, pp. 1267–1294, 2020.

[13] J. Hwan Jeon, R. V. Cowlagi, S. C. Peters, S. Karaman, E. Frazzoli, P. Tsiotras, and K. Iagnemma, "Optimal motion planning with the half-car dynamical model for autonomous high-speed driving," in *Proc. Amer. Control Conf.*, Jun. 2013, pp. 188–193.

[14] G. Williams, B. Goldfain, P. Drews, J. M. Rehg, and E. A. Theodorou, "Autonomous racing with AutoRally vehicles and differential games," 2017, *arXiv:1707.04540*.

[15] N. Li, E. Goubault, L. Pautet, and S. Putot, "Autonomous racecar control in head-to-head competition using mixed-integer quadratic programming," in *Proc. Int. Conf. Robot. Automat. (ICRA), Workshop Opportunities Challenges With Auton. Racing*. IEEE, 2021, pp. 1–6.

[16] M. Wang, Z. Wang, J. Talbot, J. C. Gerdes, and M. Schwager, "Game-theoretic planning for self-driving cars in multivehicle competitive scenarios," *IEEE Trans. Robot.*, vol. 37, no. 4, pp. 1313–1325, May 2021.

[17] P. R. Wurman, "Outracing champion Gran Turismo drivers with deep reinforcement learning," *Nature*, vol. 602, no. 7896, pp. 223–228, 2022.

[18] V. Sukhil and M. Behl, "Adaptive lookahead pure-pursuit for autonomous racing," 2021, *arXiv:2111.08873*.

[19] N. Dal Bianco, E. Bertolazzi, F. Biral, and M. Massaro, "Comparison of direct and indirect methods for minimum lap time optimal control problems," *Vehicle Syst. Dyn.*, vol. 57, no. 5, pp. 665–696, May 2019, doi: 10.1080/00423114.2018.1480048.

[20] A. Heilmeier, A. Wischnewski, L. Hermansdorfer, J. Betz, M. Lienkamp, and B. Lohmann, "Minimum curvature trajectory planning and control for an autonomous race car," *Vehicle Syst. Dyn.*, vol. 58, no. 10, pp. 1497–1527, Oct. 2020, doi: 10.1080/00423114.2019.1631455.

[21] (2020). *Tumftm Global Race Trajectory Optimization*. [Online]. Available: https://github.com/TUMFTM/global_racetrajectory_optimization/

[22] S. Ammoun and F. Nashashibi, "Real time trajectory prediction for collision risk estimation between vehicles," in *Proc. IEEE 5th Int. Conf. Intell. Comput. Commun. Process.*, Aug. 2009, pp. 417–422.

[23] N. Deo and M. M. Trivedi, "Multi-modal trajectory prediction of surrounding vehicles with maneuver based LSTMs," in *Proc. IEEE Intell. Vehicles Symp. (IV)*, Jun. 2018, pp. 1179–1184.

[24] F. Török, P. Karle, and M. Geisslinger, "Structured deep neural motion prediction of opposing vehicles for an autonomous racecar," in *Proc. Int. Conf. Robot. Automat. (ICRA), Workshop Opportunities Challenges With Auton. Racing*. IEEE, 2022, pp. 1–18.

[25] R. Verschueren, M. Zanon, R. Quirynen, and M. Diehl, "Time-optimal race car driving using an online exact Hessian based nonlinear MPC algorithm," in *Proc. Eur. Control Conf. (ECC)*, Jun. 2016, pp. 141–147.

[26] A. Liniger, A. Domahidi, and M. Morari, "Optimization-based autonomous racing of 1:43 scale RC cars," *Optim. Control Appl. Methods*, vol. 36, no. 5, pp. 628–647, 2015.

[27] A. Wischnewski, T. Herrmann, F. Werner, and B. Lohmann, "A tube-MPC approach to autonomous multi-vehicle racing on high-speed ovals," *IEEE Trans. Intell. Vehicles*, early access, Apr. 25, 2022, doi: 10.1109/TIV.2022.3169986.

[28] S. M. LaValle, "Rapidly-exploring random trees: A new tool for path planning," Comput. Sci. Dept., Iowa State Univ., Ames, IA, USA, Tech. Rep. 98-11, 1998.

[29] O. Arslan, K. Berntorp, and P. Tsiotras, "Sampling-based algorithms for optimal motion planning using closed-loop prediction," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, May 2017, pp. 4991–4996.

[30] M. O'Kelly, H. Zheng, A. Jain, J. Auckley, K. Luong, and R. Mangharam, "TUNERCAR: A superoptimization toolchain for autonomous racing," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, May 2020, pp. 5356–5362.

[31] M. Fu, J. Ni, X. Li, and J. Hu, "Path tracking for autonomous race car based on G-G diagram," *Int. J. Automot. Technol.*, vol. 19, no. 4, pp. 659–668, Aug. 2018.

[32] G. Hartmann, Z. Shiller, and A. Azaria, "Deep reinforcement learning for time optimal velocity control using prior knowledge," in *Proc. IEEE 31st Int. Conf. Tools Artif. Intell. (ICTAI)*, Nov. 2019, pp. 186–193.

[33] U. Rosolia, A. Carvalho, and F. Borrelli, "Autonomous racing using learning model predictive control," in *Proc. Amer. Control Conf. (ACC)*, May 2017, pp. 5115–5120.

[34] Z. Shiller and S. Sundar, "Emergency lane-change maneuvers of autonomous vehicles," *J. Dyn. Syst., Meas., Control*, vol. 120, no. 1, pp. 37–44, 1998.

[35] Ansys. (2021). *Ansys Vrxperience Driving Simulator*. [Online]. Available: https://www.ansys.com/products/av-simulation/ansys-vrxperience-driving-simulator

[36] T. A. Foundation. (2021). *Autoware.Auto*. [Online]. Available: https://www.autoware.org/autoware-auto

[37] (2021). *Autonomous Head-to-Head Racing in the INDY Autonomous Challenge Simulation Race*. [Online]. Available: https://youtu.be/f2nLufCZlbs

[38] A. Team. (2021). *Autonomous Simulation Race Between 6 Vehicles*. [Online]. Available: https://youtu.be/cWoA1f-YwkE

[39] G. Hartmann. (2022). *Multi-Vehicle Race Log*. [Online]. Available: https://dx.doi.org/10.21227/00cg-bt27

[40] Ansys. (2021). *The ANSYS INDY Autonomous Challenge Simulation Race*. [Online]. Available: https://youtu.be/gTjQ3sWdYh0

Deep Reinforcement Learning for Time Optimal Velocity Control using Prior Knowledge

Gabriel Hartmann
Computer Science Department, Mechanical Engineering and Mechatronics Department
Ariel University
Ariel, Israel
gavrielhartmann@gmail.com

Zvi Shiller
Mechanical Engineering and Mechatronics Department
Ariel University
Ariel, Israel
shiller@ariel.ac.il

Amos Azaria
Computer Science Department
Ariel University
Ariel, Israel
amos.azaria@ariel.ac.il

***Abstract*—Autonomous navigation has recently gained great interest in the field of reinforcement learning. However, little attention was given to the time optimal velocity control problem, i.e. controlling a vehicle such that it travels at the maximal speed without becoming dynamically unstable (rollover or sliding).**

Time optimal velocity control can be solved numerically using existing methods that are based on optimal control and vehicle dynamics. In this paper, we use deep reinforcement learning to generate the time optimal velocity control. Furthermore, we use the numerical solution to further improve the performance of the reinforcement learner. It is shown that the reinforcement learner outperforms the numerically derived solution, and that the hybrid approach (combining learning with the numerical solution) speeds up the training process.

I. Introduction

The operation of autonomous vehicles requires the synergetic application of a few critical technologies, such as sensing, motion planning, and control. This paper focuses on a subset of the motion planning problem, that is moving at the time optimal speeds to minimize travel time along a given path, while ensuring the vehicle's dynamic stability. By "dynamic stability" we refer to constraints that are functions of the vehicle speed, such as rollover or sliding [1, 2, 3, 4]. Respecting the dynamic constraints would thus ensure that the vehicle does not rollover or slide at any point along the path. Additional constraints that may affect the vehicle speeds, although they are not considered in this paper, include passenger comfort [5], traffic laws, and sensing limitations [6]. Although these constrains must be considered in most real driving scenarios, the vehicle's dynamic stability is the most challenging because it concerns the vehicle's (and passengers) safety.

As the time optimal velocity profile is affected by the vehicle's dynamic capabilities, such as its maximum and minimum acceleration, ground/wheels interaction, terrain topography, and path geometry, a complex dynamic model is required to ensure that the vehicle is dynamically stable during motion at any point along the path [1].

Since the consideration of a detailed vehicle dynamic model may be impractical for online computation, we use a simplified model to compute the vehicle's velocity profile as discussed later. In this context, one of the goals of Reinforcement Learning (RL) is to bridge the gap between the approximate and the actual vehicle model.

A large body of work on reinforcement learning has focused on autonomous driving with an emphasis on perception and steering [7, 8, 9, 10, 11]. Some works have focused on human like velocity control [12, 13] or fuel efficiency [14]. Other works use RL to track a given reference velocity [15]. In [16], a model-predictive control is used to drive a race car at high speeds along a specific track. The controller is tuned iteratively to reduce total motion time. This method is applicable to repetitive tasks, where the initial state is fixed for all iterations. Clearly, this approach is not suitable for controlling a vehicle on general paths. We are not aware of works that use reinforcement learning of time optimal speeds along general paths, while ensuring the vehicle's dynamic stability.

This paper proposes a reinforcement learning method for driving a vehicle at the time optimal speed along a known arbitrary path. It learns the acceleration (and deceleration) that maximizes vehicle speeds along the path, without losing its dynamic stability. Here, steering is not learned, but is rather determined directly by the path following controller (pure pursuit) [17].

One major challenge of RL is that, in many cases, the initial policy executed by the agent is random, and long training is required to achieve a good policy. Several methods for combining prior information about the problem into the RL process were proposed. For example, imitation learning uses expert demonstrations (either automated or human) to train an agent in order to achieve the initial policy [7, 13, 12]. The policy can then be further improved using RL [18, 19]. In this paper we propose a different method for using prior knowledge in order to allow the RL agent to begin the training with a relatively good policy. Instead of learning the

2375-0197/19/$31.00 ©2019 IEEE
DOI 10.1109/ICTAI.2019.00034

actions directly using RL, only the variation from a nominal time optimal controller is learned by the RL agent. For this purpose, we use a numerical, model-based controller [20] that controls a vehicle along a path while avoiding rollover, slipping an loosing contact with ground. This model-based method, computes a solution in a efficient way, hence it is suitable for real-time use.

The RL method, the model-based method and hybrid approach that combines both, was implemented in a simulation for a ground vehicle moving along arbitrary paths in the plane. It is shown that, the synergy between our learning based method and the model-based method, speeds-up the learning process (especially at early stages). The RL agent that uses the model-based controller, achieves at the beginning of the learning process the same velocity as the model-based controller alone, while the pure RL approach achieves low performance at the same time. Eventually both methods converge to an average velocity that is higher by about 10% than the velocity achieved by the model-based controller, while maintaining very low failure rates.

Our main contributions of this paper are (i) Applying a deep reinforcement learning-based method for driving a vehicle at time optimal speeds, subject to the vehicle's dynamic constraints, that outperforms the model-based controller; (ii) Using the model-based prior knowledge to speed up the learning process (especially at early stages).

II. Problem Statement

We wish to drive a ground vehicle along a predefined path in the plane. The steering angle is controlled by a path following controller whereas its speed is determined by the learned policy. The goal of the reinforcement learning agent is to drive the vehicle at the highest speeds without causing it to rollover or deviate from the defined path beyond a predefined limit.

The path is defined by P, $P = \{p_1, p_2, \cdots, p_N\}$, $p_i \in \mathbb{R}^2$, $i \in \{1, 2, \cdots, N\}$. The position of the vehicle's center of mass is denoted by $q \in \mathbb{R}^2$, yaw angle θ, and roll angle α. The vehicle's speed is $v \in \mathbb{R}$, $0 \leq v \leq v_{\max}$. The throttle (and brakes) command that affects the vehicle's acceleration (and deceleration) is $\tau \in [-1, 1]$. The steering control of the vehicle is performed by a path following controller (pure pursuit [17]). The deviation of the vehicle center from the desired path is denote by d_{err}, as shown in Fig. 1.

The agent's goal is to drive the vehicle at the maximal speed along the path, without losing its dynamic stability (sliding and rollover), while staying within a set deviation from the desired path, i.e. $d_{err} \leq d_{\max}$, and within a "stable" roll angle, i.e. $|\alpha| \leq \alpha_{\max}$, where $\alpha_{\max}$ is the maximal roll angle beyond which the vehicle is statically unstable.

The time optimal policy maximizes the speed along the path during a fixed distance. More formally, for every path P with length D, and a vehicle at some initial velocity v_{init}, initial position q, which is closest to point $p_i \in P$ along the path, we wish to derive the time optimal policy π^* that at every time t outputs the action $\tau = \pi^*(s_t)$ that maximizes the vehicle speed (minimizing traveling time), while ensuring that every state s_t is stable. The time optimal velocity along path P is the velocity profile $v(t)$ produced by the optimal policy π^*.

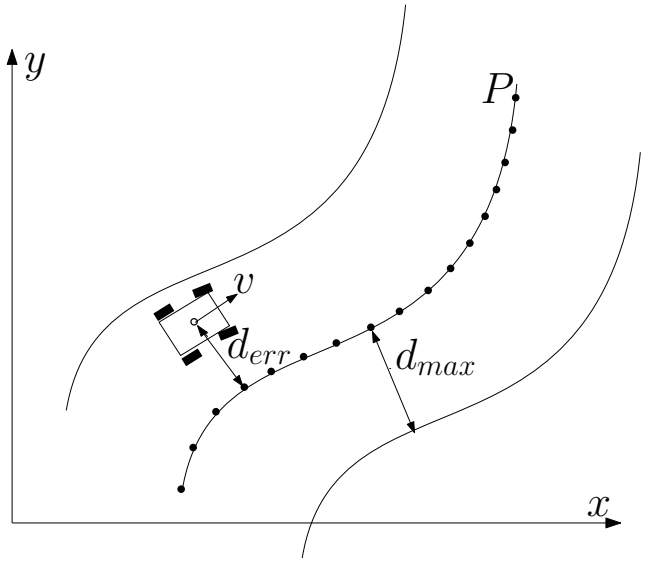


Figure 1: A vehicle, tracking path P within the allowed margin d_{max}.

III. Time Optimal Velocity Control using Reinforcement Learning

Our basic reinforcement learner is a direct adaptation of the "Deep Deterministic Policy Gradient" (DDPG) [21] to the time optimal velocity control problem. We refer to this method as the Reinforcement learning based Velocity Optimizer REVO.

A. Deep Deterministic Policy Gradient

DDPG [21] is an actor-critic, model-free algorithm for a continuous action space, A, and a continuous state space, S. The agent is assumed to receive a reward $r_t \in \mathbb{R}$ when being at state $s_t \in \mathbb{R}^{|S|}$ and taking action $a_t \in \mathbb{R}^{|A|}$. The transition function $p(s_{t+1}|s_t, a_t)$ is defined as the probability of ending at s_{t+1} when being at state s_t and taking action a_t. The goal of the DDPG algorithm is to learn a deterministic policy $\pi : S \rightarrow A$ (represented as a neural network) that maximizes the return from the beginning of the episode:

$$R_0 = \sum_{i=1}^{T} \gamma^{(i-1)} r(s_i, \pi(s_i))$$

where $\gamma \in [0, 1]$ is the discount factor. DDPG learns the policy using policy gradient. The exploration of the environment is done by adding exploration noise to the actions.

We use DDPG to train an agent for driving along any given path at the highest possible speed, while preventing a rollover or slipping away from the path. The training process consists of episodes; at each episode the vehicle moves along a randomly generated path. Training an agent on randomly generated paths allow the learned policy to be more general. Only paths that are kinematically feasible are considered, that is, the generated paths do not contain any sharp curves that exceed the vehicle's minimum turning radius (the maximum turning ability of the vehicle at zero velocity). Each path, P, is generated by smoothly connecting short path segments of random length and curvature until reaching the desired length. This ensures that the selected path respects the vehicles steering capabilities.

The state, s, includes a down-sampled limited horizon path segment, $P_s \subseteq P$, which is defined relative to the vehicle's position and the vehicle speed, v. More formally,

$$P_s = \{p_m, p_{m+d}, p_{m+2d} \cdots p_{m+kd}\}$$

where m is the index of the closest point on the path P, to the vehicle, $d \in \mathbb{N}$ is the down-sampling factor and $k \in \mathbb{N}$ is a predefined number of points. In addition to this path segment, also the current velocity of vehicle (v) is included in the state. Therefore, the state of the system is defined as $s = \{v, P_s\}$.

The DDPG agent is not provided with any information related to the path segment following p_s. Therefore, p_s is required to be long enough in order to enable the vehicle to decelerate to a safe velocity at the end of this path segment, even when driving at the maximal speed. If p_s is too short, the agent may need to drive at a lower speed to prepare for any unforeseen curve that might appear as the vehicle moves forward.

The reward function is defined as follows: If the vehicle is stable and has a positive velocity, the reward r is proportional to the vehicle's velocity ($r_t = kv_t, k \in \mathbb{R}_+$). If the vehicle encounters an unstable state, it receives a negative reward. To encourage the agent not to stop the vehicle during motion, a small negative reward is received if $v_t = 0$.

At each time step, the action is determined as $a_t = \tau_t = \pi(s_t) + \eta(t)$ where $\eta(t)$ is the exploration noise. The episode terminates at time T or if the vehicle becomes unstable.

IV. Computing the Time optimal Velocity Profile

The time optimal velocity profile of a vehicle moving along a specified path can be numerically computed using an efficient algorithm described in [20, 22, 1]. It uses optimal control to compute the fastest velocity profile along the given path, taking into account the vehicle's dynamic and kinematic models, terrain characteristics, and a set of dynamic constraints that must be observed during the vehicle motion: no slipping, no rollover and maintaining contact with the ground at all points along the specified path. This algorithm is used here as a model predictive controller, generating the desired speeds at every point along a path segment ahead of the vehicle's current position. This Velocity Optimization using Direct computation is henceforth termed VOD. The output of this controller is used to evaluate the results of the learning based optimization (REVO), and to serve as a baseline for the training process. We now briefly describe the algorithm in some details.

Given a vehicle that is moving along a given path P, the aforementioned algorithm computes the time optimal velocity, under the following assumptions:

- The dynamics of the vehicle are deterministic;
- The vehicle moves exactly on the specified path i.e. $(p_{err})_t = 0$, $t = \{0, ...N\}$;
- The vehicle is modeled as a rigid body (no suspension);
- Vehicle parameters, such as geometric dimensions, mass, the maximum torque at the wheels, the coefficient of friction between the wheels and ground, are known.

These assumptions help simplify the computation of the time optimal velocity profile. This simplification does not seriously affect our approach as the goal of the learning process is to bridge the gap between the model and reality, which may always exist, regardless of the fidelity of the theoretical model.

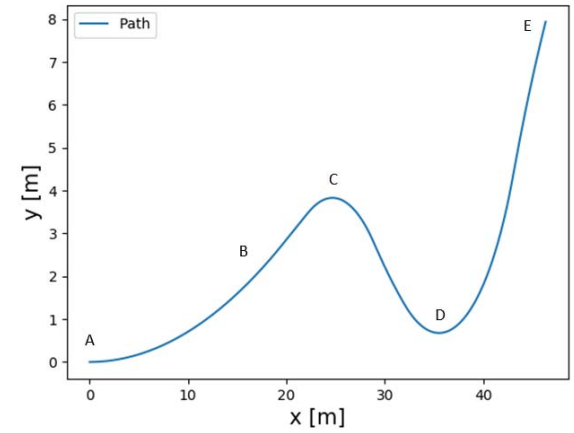
The algorithm first computes the maximal velocity profile along the path, termed the "velocity limit curve", which represents the highest vehicle speeds, above which at least one of the vehicle's dynamic constraints is violated, i.e. the vehicle either rolls-over, slides, or looses contact with the ground. The velocity limit is determined by the coefficient of friction between the wheels and ground as well as by the centripetal forces that might cause the vehicle to slide or rollover.

The time optimal velocity profile is computed by applying "Bang-Bang" acceleration, i.e. either maximum or minimum acceleration, at all points along the path. Bang-bang control is known to produce the time optimal motion of second order systems [23]. The optimal velocity profile is computed by integrating forward and backwards the extreme accelerations at every point along the path so as to avoid crossing the velocity limit curve [22].

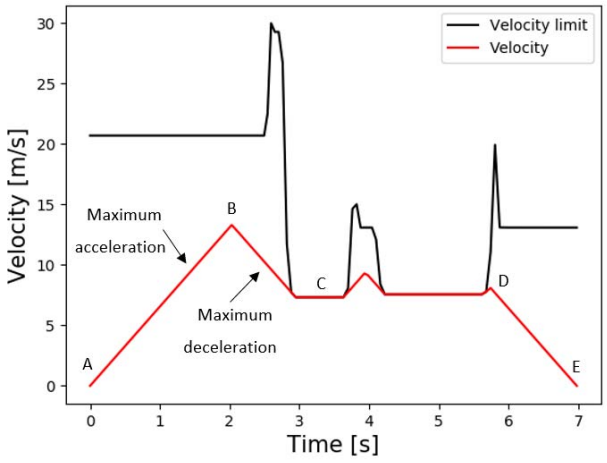
Fig. 2a shows a given planar curved path. The velocity limit curve along that path is shown in black in Fig. 2b. Note the drops in the velocity limit caused by the sharp curves C and D along the path. Clearly, moving at high speeds along these curves might cause the vehicle to either slide or rollover (which of the two occurs first, depends on the location of the vehicle's center of mass). The optimal velocity thus starts at zero (the initial boundary condition), accelerates at a constant acceleration until point B, where it decelerates to avoid crossing the velocity limit towards point c. At point D, the optimal velocity decelerates to a stop at the end point E (the assumed final boundary condition).

The velocity computed by this algorithm is used to control the vehicle along the specified path. At every time t, the op-

timal velocity profile is computed along the limited horizon path segment P_s (as was formally defined in Section III). The vehicle's speed at time t serves as the initial condition for the velocity profile computed from that point. To ensure that the vehicle can decelerate to a stop at the end of this path segment, the target velocity at the endpoint of P_s is set to zero. The action produced by the controller at time t is the initial acceleration of the velocity profile computed at time t. This acceleration is used as a command to the vehicle's engine. This controller is used as a baseline for REVO.



(a)



(b)

Figure 2: (a) A curved path segment (b) The directly computed optimal velocity profile (red) and the velocity limit curve (black). The velocity limit drops along sharp curves along the path. The optimal velocity never crosses the velocity limit curve.

V. Using Direct Computation to Enhance Reinforcement Learning

In this paper, we propose to speed-up the learning process by combining VOD (the direct velocity optimization controller) with REVO (the reinforcement learning based controller). This is done by first adding the actions τ_{VOD} and τ_{REVO} of VOD and REVO, respectively, to produce the action τ_{REVO+A} of the combined policy REVO+A (REVO+Action):

$$\tau_{REVO+A} = \tau_{VOD} + \tau_{REVO}.$$

The REVO+A policy is illustrated in Fig. 3c.

The REVO+A policy first follows the actions of the VOD controller, i.e. $\tau_{REVO+A} \approx \tau_{VOD}$ because $\tau_{REVO} \approx 0$ at the beginning of the learning process. This is significantly better than a randomly initialized policy as in π_{REVO}. It simplifies the problem for the reinforcement learner agent, which only learns the deviation from VOD, as oppose to learning the actions from ground up.

The second approach proposed in this paper to combining REVO and VOD is based on adding the action output τ_{VOD} from the VOD controller as an additional feature to the state space of the agent:

$$s = \{\tau_{VOD}, v, P_s\}.$$

We denote this method REVO+F (REVO+Feature). It is illustrated in Fig. 3d.

An intuitive justification for using REVO+F is that the reinforcement learner has the information about τ_{VOD}, and hence, the agent can use this information to improve its actions.

VI. Experimental Results

The performance of the proposed methods were tested in several experiments as detailed henceforth.

A. Settings

A simulation of a four-wheel vehicle was developed using "Unity" software [24]. A video of the vehicle driving along a path at time optimal velocity is available at [25].

The vehicle properties were set to width= $2.1m$, height= $1.9m$, length= $5.1m$, center of mass at the height of $0.9m$, mass= $3,200Kg$, and a total force produced by all wheels of $21KN$.

The maximal velocity of the vehicle was set to $v_{max} = 30m/s$ $(108km/h)$. Note that the actual speed limit is determined by the path, which may be lower in most cases than the above set limit.

The maximal acceleration of the vehicle is $6.5m/s^2$. The acceleration and deceleration are applied to all four wheels (4x4); steering is done by the front wheels (Ackermann steering). The friction coefficient between the wheels and the ground was set arbitrarily high (at 5) to focus this experiment on rollover only.

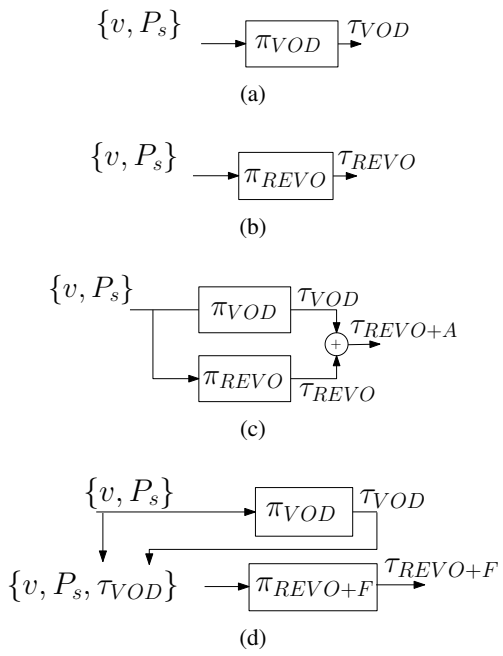


Figure 3: π is a policy, τ is an action, $\{v, P_s\}$ is the state. (a) VOD: Direct planning (b) REVO: DDPG based learning. (c) REVO + A: combines the actions of VOD with REVO. (d) REVO + F: adds the action output of VOD as a feature in the state space of REVO.

Each episode is limited to 100 time steps. The time step is set to 0.2 seconds, i.e. 20 seconds per episode. The policy updates are synchronized with the simulation time steps, two updates per step. P_s consists of 25 points along the path ahead of the vehicle ($|P_s| = 25$. The distance between one point to the next point in P_s is $1m$.

$$|p_i - p_{i+1}| = 1[m] : p_i, p_{i+1} \in P_s, i \in \{0, 1, \cdots, 25\}$$

A state is considered unstable if the roll angle of the vehicle exceeds 4 degrees ($\alpha_{\max} = 4$), and when the vehicle deviates more then $2m$ ($d_{max} = 2$) from the nominal path. The reward function was defined as:

$$\begin{cases} -1 & s \text{ is not stable} \\ 0.2v/v_{max} & s \text{ is stable} \\ -0.2 & v = 0 \end{cases}$$

All the hyper-parameters of the reinforcement learning algorithm (e.g. neural network architecture, learning rates) were set as described in [21].

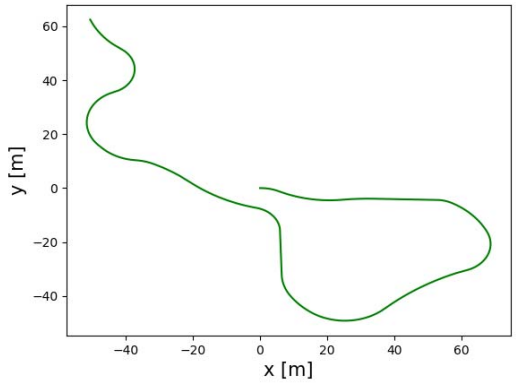
B. Experiment Protocol

During the training process, the vehicle drives along randomly generated paths using the learned policy with exploration noise. Each training process is performed until reaching 90,000 policy updates. Every 5000 updates the neural networks parameters are saved for evaluation. To evaluate the policy during the training process, the vehicle runs along 100 random paths on every saved parameter set. During the evaluation, the exploration noise is disabled. This training and evaluation process was repeated 5 times for each of the methods.

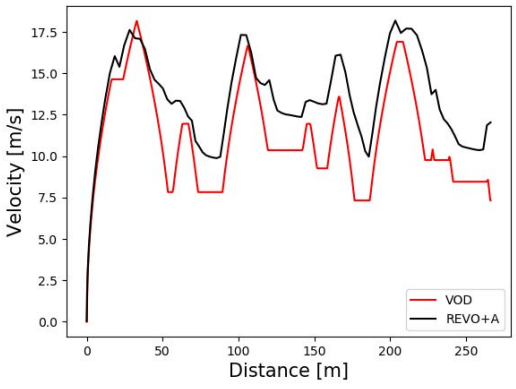
The agent's goal is to maximize its average velocity. Since the average velocity during the failed episodes was usually higher than the average velocity during successful episodes, we excluded failed episodes when presenting the average velocity of each method.

C. Results

Fig. 4a shows an example path, and Fig. 4b shows the velocity profile along this path during 20 seconds, for both the VOD controller (red) and REVO+A after convergence (black). As can be seen, the learned velocity profile of REVO+A is higher than that of the VOD.



(a)



(b)

Figure 4: (a) Example of a random path (b) The dynamics based velocity profile (VOD) and the velocity profile of a trained REVO+A agent.

Fig. 5 presents the normalized average velocity along the path during each episode. All results were normalized with

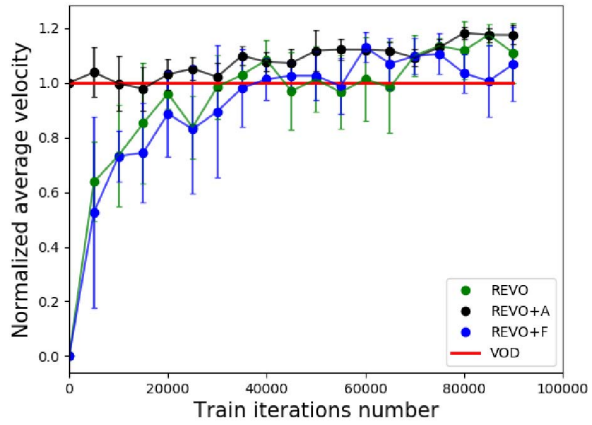


Figure 5: Average velocity on 100 random paths, measured at every 5000 training steps (normalized with respect to VOD), on 5 different training processes. The bars represent the variance between the training processes.

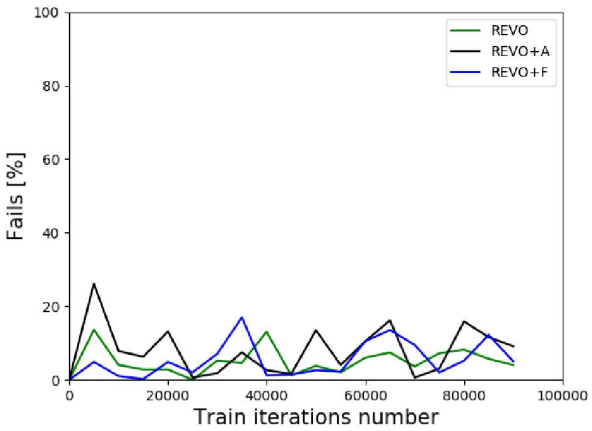


Figure 6: The failure rate of all methods during training.

respect to the VOD, hence it appears as a horizontal line at 1.0.

At the beginning of the training process, REVO did not achieve any progress; after about $40,000$ training updates, REVO achieved the same performance as that of VOD. REVO+A achieved the same performance as VOD from the very beginning. This implies that REVO+A converges much faster than REVO, because REVO+A uses VOD as a baseline.

When the training process continues, the policies learned by all methods improve the performance of the vehicle's velocity compared to using VOD by about 10%. This is expected because VOD uses a relatively simple vehicle model.

REVO+F doesn't improve the converge time compared to REVO, in this experiment. On the other hand, REVO+F performed better than REVO when used in a different setting, as was shown in Section VI-E.

The failure rate of the different methods has a relatively high variance as is depict in Fig. 6. After training and evaluation, it is possible to choose the best policy that achieves high velocity and low failure rates. When re-evaluating the best policies achieved by all methods on 1000 new episodes, the failure rate is lower than 1% and the average velocity is approved to be statistically significant higher that VOD by about 10% (using student's t-test, $p < 0.0001$).

D. Near Optimality of VOD

VOD uses a computational effective model to compute the velocity. In this section we show that the VOD velocity cannot be easily increased without resulting in high failure rates. We show that even slightly scaling up the velocity of the VOD policy, causes the vehicle to fail. This implies that the velocity computed by VOD is close to the real performance envelope.

Fig. 7 shows, that scaling up the VOD velocity, cause an increased failure rate (evaluated on 100 episodes at 6 different velocity factors between 1.00 and 1.25). As depicted by the figure, when the velocity is scaled up by 5%, the vehicle fails on 3% of the episodes, and scaling up by 20% results in a failure rate of nearly 50%.

When controlling the vehicle using the trained policies of REVO, REVO+A and REVO+F, a higher velocity (by about 10% can be achieved without increasing the failure rate.

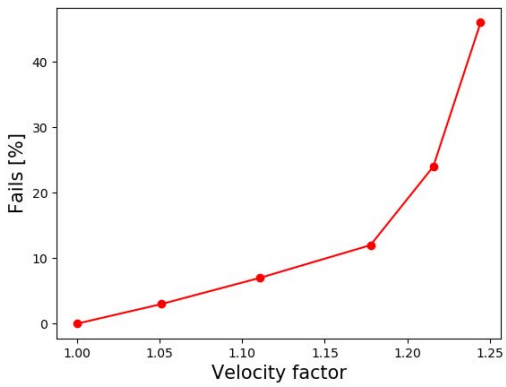


Figure 7: Failure rate of VOD when scaling up the VOD velocity

E. Closer Look at REVO+F

Before we conclude this section, we would like to take a closer look at how adding the VOD output (τ_{VOD}) as a feature to the state (REVO+F) influences the training process. When running the training process on a single

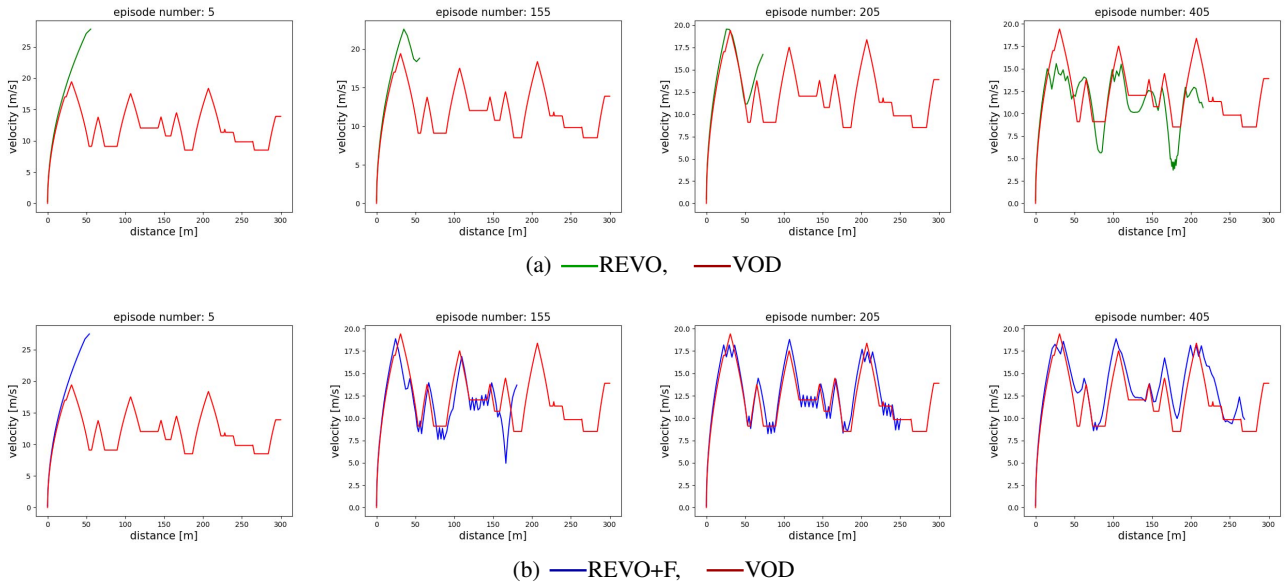


Figure 8: A comparison between learning progress of REVO and REVO+F. In episode 5, both methods accelerate until a roll-over occurs. In episode 155, REVO+F started to imitate the VOD controller, while REVO shows a very little progress. In episode 205, D-VOL shows almost full imitation of VOD velocity, while REVO is still in its initial stages of learning. In episode 405, REVO+F actions result in a higher velocity than VOD

randomly picked path (instead of training each episode on a new path) it is possible to closely track the policy improvement. In this case, as can be seen in Fig. 8, after some training, the learned policy uses the VOD information supplied through the additional feature, hence the velocity profile is similar to that of VOD; while the policy achieved by the regular training process (REVO) is still not able to complete this path. More research is required to understand this observation better.

VII. Conclusions

In this paper, we addressed the issue of deep reinforcement learning of autonomous driving at high speeds along specified paths, while accounting for the vehicle dynamics and its dynamic constraints (rollover and sliding). To this end, we proposed two methods, each combine traditional deep reinforcement learning (REVO) with a direct computation of the time optimal velocity profile along a given path (VOD). One method, denoted REVO+A, adds actions of REVO and VOD so that it is initialized at the VOD profile, and thus it learns only the required deviations from the model-based optimal speeds. The second method, denoted REVO+F, adds the action of VOD as a feature to the state of REVO.

The two methods were tested in experiments using a simulator that simulates the dynamics of a real vehicle. We show that REVO+A results in a significant improvement to the basic reinforcement learner REVO, especially at early stages of the learning process. It was shown that the REVO took around $40,000$ iterations to converge to an model-based velocity controller (VOD), compared to an immediate convergence by the combined controller (REVO+A). Another interesting result was that the learning process improved over the model-based velocity profile. This is not surprising as we used a relatively simple and computational effective vehicle model to speed up computation and the learning process.

The REVO+F method showed no significant advantage for randomly chosen paths. However, when learning to drive along a single path, it quickly converged to the VOD velocity profile. This suggests that the REVO+F agent quickly recognizes the utility of the model-based velocity profile. Further research may be required in order to take advantage of this phenomenon.

References

[1] Moshe Mann and Zvi Shiller. “Dynamic stability of off-road vehicles: Quasi-3D analysis”. In: *Robotics and Automation, 2008. ICRA 2008. IEEE International Conference on*. IEEE. 2008, pp. 2301–2306.

[2] Florent Altché, Philip Polack, and Arnaud de La Fortelle. “A simple dynamic model for aggressive, near-limits trajectory planning”. In: *Intelligent Vehicles Symposium (IV), 2017 IEEE*. IEEE. 2017, pp. 141–147.

[3] Jeong hwan Jeon et al. “Optimal motion planning with the half-car dynamical model for autonomous high-speed driving”. In: *American Control Conference (ACC), 2013*. IEEE. 2013, pp. 188–193.

[4] Toni Petrinić, Mišel Brezak, and Ivan Petrović. “Time-optimal velocity planning along predefined path for static formations of mobile robots”. In: *International Journal of Control, Automation and Systems* 15.1 (2017), pp. 293–302.

[5] Mohamed Elbanhawi, Milan Simic, and Reza Jazar. “In the passenger seat: investigating ride comfort measures in autonomous cars”. In: *IEEE Intelligent Transportation Systems Magazine* 7.3 (2015), pp. 4–17.

[6] Chris J Ostafew et al. “Speed daemon: experience-based mobile robot speed scheduling”. In: *2014 Canadian Conference on Computer and Robot Vision*. IEEE. 2014, pp. 56–62.

[7] Mariusz Bojarski et al. “End to end learning for self-driving cars”. In: *arXiv preprint arXiv:1604.07316* (2016).

[8] Jeff Michels, Ashutosh Saxena, and Andrew Y Ng. “High speed obstacle avoidance using monocular vision and reinforcement learning”. In: *Proceedings of the 22nd international conference on Machine learning*. ACM. 2005, pp. 593–600.

[9] Chenyi Chen et al. “Deepdriving: Learning affordance for direct perception in autonomous driving”. In: *Proceedings of the IEEE International Conference on Computer Vision*. 2015, pp. 2722–2730.

[10] Paul Drews et al. “Aggressive deep driving: Model predictive control with a cnn cost model”. In: *arXiv preprint arXiv:1707.05303* (2017).

[11] Chris J Ostafew et al. “Learning-based Nonlinear Model Predictive Control to Improve Vision-based Mobile Robot Path Tracking”. In: *Journal of Field Robotics* 33.1 (2016), pp. 133–152.

[12] Yi Zhang et al. “Human-like Autonomous Vehicle Speed Control by Deep Reinforcement Learning with Double Q-Learning”. In: *2018 IEEE Intelligent Vehicles Symposium (IV)*. IEEE. 2018, pp. 1251–1256.

[13] Stéphanie Lefèvre, Ashwin Carvalho, and Francesco Borrelli. “A learning-based framework for velocity control in autonomous driving”. In: *IEEE Transactions on Automation Science and Engineering* 13.1 (2016), pp. 32–42.

[14] Hasitha Dilshani Gamage and Jinwoo Brian Lee. “Reinforcement Learning based Driving Speed Control for Two Vehicle Scenario”. In: *Australasian Transport Research Forum (ATRF), 39th, 2017, Auckland, New Zealand*. 2017.

[15] Zhenhua Huang et al. “Parameterized batch reinforcement learning for longitudinal control of autonomous land vehicles”. In: *IEEE Transactions on Systems, Man, and Cybernetics: Systems* 99 (2017), pp. 1–12.

[16] Ugo Rosolia and Francesco Borrelli. “Learning model predictive control for iterative tasks. a data-driven control framework”. In: *IEEE Transactions on Automatic Control* 63.7 (2018).

[17] Jarrod M Snider et al. “Automatic steering methods for autonomous automobile path tracking”. In: *Robotics Institute, Pittsburgh, PA, Tech. Rep. CMU-RITR-09-08* (2009).

[18] Andrew Sendonaris and Gabriel Dulac-Arnold. “Learning from Demonstrations for Real World Reinforcement Learning”. In: *arXiv preprint arXiv:1704.03732* (2017).

[19] David Silver et al. “Mastering the game of Go with deep neural networks and tree search”. In: *Nature* 529.7587 (2016), pp. 484–489.

[20] Zvi Shiller and Y-R Gwo. “Dynamic motion planning of autonomous vehicles”. In: *IEEE Transactions on Robotics and Automation* 7.2 (1991), pp. 241–249.

[21] Timothy P Lillicrap et al. “Continuous control with deep reinforcement learning”. In: *arXiv preprint arXiv:1509.02971* (2015).

[22] Zvi Shiller and Hsueh-Hen Lu. “Computation of path constrained time optimal motions with dynamic singularities”. In: *Journal of dynamic systems, measurement, and control* 114.1 (1992), pp. 34–40.

[23] AE Bryson and Yu-Chi Ho. “Applied optimal control. 1969”. In: *Blaisdell, Waltham, Mass* 8 (1969), p. 72.

[24] Unity Technologies. *unity3d*. https://unity3d.com/. 2019.

[25] *Autonomous driving at time optimal velocity*. Youtube. 2019. URL: https://youtu.be/s4p9JRacdy0.

Model-Based Reinforcement Learning for Time-Optimal Velocity Control

Gabriel Hartmann, Zvi Shiller, and Amos Azaria

***Abstract*—Autonomous navigation has recently gained great interest in the field of reinforcement learning. However, little attention was given to the time-optimal velocity control problem, i.e. controlling a vehicle such that it travels at the maximal speed without becoming dynamically unstable (roll-over or sliding). Time optimal velocity control can be solved numerically using existing methods that are based on optimal control and vehicle dynamics. In this letter, we develop a model-based deep reinforcement learning to generate the time-optimal velocity control. Moreover, we introduce a method that uses a numerical solution that predicts whether the vehicle may become unstable and intervenes if needed. We show that our combined model outperforms several baselines as it achieves higher velocities (with only one minute of training) and does not encounter any failures during the training process.**

***Index Terms*—Autonomous vehicle navigation, reinforcement learning, motion and path planning.**

I. Introduction

THE operation of autonomous vehicles requires the synergetic application of a few critical technologies, such as sensing, localization, motion planning, and control. This letter focuses on a subset of the motion planning problem, that is moving at the time optimal speeds to minimize travel time along a given path, while ensuring the vehicle's dynamic stability. By "dynamic stability" we refer to constraints on the vehicle that are functions of its speed, such as not rolling-over and not sliding [1]–[3]. Other constraints that may affect the vehicle speeds, such as traffic laws and passenger comfort [4], while important for driving autonomous vehicles, are not considered in this letter as they are secondary to ensuring the vehicle stability at high speeds.

Time optimal velocity profile is affected by the vehicle's dynamic capabilities, such as its maximum and minimum acceleration, ground/wheels interaction, terrain topography, and path geometry. Therefore, the actual underlying dynamic model of the vehicle is very complex. Yet such a model is required to ensure that the vehicle is dynamically stable during motion at any point along the path [1].

Model-based reinforcement learning is an effective way to learn the complex dynamic model of the vehicle from its actual responses, thus bridging the gap that separates the real vehicle dynamics from its analytical model. Furthermore, the automated learning process does not require exact information of the vehicle's physical properties. Therefore, we present a model-based reinforcement learning method for driving a vehicle at near time optimal speeds along any given path. The dynamic model of the vehicle is learned and used for planning the acceleration actions that maximize vehicle speeds along the path, without compromising its dynamic stability.

Despite its obvious advantages, reinforcement learning has its limitations due to the limited time available for any learning process, which results in a limited exploration of the state-space. This is particularly evident at the beginning of the learning process, when the learned model may still be highly inaccurate [5]. This is critical when attempting to reach the vehicle's performance limits during the learning process, as it may result in a failure (i.e. vehicle instability).

To this end, we propose a dual-model approach that protects the vehicle from reaching dynamically unstable states. A simplified and conservative analytical model that accounts for vehicle dynamic safety is used to predict if every vehicle command is dynamically safe. If an unsafe maneuver is attempted, an alternative safe local maneuver is executed. By relying on both learned and analytical models, our method gains the best of both worlds: the high performance of the learned model and the safety of the analytical model.

The proposed methods were implemented in a simulation for a ground vehicle, moving along arbitrary paths in the plane. We show that the model-based RL agent learns to drive the vehicle at high speeds after only one minute of real-time driving. By intervening in potentially unsafe situations, our method eliminates any failures during the entire learning process and even achieves higher velocities as compared to the velocities achieved by the model-based RL agent alone. We also compare the results to a model-free RL method (DDPG) and show that our proposed model-based RL agent converges in a significantly faster time and achieves higher performance.

The main contributions of this letter are thus twofold: (i) a model-based reinforcement learning method for driving a vehicle at near time-optimal speeds along any given path in

Manuscript received February 24, 2020; accepted July 6, 2020. Date of publication July 27, 2020; date of current version August 7, 2020. This letter was recommended for publication by Associate Editor P. Tokekar and Editor D. Popa upon evaluation of the Reviewers' comments. This work was supported by the Ministry of Science & Technology, Israel. *(Corresponding author: Gabriel Hartmann.)*

Gabriel Hartmann is with the Department of Mechanical Engineering and Mechatronics, Ariel University, 40700 Ariel, Israel, and also with the Department of Computer Science, Ariel University, 40700 Ariel, Israel (e-mail: gavrielhartmann@gmail.com).

Zvi Shiller is with the Department of Mechanical Engineering and Mechatronics, Ariel University, 40700 Ariel, Israel (e-mail: shiller@ariel.ac.il).

Amos Azaria is with the Department of Computer Science, Ariel University, 40700 Ariel, Israel (e-mail: amos.azaria@ariel.ac.il).

Digital Object Identifier 10.1109/LRA.2020.3012128

less than one minute of real-time learning; (ii) A dual-model approach, that combines model-based RL with an analytical planner. This approach protects the vehicle from reaching dynamically unstable states. Extensive experiments show that our method outperforms other baselines.

II. Related Work

Autonomous driving at high speeds that approach the vehicle's performance limits was demonstrated so far in simulations [2], [6], on small-size vehicles [7], [8] and even on full-size vehicles [9], [10]. In these applications, the vehicle behavior is usually described by a mathematical dynamic model. While these works demonstrate impressive vehicle performance, they are model specific and require exact information of the vehicle's physical properties.

The need to develop more general techniques lead to the use of reinforcement learning (RL) to learn the required functions for autonomous driving. There are two types of RL algorithms: model-free RL and model-based RL; in model-free RL the policy is learned based on a reward signal received during the interaction with the environment. Contrary, model-based RL first learns a model of the transition dynamics and a policy is then composed based on this model. Both model-free and model-based RL are successfully used in the field of autonomous driving. Most works have focused on perception and steering [11]–[13]. Other works have considered human imitation for velocity control [14], [15], tracking a given reference velocity [16], and achieving fuel efficiency [17].

Some recent works use RL for aggressive driving: Jaritz *et al.* [18] use model-free RL for simulated end-to-end racing. However, the millions of training steps are required to converge, which is impractical for real applications, and the safety of the learned driving policy is not guaranteed. Williams *et al.* [19] develop an agent that drives a small-size vehicle by learning its dynamic model. While they achieve good results with respect to vehicle velocity, their work does not focus on vehicle's safety and their planning method is based on intense sampling. Furthermore, the vehicle must be manually driven in order to initialize the model.

Safe learning is an active field of research; some works learn a safe policy, that avoids failure after convergence by defining a safety-directed optimization criterion or guiding the learning process by external knowledge [5]. Other works emphasize safety during the learning process. One way is to update the policy while preventing the policy to reach unsafe situations by using a lyapunov function [20], [21]. However, we show that it is challenging to construct a policy that will be safe with high probability when driving near the performance limits. That leads us to the dual-model approach, which enables the driving policy to focus on performance without considering safety with high probability. Existing works propose general methods that are not domain specific that define a safety backup policy by a formal specification language [22], [23], others use action pruning in a discrete state space game [24]. A similar approach to our safety module uses model predictive control to ensure the safety of a learned policy [25]. However, they assume known system dynamics contrary to our dual approach that learns the dynamic model and uses an analytical model as a backup. All these work are not related to time-optimal velocity control where the system is pushed to drive near the performance limits. Because of the complex vehicle dynamics, and the importance of human safety when driving at high speeds we propose a domain-specific method that allows us to speed up the learning process and minimize the number of failures to zero, as was demonstrated in the paper. Furthermore, we rely on the stabilization policy to drive the vehicle closer to the performance limits and thus enabling higher safe velocity.

III. Problem Statement

In our setting, an agent is faced with a vehicle and a given path, and the agent must determine the acceleration and deceleration in order to complete the path without failure at minimal time. These commands are learned by the agent, whereas the steering angle is controlled by an analytical path following controller [26].

The path P is defined by N discrete points, $P = \{p_1, p_2, \ldots, p_N\}$. The position of the vehicle's center of mass is denoted by $q \in \mathbb{R}^2$ and yaw angle θ. The vehicle's speed is $v_y \in \mathbb{R}, v_y \geq 0$ and the steering angle is denoted by δ. The action is defined as $a = \{u, \delta_d\}$, where u controls the throttle and brake (which affects vehicle's acceleration (and deceleration)), and δ_d controls the steering. The steering command δ_d of the vehicle is provided by a path following controller (pure pursuit [26]) π_δ, which receives as input the vehicle's state and the path relative to vehicle's position.

The time optimal policy maximizes the speed along the path during a fixed distance. That is, for every path P,

the time optimal policy π^* outputs, at every time t, the action $u = \pi^*(s_t)$ that maximizes the vehicle speed (minimizing traveling time), while ensuring that every state s_t is stable.

The time optimal velocity along path P is the velocity profile $v_y(t)$ produced by the optimal policy π^*.

IV. Dynamic Model

The dual-model approach is based on a learned model that is used to drive the vehicle at the maximal speeds, and on an analytical model that is used to determine the dynamic stability of the current vehicle state. We first describe the analytical model that is based on a simple planar bicycle model, as described next.

A. Bicycle Model

The bicycle model represents the vehicle by only two wheels, by collapsing the two front and rear wheels into one, as shown in Fig. 1. The bicycle is steered by the front wheel at the steering angle δ and is assumed, for simplicity, to be driven by the force F_t applied on the rear wheel. The radius of curvature R, measured from the center of rotation to the center of mass, is easily derived from the steering angle δ:

$$R = \sqrt{l_r^2 + \left(\frac{l_r + l_f}{\tan\delta}\right)^2}, \tag{1}$$

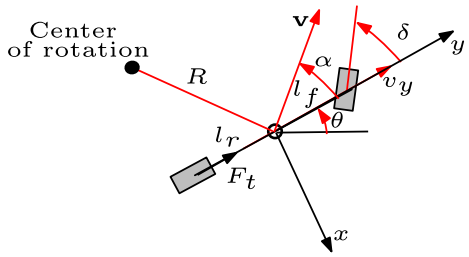


Fig. 1. The bicycle model.

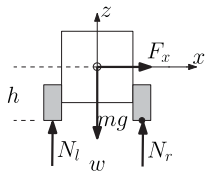


Fig. 2. Parameters for the simple roll model.

where l_f and l_r are the distances of the front and rear wheels from the center of mass, respectively. The angle between the velocity v of the vehicle center mass and its y axis is the slip angle α. The motion of the bicycle is influenced by two parameters: the steering angle δ and the driving force F_t.

The bicycle's equations of motion are thus:

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \\ \dot{\delta} \\ \dot{v_y} \end{bmatrix} = \begin{bmatrix} -v \sin \alpha \\ v \cos \alpha \\ v/R \\ k(\delta_d - \delta) \\ F_t/m \end{bmatrix} \tag{2}$$

where $\dot{x}$ and $\dot{y}$ are the projections of v on the vehicle's local coordinate frame, and

$$\alpha = \arcsin \frac{l_r}{R} \quad v = \frac{v_y}{\cos \alpha} \tag{3}$$

To account for the time response of the steering system, the steering angle δ is driven by a proportional controller with a desired angle δ_d.

B. Dynamic Stability

To account for the dynamic stability of a vehicle moving on a flat surface, we consider sliding and roll-over. For the sake of simplicity, we focus only on roll-over. Referring to Fig. 2, showing the vehicle from its rear view, roll-over may occur during a counter clockwise turn if the vehicle moves at a high speed at which the left wheel separates from the ground. We use the absolute Lateral load Transfer Rate (LTR), to estimate how close the vehicle is to a roll-over. The LTR describes the different between the load on the left and the load on the right wheels and is defined as:

$$LTR = \frac{|N_r - N_l|}{N_r + N_l} \in [0, 1]. \tag{4}$$

For our roll model, the LTR is computed by summing the moments acting on the vehicle around the point of contact with the ground:

$$LTR = \frac{2v^2\, h}{Rwg}. \tag{5}$$

The maximal velocity for a given state is when $LTR = 1$ i.e. when the load on one of the wheels is zero. The LTR is later used to determine the potential instability of a given action, as is discussed later in Section VI-B.

V. Time Optimal Velocity Control Using Model-Free Reinforcement Learning

We briefly describe a model-free RL algorithm to drive the vehicle time optimally (we later present our model-based RL algorithm).

This approach is a direct adaptation of "Deep Deterministic Policy Gradient" (DDPG) [27] to the time optimal velocity control problem. DDPG is an actor-critic, model-free algorithm for a continuous action space, A, and a continuous state space, S. DDPG learns the policy using policy gradient. The exploration of the environment is done by adding exploration noise to the actions.

The training process consists of episodes; at each episode the vehicle moves along a randomly generated path. Each path, P, is generated by smoothly connecting short path segments of random length and curvature until reaching the desired length. This ensures that the selected path respects the vehicles steering capabilities.

The DDPG state, s^{DDPG}, includes a down-sampled limited horizon path segment, $P_s \subseteq P$, which is defined relative to the vehicle's position. In addition to this path segment, also the current velocity of vehicle (v_y) and steering (δ) is included in the state. Therefore, the full state of the system is defined as $s^{DDPG} = \{v_y, \delta, P_s\}$.

The reward function is defined as follows: If the vehicle is stable and has a positive velocity, the reward r is proportional to the vehicle's velocity ($r_t = kv_t, k \in \mathbb{R}_+$). If the vehicle encounters an unstable state or exceeds the maximal allowed LTR, it receives a negative reward. To encourage the agent not to stand without moving at all, a small negative reward is received if $v_t = 0$. The episode terminates at time T or if the vehicle becomes unstable. (see [28] for more details on the adaptation of DDPG to the time optimal velocity problem).

The results in section VIII-D infer that DDPG is able to achieve high velocities. However this approach has several limitations; the training time is too long for being practical in real vehicles and there are still an unacceptable probability of becoming unstable. The nontransparent nature of model-free approaches makes it even more difficult to trust the RL policy. We use DDPG as a baseline to our model-based RL methods.

VI. Time Optimal Velocity Control Using Model-Based Reinforcement Learning (LMVO)

We propose a Model-based RL algorithm (Learned Model Velocity Optimization - LMVO). Model-based RL is considered

to be more sample efficient than model-free RL [29], [30], which may be important in real-world applications. The reason for that efficiency is that the general behavior of the system is learned, therefore, it is possible to choose the best actions sequences without explicitly trying these sequences before. Therefore, The agent can drive safely and near performance limits along arbitrary paths after training on just a few different paths.

Clearly, the performance of the model-based RL algorithm depends on the accuracy of the learned model. However, not in all environments it is simple to learn an accurate model. Vehicle dynamics are relatively predictable, therefore, model-based RL, is expected to be useful for vehicle motion control.

Unlike DDPG, that learns the driving policy directly, LMVO includes two parts: learning the dynamics of the vehicle, which are not known (model learning), and using the learned model to compute the actions (policy).

A. LMVO Model Learning

There are several approaches for learning the model, such as parameter tuning of an expert-designed model [31], using Gaussian processes [32], and neural networks [30], [33]. We approximated the vehicle's model by a deep neural network; the advantage of deep neural networks is their power of approximating general functions, including highly non-linear functions.

The vehicle dynamic model are described by a global prediction transition function f^v that given the state s_t^v and action a, outputs the state s_{t+1}^v after one time step: $s_{t+1}^v = f^v(s_t^v, a_t)$.

1) Vehicle's State Definition: The vehicle's state s_t^v is defined as: $s_t^v = \{x, y, \theta, v_y, \delta\}$, where x, y are the incremental change in the vehicle position from position q_{t-1} to position q_t (in the frame attached to the vehicle at time $t-1$), angle $\theta = \theta_t - \theta_{t-1}$, v_y is the linear velocity of the vehicle, and δ is the steering angle. The action $a = \{u, \delta_d\}$ consist of the throttle command u and the steering command δ_d.

The next state s_{t+1}^v does not depend on the position x, y, θ of the current state s_t^v, therefore, the relative position is not used as an input to the prediction transition function f^v. During training, the vehicle drives along randomly created paths. At each time step t during the driving, the state s_t^v and the action a_t are saved to a state buffer D that stores the data for training f^v.

The neural network that approximates the model function f^v is trained by gradient descent, mean-squared error minimization on the collected samples stored in D (excluding the final samples, which do not have any following state).

2) Prediction Transition Function: Instead of directly predicting the next state s_{t+1}^v, we use a neural network to predict the difference between the current state s_t^v and the next state s_{t+1}^v: $\Delta s^v = s_{t+1}^v - s_t^v$, as described in [30]. The effect of the action on state's change is more significant compared to the effect on the next state itself, therefore, the neural network can represent the model more accurately.

The prediction transition function f^v has the following architecture: the features of the state s^v (except the position) are inserted to a fully connected layer (100 neurons) and the outputs of that layer are inserted to 5 separated sections that consist of two fully connected layers (20 neurons in each layer). Each of these sections outputs one output feature. Fig. 3 summarizes this architecture.

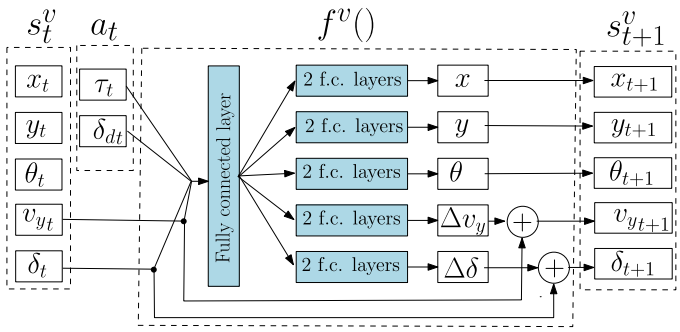


Fig. 3. The prediction transition function $s_{t+1}^v = f^v(s_t^v, a_t)$ architecture. The blue rectangles are the fully connected layers of the neural network.

3) Multi-Step Roll-Outs: If at time step t the n future actions $a_t, a_{t+1}, \ldots a_{t+n}$ are known, it is possible to predict the n next states (roll-out). To make a multi-step roll-out, a sequential single-step prediction is performed. The predicted single-step relative positions are integrated during the n-step roll-out, resulting in a geometric path relative to the position at time-step t and the velocity and the steering in the feature states. An example of such roll-out is illustrated in Fig. 8.

The next section explains the usage of this learned model to choose optimal actions.

B. LMVO Policy

The action u considered by the LMVO policy is either the maximal throttle, $u_{\max}$, or minimal acceleration (i.e. maximal brake), $u_{\min}$.We note that in order to ensure that the vehicle does not result in an unavoidable failure it is enough to test whether the vehicle can decelerate to a full stop from a given state, s_t^v. Therefore, LMVO rolls-out future states when the acceleration action is $u_{\min}$ (maximal brake) and the steering commands are computed by the path following controller π_δ on the predicted future states. For all rolled-out future states, it is checked if the predicted LTR is lower than 1, which indicates that the vehicle is expected to remain safe.

To decide whether the vehicle is allowed to accelerate (by applying $u_{\max}$ for a single time step) at state s_t^v, LMVO ensures that the vehicle can stop safely after this acceleration step. If the vehicle cannot stop safely from s_{t+1}^v, the vehicle must decelerate, and therefore, $u_{\min}$ is applied. Assuming a perfectly accurate model, if for every time step during the motion the vehicle accelerates only if possible, the vehicle always stays within a safe velocity envelope. Algorithm 1 describes LMVO's driving policy.

1) Margin of Safety: The model f^v predicts the expected values of the future states. However, the actual value may be different due to model error or stochastic dynamics. We note that when the vehicle drives on the limit of performance, an under-estimated prediction of the future LTR is unacceptable because it may lead to instability of the vehicle. Therefore, a margin of safety is added to the expected LTR. Since the

Algorithm 1: LMVO's Driving Policy. The Algorithm Receives as Input the Current State s_0^v and a Short Path Segment P_0 Relative to Current Vehicle's Position and Returns the Acceleration Action u. f^v is the Prediction Transition Function. The Function $transformPath$ Transforms P_t Relative to the New Position of the Vehicle.

Input: s_0^v, P_0
Output: u
$\delta_d \leftarrow \pi_\delta(s_0^v, P_0)$;
$s_1^v \leftarrow f^v(s_0^v, \{u_{max}, \delta_d\})$;
if s_1^v *is unstable* **then return** u_{min};
else
  $t \leftarrow 1$;
  while *vehicle speed in* $s_t^v > 0$ **do**
    $P_t \leftarrow transformPath(P_{t-1}, s_t^v)$;
    $s_{t+1}^v \leftarrow f^v(s_t^v, \{u_{min}, \pi_\delta(s_t^v, P_t)\})$;
    if s_{t+1}^v *is unstable* **then return** u_{min};
    $t \leftarrow t + 1$;
  end
  return u_{max};
end

multi-step predictions are computed iteratively based on the previous step, the error between the predicted and actual values is expected to grow with the number of steps. For simplicity, we take a safety factor that is linear with the number of future steps. More formally, for the rolled-out future LTR values $LTR_t, LTR_{t+1}, LTR_{t+2}, \ldots, LTR_{t+n}$ the vehicle is expected to be stable up to time step $t + n$ if

$$\forall LTR_i, i \in \{0, n\}, LTR_i + \beta i < 1 \tag{6}$$

where β is the safety factor constant. Section VIII compares between a variety of values of β.

2) Computing Time: At every time step t, the action a_t must be applied immediately. However, since the computing time is not negligible, the command is applied with some delay. To solve this problem, instead of computing action a_t at time t, action a_{t+1} is computed based on the *predicted* next state s_{t+1} and a_{t+1} is applied immediately when obtaining the actual state s_{t+1} at time $t + 1$ (which may be slightly different than the model's prediction for that state).

VII. Stabilization by Using a Prior Dynamic-Model

Since the significance of failure is very severe, even situations with a low failing probability should be avoided. However, if taken to the extreme, the vehicle will not be able to drive at high velocities at all, since there may be rare occasions in which the vehicle might fail (see Fig. 9). To avoid the need to consider the rare cases of high errors and therefore allowing a smaller margin of safety, we propose the use of the Failure Prediction and Intervention Module (FIM). The general idea behind FIM is that if a failure is predicted, FIM overrides the vehicle actions, so that the failure state is not reached. This allows the use of a much lower safety factor.

Algorithm 2: The Failure Prediction and Intervention Module's Policy (π^{FIM}). $f^s(s_t, a_t)$ is a Prediction Transition Function. $\pi(s_t)$ and $\pi^s(s_t)$ are Given Policies and $SAFE(s_t, \pi')$ is a Function that Predicts if Following Policy π' from State s_t is Safe.

Input: state s_0
Output: action a
  $a_0 \leftarrow \pi(s_0)$;
  $s_{t+1} \leftarrow f^s(s_0, a_0)$;
  If $SAFE(s_{t+1}, \pi^s) = False$ **then return** $\pi^s(s_0)$
  return a_0

A. Prediction and Intervention Module (FIM)

We define a prediction transition function $f^s(s_t, a_t)$, and two policies $\pi(s_t)$ and $\pi^s(s_t)$. $SAFE(s_t, \pi')$ is a function that predicts if following policy π' from state s_t is safe (e.g. the vehicle remains stable). The Failure prediction and Intervention Module policy $\pi^{FIM}(s_0)$ predicts using f^s if it will be safe to execute $a_0 = \pi(s_0)$ and then following π^s. If all future states are safe, π^{FIM} returns a_0 otherwise it returns a safe action $a_s = \pi^s(s_0)$. Algorithm 2 describes π^{FIM}. We note that if $SAFE(s_0, \pi^s) = True$ then it is guaranteed that also $SAFE(s_0, \pi^{FIM}) = True$. This is true regardless of the specific policy π.

We now introduce the use of the FIM module for our driving agent, which will be termed as the LMVO+FIM method. LMVO+FIM uses the LMVO driving policy (algorithm 1) as its π. As mentioned, the safety of π does not impact the safety of π^{FIM}, therefore, π^{FIM} is expected to be safe also before the learned model f^v converges. The safety policy π^{FIM} on the other hand, is not required to obtain high performance (i.e high velocities) as it is only responsible of overriding actions that may lead to instability according to f^s. For f^s the bicycle model is used; the SAFE function that is used by the FIM module, tests by performing a roll-out using the bicycle model (f^s) whether the vehicle can safely reach a full stop. If so, it returns True, otherwise it returns False. LMVO+FIM's safety policy, π^s, first tests whether $SAFE(f^s(s^v, \{u_{\min}, \delta_d\}), \pi^s) = True$; if so it returns $a = \{u_{\min}, \delta_d\}$ otherwise, the steering action straightens the steering wheel and the acceleration action is $u_{\min}$ (i.e. maximal brake). That is, π^s tries to brake while using the regular controller for steering, but if it predicts that the vehicle will still result in an unstable state, it also straightens the steering wheel which will prevent the expected roll-over by reducing the radius of curvature of the future state and following that, reducing LTR. In addition to ensuring that the vehicle will not roll-over, the FIM module ensures that this safety maneuver does not cause the vehicle to deviate from the road.

VIII. Experimental Results

The performance of the proposed methods were tested in several experiments as detailed henceforth.

A. Settings

A simulation of a four-wheel vehicle was developed using "Unity" [34] software,[1] which simulates realistic vehicle dynamics. The vehicle properties are: mass$= 3,200Kg$, a total force produced by all wheels of $21KN$, center of mass (COM) height $h = 1.0$ m, COM to rear wheel distance $l_r = 1.55$ m, COM to front wheel distance $l_f = 1.55$ m and width$= 2.1$ m. The vehicle is driven by all wheels (4×4); steering is done by the front wheels (Ackermann steering). For DDPG we use hyper-parameters as described in [27]. For LMVO, we use batch size of 64 with batch normalization and learning rate 10^{-4}.Each episode is limited to 100 time steps. The time step is set to 0.2 seconds, i.e. 20 seconds per episode. An episode is considered as a failure if the vehicle rolled-over or deviated more then 2 m from the nominal path.

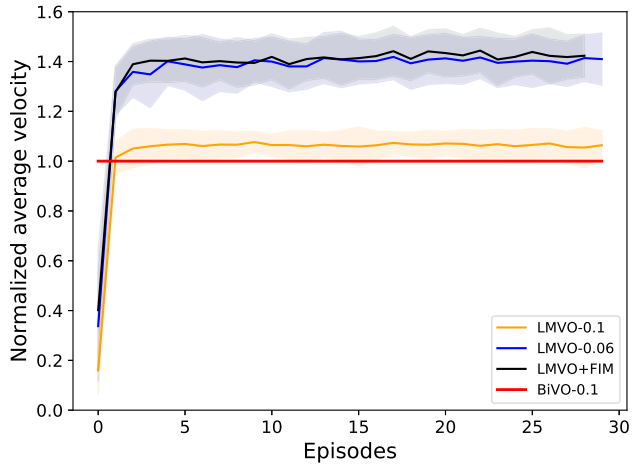
During the training process, the vehicle drives along randomly generated paths using the learned policy, 30 episodes in each learning process (at every episode the vehicle drives along a different path). The learning process is repeated 100 times with different seed and paths. During the training of the DDPG agent, exploration noise is added to the actions and the noise is disabled during evaluation because the exploration noise will cause the vehicle to fail when driving near the performance limits. Different types and levels of noise don't significantly affect the learning process, therefore we used the same noise as [27] (OrnsteinUhlenbeck process). Because of the longer convergence time for the DDPG agent, it is trained for 200 episodes.

Since the average velocity during the failed episodes was usually higher than the average velocity during successful episodes, we excluded failed episodes when presenting the average velocity of each method. The baseline is a controller that uses the same policy as LMVO but instead of learning the model, the bicycle model is used (BiVO) with a safety factor of $\beta = 0.1$ to ensure safe driving. The average velocity at each time during the training is normalized with respect to the baseline.

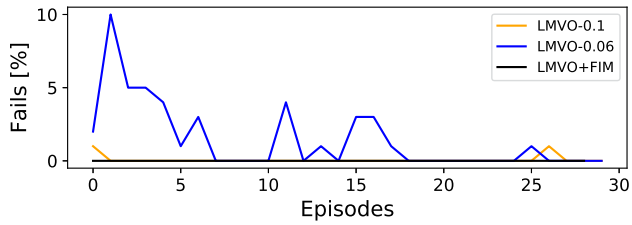
B. Learning Processes With and Without Intervention

Fig. 4 compares three different methods: LMVO with a safety factor of $\beta = 0.1$ ((LMVO-0.1), LMVO with a safety factor $\beta = 0.06$ (LMVO-0.06) and LMVO+FIM with a safety factor of $\beta = 0.05$. A comparison to DDPG is shown in section VIII-D. As depicted by the figures, an important advantage of LMVO+FIM over the other methods is that it maintains safety also during the beginning of the training process, where the learned model may be inaccurate. LMVO-0.1 achieves lower velocity compared to LMVO-0.06 and LMVO+FIM but has a lower failure rate compared to LMVO-0.06. LMVO+FIM maximizes the velocity without failing and even achieves a slightly higher velocity compared to LMVO-0.06. These results validate our assumption that LMVO+FIM can learn how to maximize the velocity without failing and can achieve velocity that is not lower than a not as safe driving policy (i.e. LMVO-0.06).

[1]A video of the vehicle driving along a path at time optimal velocity is available at: https://youtu.be/Ffo3SYonwPk.



(a) Normalized average velocity during 100 learning processes (higher is better).



(b) Failure rate during 100 learning processes (lower is better). Note that by using the stabilization policy LMVO+FIM, there are no failures during the entire training process.

Fig. 4. (a) Normalized average velocity during 100 learning processes (higher is better). (b) Failure rate during 100 learning processes (lower is better). Note that by using the stabilization policy LMVO+FIM, there are no failures during the entire training process.

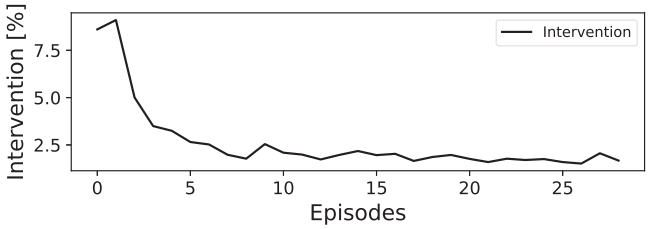


Fig. 5. Intervention of FIM module during training LMVO+FIM.

Fig. 5 describes the intervention of the stabilization policy during the training process. As expected, the number of interventions decrease during training.

C. FIM Analysis

Fig. 6 shows the influence of the safety factor on the average velocity. An LMVO agent was trained for 5 minutes (1500 samples) and was evaluated on different safety factors (β) with and without the intervention policy (FIM). For $0.14 > \beta > 0.08$ LMVO (blue) and LMVO+FIM (black) achieve the same velocity because the intervention policy did not require to intervene in the safe driving policy. For $\beta < 0.07$, the velocity of

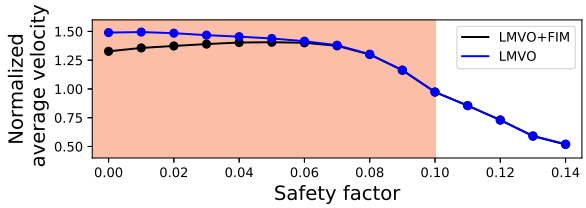
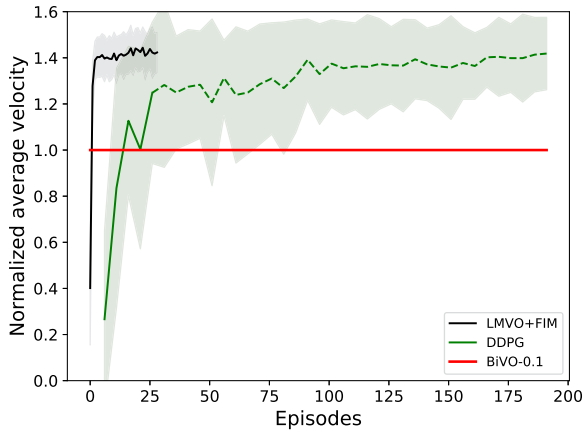
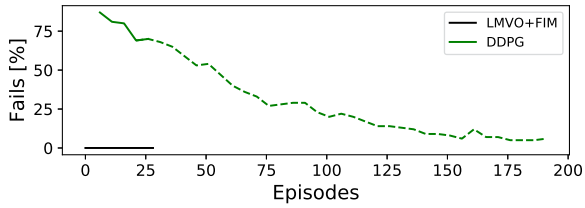


Fig. 6. Normalized average velocity as a function of linear safety factor (β). The red area marks the range of safety factors that may cause failure when used by LMVO without FIM.



(a) Average velocity, (normalized with respect to the baseline).



(b) Failure rate

Fig. 7. (a) Average velocity, (normalized with respect to the baseline). (b) Failure rate. LMVO+FIM compare to DDPG on 100 random learning processes. LMVO+FIM achieves higher velocity, in approximately 1% of the time that is required by DDPG, while completely preventing failure.

LMVO+FIM is lower than LMVO due to intervention policy that causes the vehicle to slow down and to deviate from the desired path. The continual decrease of the LMVO+FIM velocity for $\beta < 0.06$ is expected, because more interventions cause less optimal driving. As discussed in section VIII-B, LMVO must use a safety factor of 0.1 or higher in order to maintain a low failure rate (therefore LMVO cannot safely reach the hypothetical velocities depict in Fig. 6 where the safety factor is less than 0.1).

D. Comparison to Model-Free RL

Fig. 7 a compares LMVO+FIM to DDPG. As expected, LMVO+FIM converges to a good solution significantly faster than DDPG. For DDPG, approximately 200 episodes are required to achieve similar performance to what LMVO+FIM achieves after only 2 episodes (less than a minute). Furthermore, the failure rate achieved by LMVO+FIM is constantly zero,

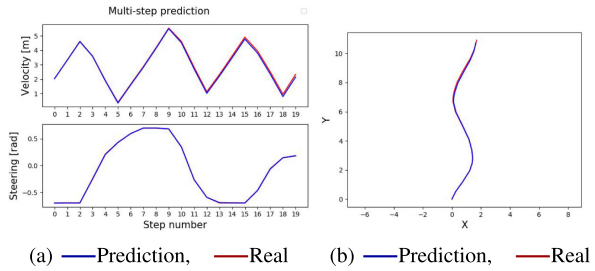


Fig. 8. (a) Prediction, Real (b) Prediction, Real. An example of a 20 time-steps roll-out. (A) Prediction of the state features as a function of time-steps and the actual values, (B) Predicted and actual paths relative to the vehicle located at the origin. Note that the prediction values merge with the actual values.

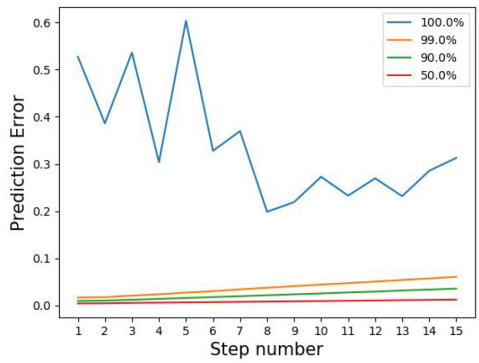


Fig. 9. Error between the predicted and the actual LTR as a function of the roll-out depth. Note the difference between the maximal error of the 99th percentile of samples and all samples.

compared to a very high failure rate starting at over 90% and even after 200 episodes, the failure rate is still approximately 5%.

E. Model Accuracy

For statistically analyzing the accuracy of the prediction function f^v, 100,000 samples of data were collected from driving the simulated vehicle by the LMVO policy. The model was trained on five minutes (1500 samples) of the collected samples and tested on the remaining samples. The results below are on the test samples (i.e. they were not used in the training process). To determine the action commands, a multi-step roll-out is used as described in VI-B. Fig. 8 shows an example of one roll-out of 20 time steps. The actual and the predicted future positions of the vehicle and the features prediction are depicted. The learned model predict the future states accurately even after 20 time steps (4 seconds). It is important to evaluate the maximum error between the predicted and the actual LTR, because the future LTR is used for deciding if the vehicle will remain stable in the future states. Fig. 9 shows the error between the predicted and the actual LTR as a function of the roll-out depth. We note that the maximum one step error of all samples is above 0.5, while for the 99th percentile, the maximal error is only 0.03. This result justifies the need described in section VII-A to disregard the extremely rare occasions, as including them would result in a

much larger safety factor, which in-turn may extremely reduce performance.

IX. Conclusion

In this letter, we addressed the issue of model-based deep reinforcement learning of autonomous driving at high speeds along paths, while accounting for the vehicle dynamics and its dynamic constraints. We proposed a method (LMVO) that learns to drive a vehicle by learning the dynamic model of the vehicle. Additionally, we used an analytical dynamic model (bicycle model) for predicting the stability of the future states (FIM). We combined the analytical dynamic model with the learned model to ensure vehicle safeness (LMVO+FIM). LMVO+FIM achieved the best performance; i.e. the highest velocity and no failures during the learning process. We showed that in five minutes LMVO learned a significantly more accurate dynamic model of the vehicle compared to the bicycle model (BiVO). Because the learned model was more accurate, both LMVO and LMVO+FIM achieved higher velocities than BiVO. We also compared LMVO+FIM to a model-free reinforcement method (DDPG). DDPG required about one hour to achieve the same velocity that LMVO+FIM achieved after less than one minute. furthermore, the failure rate of DDPG was significantly higher compared to that of LMVO+FIM.

References

[1] M. Mann and Z. Shiller, "Dynamic stability of off-road vehicles: Quasi-3d analysis," in *Proc. Int. Conf. Robot. Autom.*, 2008, pp. 2301–2306.

[2] F. Altché, P. Polack, and A. de La Fortelle, "A simple dynamic model for aggressive, near-limits trajectory planning," in *Proc. IV,*, 2017, pp. 141–147.

[3] T. Petrinić, M. Brezak, and I. Petrović, "Time-optimal velocity planning along predefined path for static formations of mobile robots," *Int. J. Control, Autom. Syst.*, 2017.

[4] M. Elbanhawi, M. Simic, and R. Jazar, "In the passenger seat: investigating ride comfort measures in autonomous cars," *IEEE Intell. Transp. Syst. M.*, vol. 7, no. 3, pp. 4–17, 2015.

[5] J. Garcıa and F. Fernández, "A comprehensive survey on safe reinforcement learning," *J. Mach. Learn. Res.*, vol. 16, no. 1, pp. 1437–1480, 2015.

[6] J. Liu, P. Jayakumar, J. L. Stein, and T. Ersal, "An mpc algorithm with combined speed and steering control for obstacle avoidance in autonomous ground vehicles," in *Proc. Am. Soc. Mech. Eng. Dyn. Syst. Control Conf.*, 2015.

[7] A. Liniger, A. Domahidi, and M. Morari, "Optimization-based autonomous racing of 1: 43 scale rc cars," *Optimal Control Appl. Methods*, vol. 36, no. 5, pp. 628–647, 2015.

[8] G. Williams, P. Drews, B. Goldfain, J. M. Rehg, and E. A. Theodorou, "Aggressive driving with model predictive path integral control," in *Proc. IEEE,Int. Conf. Robot. Autom.*, 2016, pp. 1433–1440.

[9] J. Funke *et al.*, "Up to the limits: Autonomous audi tts," in *Proc. IEEE Intell. Vehicles Symp.*, 2012, pp. 541–547.

[10] J. Kabzan *et al.*, "Amz driverless: The full autonomous racing system," 2019, *arXiv:1905.05150*.

[11] M. Bojarski *et al.*, "End to end learning for self-driving cars," 2016, *arXiv:1604.07316*.

[12] C. Chen, A. Seff, A. Kornhauser, and J. Xiao, "Deepdriving: Learning affordance for direct perception in autonomous driving," in *Proc. IEEE Int. Conf. Comput. Vision*, 2015, pp. 2722–2730.

[13] P. Drews, G. Williams, B. Goldfain, E. A. Theodorou, and J. M. Rehg, "Aggressive deep driving: Combining convolutional neural networks and model predictive control.," in *Conf. Robot Learn.*, Oct. 2017, pp. 133–142.

[14] Y. Zhang, P. Sun, Y. Yin, L. Lin, and X. Wang, "Human-like autonomous vehicle speed control by deep reinforcement learning with double q-learning," in *Proc. IEEE Intell. Vehicles Symp. (IV)*, 2018, pp. 1251–1256.

[15] S. Lefèvre, A. Carvalho, and F. Borrelli, "A learning-based framework for velocity control in autonomous driving," *IEEE Trans. Autom. Sci. Eng.*, vol. 13, no. 1, Jan. 2016.

[16] Z. Huang, X. Xu, H. He, J. Tan, and Z. Sun, "Parameterized batch reinforcement learning for longitudinal control of autonomous land vehicles," *IEEE Trans. Syst., Man, Cybern.: Syst.*, vol. 49, no. 4, pp. 1–12, 2019.

[17] H. D. Gamage and J. B. Lee, "Reinforcement learning based driving speed control for two vehicle scenario," in *Proc. 39th Australas. Transport. Res. Forum (ATRF)*, 2017.

[18] M. Jaritz, R. de Charette, M. Toromanoff, E. Perot, and F. Nashashibi, "End-to-end race driving with deep reinforcement learning," in *Proc. IEEE, Int. Conf. Robot. Autom.*, 2018, pp. 2070–2075.

[19] G. Williams, N. Wagener, B. Goldfain, P. Drews, J. M. Rehg, B. Boots, and E. A. Theodorou, "Information theoretic mpc for model-based reinforcement learning," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, 2017, pp. 1714–1721.

[20] Y. Chow, O. Nachum, E. Duenez-Guzman, and M. Ghavamzadeh, "A lyapunov-based approach to safe reinforcement learning," in *Proc. Adv. Neural Inf. Process. Syst.*, 2018, pp. 8092–8101.

[21] F. Berkenkamp, M. Turchetta, A. Schoellig, and A. Krause, "Safe model-based reinforcement learning with stability guarantees," in *Proc. Adv. Neural Inf. Process. Syst.*, 2017, pp. 908–918.

[22] M. Alshiekh, R. Bloem, R. Ehlers, B. Könighofer, S. Niekum, and U. Topcu, "Safe reinforcement learning via shielding," in *Thirty-Second AAAI Conf. Artif. Intell.*, 2018.

[23] X. Li, Z. Serlin, G. Yang, and C. Belta, "A formal methods approach to interpretable reinforcement learning for robotic planning," *Sci. Robot.*, vol. 4, no. 37, 2019.

[24] C. Gao, B. Kartal, P. Hernandez-Leal, and M. E. Taylor, "On hard exploration for reinforcement learning: A case study in pommerman," in *Proc. AAAI Conf. Artif. Intell. Interact. Digital Entertainment*, vol. 15, no. 1, 2019, pp. 24–30.

[25] O. Bastani, "Safe planning via model predictive shielding," 2019, *arXiv:1905.10691*.

[26] J. M. Snider *et al.*, "Automatic steering methods for autonomous automobile path tracking," *Robot. Inst., Pittsburgh, PA, Tech. Rep.*, 2009.

[27] T. P. Lillicrap *et al.*, "Continuous control with deep reinforcement learning," 2015, *arXiv:1509.02971*.

[28] G. Hartmann, Z. Shiller, and A. Azaria, "Deep reinforcement learning for time optimal velocity control using prior knowledge," *Int. Conf. Tools with Artif. Intell.*, 2019.

[29] M. P. Deisenroth *et al.*, "A survey on policy search for robotics," *Foundations and Trends in Robotics*, vol. 2, no. 1–2, pp. 1–142, 2013.

[30] A. Nagabandi, G. Kahn, R. S. Fearing, and S. Levine, "Neural network dynamics for model-based deep reinforcement learning with model-free fine-tuning," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2018, pp. 7559–7566.

[31] U. Rosolia, A. Carvalho, and F. Borrelli, "Autonomous racing using learning model predictive control," in *Proc. Am. Control Conf.*, 2017, pp. 5115–5120.

[32] C. E. Rasmussen, "Gaussian processes in machine learning," in *Proc. Summer School Mach. Learn.* Springer, 2003, pp. 63–71.

[33] I. Clavera, J. Rothfuss, J. Schulman, Y. Fujita, T. Asfour, and P. Abbeel, "Model-based reinforcement learning via meta-policy optimization," 2018, *arXiv:1809.05214*.

[34] U. Technologies, "unity3d," 2019, https://unity3d.com/.

Meta-Reinforcement Learning Using Model Parameters

Gabriel Hartmann[1,2] and Amos Azaria[2,3]

Abstract— **In meta-reinforcement learning, an agent is trained in multiple different environments and attempts to learn a meta-policy that can efficiently adapt to a new environment. This paper presents RAMP, a *Reinforcement learning Agent using Model Parameters* that utilizes the idea that a neural network trained to predict environment dynamics encapsulates the environment information. RAMP is constructed in two phases: in the first phase, a multi-environment parameterized dynamic model is learned. In the second phase, the model parameters of the dynamic model are used as context for the multi-environment policy of the model-free reinforcement learning agent. We show the performance of our novel method in simulated experiments and compare them to existing methods.**

I. Introduction

Common approaches for developing controllers do not rely on machine learning. Instead, engineers manually construct the controller based on general information about the world and the problem. After repetitively testing the controller in the environment, the engineer improves the controller based on the feedback from these tests. That is, a human is an essential part of this iterative process. Reinforcement Learning (RL) reduces human effort by automatically learning from interaction with the environment. Instead of explicitly designing and improving a controller, the engineer develops a general RL agent that learns to improve the controller's performance without human intervention. The RL agent is usually general and does not include specific information about the target environment; this allows it to adapt to different environments. Indeed, RL agents may achieve higher performance compared to human-crafted controllers [1]–[3]. However, RL agents usually require training from the ground up for every new environment, which requires extensive interaction in the new environment.

One solution to speed up the training time is to explicitly provide human-crafted information about the environment (context) to the RL agent [4]. However, such a solution requires explicitly analyzing the target environment, which may be challenging and time-consuming.

Instead of relying on the human understanding of the problem for providing such context, a *meta-Reinforcement Learning* (meta-RL) agent can learn to extract a proper environmental context. To that end, a meta-RL agent is trained on extended interaction in multiple different environments, and then, after a short interaction in a new, unseen environment, it is required to perform well in it [5], [6]. Specifically, a meta-RL algorithm that is based on context extraction is composed of two phases. First, in the meta-learning phase, the agent learns a general policy suitable to all environments given a context. Additionally, in this phase, the meta-RL agent learns how to extract a context from samples obtained from an environment. Secondly, in the adaptation phase, the meta-RL agent conducts a short interaction in the new environment, and the context is extracted from it. This context is then fed to the general policy, which acts in the new environment.

One common approach for context extraction is using a Recurrent Neural Network (RNN). That is, the RNN receives the history of the states, actions, and rewards and is trained to output a context that is useful for the general policy. However, the RNN long-term memory capability usually limits the effective history length [7]. Additionally, since the context vector is not explicitly explainable, it is difficult to examine the learning process and understand if the RNN learned to extract the representative properties of the environments.

In this paper, we introduce RAMP – a Reinforcement learning Agent using Model Parameters. We utilize the idea that a neural network trained to predict environment dynamics encapsulates the environment properties; therefore, its parameters can be used as the context for the policy. That is, the RL agent should be able to perform well in a new environment if the environment dynamics are known and the environment dynamics are encapsulated into the neural network's parameters. During the meta-RL phase, RAMP learns a neural network that predicts the environment dynamics for each environment. However, since the number of the neural network's parameters is usually high, it is challenging for the policy to use the entire set of parameters as its context. Therefore, the majority of the model's parameters are shared between all environments, and only a small set of parameters are trained separately in each environment. In that way, the environment-specific parameters represent the specific environment properties. Consequently, a general policy uses only these parameters as context and outputs actions that are suitable for that particular environment. One advantage of RAMP is that the history length used for the context extraction is not limited because the context is extracted from a global dynamic model. Additionally, the combination of model learning and RL in RAMP makes the training process more transparent since it is possible to evaluate the performance of the model learning process independently. We demonstrate the effectiveness of RAMP

This research was supported, in part, by the Ministry of Science & Technology, Israel.

[1] Department of Mechanical Engineering and Mechatronics, Ariel University, Israel

[2] Department of Computer Science, Ariel University, Israel

[3] Machine Learning Department, Carnegie Mellon University, Pittsburgh, PA, USA

gabrielh@ariel.ac.il, amos.azaria@ariel.ac.il

in several simulated experiments in Sec. V.

To summarize, the contributions of this paper are:

- Suggesting a novel method for meta-reinforcement learning.
- Presenting a multi-environment dynamic model learning method that adapts to new environments by updating only a few parameters.
- Using the dynamic model parameters directly as a context for the general policy.
- Combining model-based and model-free RL.

II. Related Work

RL has shown success in numerous domains, such as playing Atari games [1], [8], playing Go [9], and driving autonomous vehicles [3], [10]. Some are designed for one specific environment [3], [11], while others can learn to master multiple environments [2], [8]; however, many algorithms require separate training for each environment.

Several approaches were proposed to mitigate the need for long training times by using meta-RL methods. We begin by describing methods that, similarly to ours, learn a context-conditioned, general policy. However, they constructed the context vector in different ways. We note that some previous works term the different training environments "tasks" since they emphasize the changes in the reward function. However, since our work focuses on environments with different dynamics (transition functions), we use the term "environments". In [12], the environment properties are predicted by a neural network based on a fixed, small number of steps. However, this approach requires explicitly defining the representative environment properties. Moreover, it assumes that these properties can be estimated based on the immediate environmental dynamics. Rasool et al. [13] introduce TD3-context, a TD3-based RL agent that uses a recurrent neural network (RNN) to create a context vector, which receives the recent states and rewards as input. However, even though types of RNNs such as LSTM [14] and GRU [15] are designed for long-term history, in practice, the number of previous states considered by the RNN is limited [7]. Therefore, if an event that defines an environment occurs too early, the RNN will "forget" it and not provide an accurate context to the policy. In our method, RAMP, the context consists of the parameters of a global, dynamic model, which is not limited by the history length. Other approaches use the RNN directly as a policy, based on the transitions and rewards during the previous episode [6], [16], instead of creating a context vector for a general policy. These approaches are also vulnerable to this RNN memory limitation.

Finn et al. [5] proposed a different principle for meta-learning termed "Model-Agnostic Meta-Learning (MAML)." In MAML, the neural network parameters are trained such that the model will be adapted to a new environment by updating all parameters only with a low number of gradient-descent steps. However, the training process of MAML may be challenging [17]. Furthermore, MAML uses on-policy RL and therefore is unsuitable for the more sampling-efficient off-policy methods as in our approach. Nevertheless, since MAML can also be used for regression, we compare our multi-environment dynamic model learning method to MAML in Sec. V-A.

Some proposed meta-RL methods are suitable for off-policy learning [13], [18]. Meta-Q-learning (MQL) [13] updates the policy to new environments by using data from multiple previous environments stored in the replay buffer. The transitions from the replay buffer are reweighed to match the current environment. We compare our method, RAMP, to MQL in our testing environment in Sec. V-B.2.

As opposed to all these meta-RL methods, which are model-free, also model-based meta-RL methods were proposed. In model-based meta-RL, the agent learns a model that can quickly adapt to the dynamics of a new environment. Ignasi et al. [19] propose to use recurrence-based or gradient-based (MAML) online adaptation for learning the model. Similarly, Lee et al. [20] train a model that is conditioned on the encoded, previous transitions. In contrast to model-free RL, which learns a direct mapping (i.e., a policy) between the state and actions, model-based RL computes the actions by planning (using a model-predictive controller) based on the learned model. In our work, we combine the model-free and model-based approaches resulting in rapid learning of the environment dynamic model and a direct policy without the need for planning.

III. Problem Definition

We consider a set of N environments that are modeled as a Markov Decision Processes $\mathcal{M}^k = \{\mathcal{S}, \mathcal{A}, \mathcal{T}^k, \mathcal{R}\}$, $k = \{1, \ldots, N\}$. All environments share the same state space $\mathcal{S}$, action space $\mathcal{A}$, and reward function $\mathcal{R}$ and differ only by their unknown transition function $\mathcal{T}$. These N environments are randomly split into training environments $\mathcal{M}_{\text{train}}$ and testing environments $\mathcal{M}_{test}$.

The meta-RL agent is trained on the $\mathcal{M}_{\text{train}}$ environments and must adapt separately to each of the $\mathcal{M}_{\text{test}}$ environments. That is, the agent is permitted to interact with the $\mathcal{M}_{\text{train}}$ environments for an unlimited number of episodes. Then, the meta-RL agent is given only a short opportunity to interact with each of the $\mathcal{M}_{\text{test}}$ environments (e.g., a single episode, a number of time steps, etc.), and update its policy based on this interaction. Overall, the agent's goal is to maximize the average expected discounted for each of the $\mathcal{M}_{\text{test}}$ environments.

IV. RAMP

RAMP is constructed in two phases: in the first phase, a multi-environment dynamic model is learned, and in the second phase, the model parameters of the dynamic model are used as context for the multi-environment policy of the reinforcement learning agent. The following sections first describe how the multi-environment dynamic model is learned by exploiting the environments' common structure. In the second part, we describe the reinforcement learning agent.

A. Multi-Environment Dynamic Model

Attempting to approximate the transition function of each environment $\mathcal{T}^k$ by an individual neural network is likely to work well for the training environments. However, it is unlikely to generalize to the testing environments, as we have only a limited set of data points for them. However, since the environments share a common structure, it will be more efficient to train a neural network that has shared components between all the environments. Namely, we intend to train a general neural network based on the training environments such that it can be adapted to each testing environment using only a limited set of data points.

In addition, since RAMP's second phase uses the neural network's parameters' values directly as a context for the RL agent, we wish to use only a small number of parameters that should represent the properties of each specific environment dynamics. Therefore, the general neural network shares the vast part of the parameters between all environments and includes only a small set of environment-specific parameters. The values of the small set of parameters represent the specific environment dynamics since only they are unique to each environment. We show in Section 4 that it is possible to predict the unique environmental properties based only on these parameters. The environment-specific parameters are, in fact, a compact representation of each environment; therefore, they can be used by RAMP as a context vector (as described in Sec. IV-B).

The number of the environment-specific parameters should match the degree of freedom between the environments' dynamics. That is, there should be enough parameters to describe the difference between the environments. However, this number is usually not precisely known, instead, it can be tuned as a hyperparameter. Nevertheless, a greater size of this set will not prevent the succes of the multi-environment model learning as shown in Section V.

We approximate the transition function of all environments by a neural network with parameters indexed by φ, which are split to environment-specific parameters indexes $\omega \subseteq \varphi$, and to the remaining parameters indexes $\sigma = \varphi \setminus \omega$. The values of the parameters of each environment k are denoted by $\hat{\varphi}^k$ and the environment-specific parameters' values by $\hat{\omega}^k$. The shared parameters' values, which do not depend on a specific environment, are denoted by $\hat{\sigma}$. Our multi-environment dynamic model is denoted by $f_{\hat{\sigma},\hat{\omega}_k}$. The multi-environment dynamic model is given a state s and action a and outputs a prediction of the state at the following time step s' for each environment $\mathcal{M}^k$, i.e., $s' = f_{\hat{\sigma},\hat{\omega}_k}(s,a)$.

We now describe how to select ω and how to train the neural network parameters $\hat{\sigma}$ and $\hat{\omega}^k$. At first, we gather sufficient data in the form of $D^k = \{(s,a,s')\}$, for each environment k. At the beginning of the training process $\omega = \varnothing$ and $\sigma = \varphi$. The network is trained using the gradient descent algorithm to minimize the loss, which is the squared error between the predicted and real next state for each environment:

$$\mathcal{L}(D^k, \hat{\sigma}, \hat{\omega}^k) = \sum_{s,a,s' \in D^k} (s' - f_{\hat{\sigma},\hat{\omega}^k}(s,a))^2. \tag{1}$$

Initially, $\hat{\sigma}$ are trained in all environments to achieve an average model prediction:

$$\hat{\sigma} = \arg\min_{\hat{\sigma}} \sum_{k=1}^{|\mathcal{M}_{\text{train}}|} \mathcal{L}(D^k, \hat{\sigma}, \hat{\omega}^k) \tag{2}$$

After the initial training phase, the parameters ω are selected from φ by the algorithm, one-at-a-time. Intuitively, the algorithm should select parameters for ω that have the greatest impact on the difference between the environments. Therefore, at each gradient step, the gradient of the loss function $\mathcal{L}(D^k, \hat{\sigma}, \hat{\omega}^k)$ relative to $\hat{\varphi}^k$ is computed for each environment k:

$$g^k = \nabla_{\hat{\varphi}^k} \mathcal{L}(D^k, \hat{\sigma}, \hat{\omega}^k), \tag{3}$$

and the parameter with the highest variance between all gradients g^k is added to ω:

$$\omega \leftarrow \omega \cup \arg\max_{i \in \varphi \setminus \omega} \text{var}(g_i^0, g_i^1, \ldots, g_i^{|\mathcal{M}_{\text{train}}|}). \tag{4}$$

Then, the network is trained to minimize the loss function in all environments:

$$\min_{\hat{\sigma}} \sum_{k=1}^{|\mathcal{M}_{\text{train}}|} \min_{\hat{\omega}^k} \mathcal{L}(D^k, \hat{\sigma}, \hat{\omega}^k). \tag{5}$$

That is achieved by updating the environment-specific parameters $\hat{\omega}^k$ by the corresponding gradient:

$$\hat{\omega}^k \leftarrow \hat{\omega}^k - \alpha_\omega g^k, \tag{6}$$

and updating the shared parameters by the average gradient:

$$\hat{\sigma} \leftarrow \hat{\sigma} - \alpha_\sigma \frac{1}{|\mathcal{M}_{\text{train}}|} \sum_{i=0}^{|\mathcal{M}_{\text{train}}|} g^i, \tag{7}$$

where α_ω and α_σ are the learning rates. During the training, parameters continue to be added to $|\omega|$ until it reaches a predefined size n_ω. Algorithm 1 summarizes the multi-environment dynamic model learning.

Finally, at the end of the training process (after achieving a low loss value), only parameters ω need to be adjusted for a new environment to get an accurate dynamic model, while parameters σ remain constant. That is,

$$\hat{\omega}^k = \arg\min_{\hat{\omega}^k} \mathcal{L}(D^k, \hat{\sigma}, \hat{\omega}^k). \tag{8}$$

B. Reinforcement Learning With Model Parameters Context

The multi-environment dynamic model parameters, described in the previous section, are used as a context for the RL agent. That is, RAMP concatenates the environment-specific parameters' values $\hat{\omega}^k$ to the state s for training the RL agent.

Unfortunately, the environment-specific parameters $\hat{\omega}^k$ do not necessarily converge to the same value when trained in

Algorithm 1 Model learning with RAMP

Require: n_ω ▷ Number of environment-specific parameters
Require: n_{init} ▷ Number of steps for initial training
Require: n_{tot} ▷ Number of total training steps
Require: $\alpha_\omega, \alpha_\sigma$ ▷ Learning rates
Require: $\{D^0, \ldots, D^{|\mathcal{M}_{\text{train}}|}\}$ ▷ Data from the environments
  $\omega \leftarrow \varnothing$
  for $i \leftarrow 1$,number of training steps **do**
    for $\mathcal{M}^k \in \mathcal{M}_{\text{train}}$ **do**
      sample a batch of transitions $b^k \in D^k$
      $g^k \leftarrow \nabla_{\hat{\omega}^k} \mathcal{L}(b^k, \hat{\sigma}, \hat{\omega}^k)$
      $\hat{\omega}^k \leftarrow \hat{\omega}^k - \alpha_\omega g^k$
    $\hat{\sigma} \leftarrow \hat{\sigma} - \alpha_\sigma \cdot \text{avg}(g^0, \ldots, g^{|\mathcal{M}_{\text{train}}|})$
    if $|\omega| \leq n_\omega$ and $i > n_{\text{init}}$ **then**
      $\omega \leftarrow \omega \cup \arg\max_{i \in \varphi \setminus \omega} \text{var}(g_i^0, g_i^1, \ldots, g_i^{|\mathcal{M}_{\text{train}}|})$

the same environment since the amount of these parameters may be greater than the degree of freedom between the environments. That is, there may be more than one way (i.e., single parameters' values) to minimize the multi-environment dynamic model network. Therefore, the RL agent should be trained on multiple possible representations of each environment. To achieve this, the environment-specific parameters' values $\hat{\omega}^k$ are retrained every H episodes for each environment k by collecting data from a single episode with the current policy. These values are stored in Ω^k. The shared parameters, $\hat{\sigma}$, remain constant during the entire RL multi-environment training phase.

If the RL algorithm uses a replay buffer, the current environment index k is added to each tuple in addition to the standard data stored in the replay buffer (i.e., state, action, next state, reward, and done). When sampling from the replay buffer, a context vector is concatenated to each state according to the tuple's environment index k. That context vector, which is the values of the environment-specific parameters $\hat{\omega}^k$, is randomly sampled from Ω^k.

We note that RAMP can be used with any RL algorithm and also supports off-policy algorithms, which are considered to be more efficient. In this work, we use TD3 [21], which is an off-policy, actor-critic RL algorithm. The TD3 agent contains critic neural networks that estimate the action-value function. The critic is trained by minimizing the Bellman function. The actor, which is a policy represented by a neural network, aims to maximize the expected discounted infinite episode reward by maximizing the action-value function. RAMP using the TD3 algorithm is summarized in 2.

V. Experimental Evaluation

We evaluate RAMP on two domains. The first domain, a sine waves regression test, evaluates the first phase of RAMP alone, i.e., the multi-environment dynamic model learning algorithm. The second domain is the vehicle target-reaching domain, in which vehicles with different dynamics aim to reach a target. The vehicle target-reaching domain tests the complete RAMP algorithm, composed of both phases.

Algorithm 2 RAMP (using TD3)

Require: n_{tot} ▷ Number training steps
Require: $\mathcal{M}_{\text{train}}$ ▷ Training environments
Require: $\hat{\omega}^k, k = \{1, |\mathcal{M}_{\text{train}}|\}$
  Initialize critic, actor, and replay buffer $\mathbf{B}$
  Add $\hat{\omega}^k$ to Ω^k for all $k = \{1, |\mathcal{M}_{\text{train}}|\}$
  while $i < n_{\text{tot}}$ **do**
    Select environment $\mathcal{M}^k$ from $\mathcal{M}_{\text{train}}$ randomly
    Select parameters $\hat{\omega}^k$ from Ω^k randomly
    while not done **do**
      Observe s
      Execute action $a = \pi_\Phi(s, \hat{\omega}^k) + \epsilon)$, $\epsilon \sim \mathcal{N}(0, \sigma)$
      Observe new state s', reward r and done flag d
      Add (s, a, s', r, d, k) to replay buffer $\mathbf{B}$
      if i mod $H = 0$ **then**
        for all $\mathcal{M}^k \in \mathcal{M}_{\text{train}}$ **do**
          Select random parameters $\hat{\omega}^k$ from Ω^k
          Sample one episode from $\mathcal{M}^k$:
          $D^k \leftarrow \{(s_t, \pi(s_t, \hat{\omega}^k), s_{t+1})\}_{t=\{1,T\}}$
          Retrain $\hat{\omega}^k_{\text{new}}$ with D^k and add to Ω^k
      Sample random batch $b \subset \mathbf{B}$
      for all $(s, a, s', r, d, k) \in b$ **do**
        Sample $\hat{\omega}^k$ from Ω^k
        Set $s' \leftarrow (s', \hat{\omega}^k)$
        Set $s \leftarrow (s, \hat{\omega}^k)$
      Update critics using b
      if i mod $n_{\text{training}} = 0$ **then**
        Update actor using b
      $i \leftarrow i + 1$

A. Sine Waves Regression

We used a sine waves regression test similar to [5]. The multi-environment dynamic model was trained on random samples of a sine wave function with different amplitudes A and phases ϕ:

$$y = A \sin(x + \phi). \tag{9}$$

The input to the function, x, is sampled uniformly from the range $[-5, 5]$. The amplitudes of the different functions, are sampled from $A \in [0.1, 5]$, and the phases are sampled from $\phi \in [0, \pi]$. The network consists of two fully connected hidden layers, with 40 neurons in each layer and ReLU activation. The size of the environment-specific parameters is limited to 10, i.e. $n_\omega = 10$, out of a total 1761 parameters. Contrary to the dynamic model prediction, which receives an action in addition to the current state to predict the next state, in this simple sine regression problem, there is a single input and a single output. The multi-environment model was trained on 100 random sine waves with 10 samples each. It was then retrained by updating only environment-specific parameters, ω, on 10 samples of new sine waves. We compare the multi-environment dynamic model of RAMP to a small network composed of only 10 parameters trained on each new sine wave separately and to MAML [5], which updates the entire network (1761 parameters).

The multi-environment dynamic model achieved a Mean Squared Error (MSE) of 0.021. This result is slightly lower than MAML, which achieved an MSE of 0.037. Nevertheless, since MAML uses 1761 parameters, it is impractical to use them as a context for the RL agent. As expected, the network that contains only 10 parameters resulted in a very high average MSE, 19.25. When training on all sine waves together (i.e., all model's parameters are shared without environment-specific parameters), the MSE was 1.9. Figure 1 depicts the performance of the multi-environment dynamic model of RAMP on the test set.

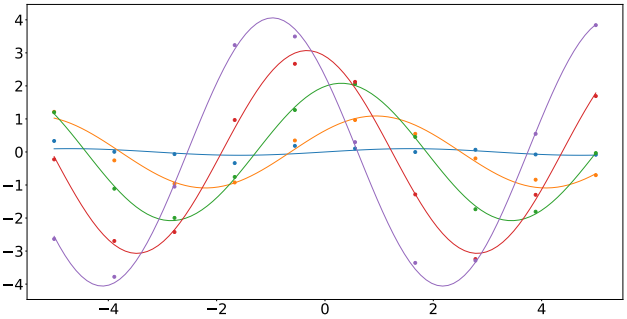


Fig. 1: Evaluation of sine functions with different amplitudes and phases. The solid lines represent the ground-truth functions, and the dots are the predictions.

B. Vehicle Target-Reaching Domain

The vehicle target-reaching domain is a simple domain that enables us to provide a precise analysis of RAMP's behavior and demonstrate the concepts behind RAMP. In this domain, an agent controls the vehicle's throttle and brake and aims to reach a target line in a minimum time. The vehicle must reach the target line at a speed of at most $v_{\max}$. The state, $s = \{v, d\}$, consists of the current vehicle's speed, v, and the distance to the target d. v ranges from 0 to 30 m/s, the distance to the target at the beginning of the episode is $d = 40$ m, and the desired maximal speed at the target line $v_{\max}$ is 5 m/s. The continuous throttle/brake action, a ranges from -1 to 1. The sampling frequency is 25 Hz. The reward function returns -0.002 at each non-terminal step. When approaching the target line with a higher speed, than $v_{\max}$ the reward is $0.01(v - v_{\max})^2$ otherwise 0.

We construct 24 vehicle target-reaching environments, split into 22 for training the multi-environment model and two for testing. All vehicles from the different environments have identical acceleration but a different deceleration, which is unknown to the agent. Specifically, the throttle command $a = [0, 1]$ causes an acceleration value $\dot{v} = [0, 42]$ m/s^2 in all environments. However, the brake command $a = [-1, 0)$ causes a deceleration value $\dot{v} = [0, 42k_a]$ that is scaled down by the braking factor k_a, which has a value between 0.1 and 1. The braking factor in the test environment is $k_a = 0.925$ and $k_a = 0.175$, which are close to the extremes of all factors.

We begin by evaluating the performance of the multi-environment dynamic model learning process, and then we evaluate the performance of the RL learning procedure. Finally, we show the adaptation process in a new environment.

1) Multi-Environment Dynamic Model Learning: The multi-environment neural network is identical to the network used for the sine wave regression. The dynamic model state consists only of the vehicle's speed, and the network predicts the difference between the current and new states. In each environment, 100 points are randomly sampled for training, and only 10 points are sampled from new environments for the adaptation process. Figure 2 shows speeds of 3 different vehicles. All accelerate at the same rate until reaching the maximal speed, and then, each vehicle applies a maximal braking action ($a = -1$), resulting in different deceleration values. The points represent the predicted speeds, and the solid lines are the real speeds. Our multi-environment dynamic model results in an MSE of 0.00029 on new environments compared to an MSE of 0.045 when trained on all environments together.

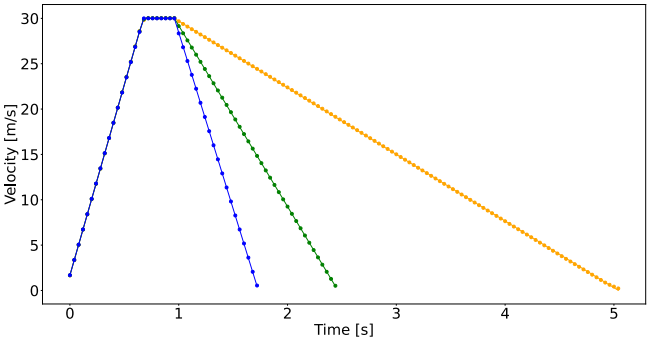


Fig. 2: Speeds prediction of different vehicles. Points are the predictions, and solid lines are the ground-truth speeds.

Recall that the environment-specific parameters ω are retrained multiple times during the RL training process as described in Sec. IV-B. Figure 3 shows the values of each of the 10 parameters for every environment, in different colors. The different parameter sets are slightly shifted along the horizontal axis. As depicted by the figure, the environment-specific parameters converged to similar values; this can be seen by the consistency of the values between the parameter sets. In addition, the figure shows that the different environments result in noticeable, different values for the first three parameters. In contrast, the remaining parameters show only a minor variance between the environments. This result seems reasonable, since not all parameters are required to determine the variance of the vehicle dynamics, which in fact, has only one degree of freedom.

Recall that in the RL training phase, the general policy must extract the properties of the environments from only the environment-specific parameters. Therefore, beyond the low loss of the prediction, we tested that it is possible to directly predict the braking factor from the environment-specific parameters. To that end, we trained a dedicated regressor, which is not used by RAMP, on 58 sets of the

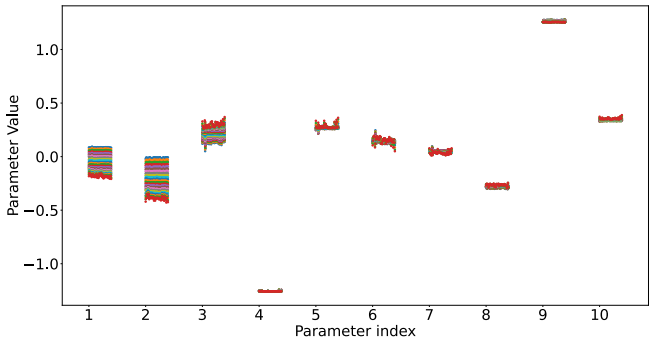


Fig. 3: Values of the 10 environment-specific parameters. Different colors represent different environments. Multiple parameter sets are shown with a shift along the horizontal axis.

trained environment-specific parameters created during the RL training process. 40 environments were used as a training set, and the 18 remaining environments were used as a test set. The regressor is composed of a neural network with two hidden layers with 100 neurons each. Figure 4 shows the prediction error distribution of the braking factor. As depicted by the figure, the regressor predicts the braking factor, which ranges from 0.1 to 1.0, with an average error of -0.0077.

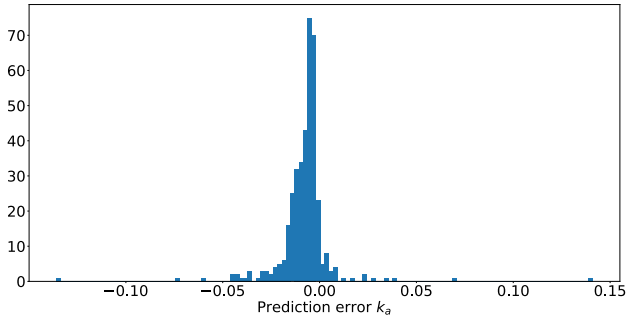


Fig. 4: Error distribution of predicting the real braking factor of environment k, based only on the environment-specific parameters $\hat{\omega}^k$.

2) Multi-Environment Reinforcement Learning: We compared RAMP to the following 3 other RL agents. The Oracle RL receives explicit information about the vehicle; that is, the braking factor is added to the state. With full knowledge of the environmental properties, the Oracle RL is expected to find a nearly optimal solution. Next, we consider a basic RL that is trained in all environments together, without any identification input, and thus cannot distinguish between different vehicles. Therefore, it is expected to learn a conservative policy that enables safe deceleration to the target line, even for the vehicle with the lowest braking capability. The third RL agent is the meta-Q-learning (MQL) algorithm [13].

The training process of all methods was repeated 5 times with different random seeds and is shown in Fig. 5. Our method's performance during the training process is comparable to the Oracle RL, achieving consistently higher episode rewards than the basic RL and MQL.

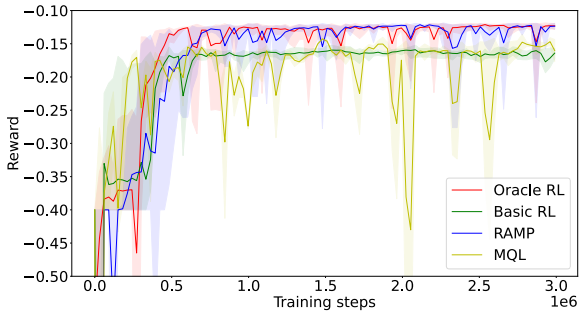


Fig. 5: Comparison between the training processes. RAMP is close to the Oracle RL.

Table I summarizes the average performance at the end of the training procedure. The table shows for all agents: the average reward in both test environments, the time to reach the target line by the vehicles with a low and high braking factor, and the average time. As depicted by the table, RAMP reaches an average reward that is very close to the Oracle's and also has a very similar average time.

	Average Reward	**Low k_a Time [s]**	**High k_a Time [s]**	**Average Time [s]**
Basic RL	-0.1656	3.160	3.376	3.268
Oracle RL	-0.1226	3.0	1.976	2.488
RAMP	-0.1230	3.048	2.04	2.544
MQL	-0.1510	3.18	2.62	2.90

TABLE I: The performance of all RL agents on the test environments. The table shows for each agent: 1. The loss, averaged over the test episodes following the 5 separate training processes; 2. The average time achieved by the vehicle with the low braking factor; 3. The average time achieved by the vehicle with the high braking factor; 4. The average between these times.

Next, we analyze the speed profiles of different vehicles driven by policies trained by the different RL agents. The speed profile of the basic RL, Oracle RL, RAMP, and MQL are shown in Figures 6a, 6b, 6c, and 6d respectively. The orange lines represent speed profiles of vehicles with a high braking factor $k_a = 0.925$, and the blue lines represent low braking factors $k_a = 0.175$. These values are close to the extremes of the braking factor range to demonstrate the difference between the environments. The bold columns represent the maximal permitted speed at the target for each of the two environments. As depicted by Fig. 6a, the basic RL begins to brake on both vehicles at the same point in time. This happens because the agent cannot know if the vehicle has a higher braking capability that allows braking later or not, which leads to a conservative policy. As shown in Fig. 6b, the Oracle RL begins braking on time in both environments and arrives at the destination at the required maximum target speed. As shown in Fig. 6c,

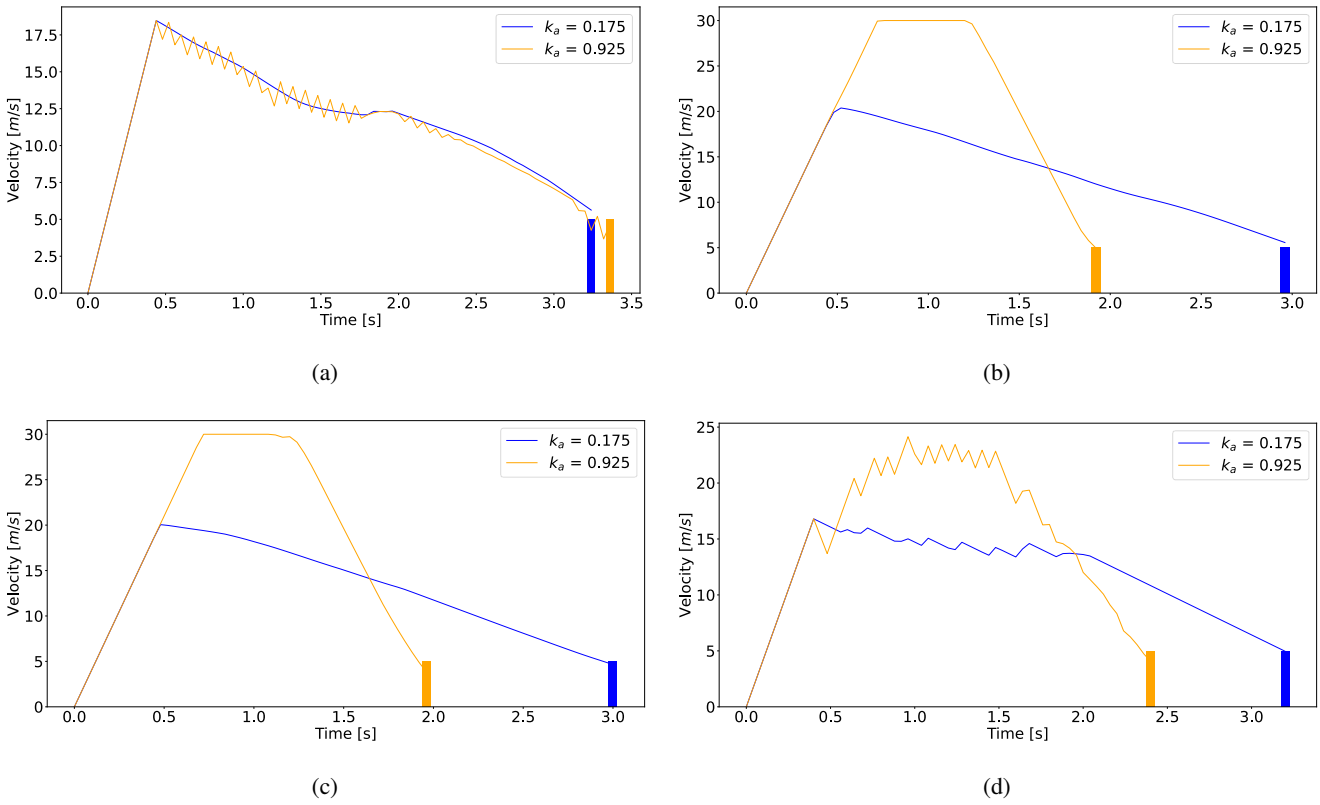


Fig. 6: Speed profiles that were achieved by our method and the other agents. Blue - low braking factor, orange - high braking factor, bold line - the maximal desired speed at the target. (a) The basic RL agent resulted in a conservative solution. (b) The Oracle RL agent achieved optimal speed profiles. (c) Our agent, RAMP, achieves similar optimal results without prior knowledge about the vehicle dynamics. (d) MQL resulted in sub-optimal results compared to RAMP.

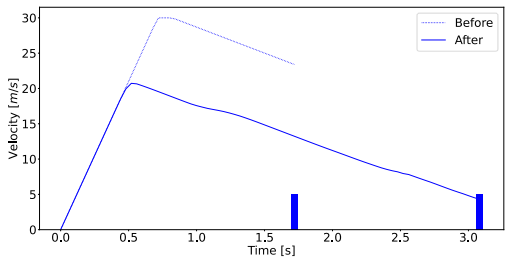
RAMP results in a similar speed profile as the Oracle RL. However, unlike the Oracle agent, RAMP does not receive any explicit information about the environment; instead, it learns this information from the trajectory sampled during one episode. Figure 6d illustrates that the MQL agent can distinguish between the vehicles' braking differences because the vehicle with the higher braking factor is allowed to gain more speed. However, MQL's speed profile is not as good as RAMP's since the MQL agent does not accelerate and decelerate at the maximal values, therefore resulting in longer driving times.

To conclude the evaluation of RAMP's performance in the target-reaching domain, we analyze RAMP's adaptation process. As opposed to the Oracle RL, which is given the braking factor information, RAMP must learn it from the driving experience. That is, in the first episode, RAMP collects data points, and the environment-specific parameters are trained on it; in the second episode, RAMP drives the vehicle with the updated context. Figure 7 shows the speed profile of a vehicle during two subsequent episodes for the low braking factor vehicle (Fig. 7a) and for the high breaking-factor (Fig. 7b). As depicted by Fig. 7a, in the first episode (represented by the dashed line), the vehicle brakes too late and therefore crosses the target line at too high a speed. In the second episode (represented by a solid line), the vehicle brakes earlier and cross the target line at a speed that is within the speed limit. Similarly, for the vehicle with a higher braking factor, RAMP learns that the vehicle can brake later. Therefore, in the second episode, it crosses the finish line earlier than in the first episode.

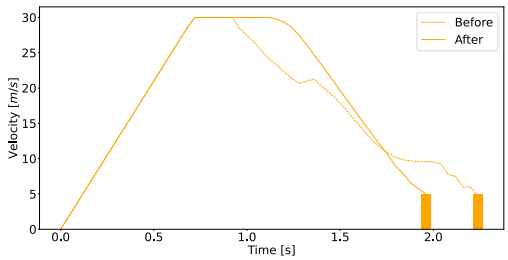
VI. Conclusions and Future Work

This paper presented RAMP, a novel meta-reinforcement learning algorithm. RAMP is constructed in two phases: learning a multi-environment dynamic model and training a general reinforcement learning policy that uses the model parameters as context. The multi-environment dynamic model is trained on data from multiple environments. The shared parameters are updated by the average gradient computed from the loss resulting from all environments, and the environment-specific parameters are trained separately on data from each environment. The low number of environment-specific parameters allows direct use of them as context for the general policy. That general policy is trained by TD3, an actor-critic, off-policy RL algorithm.

We evaluated the performance of RAMP in simulated experiments. First, we tested the multi-environment dynamic model performance by a sine-wave regression test which we show to achieve a slightly lower loss compared to MAML



(a)



(b)

Fig. 7: RAMP evaluation during two subsequent episodes. The dashed line represents the speed profile used for collecting data in the first episode and the solid line in the second. (a) Low braking factor: the agent learned that this vehicle must brake earlier. (b) High braking factor: the agent learned that this vehicle can brake later.

[5]. Then, we tested RAMP in a simple driving domain where every vehicle had a different deceleration rate. We showed that RAMP achieved similar performance to an Oracle RL agent, which is provided with full knowledge of the environment properties.

In future work, we plan to test RAMP in more challenging domains, such as controlling the steering of an autonomous vehicle and following a given path. Recall that RAMP assumes that the environments differ by their dynamics and not by their reward function, while most previous works consider the opposite. In order to adapt to environments that also differ by their reward functions, a future extension can be to learn a reward prediction function and uses its parameters as a context in addition to the multi-environment model parameters.

References

[1] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, *et al.*, "Human-level control through deep reinforcement learning," *nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[2] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, *et al.*, "A general reinforcement learning algorithm that masters chess, shogi, and go through self-play," *Science*, vol. 362, no. 6419, pp. 1140–1144, 2018.

[3] F. Fuchs, Y. Song, E. Kaufmann, D. Scaramuzza, and P. Dürr, "Super-human performance in gran turismo sport using deep reinforcement learning," *IEEE Robotics and Automation Letters*, vol. 6, no. 3, pp. 4257–4264, 2021.

[4] G. Hartmann, Z. Shiller, and A. Azaria, "Deep reinforcement learning for time optimal velocity control using prior knowledge," in *2019 IEEE 31st International Conference on Tools with Artificial Intelligence (ICTAI)*. IEEE, 2019, pp. 186–193.

[5] C. Finn, P. Abbeel, and S. Levine, "Model-agnostic meta-learning for fast adaptation of deep networks," in *International conference on machine learning*. PMLR, 2017, pp. 1126–1135.

[6] Y. Duan, J. Schulman, X. Chen, P. L. Bartlett, I. Sutskever, and P. Abbeel, "Rl^2: Fast reinforcement learning via slow reinforcement learning," *arXiv preprint arXiv:1611.02779*, 2016.

[7] A. Graves, G. Wayne, and I. Danihelka, "Neural turing machines," *arXiv preprint arXiv:1410.5401*, 2014.

[8] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," *arXiv preprint arXiv:1312.5602*, 2013.

[9] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, *et al.*, "Mastering the game of go with deep neural networks and tree search," *nature*, vol. 529, no. 7587, pp. 484–489, 2016.

[10] B. R. Kiran, I. Sobh, V. Talpaert, P. Mannion, A. A. A. Sallab, S. Yogamani, and P. Pérez, "Deep reinforcement learning for autonomous driving: A survey," *IEEE Transactions on Intelligent Transportation Systems*, vol. 23, no. 6, pp. 4909–4926, 2022.

[11] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, *et al.*, "Mastering the game of go without human knowledge," *nature*, vol. 550, no. 7676, pp. 354–359, 2017.

[12] W. Yu, J. Tan, C. K. Liu, and G. Turk, "Preparing for the unknown: Learning a universal policy with online system identification," in *Robotics: Science and Systems XIII, Massachusetts Institute of Technology, Cambridge, Massachusetts, USA, July 12-16, 2017*, 2017. [Online]. Available: http://www.roboticsproceedings.org/rss13/p48.html

[13] R. Fakoor, P. Chaudhari, S. Soatto, and A. J. Smola, "Meta-q-learning," in *International Conference on Learning Representations*, 2020. [Online]. Available: https://openreview.net/forum?id=SJeD3CEFPH

[14] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[15] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, "Empirical evaluation of gated recurrent neural networks on sequence modeling," *arXiv preprint arXiv:1412.3555*, 2014.

[16] J. X. Wang, Z. Kurth-Nelson, H. Soyer, J. Z. Leibo, D. Tirumala, R. Munos, C. Blundell, D. Kumaran, and M. M. Botvinick, "Learning to reinforcement learn," in *CogSci*, 2017. [Online]. Available: https://mindmodeling.org/cogsci2017/papers/0252/index.html

[17] A. Antoniou, H. Edwards, and A. Storkey, "How to train your MAML," in *International Conference on Learning Representations*, 2019. [Online]. Available: https://openreview.net/forum?id=HJGven05Y7

[18] K. Rakelly, A. Zhou, C. Finn, S. Levine, and D. Quillen, "Efficient off-policy meta-reinforcement learning via probabilistic context variables," in *International conference on machine learning*. PMLR, 2019, pp. 5331–5340.

[19] I. Clavera, A. Nagabandi, S. Liu, R. S. Fearing, P. Abbeel, S. Levine, and C. Finn, "Learning to adapt in dynamic, real-world environments through meta-reinforcement learning," in *International Conference on Learning Representations*, 2019. [Online]. Available: https://openreview.net/forum?id=HyztsoC5Y7

[20] K. Lee, Y. Seo, S. Lee, H. Lee, and J. Shin, "Context-aware dynamics model for generalization in model-based reinforcement learning," in *Proceedings of the 37th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, vol. 119. PMLR, 13–18 Jul 2020, pp. 5757–5766. [Online]. Available: https://proceedings.mlr.press/v119/lee20g.html

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

Chapter 3

Discussion and Conclusions

This section reviews the main results presented in the papers and concludes this work.

One of the interesting results accomplished by the racing controller presented in the first paper [26] is its ability to balance competitiveness and safety. This was demonstrated by driving 6 vehicles for 30 laps each, which accounts for 180 laps over 720 km, without *any* collision, and in the IAC simulation race, where our controller drove over a total of 17 laps with no collision.

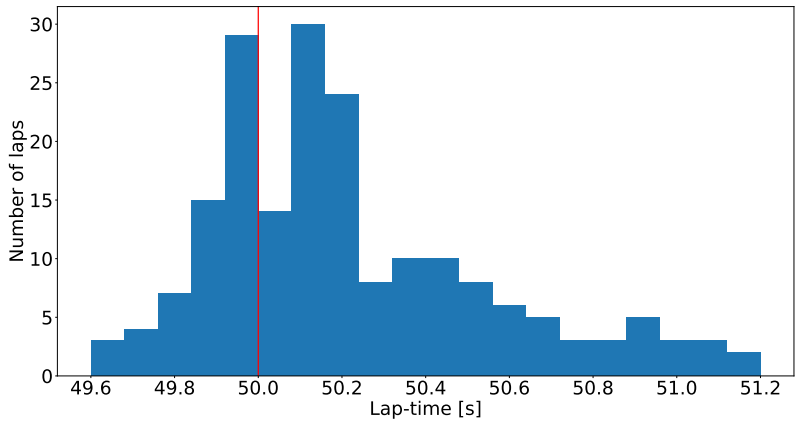


FIGURE 3.1: Lap times distribution of all 6 vehicles over 30 laps. The red line represents the solo-lap time.

This notable result did not diminish the racing controller's competitiveness, as demonstrated by the 6-vehicle race, where the average lap-time of all vehicles was only 0.25 seconds greater than the optimal solo-lap time, as shown in Fig. 3.1. This implies that the vehicles were running in close proximity to each other, which demonstrates the high competitiveness of our controller.

Despite the unprecedented achievements of autonomous vehicles in the Indy Autonomous Challenge [31], human drivers still defeat autonomous racing. One reason for the superiority of human drivers is that humans' actions are based on complex ,conscious and subconscious decisions that are challenging to model in directly-designed algorithms such as the described approach.

The subsequent papers explored the use of learning-based approaches that eliminate the need to explicitly define the controller, while achieving higher performance. The learning methods in these papers were demonstrated on maximizing speed and path following along a given path. Figure 3.2, from the second paper [28], presents the normalized average velocity along the path during training. All results were normalized relative to the directly-designed velocity controller (VOD), which appears as a horizontal line at 1.0.

At the beginning of the training process, the RL agent (REVO) drove very slowly and achieved the same performance as VOD after about $40,000$ training updates. REVO+A achieved the same performance as VOD from the beginning, which implies that REVO+A converges much faster than REVO because REVO+A uses VOD as a baseline.

When the training process continues, the policies learned by all methods increase the vehicle's velocity, compared to using VOD, by about 10%. This performance improvement is expected because VOD uses a simplified vehicle model.

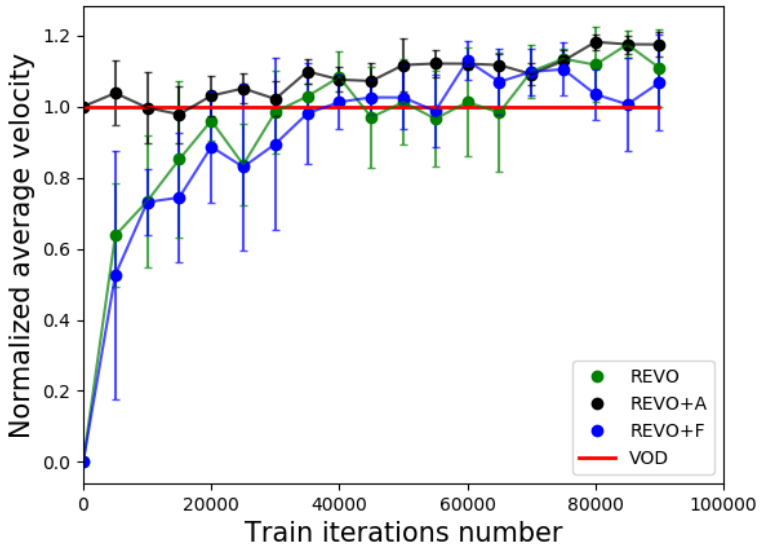
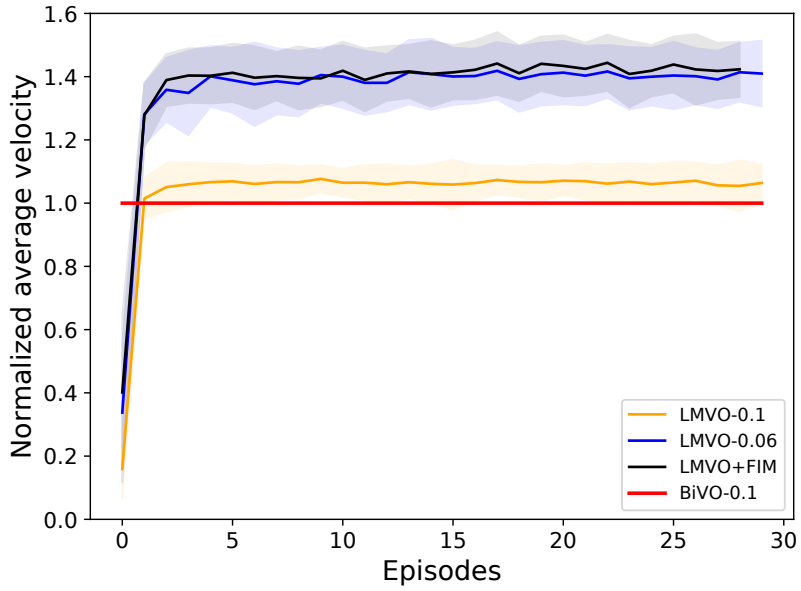


FIGURE 3.2: The average velocity on 100 random paths, measured at every 5000 training steps (normalized relative to VOD), on 5 different training processes. The bars represent the standard deviation between the training processes.

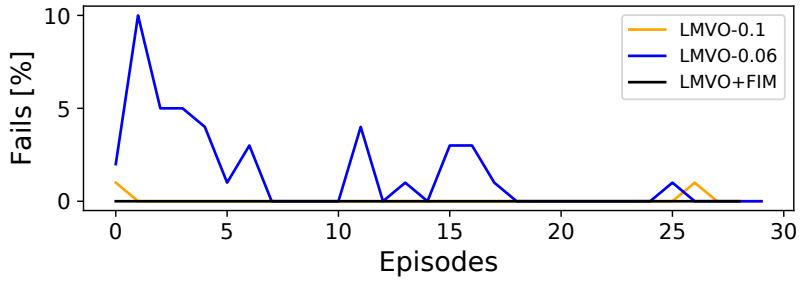
One of the challenges of RL is the long training times. The third paper [29] presented a model-based RL approach (LMVO) which is more efficient compared to the model-free RL algorithm (REVO) used in the previous paper. Ensuring safety is another challenge of learning-based approaches. Therefore, a safety module (FIM) that is based on a direct solution is presented in this paper.

Figure 3.3 compares three different methods: LMVO with a safety factor of 0.1 (LMVO-0.1), LMVO with a safety factor 0.06 (LMVO-0.06) and LMVO+FIM with a safety factor of 0.05.

LMVO-0.1 achieves lower velocity than LMVO-0.06 and LMVO+FIM but also has a lower failure rate than LMVO-0.06. LMVO+FIM maximizes the velocity without failing and achieves a slightly higher velocity than LMVO-0.06. These results validate our assumption that LMVO+FIM can learn how to maximize the velocity without failing and can achieve a velocity that is not lower than a not-as-safe driving policy (i.e., LMVO-0.06).



(A)



(B)

FIGURE 3.3: Comparison between model-based RL methods during 100 learning processes. (A) Normalized average velocity (higher is better). (B) Failure rate (lower is better). Note that when using the stabilization policy LMVO+FIM, there are no failures during the entire training process.

As depicted by Fig. 3.3b, an important advantage of LMVO+FIM over the other methods is that it maintains safety also during the beginning of the training process,

where the learned model may be inaccurate.

All these model-based RL methods converged to high-performance solutions in only a few episodes compared to the 200 episodes required for the model-free RL method (REVO). However, the model-based approach requires planning the action using the learned dynamic model in contrast to the immediate solution that results from the model-free RL policy.

A meta-RL algorithm (RAMP) was proposed in the fourth paper [30] to eliminate the need for directly developing a solution for speeding up the RL learning process or ensuring safety. RAMP was validated in a target-reaching problem where each vehicle has a different, unknown braking capability. It was compared to the following 3 other RL agents in Fig. 3.4. The Oracle RL received explicit information about the vehicle. With full knowledge of the environmental properties, it is expected to find a nearly optimal solution. Next, a basic RL that was trained in all environments together, without any identification input, and thus cannot distinguish between different vehicles. Therefore, it is expected to learn a conservative policy. The third RL agent is the meta-Q-learning (MQL) algorithm [32]. Our method's performance during the training process is comparable to the Oracle RL, achieving consistently higher episode rewards than the basic RL and MQL.

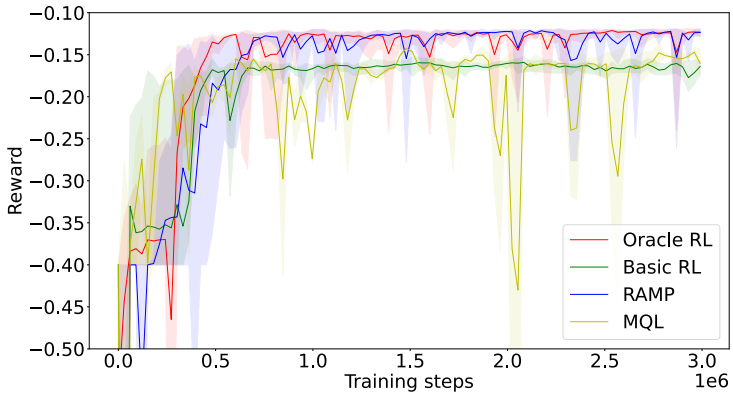


FIGURE 3.4: Comparison between the training processes. RAMP is close to the Oracle RL.

This doctoral thesis presented several algorithms for time-optimal velocity control and competitive driving that use both the direct-design approach, as well as the learning-based approach. The first paper described a directly designed algorithm for autonomous racing. The second and third papers used directly designed solutions for speeding up and ensuring the safety of an RL agent. The fourth paper described a new approach for meta-RL.

Bibliography

[1] Chris J Ostafew et al. “Speed daemon: experience-based mobile robot speed scheduling”. In: *2014 Canadian Conference on Computer and Robot Vision*. IEEE. 2014, pp. 56–62.

[2] Mohamed Elbanhawi, Milan Simic, and Reza Jazar. “In the passenger seat: investigating ride comfort measures in autonomous cars”. In: *IEEE Intelligent Transportation Systems Magazine* 7.3 (2015), pp. 4–17.

[3] Moshe Mann and Zvi Shiller. “Dynamic stability of off-road vehicles: Quasi-3D analysis”. In: *Robotics and Automation, 2008. ICRA 2008. IEEE International Conference on*. IEEE. 2008, pp. 2301–2306.

[4] Florent Altché, Philip Polack, and Arnaud de La Fortelle. “A simple dynamic model for aggressive, near-limits trajectory planning”. In: *IV, 2017 IEEE*. 2017, pp. 141–147.

[5] Jeong hwan Jeon et al. “Optimal motion planning with the half-car dynamical model for autonomous high-speed driving”. In: *American Control Conference (ACC), 2013*. IEEE. 2013, pp. 188–193.

[6] Toni Petrinić, Mišel Brezak, and Ivan Petrović. “Time-optimal velocity planning along predefined path for static formations of mobile robots”. In: *International Journal of Control, Automation and Systems* (2017).

[7] José L. Vázquez et al. “Optimization-Based Hierarchical Motion Planning for Autonomous Racing”. In: *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2020, pp. 2397–2403. DOI: 10.1109/IROS45743.2020.9341731.

[8] Juraj Kabzan et al. “Amz driverless: The full autonomous racing system”. In: *Journal of Field Robotics* 37.7 (2020), pp. 1267–1294.

[9] Alexander Liniger, Alexander Domahidi, and Manfred Morari. “Optimization-based autonomous racing of 1: 43 scale RC cars”. In: *Optimal Control Applications and Methods* 36.5 (2015), pp. 628–647.

[10] Paolo Fiorini and Zvi Shiller. “Motion planning in dynamic environments using velocity obstacles”. In: *The International Journal of Robotics Research* 17.7 (1998), pp. 760–772.

[11] Xiaohui Li et al. "Real-time trajectory planning for autonomous urban driving: Framework, algorithms, and verifications". In: *IEEE/ASME Transactions on mechatronics* 21.2 (2015), pp. 740–753.

[12] Chris Urmson et al. "Autonomous driving in urban environments: Boss and the urban challenge". In: *Journal of Field Robotics* 25.8 (2008), pp. 425–466.

[13] Johannes Betz et al. "Autonomous Vehicles on the Edge: A Survey on Autonomous Vehicle Racing". In: *IEEE Open Journal of Intelligent Transportation Systems* 3 (2022), pp. 458–488. DOI: 10.1109/OJITS.2022.3181510.

[14] Alexander Wischnewski et al. "A Tube-MPC Approach to Autonomous Multi-Vehicle Racing on High-Speed Ovals". In: *IEEE Transactions on Intelligent Vehicles* (2022), pp. 1–1. DOI: 10.1109/TIV.2022.3169986.

[15] Peter R Wurman et al. "Outracing champion Gran Turismo drivers with deep reinforcement learning". In: *Nature* 602.7896 (2022), pp. 223–228.

[16] Volodymyr Mnih et al. "Playing atari with deep reinforcement learning". In: *arXiv preprint arXiv:1312.5602* (2013).

[17] Mariusz Bojarski et al. "End to end learning for self-driving cars". In: *arXiv preprint arXiv:1604.07316* (2016).

[18] Maximilian Jaritz et al. "End-to-end race driving with deep reinforcement learning". In: *2018 IEEE,ICRA*. 2018, pp. 2070–2075.

[19] Tobias Glasmachers. "Limits of end-to-end learning". In: *arXiv preprint arXiv:1704.08305* (2017).

[20] Javier Garcıa and Fernando Fernández. "A comprehensive survey on safe reinforcement learning". In: *Journal of Machine Learning Research* 16.1 (2015), pp. 1437–1480.

[21] Zvi Shiller and Y-R Gwo. "Dynamic motion planning of autonomous vehicles". In: *IEEE Transactions on Robotics and Automation* 7.2 (1991), pp. 241–249.

[22] Jeong hwan Jeon et al. "Optimal motion planning with the half-car dynamical model for autonomous high-speed driving". In: *2013 American Control Conference*. 2013, pp. 188–193. DOI: 10.1109/ACC.2013.6579835.

[23] Volodymyr Mnih et al. "Human-level control through deep reinforcement learning". In: *nature* 518.7540 (2015), pp. 529–533.

[24] David Silver et al. "A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play". In: *Science* 362.6419 (2018), pp. 1140–1144.

[25] Florian Fuchs et al. "Super-Human Performance in Gran Turismo Sport Using Deep Reinforcement Learning". In: *IEEE Robotics and Automation Letters* 6.3 (2021), pp. 4257–4264. DOI: 10.1109/LRA.2021.3064284.

[26] Gabriel Hartmann, Zvi Shiller, and Amos Azaria. "Competitive Driving of Autonomous Vehicles". In: *IEEE Access* 10 (2022), pp. 111772–111783. DOI: 10.1109/ACCESS.2022.3215984.

[27] Omead Amidi and Chuck E Thorpe. "Integrated mobile robot control". In: *Mobile Robots V*. Vol. 1388. SPIE. 1991, pp. 504–523.

[28] Gabriel Hartmann, Zvi Shiller, and Amos Azaria. "Deep Reinforcement Learning for Time Optimal Velocity Control using Prior Knowledge". In: *2019 IEEE 31st International Conference on Tools with Artificial Intelligence (ICTAI)*. 2019, pp. 186–193. DOI: 10.1109/ICTAI.2019.00034.

[29] Gabriel Hartmann, Zvi Shiller, and Amos Azaria. "Model-Based Reinforcement Learning for Time-Optimal Velocity Control". In: *IEEE Robotics and Automation Letters* 5.4 (2020), pp. 6185–6192. DOI: 10.1109/LRA.2020.3012128.

[30] Gabriel Hartmann and Amos Azaria. *Meta-Reinforcement Learning Using Model Parameters*. 2022. DOI: 10.48550/ARXIV.2210.15515.

[31] *Indy Autonomous Challenge*. https://www.indyautonomouschallenge.com/. 2021.

[32] Rasool Fakoor et al. "Meta-Q-Learning". In: *International Conference on Learning Representations*. 2020. URL: https://openreview.net/forum?id=SJeD3CEFPH.

תקציר

נהיגה אוטונומית כוללת תתי נושאים כגון חישה, תכנון תנועה ובקרה. ישנם מטרות שונות לתכנון התנועה, שכוללים שיפור יעילות האנרגיה וקיצור זמן הנסיעה. זמן הנסיעה מקוצר (כלומר, נסיעה קרוב למהירות האופטימלית בזמן) על ידי מיקסום המהירות, שיכולה להיות מוגבלת על ידי אילוצים שונים כגון חוקי תנועה, מגבלות החיישנים, ונוחות הנסיעה. על כל פנים, האילוץ הקריטי ביותר שצריכים להתחשב בו בכל מצב הוא היציבות הדינמית. הכוונה ביציבות דינמית היא אילוצים שתלויים במהירות הרכב כגון מניעת התהפכות או החלקה. העבודה הזאת מתמקדת בתנועה אופטמלית בזמן בסביבות שונות.

קיימות שתי גישות לפתרון פעיות מהסוג הזה: פתרון ישיר, שבו הפתרון מבוסס על הבנת הבעיה על ידי מומחה, ולמידת מכונה שמאפשר ללמוד את הפתרון באמצעות מידע באופן אוטומטי. העבודה הזאת מציעה כמה אלגוריתמים עבור נהיגה אוטונומית שמדגימים את התועלת של פתרונות ישירים בפני עצמן והשילוב עם שיטות מבוססות למידה כדי להאיץ את תהליך הלמידה ולשמור על הבטיחות של הפתרון הנלמד.

אוניברסיטת אריאל בשומרון

שיפור למידת חיזוקים באמצעות שילוב אלגוריתמים שפותחו באופן ישיר עבור נהיגה אוטונומית

חיבור זה הוגש כחלק מדרישות התואר
"דוקטור לפילוסופיה"

מאת

גבריאל הרטמן

עבודה זו נכתבה בהנחיית פרופ' צבי שילר ופרופ' עמוס עזריה

מוגש לסנאט אוניברסיטת אריאל בשומרון

12.2022