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

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^*.

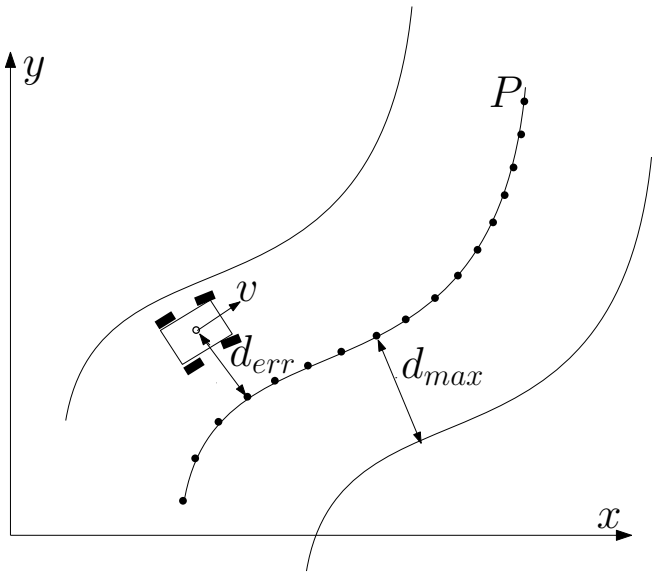


Figure 1: A vehicle, tracking path P within the allowed margin d_{max}.

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.

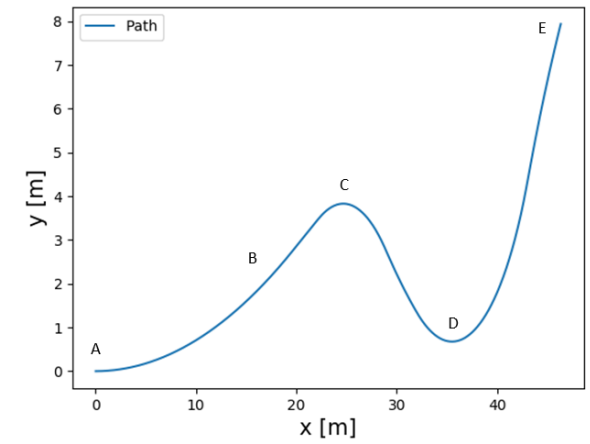
The algorithm first computes the maximal velocity profile along the path, termed the "velocity limit curve", which represents the highest vehicle speeds, above which at least one of the vehicle's dynamic constraints is violated, i.e. the vehicle either rolls-over, slides, or looses contact with the ground. 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].

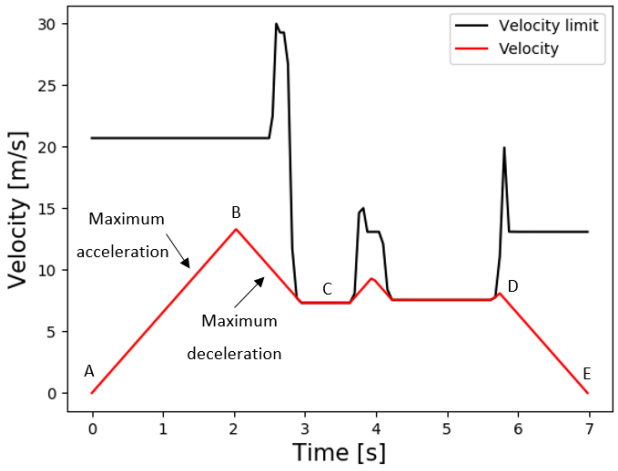
Fig. 2a shows a given planar curved path. The velocity limit curve along that path is shown in black in Fig. 2b. Note the drops in the velocity limit caused by the sharp curves C and D along the path. Clearly, moving at high speeds along these curves might cause the vehicle to either slide or rollover (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.

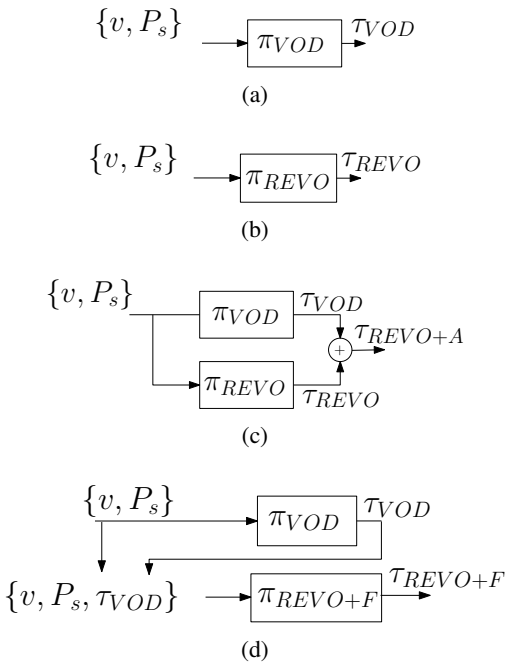


Figure 3: π is a policy, τ is an action, $\{v, P_s\}$ is the state. (a) VOD: Direct planning (b) REVO: DDPG based learning. (c) REVO + A: combines the actions of VOD with REVO. (d) REVO + F: adds the action output of VOD as a feature in the state space of REVO.

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].

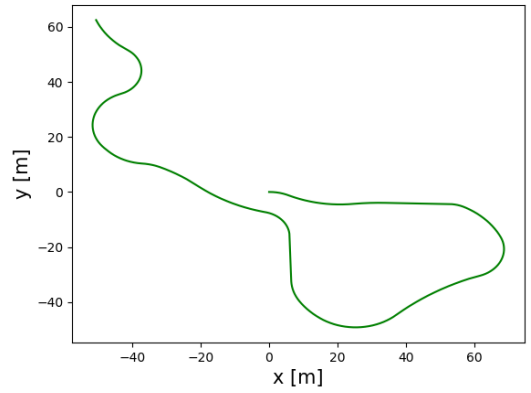
B. Experiment Protocol

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.

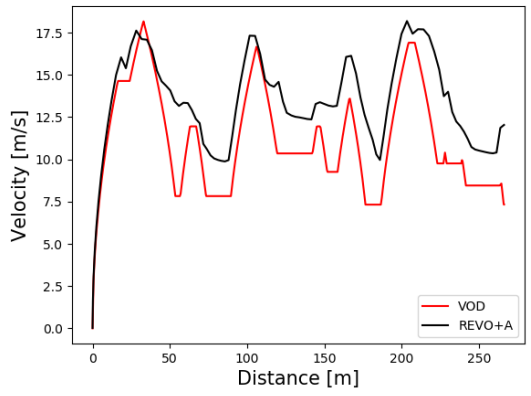
The agent's goal is to maximize its average velocity. Since the average velocity during the failed episodes was usually higher than the average velocity during successful episodes, we excluded failed episodes when presenting the average velocity of each method.

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

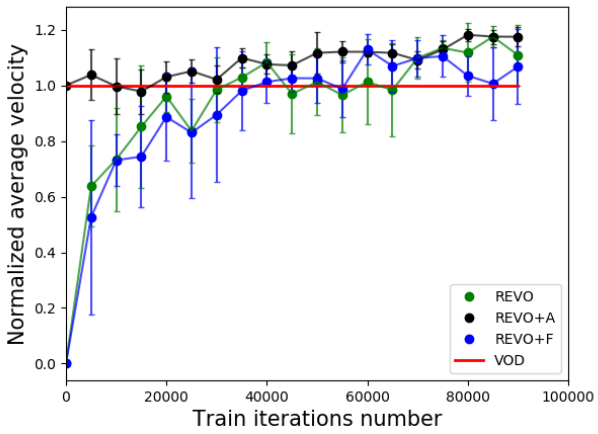


Figure 5: Average velocity on 100 random paths, measured at every 5000 training steps (normalized with respect to VOD), on 5 different training processes. The bars represent the variance between the training processes.

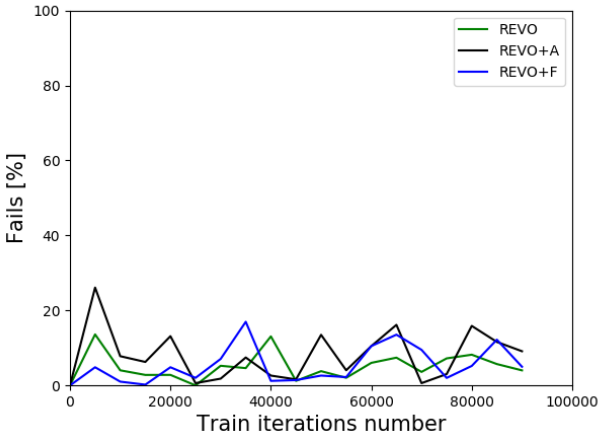


Figure 6: The failure rate of all methods during training.

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.

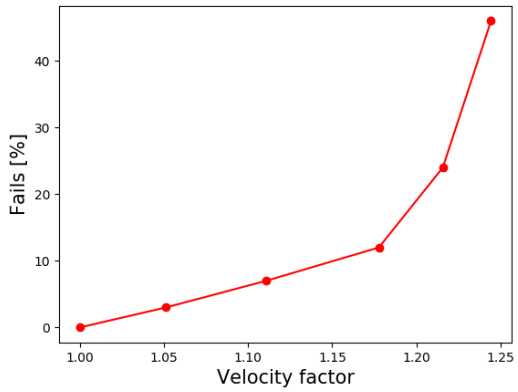


Figure 7: Failure rate of VOD when scaling up the VOD velocity

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

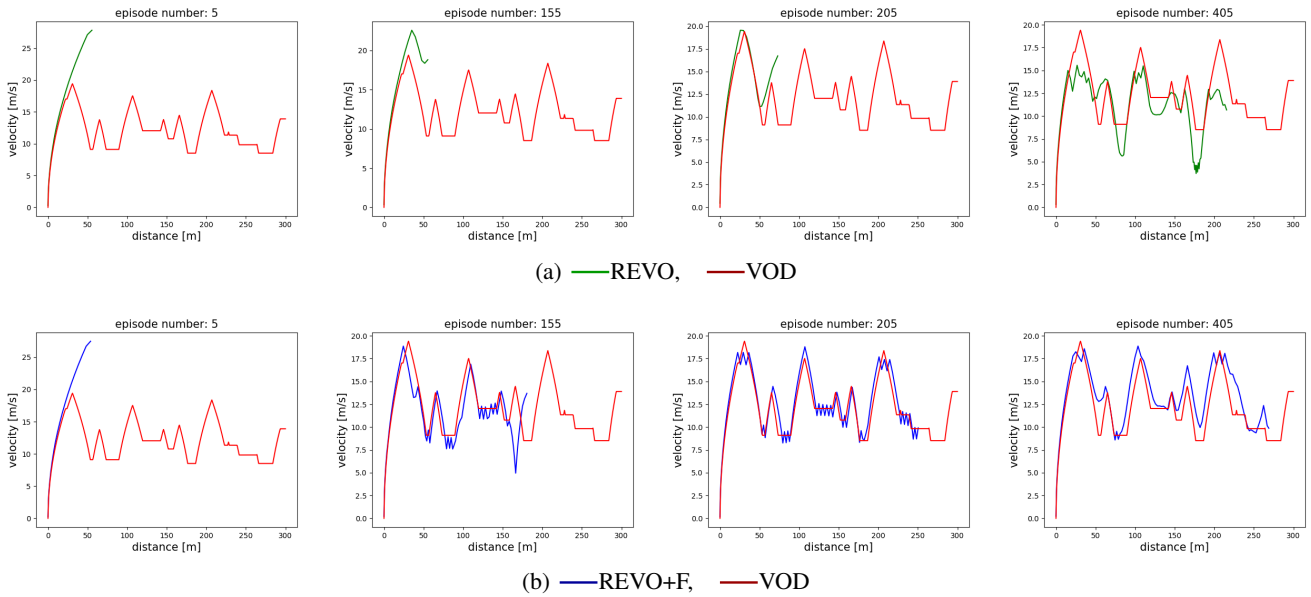


Figure 8: A comparison between learning progress of REVO and REVO+F. In episode 5, both methods accelerate until a roll-over occurs. In episode 155, REVO+F started to imitate the VOD controller, while REVO shows a very little progress. In episode 205, D-VOL shows almost full imitation of VOD velocity,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