Deep Reinforcement Learning for Time Optimal Velocity Control using Prior Knowledge

Gabriel Hartmann[1], Zvi Shiller[2], Amos Azaria[3]

arXiv:1811.11615v2 [cs.RO] 12 Mar 2019

***Abstract*— While autonomous navigation has recently gained great interest in the field of reinforcement learning, only a few works in this field have focused on the time optimal velocity control problem, i.e. controlling a vehicle such that it travels at the maximal speed without becoming dynamically unstable. Achieving maximal speed is important in many situations, such as emergency vehicles traveling at high speeds to their destinations, and regular vehicles executing emergency maneuvers to avoid imminent collisions. 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 learning 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 addresses the time optimal velocity control along a known path. Driving at the time optimal velocity would yield the shortest travel time possible along the path while ensuring that the vehicle does not rollover or slide at any point along the path.

Driving at the time optimal velocity is important when motion time is critical, e.g. emergency vehicles such as ambulances, firefighters vehicles attempting to reach their destinations at high speeds, and common vehicles attempting to execute emergency maneuvers to avoid collisions.

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, an accurate vehicle dynamic model is required to ensure that the vehicle does not lose its dynamic stability during motion at the time optimal speeds at any point along the path. Since the consideration of a detailed vehicle dynamic model is impractical for online computation, a simplified model is usually used to compute the vehicle's velocity profile [1]. It is therefore useful to use Reinforcement Learning to bridge the gap between the approximate and the actual vehicle model.

[1]Department of Computer Science and department of Mechanical Engineering and Mechatronics, Ariel University, Ariel 40700, Israel gabriel.hartmann@msmail.ariel.ac.il

[2]Department of Mechanical Engineering and Mechatronics, Ariel University, Ariel 40700, Israel shiller@ariel.ac.il

[3]Department of Computer Science, Ariel University, Ariel 40700, Israel amos.azaria@ariel.ac.il

While a large body of work was published on reinforcement learning for autonomous vehicles the majority have focused on perception and steering [2, 3, 4], and only a few have focused on velocity control [5, 6, 7]. We are not aware of works focusing on time optimal velocity control.

Rosolia and Borrelli [6] use model-predictive control to drive a race car at high speeds along a given track. The controller is tuned iteratively to reduce total motion time.

Several works in the context of autonomous driving use prior knowledge on the task as a baseline for the learning process. For example, imitation learning uses demonstrations to teach an agent to steer a vehicle along arbitrary paths [2]. Lefèvre et al. [7] learn driving styles from a driver's demonstrations. While imitation learning cannot outperform the demonstrator's performance, it can serve as a baseline for further improvements using reinforcement learning [8, 9].

Another well-studied approach, known as reward shaping, encodes the knowledge in a reward function, which speeds up the learning process [10].

This paper proposes a reinforcement learning method for driving a vehicle at the time optimal speed along a given path. It learns the acceleration (and deceleration) that maximizes vehicle speeds along the path, without losing its dynamic stability (rollover). Steering is not learned, but is rather determined directly by the path following controller (pure pursuit) [11]. We then enhance the learning process, to speed up convergence, by reducing the learning process to learn only the variation from a nominal time optimal acceleration profile, computed numerically using the available vehicle model. The algorithm was implemented in simulation for a ground vehicle moving along arbitrary paths in the plane. It is shown that the synergy between these two methods outperforms each one separately.

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 learning process. 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 $v \in \mathbb{R}$, $0 \le v \le v_{\max}$. 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 [11]). The deviation of the vehicle center from the desired path is denote by d_{err}, as shown in Fig. 1.

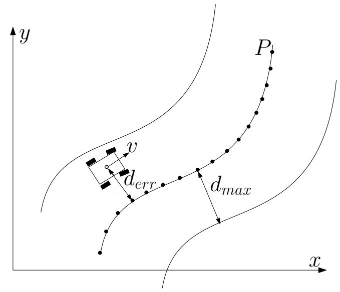


Fig. 1: A vehicle, tracking path P within the allowed margin d_{max}.

The goal of the reinforcement learning agent 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 time interval. 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^*.

III. Time Optimal Velocity Control using Reinforcement Learning

Our basic reinforcement learner is based upon "Deep Deterministic Policy Gradient" (DDPG) [12], which we adapt to the time optimal velocity control problem. We term this Reinforcement learning based Velocity Optimization REVO.

A. Deep Deterministic Policy Gradient

Deep Deterministic Policy Gradient (DDPG) [12] is an actor-critic, model-free algorithm for continuous action space A, and continuous state space S. With the assumption that the state is fully observed, the agent receives 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 the probability of ending at s_{t+1} when being at s_t and taking action a_t. The goal of the DDPG algorithm is to learn a deterministic policy $\pi : S \to A$ (represented as a neural network) which maximizes the return from start 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 action.

The goal is to learn a policy that generates the time optimal velocity along any given path.

The learning process consists of episodes, at each the vehicle is moving along a randomly generated path from its current state. The agent is trained on a diverse set of paths to make the policy general as possible. Only paths that are kinematically feasible are considered, that is, avoiding sharp curves that exceed the vehicle's minimum turning radius. Each path P is generated by connecting short path segments of random length and curvature until reaching the desired length of P. The curvature is selected to ensure continuity between consecutive segments. This ensures that the selected path respects the vehicles steering capabilities.

At each time step t during the episode, the agent is in state s_t that consists of a limited horizon path segment and the vehicles speed v. A reward r_t is received according to the reward function that encourages the agent to drive at high speeds while preserving the vehicle stability. The episodes terminates after time T or if the vehicle became unstable.

At each time step the agent has information only on a limited horizon path segment $P_s \subseteq P$. Formally, $P_s = \{p_m, p_{m+1}, \cdots p_{m+k}\}$, where m is the index of the closest point on the path P to the vehicle, and $k \in \mathbb{N}$ is a predefined number of points.

P_s is down-sampled to reduce the dimensionality of the state. In addition, every point p in the down-sampled path $P_s^{d.s.}$ is transformed to a local vehicle reference frame at its current state. The state of the system is formally defined as $s = \{v, P_s^{d.s.}\}$.

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, it obtains 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 $a_t = \tau_i = \pi(s_t) + \eta(t)$ where $\eta(t)$ is the exploration noise.

Next we describe the direct approach to computing the time optimal velocity profile, which is based on an analytically derived vehicle model.

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 [1, 13, 14]. 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 learning 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 friction between the wheels and ground are known.

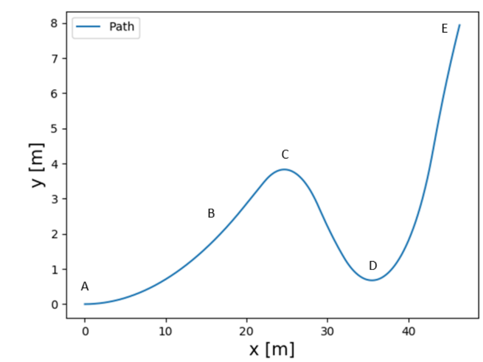
The algorithm computes the maximal velocity profile along the path for given boundary condition (initial and final speeds for the given path segment), that ensures that the vehicle stays dynamically stable at all times, the centrifugal forces required to keep the vehicle along the path stay below the limit, determined by the coefficient of friction between the wheels and ground and by the centripetal forces that might cause the vehicle to rollover.

Since the vehicle position and orientation depend only on the path geometry (as assumed), it is possible to compute the forces applied on the vehicle at any point along the path given the velocity of the vehicle at that point. The velocity limit curve is a set of velocities along the path, above which one of the dynamic constraints is violated. That is, the highest velocity at which the vehicle is dynamically stable.

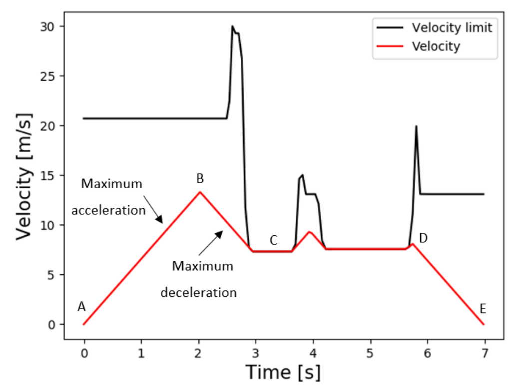
The time optimal velocity profile is computed by applying "Bang-Bang" acceleration, i.e. either maximum or minimum, at all points along the path. "Bang-bang control is known to produce the time optimal motion of second order systems [15]. The optimal velocity profile is computed by integrating forward and backwards the extreme accelerations at every point along the path so as to avoid crossing the velocity limit curve [13].

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 may cause the vehicle to either slide or rollover (which 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 final assumed boundary condition).

The velocity computed by this algorithm is used to control the vehicle along the specified path. At every time t the optimal velocity profile is computed on the limited horizon path segment P_s (as was formally defined in III, with the current vehicle velocity serving as the initial velocity for this computation. To ensure that it will be possible to stop at the end of this path segment, the target velocity of the endpoint of P_s is set to zero. The acceleration at the first step of the computed velocity profile is used as the action output of the controller. Since it is repeated every time step, and the path segment is long enough, the vehicle is not influenced by the zero target velocity.



(a)



(b)

Fig. 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

We propose a method to combine VOD, the direct velocity optimization controller and REVO, the reinforcement learning based controller. This method is based upon two concepts. The first combines VOD and REVO through the actions, it is termed REVO+A. The value of the action output from the direct planner (τ_{VOD}) is added to the value of the action output from the Reinforcement learner (τ_{REVO}) such that the action output from the REVO+A controller is $\tau_{REVO+A} = \tau_{VOD} + \tau_{REVO}$. REVO+A is illustrated in Fig. 3 (c). In this form, at the beginning of the learning process the vehicle follows the actions from the computed direct optimization (VOD), that is, at each time t, $\tau_{REVOt} \approx \tau_{VODt}$ because $\tau_{REVOt} \approx 0$ at the beginning of the training process because of the initialization method. The policy of the VOD controller (π_{VOD}) is a good initial policy, hence even at the beginning the policy π_{REVO+A} is much better than a random initialized policy as π_{REVO}. This simplifies the problem for the reinforcement learner agent, which only needs to learn how much to deviate from the direct computed optimal velocity and is not required to start from the ground up.

The second concept for combining REVO and VOD is based on combining them through the state space, we term this method REVO+F. In the REVO+F method, the action output (τ_{VOD}) from the VOD controller is added as an additional feature to the state space, that is, $s = \{\tau_{VOD}, v, P_s\}$. REVO+F is illustrated in Fig. 3 (d). An intuitive justification to this is, that the reinforcement learner has the information about τ_{VOD}, hence it can decide to act directly like τ_{VOD}, or to "understand" when to act like it and when it is better to deviate from these actions. In the results section, it will be shown that in a certain case, REVO+F, the learning process is faster, and it can be seen that the reinforcement learner acts according to τ_{VOD}. An additional improvement above REVO+A and REVO+F is adding this to methods together (REVO+FA), as illustrated in Fig. 3 (e).

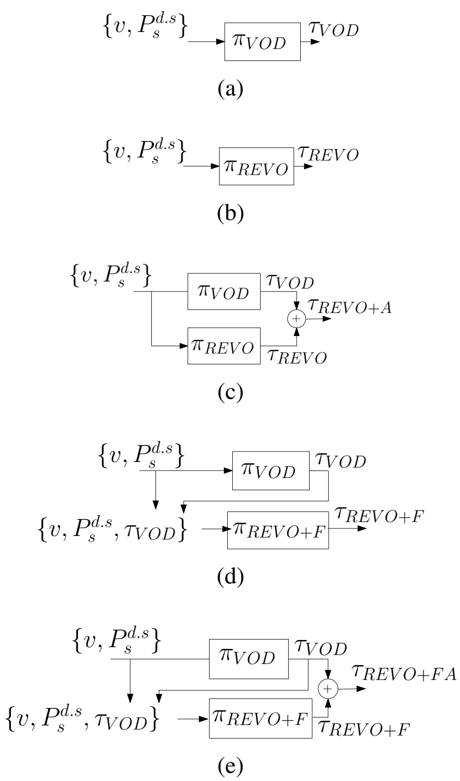


Fig. 3: π is a policy, τ is a action, $\{v, P_s^d.s.\}$ is the state. (a) VOD: Direct planning (b) REVO: DDPG based learning. (c) REVO + A: combine actions of VOD and REVO. (d) REVO + F: adding action output of VOD as feature in state for REVO. (e) REVO+FA: combine actions and add as feature.

VI. Experimental Results

A. Settings

A simulation of a four-wheel vehicle was developed using "Unity" software [16]. A video of the vehicle driving along a path at time optimal velocity is available here: [17]. The vehicle properties were set to width 2.08 [m], height 1.9 [m], length 5.1 [m], center of mass at the height of 0.94 [m], mass 3200 [Kg], and a total force produced by all wheels of 21 [KN]. The maximum velocity of the vehicle is $v_{max} = 30$ [m/s] (108 km/h). The maximum acceleration of the vehicle is 6.5 m/s^2. Acceleration and deceleration are applied to all four wheels (4x4); steering is done by the front wheels (Ackermann steering). To focus this experiment on rollover only, the friction coefficient between the wheels and the ground was set arbitrarily high (at 5).

As for the specific settings for the Reinforcement agents, a state consists of 25 points along the path ahead of the vehicle. The distance between one point to the next point is 1 [m]. ($\left|P_s^{d.s.}\right| = 25$,$dis(p_i, p_{i+1}) = 1[m]$, $p_i, p_{i+1} \in P_s^{d.s.}$, $i \in \{0, 1, \cdots, 25\}$). The state was normalized to range $[-1, 1]$ The reward function was defined as:

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

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 2 [m] ($d_{max} = 2$) from the nominal path.

All the hyper-parameters of the reinforcement learning algorithm (e.g. neural network architecture, learning rates) were set as described in [12].

B. Experiment Description

The goal of the experiment is to compare the five discussed methods: Direct, numeric-based method (VOD), DDPG based learning method (REVO), the combination of VOD and REVO trough actions (REVO+A), the combination of them through state (REVO+F) and the combination of REVO+A and REVO+F (REVO+FA).

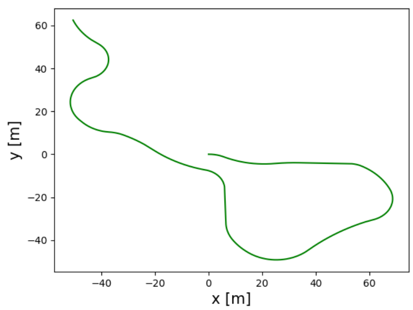
Every learning process includes 500 episodes, each limited to 100 time steps. The time step is 0.2 seconds, hence 20 seconds per episode. The policy updates are synchronized with the simulation time steps, two updates per step. After every 10 episodes, an evaluation process is performed on 5 fixed random paths. During this process, the exploration noise was disabled to evaluate the policy accurately and no data is collected to prevent over-fitting to these fixed paths. The results are averaged on 5 independent such learning processes.

The goal is that the vehicle achieves the most progress possible along the path during each episode. The progress that the vehicle achieved along the path in an episode is denoted as D, the distance achieved by the VOD controller is denoted as D_{VOD}. the normalized progress is $D_n = D/D_{VOD}$. If the vehicle fails, the distance achieved is irrelevant (even the progress until the failure is not informative because every P_s

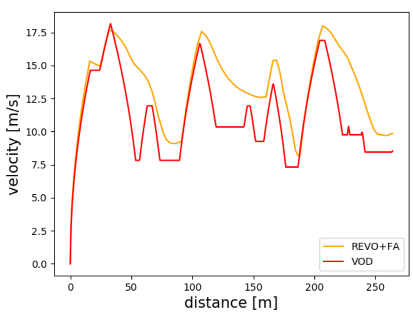
is independent of the rest of the path, hence, on the same policy the distance until the failure depends only on the path), therefore the learning process for each method is described as D_n and the failure rate as function of policy updates. The failure rate is a running average over 60 evaluation episodes.

C. Results

Fig. 4a shows one of the paths used for the evaluation during the training process and the velocity along it during 20 seconds. It can be seen, that the learned velocity profile of REVO+FA is higher compared to dynamics based planned velocity.



(a)



(b)

Fig. 4: (a) One of the random paths used for tests and (b) The dynamics based velocity profile (VOD) and a learned velocity

Fig. 5 present the normalized progress on the path D_n on successful episodes (i.e. unsuccessful episodes were filtered out from this graph). All the results are normalized with respect to VOD, hence VOD is a horizontal line at 1. The Reinforcement learning method (REVO) reach VOD after 6500 training iterations, while REVO+AF and REVO+A are always above the VOD performance (on the successful episodes). A little preference of REVO+AF over REVO+A can be seen. REVO+F shows worse performance than REVO, But in section VI-D in a different setting, the advantage of REVO+F can be noticed. All methods outperform eventually the VOD controller performance by 15%.

Fig. 6 shows the percentage of failed episodes during the learning process. As depicted by the figure, the failure rate of REVO+A and REVO+FA is below the failure rate of

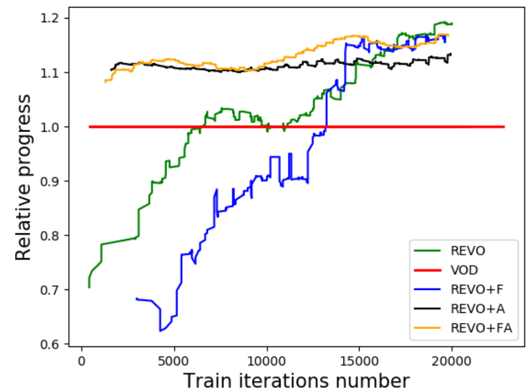


Fig. 5: Normalized progress in 20 seconds along 5 test paths, during training

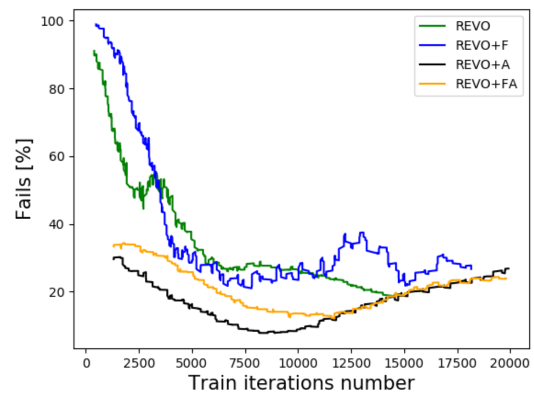


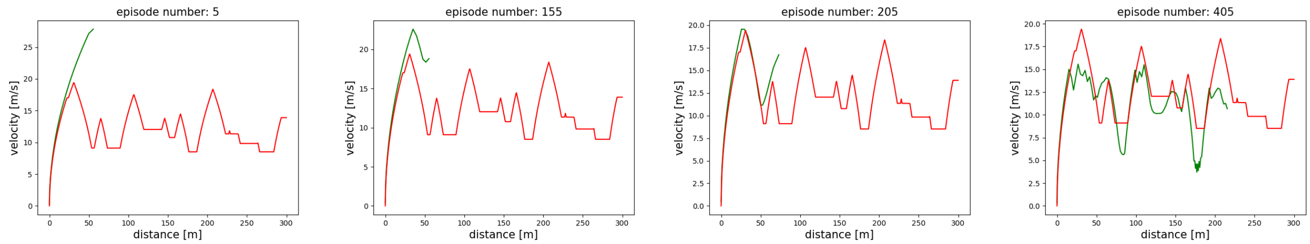
Fig. 6: Failure rate during training

REVO during the whole training. The conclusion from this results is that the training process was faster when combining the model-based controller (VOD) to the reinforcement controller (REVO) through the actions.

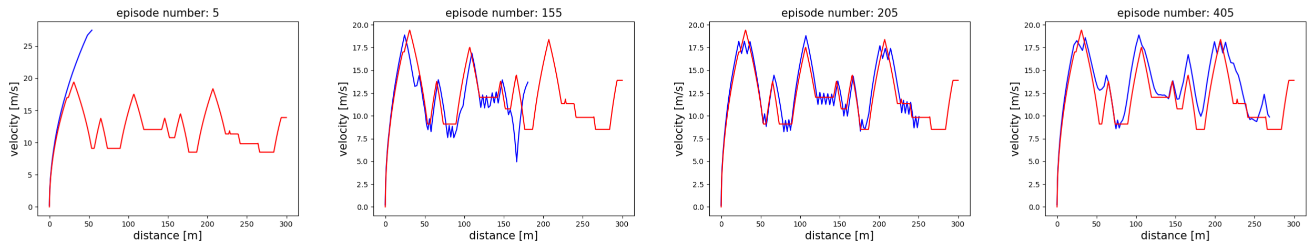
Fig. 6 shows that after the fails average decreases, it increases again. A possible explanation is that the velocity gets closer to the handling limits of the vehicle and when driving near the limit the possibility to fail is much higher. Therefore, in reality, it is better to avoid such close-to-limit velocity, and down-scale this velocity by some factor. However, if this limit is unknown it is also unknown how far is the velocity from the limit, wish may be dangerous.

D. 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 learning process. When running the learning process on the same path (instead of on random paths) it is possible to closely track the policy improvement. In this case, it can be seen in 7 that after some training, the learned policy acts like the VOD input, while the regular learning process (REVO) is still not able to complete this path.



(a) —REVO, —VOD



(b) —REVO+F, —VOD

Fig. 7: 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

VII. Conclusions & Future Work

Current methods for time optimal velocity control are usually not data driven but are analytically derived based upon vehicle dynamics. Previous works on reinforcement methods for autonomous navigation usually do not try to achieve time optimal velocity control. In this paper, we explore the problem of time optimal velocity control of a simulated vehicle. We introduce the usage of deep reinforcement learning to solving this problem and outperform an analytically derived solution.

We propose a method, which is based upon two novel concepts for improving the performance of a reinforcement learner agent. The first is adding the output of the dynamics based method as another feature to the reinforcement learner. The second includes adding the dynamics based action to the action taken by the agent, that is, the agent's action acts as a delta distance from the dynamics based method. We show that this novel solution results in a significant improvement to the reinforcement learner, especially at early stages of learning. It was shown that the regular reinforcement learning (REVO) took around $15,000$ iterations to converge to an optimal velocity controller, compared to an almost immediate convergence by the combined controller (REVO+FA).

This method may have a great impact on robot agents in domains in which an agent that has the ability to take reasonable actions already exists. Future work will include the deployment of REVO+FA in an actual (small) autonomous vehicle. Aside of the engineering challenges that we will be facing, such deployment will also require research challenges such as the need for transfer learning [18], in which the agent must transfer knowledge gathered in simulation to the actual vehicle. While the reinforcement learning agent developed in simulation will not work as is in an actual vehicle, the advantage of REVO+FA (over a standard reinforcement learner) may become more pronounced when interacting with an actual autonomous vehicle, as such interaction requires much longer duration, so that starting with a completely random reinforcement learner becomes impractical.

References

[1] 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.

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

[3] 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.

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

[5] David Stavens and Sebastian Thrun. "A self-supervised terrain roughness estimator for off-road autonomous driving". In: *arXiv preprint arXiv:1206.6872* (2012).

[6] 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).

[7] 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.

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

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

[10] Andrew Y Ng, Daishi Harada, and Stuart Russell. “Policy invariance under reward transformations: Theory and application to reward shaping”. In: *ICML*. Vol. 99. 1999, pp. 278–287.

[11] 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).

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

[13] Z. Shiller and H. Lu. “Computation of path constrained time-optimal motions with dynamic”. In: *ASME* 14.1 (1992), pp. 34–40.

[14] 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.

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

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

[17] *Autonomous driving at time optimal velocity*. Youtube. 2019. URL: `https://youtu.be/G2KwEHGsPoo`.

[18] Sinno Jialin Pan and Qiang Yang. “A survey on transfer learning”. In: *IEEE Transactions on knowledge and data engineering* 22.10 (2010), pp. 1345–1359.