Model-based Reinforcement Learning for Time-optimal Velocity Control

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

***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 paper, 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.

***Keywords*-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 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 on the vehicle that are functions of its speed, such as not rolling-over and not sliding [1], [2], [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 paper 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].

Manuscript received: February, 24, 2020; Revised May, 31, 2020; Accepted July, 6, 2020.

This paper was recommended for publication by Editor Dan Popa upon evaluation of the Associate Editor and Reviewers' comments. This research was supported, in part, by the Ministry of Science & Technology, Israel.

[1]Gabriel Hartmann and Zvi Shiller are with the Department of Mechanical Engineering and Mechatronics, Ariel University, Israel gabriel.hartmann@msmail.ariel.ac.il shiller@ariel.ac.il

[2]Amos Azaria and Gabriel Hartmann are with the Department of Computer Science, Ariel University, Israel amos.azaria@ariel.ac.il

Digital Object Identifier (DOI): see top of this page.

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 paper are thus twofold: (i) a model-based reinforcement learning method for driving a vehicle at near time-optimal speeds along any given path in 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 [6], [2], 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], [12], [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, $P = \{p_1, p_2, \cdots, p_N\}$. 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:

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

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 $\boldsymbol{v}$ 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.

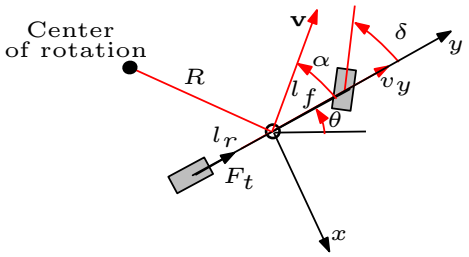


Fig. 1: The bicycle model

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 $\boldsymbol{v}$ 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

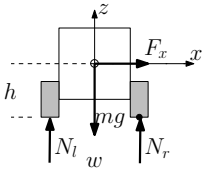


Fig. 2: Parameters for the simple roll model

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.

3) Multi-step Roll-outs: If at time step t the n future actions $a_t, a_{t+1}, ...a_{t+n}$ 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.

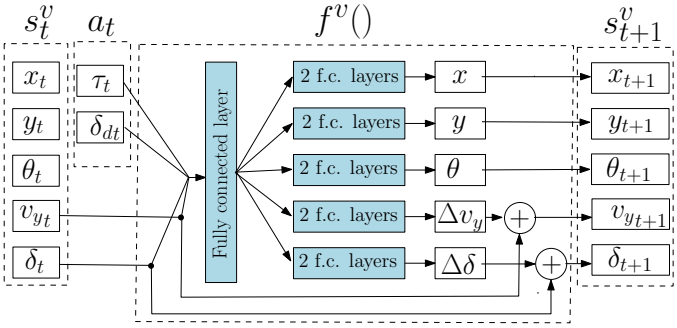


Fig. 3: The prediction transition function $s_{t+1}^v = f^v(s_t^v, a_t)$ architecture. The blue rectangles are the fully connected layers of the neural network.

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 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 $LTR_t, LTR_{t+1}, LTR_{t+2}, ..., LTR_{t+n}$ 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.

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

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.

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.

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.

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

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.

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 $h = 1.0m$, COM to rear wheel distance $l_r = 1.55m$, COM to front wheel distance $l_f = 1.55m$ and width= $2.1m$. The vehicle is driven by all wheels (4x4); 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

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

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 $2m$ 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] (Ornstein–Uhlenbeck process). Because of the longer convergence time for the DDPG agent, it is trained for 200 episodes.

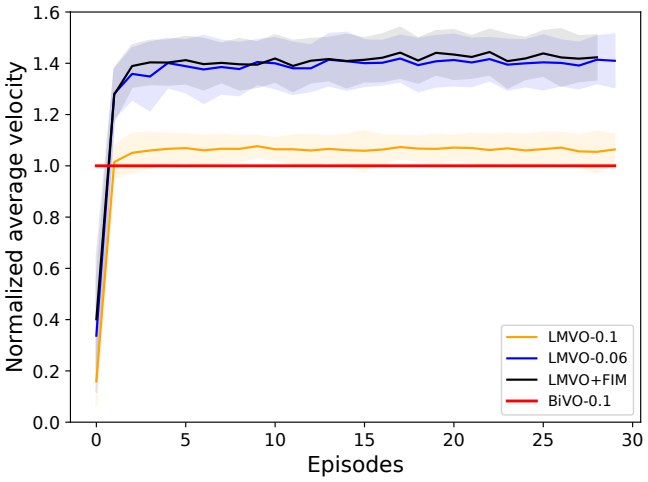
Since the average velocity during the failed episodes was usually higher than the average velocity during successful episodes, we excluded failed episodes when presenting the average velocity of each method. The baseline is a controller that uses the same policy as LMVO but instead of learning the model, 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

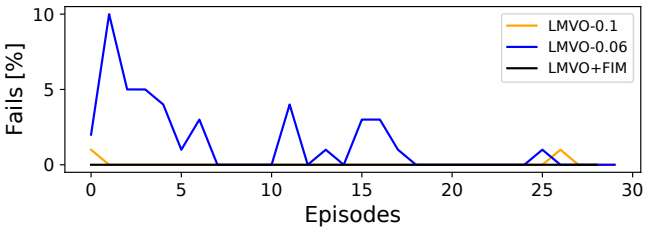
Fig. 4 compares three different methods: LMVO with a safety factor of $\beta = 0.1$ ((LMVO-0.1), LMVO with a safety factor $\beta = 0.06$ (LMVO-0.06) and LMVO+FIM with a safety factor of $\beta = 0.05$. A comparison to DDPG is shown in section VIII-D. As depicted by the figures, an important advantage 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). 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 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).



(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

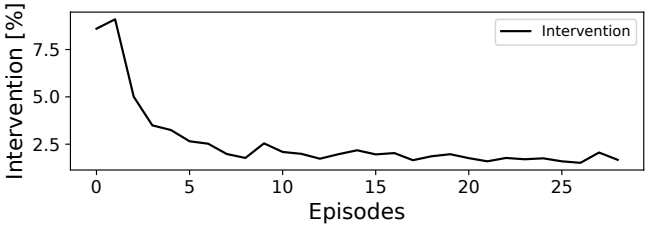


Fig. 5: Intervention of FIM module during training LMVO+FIM.

D. Comparison to Model-free RL

Fig. 7a 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). Further-

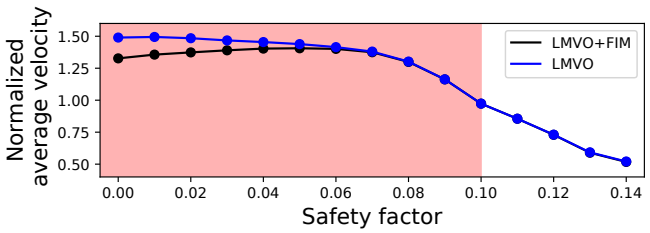
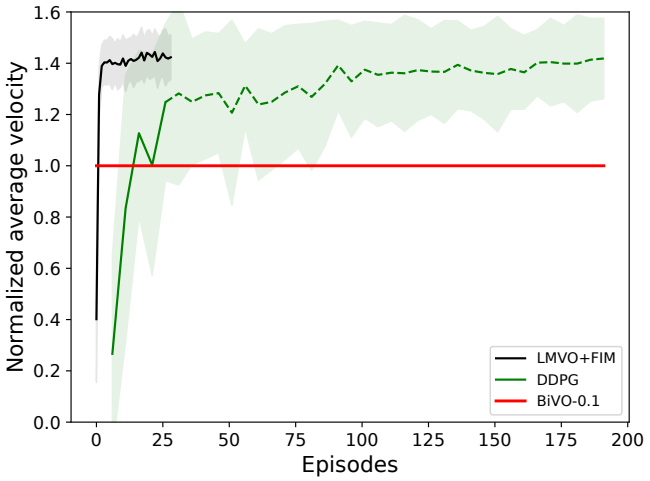
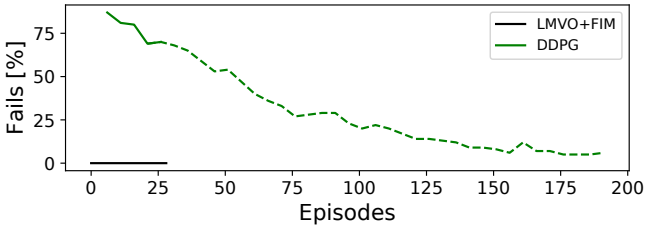


Fig. 6: Normalized average velocity as a function of linear safety factor (β). The red area marks the range of safety factors that may cause failure when used by LMVO without FIM.

more, the failure rate achieved by LMVO+FIM is constantly zero, compared to a very high failure rate starting at over 90% and even after 200 episodes, the failure rate is still approximately 5%.



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

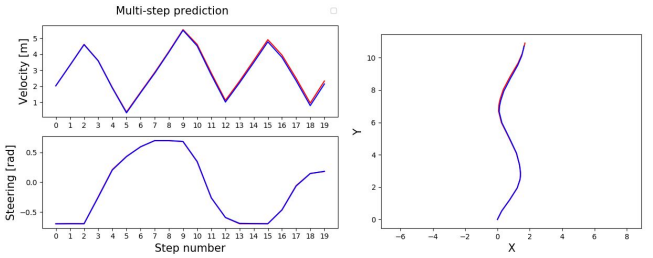


(b) Failure rate

Fig. 7: 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.

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



(a) —Prediction, —Real (b) —Prediction, —Real

Fig. 8: 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.

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.

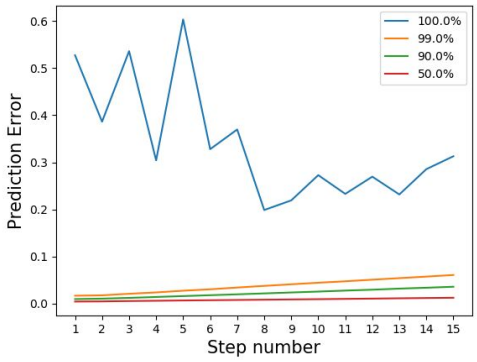


Fig. 9: Error between the predicted and the actual LTR as a function of the roll-out depth. Note the difference between the maximal error of the 99th percentile of samples and all samples.

IX. Conclusions

In this paper, 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 *ICRA 2008. IEEE*, 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 *IV, 2017 IEEE*, 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," *International Journal of Control, Automation and Systems*, 2017.

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

[5] J. Garcıa and F. Fernández, "A comprehensive survey on safe reinforcement learning," *Journal of Machine Learning Research*, 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 *ASME 2015 dynamic systems and control conference*, 2016.

[7] A. Liniger, A. Domahidi, and M. Morari, "Optimization-based autonomous racing of 1: 43 scale rc cars," *Optimal Control Applications and 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 *2016 IEEE,ICRA*, 2016, pp. 1433–1440.

[9] J. Funke, P. Theodosis, R. Hindiyeh, G. Stanek, K. Kritatakirana, C. Gerdes, D. Langer, M. Hernandez, B. Müller-Bessler, and B. Huhnke, "Up to the limits: Autonomous audi tts," in *2012 IEEE Intelligent Vehicles Symposium*. IEEE, 2012, pp. 541–547.

[10] J. Kabzan, M. d. l. I. Valls, V. Reijgwart, H. F. C. Hendrikx, C. Ehmke, M. Prajapat, A. Bühler, N. Gosala, M. Gupta, R. Sivanesan, *et al.*, "Amz driverless: The full autonomous racing system," *arXiv preprint arXiv:1905.05150*, 2019.

[11] M. Bojarski, D. Del Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. Monfort, U. Muller, J. Zhang, *et al.*, "End to end learning for self-driving cars," *arXiv preprint arXiv:1604.07316*, 2016.

[12] C. Chen, A. Seff, A. Kornhauser, and J. Xiao, "Deepdriving: Learning affordance for direct perception in autonomous driving," in *Proceedings of the IEEE International Conference on Computer Vision*, 2015, pp. 2722–2730.

[13] P. Drews, G. Williams, B. Goldfain, E. A. Theodorou, and J. M. Rehg, "Aggressive deep driving: Model predictive control with a cnn cost model," *arXiv preprint arXiv:1707.05303*, 2017.

[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 *2018 IEEE Intelligent Vehicles Symposium (IV)*, 2018, pp. 1251–1256.

[15] S. Lefèvre, A. Carvalho, and F. Borrelli, "A learning-based framework for velocity control in autonomous driving," *IEEE Transactions on Automation Science and Engineering*, 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 Transactions on Systems, Man, and Cybernetics: Systems*, no. 99, pp. 1–12, 2017.

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

[18] M. Jaritz, R. de Charette, M. Toromanoff, E. Perot, and F. Nashashibi, "End-to-end race driving with deep reinforcement learning," in *2018 IEEE,ICRA*, 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 *2017 IEEE International Conference on Robotics and Automation (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 *Advances in neural information processing systems*, 2018, pp. 8092–8101.

[21] F. Berkenkamp, M. Turchetta, A. Schoellig, and A. Krause, "Safe model-based reinforcement learning with stability guarantees," in *Advances in neural information processing systems*, 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 Conference on Artificial Intelligence*, 2018.

[23] X. Li, Z. Serlin, G. Yang, and C. Belta, "A formal methods approach to interpretable reinforcement learning for robotic planning," *Science Robotics*, 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 *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, vol. 15, no. 1, 2019, pp. 24–30.

[25] O. Bastani, "Safe reinforcement learning via online shielding," *arXiv preprint arXiv:1905.10691*, 2019.

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

[27] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *arXiv preprint arXiv:1509.02971*, 2015.

[28] G. Hartmann, Z. Shiller, and A. Azaria, "Deep reinforcement learning for time optimal velocity control using prior knowledge," *ICTAI*, 2019.

[29] M. P. Deisenroth, G. Neumann, J. Peters, *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 *2018 IEEE ICRA*, 2018, pp. 7559–7566.

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

[32] C. E. Rasmussen, "Gaussian processes in machine learning," in *Summer School on Machine Learning*. 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," *arXiv preprint arXiv:1809.05214*, 2018.

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