Learning Path Tracking for Real Car-like Mobile Robots From Simulation

Danial Kamran, Junyi Zhu, Martin Lauer
Institute of Measurement and Control Systems, Karlsruhe Institute of Technology (KIT), Karlsruhe, Germany
Email: {danial.kamran,lauer}@kit.edu, junyi.zhu@gmx.de

***Abstract*—In this paper we propose a Reinforcement Learning (RL) algorithm for path tracking of a real car-like robot. The RL network is trained in simulation and then evaluated on a small racing car without modification. We provide a big number of training data during off-line simulation using a random path generator to cover different curvatures and initial positions, headings and velocities of the vehicle for the RL agent. Comparing to similar RL based algorithms, we utilize Convolutional Neural Network (CNN) as image embedder for estimating useful information about current and future position of the vehicle relative to the path. Evaluations for running the trained agent on the real car show that the RL agent can control the car smoothly and reduce the velocity adaptively to follow a sample track. We also compared the proposed approach with a conventional lateral controller and results show smoother maneuvers and smaller cross-track errors for the proposed algorithm.**

I. Introduction

Controlling an automated vehicle in order to follow the desired path is always an important task specifically for complex curvatures. Assuming non-holonomic one track model, the control law can be applied to command the steering and acceleration of the vehicle in order to minimize cross-track error (i.e. the distance between vehicle and path) and follow the desired reference path [1]–[4]. The main drawback for these approaches is considering only a few parameters for representing the path and vehicle state such as distance and curvature of the path at current and near future moments. Such low dimensional representation prevents to control the vehicle with long term optimal maneuvers similar to human driving style that pays attention to both factors: distance to the path and also lateral jerk.

Reinforcement Leaning (RL) is an efficient learning framework for robotics that can learn optimal control commands according to predefined reward function. Some researchers used RL for training a path-following controller for robots and specifically vehicles in order to have more intelligent and smoother maneuvers [5]–[7]. However, all of these works consider only a few parameters for representing the path and vehicle pose characteristics similar to model based control approaches. As a result, they still suffer from lack of information for estimating the situation and can not provide long term optimal solutions according to the provided reference path.

There are so called end-to-end learning approaches that take raw sensor data as input and directly compute control commands in the output [8]. They extract meaningful information from sensor data and then learn the best policy for optimal control commands. Although such methods show promising outcomes, they are not suitable for widely accepted modular structure of automated driving which require specific modules for specific tasks such as perception, planning and control with better feasibility of maintenance and safety verifications [9]. Another drawback for all existed learning based control approaches is restriction in amount of training data due to training only on the real car [5], [8] or only training and evaluating in simulation without any experiments in reality [6], [7], [9].

The main contribution of this paper is in two folds. First, we apply a model free state representation for the path by utilizing a convolutional neural network (CNN) as image embedder for the task of path representation, see Figure 1. This part estimates features that represent the path curvature and pose errors of the vehicle for the current and future horizon according to the history of local path images. Comparing to similar approaches like [5]–[7] that only use parameterized representation of the path, the proposed approach can follow more complex paths and produce smoother maneuvers. It also becomes less sensitive to the measurement error for estimating path parameters that may come from noise in lane detection or road mapping.

The second contribution is learning a generic path tracking controller from simulation and evaluating it on the real car-like robot without any modification. We propose domain randomization for generating different kinds of path with random curvature and length as well as initial lateral and heading errors for the vehicle during simulation in order to cover all possible situations for training. Such strategy prevents the network from becoming over-fitted to the few training samples from dataset or small number of scenarios in simulation which makes it more generic than similar approaches [5]–[9]. According to our experiments, without further learning on real car and any modification, the trained path tracker follows the path smoothly and reduces the speed at locations with big curvatures.

II. Preliminaries

A. Reinforcement Learning

Reinforcement Learning (RL) is an efficient learning approach to solve different control problems for robotics through

978-1-7281-3605-9/19/$31.00

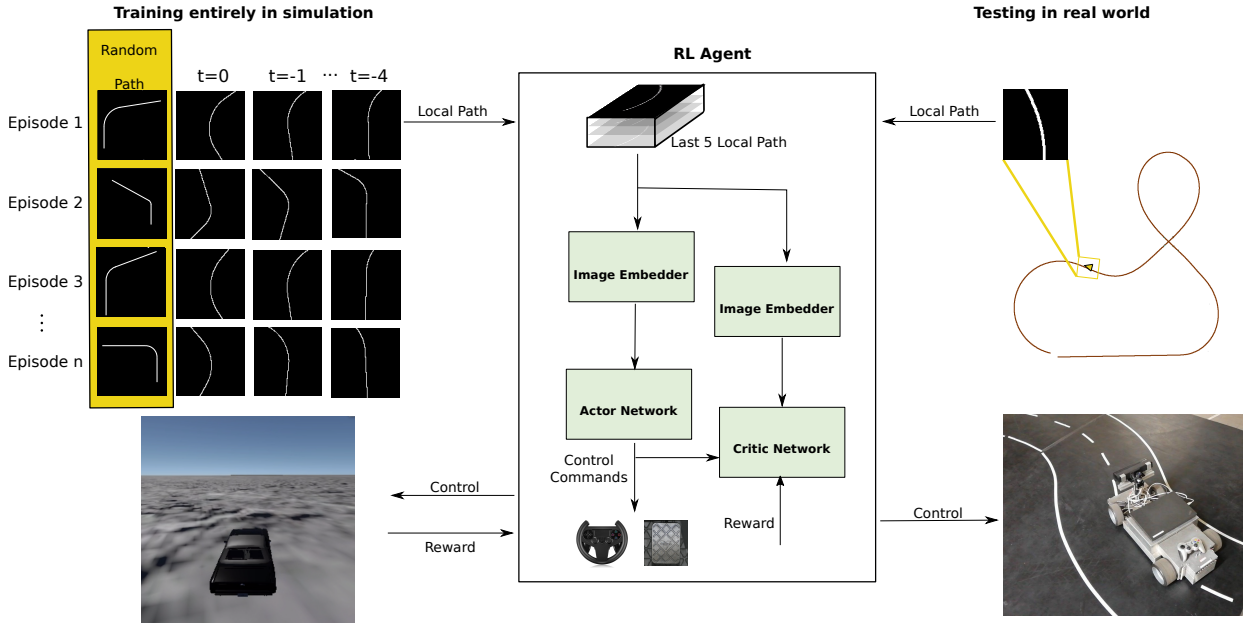


Figure 1: Path randomization for training only in simulation (left) and testing the trained agent for real small car (right).

learning the best policy which provides most desired behavior. RL models such problem as a Markov Decision Process (MDP). At each discrete time step t, the RL agent considers the current state coming from environment (s_t) and decides about an action (a_t). The action is executed and the agent gets new state from the environment (s_{t+1}). It also receives a reward value $r(s_t, a_t)$ according to the new situation in the whole environment. A policy function maps each state from the environment into action. The main goal of the reinforcement learning is to learn the best policy that provides actions for each step with maximum discounted future reward, as defined below assuming $\gamma \in [0, 1]$:

$$R_t = \Sigma_{i=t}^{T} \gamma^{(i-t)} r(s_i, a_i) \tag{1}$$

In order to find the best policy, action-value function helps to find out the future reward of each action for each state:

$$Q^{\pi}(s_t, a_t) = \mathbb{E}_{r_t, s_{t+1} \sim E}[r_t + \gamma Q^{\pi}(s_{t+1}, \pi(s_{t+1}))] \tag{2}$$

If we have optimal Q function, a greedy policy selects actions with maximum Q value as the best action for each state [10]:

$$\mu(s_i) = arg\ max_a\ Q(s_i, a|\theta^Q) \tag{3}$$

The Q function can be approximated using deep neural networks known as Deep Q Networks (DQN)[11].

Assuming θ^Q as the parameters of the Q function estimator, the Q function is learned through minimizing the loss function using B random samples from reply buffer:

$$L(\theta^Q) = \sum_{i=1}^{B} (y_i - Q(s_i, \mu(s_i)|\theta^Q)) \tag{4}$$

where y_i is the network training target:

$$y_i = r(s_i, a_i) + \gamma Q(s_{i+1}, \mu(s_{i+1})|\theta^Q) \tag{5}$$

DQN is developed for discrete action spaces which makes it usually not well-suited for continuous control problems.

B. Continuous Control with Deep Deterministic Policy Gradient

Deep Deterministic Policy Gradient [12] is an actor-critic network suitable for learning policies with continuous action space. The actor network is a policy network $\pi(s_t|\theta^{\pi})$ with parameters θ^{π} that generates continuous actions according to current state. The critic network $Q(s, a|\theta^Q)$ estimates Q value for each action and state pair with parameters θ^Q. The critic network is learned similar to DQN by minimizing the loss function for estimating Q values:

$$L(\theta^Q|\theta^{\pi}) = \sum_{i=1}^{B} (y_i - Q(s_i, \pi(s_i|\theta^{\pi})|\theta^Q)) \tag{6}$$

where y_i is computed assuming actor network as policy function:

$$y_i = r(s_i, a_i) + \gamma Q(s_{i+1}, \pi(s_{i+1}|\theta^{\pi})) \tag{7}$$

The main advantage of DDPG is benefiting from the critic network in order to estimate gradients of Q value for the actions selected by actor network and utilize that in order to update the actor network via policy gradient:

$$\nabla_{\theta^{\pi}} J = \sum_{i=1}^{B} (\nabla_{\pi(s_i)} Q(s_i, \pi(s_i)|\theta^Q) \nabla_{\theta^{\pi}} \pi(s_i|\theta^{\pi})) \tag{8}$$

Such strategy makes DDPG to be able to learn continuous actions with higher Q values that provide higher cumulative future reward. Therefore, comparing to discrete action space, DDPG is more suitable for learning steering and throttle control commands for robots and vehicles similar to our problem.

III. Learning based Path Tracking Module

In this section we explain our RL algorithm for the task of path following. The goal to train the network is not only minimizing cross-track error, but also generating smoother control commands in order to provide maneuvers more similar to human.

A. State & Action Representation

For a better understanding of the situation, the network has to know about the vehicle pose relative to the reference path not only for the current moment but also for previous vehicle poses, see Figure 1. Therefore, we feed last five local path images which are received during last second (the RL is updated every 0.2 second assuming 5 Hz for control rate is enough). Using such sequence of images, the network can obtain information about jerk as well as speed when reaching curvatures. However, for the situations where the path is a straight line with zero curvature, the local path image would not change if the car follows it perfectly. Furthermore, besides vehicle heading the RL agent needs to know previous steering to have less lateral jerk. Considering these reasons, we also feed to the RL network last 5 throttle and steering values executed on the vehicle before. Therefore, the final state is defined as here:

$$s_t = (p_t, \cdots, p_{t-4}, a_{t-1}, \cdots, a_{t-5}) \tag{9}$$

where p_i are binary matrices that specify the local path in vehicle coordinates and a_i are the last actions from RL executed on the vehicle.

Since reference path is only one pixel wide in the local path image, this causes input of network being very sparse which give rise to an inefficient learning. So we apply a Gaussian blur on the path matrix before giving it to the network. Gaussian blur can also make input images more dense while keeping detailed information of the path curvature. It is also beneficial by removing noises in the input path, as in real world the given reference path can be noisy due to measurement errors in estimating drivable corridor or during mapping of the desired path as road-maps.

In the output of actor network, the proposed RL algorithm decides about current steering and throttle values of the vehicle. Therefore, the action is defined as a two dimensional vector:

$$a = (\delta, T)\,; \quad \delta, T \in [-1, 1] \tag{10}$$

where δ is vehicle front wheels steering and T is vehicle throttle. Note that both of these values should be scaled before sending for the vehicle actuators:

$$\delta_{actuator} = \frac{\pi}{4}\delta \tag{11}$$

$$T_{actuator} = \frac{T+1}{2} \tag{12}$$

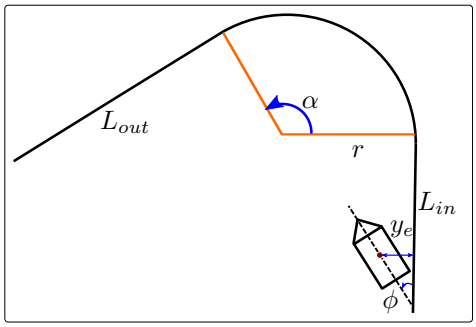


Figure 2: Illustration of randomized reference path generation during training.

B. Domain Randomization for Generating Training Data

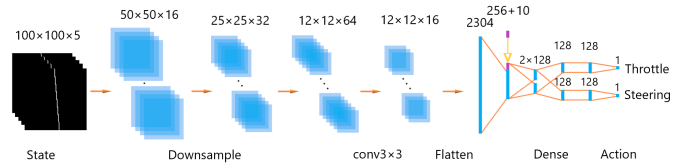
Domain randomization is a key point of proposed approach in order to generate diverse data in simulator during training which helps to learn a generic and robust path tracker network that can be also applied for real car. Without such randomization, the agent only learns to follow sample paths used during training and can not be general. For that purpose, a module called random path generator provides random curve path for each episode (Figure 2). The random path consists of curve radius (r), arc angle (α), length of curve initial part (L_{in}) and vehicle initial pose (y_e, φ). L_{out} is a big constant value to ensure that the agent learns to drive stable after turning. According to the length of the initial part of the path, the vehicle can approach the curve by different speeds which also provide more random situation during training.

C. Network Architecture

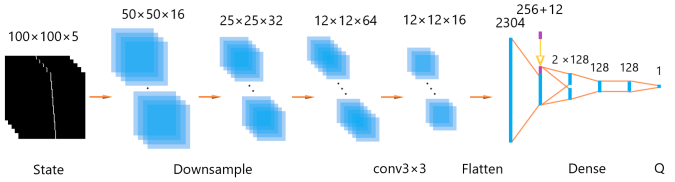
Figure 3 depicts the architecture of proposed DDPG network. In both actor and critic networks, we use several convolutional layers to extract features from local path image as image embedder. The outputs are then flatten into a feature vector and concatenated with last 5 action pairs before feeding to fully connected layers to compute action and Q values in actor and critic networks respectively. In addition to state values, the critic network also gets current action pair as input and also the reward for learning the Q value. For down-sampling of input path image, we use 3×3 convolutional layer with stride 1 and then 3×3 max pooling layer with strides 2. All convolutional layers and dense layers except the last dense layer use batch normalization. Activation function for convolutional layer is ReLU and for dense layer is Leaky ReLU with slope 0.2, except the last one which does not have activation function. In the second dense layer one branch uses square function as activation function.

D. Reward Function

We make a combination of different targets, including stability, utility and comfort in order to design the reward function of proposed RL algorithm. For that purpose, we take into account current cross-track error (y_e), steering (δ) and throttle (T) for computing the reward at each state:



(a) Image embedder and actor network



(b) Image embedder and critic network

Figure 3: Overview of image embedder, actor and critic networks utilized in the proposed RL algorithm.

$$R = \begin{cases} -1 & \text{if } |y_e| > 0.5 \\ 0 & \text{if } |\dot{\delta}| > D \text{ or } T < 0.1 \\ 1 - 0.8y_e - 0.3\delta - 0.3\frac{|0.7-T|}{0.7} & \text{otherwise} \end{cases} \tag{13}$$

Using such reward function, the agent will have a big punishment for having lateral distance to the path greater than 0.5 meter. It does not get any reward for jerky maneuvers where difference of current and last steering values is larger than D that is a variable getting smaller during training:

$$D_n = \left\{ \begin{array}{cc} 0.9997D_{n-1} & \text{if } D_{n-1} > 0.05 \\ 0.05 & \text{otherwise} \end{array} \right\} \tag{14}$$

where n is the sequence number of episodes during training.

Moreover, we set the reward to 0 if throttle that is sent for actuator is less than 0.1 in order to enforce the car to drive forward and never stop. During our experiments, we figured out that the network can learn quickly to avoid actions assigned with minimal reward (-1 or 0). However, constant reward values like -1 or 0 does not provide too much information for the network during back-propagation to learn best policy due to same punishment for all actions. For fast convergence and allowing the network to explore steering changes very rapidly at the beginning, we set the steering punishment threshold D being a relative large value 1 at the beginning, and let it to decay with training.

The part of positive reward represents the goal of smooth and fast path tracking. We give higher reward when the vehicle has lower cross-track error, lower steering and has throttle similar to the desired throttle which is set to 0.7 in our experiments.

E. Policy Exploration

Policy exploration is very helpful for our RL algorithm in order to experience different situations and compare the impact of different actions during training. Most exploration heuristics rely on random perturbations of the agent's policy,

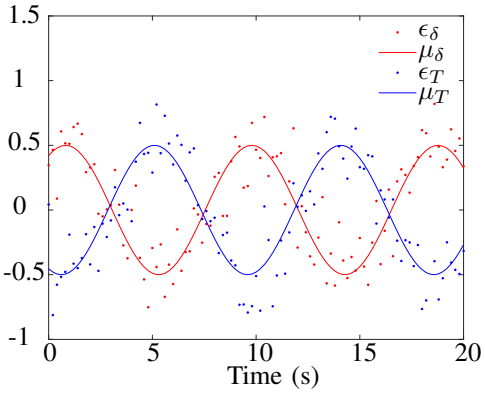


Figure 4: Example of random noise during one episode.

for instance by adding a zero mean Gaussian distributed noise to the output. However, we observed in our experiments that such kind of independent exploration noise is inefficient for fast convergence of RL algorithm, mainly because of the reason that the vehicle system is a low pass filter for high frequency changes in control commands and responses slowly to big changes. Moreover, we believe that for specific state value, a group of neighbor values in action space are better than others and the agent should only search for the optimal one by exploring in that area. Therefore, sudden big variations of control values can not demonstrate effective impacts for the RL algorithm to realize the best policy. For that purpose, we use Gaussian distributions with random sinusoid means and random deviations as steering and throttle noises which are called ϵ_δ and ϵ_T respectively:

$$\begin{aligned} \epsilon_\delta, \epsilon_T &\sim N\left(\mu, \sigma^2\right) \\ \mu_\delta &= A sin\left(\omega t + \varphi_\delta\right) \\ \mu_T &= A sin\left(\omega t + \varphi_T\right) \end{aligned} \tag{15}$$

where σ^2, A, ω are sampled from zero mean Gaussian distribution and φ_δ, φ_T are sampled from uniform distribution of range $(-\pi, \pi)$ all at the beginning of each episode and will be constant during that episode. Figure 4 depicts an example noise values that were selected during RL steps for one specific episode.

Since the output of actor network is theoretically between $(-\infty, \infty)$, we have to clip the throttle and steering into range $[-1, 1]$. Then the problem is that the actions are mostly either minimum or maximum as all the values out of this range have been clipped to the boundaries. This would make the exploration invalid. In consideration of this the network explores with pure noise between -1 and 1 in the first $n = 500$ episodes, which help the actor network to experience more action values in this range . After that, the network further explores adding noise to the agent's actor network outputs as usual.

IV. Results and Evaluation

A. Simulation for Training

We have used Carla simulator [13] in order to train the proposed RL agent as an off-line process benefiting from unlimited amount of training data. As visible in Figure 1, the vehicle is driving in an empty playground without any obstacle. It drives in order to follow the desired path generated from random path generator at the beginning of each episode as explained in III-B. During the training in each episode, state, action and reward tuple for each step is saved in the memory with size of 20000 elements. Each episode can be terminated by one of these conditions: Driving to the end of sample path, having cross-track error bigger than 2 meters or finishing the time limit (20 seconds). After terminating an episode, random samples from memory are selected in order to train actor and critic networks which both have one evaluation and one target network using the update formulas for the DDPG network explained in II-B. The training is only applied on the evaluation network and then the target network is updated using soft replacement with TAU of 0.005 which progressively replaces small amounts of the target network. See [14] for some videos.

Figure 5 shows the average lateral error and steering values over evaluation episodes during training. As visible in this graph, the average lateral error and steering values are both getting lower during training. The lateral error even simply converges after the first 100000 steps which is about 6 hours. In comparison to conventional zero mean Gaussian noise which even could not converge in our experiments at all, our new effective exploration helps the network to find optimal actions much faster and more efficiently.

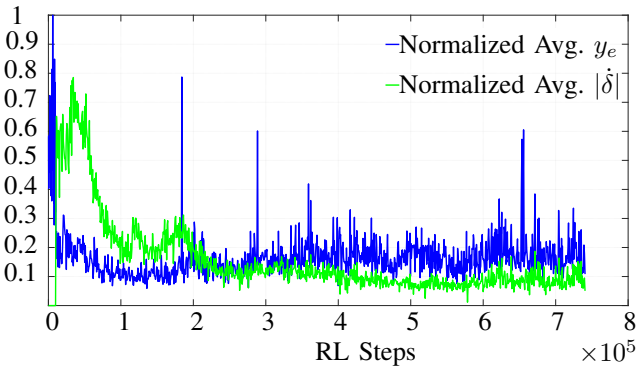


Figure 5: Average lateral error and steering difference for each episode during training. Note that both error terms are normalized.

B. Experiments on Real Car

In order to test how the learning approach is able to transfer to the real world, we run the trained RL network on a real small car in order to follow a sample track similar to a coupled lateral and longitudinal controller. See [14] for a video about testing the trained algorithm on a real car without further training. The car we used for experiments is visible in Figure 1 which has one motor on rear wheels for forward movement and one motor to control the steerings of the front wheels. The location of robot during experiments is provided by an indoor localization system called Stargazer similar to [5] which provides X-Y position and headings of the car. The orbital tracking controller proposed in [4] is also implemented in order to compare with the proposed RL based path follower.

Figure 6 depicts the path traversed using the lateral controller and our proposed RL based path follower both with the same constant velocity of 0.5 m/s. In other words, here we only used steering commands from RL agent. According to this Figure, the RL agent has smoother maneuvers and less cross-track error specially at parts with high curvatures.

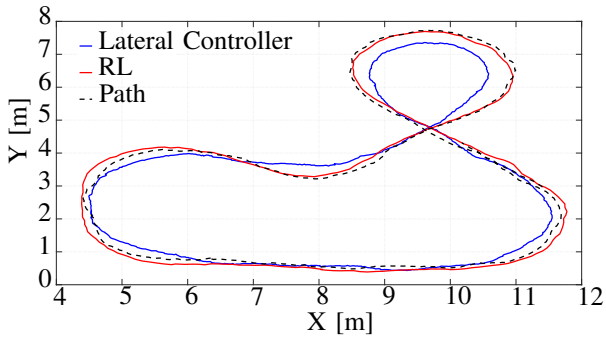


Figure 6: Traversed path by lateral controller and proposed RL algorithm both with constance velocity of 0.5 m/s.

We have also compared cross track error and steering commands of the two controllers running with constant velocity of 0.5 m/s (low speed test) and 1.1 m/s (high speed test) in Figure 7 (a and b). As it is visible, the cross track error for RL control is close to zero and much smaller than lateral controller for both low speed and high speed tests.

Finally, the trained RL agent has been used to control both steering and throttle commands of the car in order to evaluate coupled longitudinal and lateral control of the vehicle. In this case, the velocity is not constant anymore and can be adapted by RL agent according to its state observation. Note that for running the trained network on real car, $T_{actuator}$ is rescaled in order to make sure that the vehicle never exceeds the stable velocity which is 2 m/s. Figure 7 (c) shows the cross-track error of the vehicle using RL agent in this experiment. The cross-track error is smaller than high speed test and close to zero similar to the low speed cross-track error. The main reason for this improvement is that the agent can lower the speed according to its speed and upcoming path curvature in order to prevent the vehicle from being unstable and have large cross-track error.

V. Conclusions and Future Work

In this paper we presented a reinforcement learning based path tracker in order to control a real vehicle smoothly. In contrast to other RL based approaches, we used high dimensional path representation as images of path in vehicle coordinates. We utilized CNN layer as feature extractors in

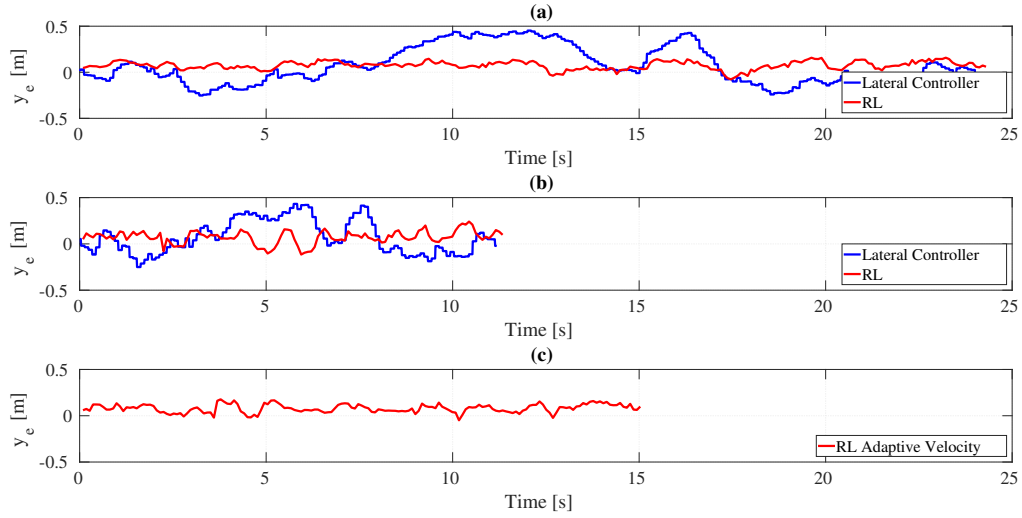


Figure 7: Cross track error for the lateral controller and RL algorithm in low speed (a), high speed (b) and adaptive RL velocity (c). All experiments started and ended at the same positions on the same track.

order to provide important information about vehicle state and its position for the RL agent. The second novelty of this paper is training the algorithm in simulation with several random configurations in order to cover variety of situations and using sinusoid based random noise for fast convergence. The trained network was then directly executed to control a real car in order to follow a sample track. According to our experiments, the trained network can control the vehicle in order to have minimum cross-track error and lateral jerk comparing to other lateral controller.

Acknowledgements

This research is accomplished within the project "UNICARagil" (FKZ 16EMO0287). We acknowledge the financial support for the project by the Federal Ministry of Education and Research of Germany (BMBF).

References

[1] R. C. Coulter, "Implementation of the pure pursuit path tracking algorithm", Carnegie Mellon University, Pittsburgh, PA, Tech. Rep., Jan. 1992.

[2] J. Ziegler, P. Bender, M. Schreiber, H. Lategahn, T. Strauss, C. Stiller, *et al.*, "Making bertha drive—an autonomous journey on a historic route", *IEEE Intelligent Transportation Systems Magazine*, vol. 6, no. 2, pp. 8–20, 2014.

[3] S. Thrun, M. Montemerlo, H. Dahlkamp, D. Stavens, A. Aron, J. Diebel, *et al.*, "Stanley : the robot that won the darpa grand challenge", vol. 23, no. April, pp. 661–692, 2006.

[4] M. Werling and L. Groll, "Low-level controllers realizing high-level decisions in an autonomous vehicle", *2008 IEEE Intelligent Vehicles Symposium*, pp. 1113–1118, 2008.

[5] M. Lauer, "A case study on learning a steering controller from scratch with reinforcement learning", *2011 IEEE Intelligent Vehicles Symposium (IV)*, pp. 260–265, 2011.

[6] A. B. Martinsen and A. M. Lekkas, "Curved path following with deep reinforcement learning: Results from three vessel models", *OCEANS 2018 MTS/IEEE Charleston*, pp. 1–8, 2018.

[7] J. Baltes and Y. Lin, "Path tracking control of non-holonomic car-like robot with reinforcement learning", *RoboCup-99: Robot Soccer World Cup III*, pp. 162–173, 2000.

[8] A. Kendall, J. Hawke, D. Janz, P. Mazur, D. Reda, J. Allen, *et al.*, "Learning to drive in a day", *CoRR*, vol. abs/1807.00412, 2018. arXiv: 1807.00412.

[9] G. Devineau, P. Polack, F. Altch, and F. Moutarde, "Coupled longitudinal and lateral control of a vehicle using deep learning", *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*, pp. 642–649, 2018.

[10] C. J. C. H. Watkins and P. Dayan, "Q-learning", *Machine Learning*, vol. 8, no. 3, pp. 279–292, 1992.

[11] V. Mnih and D. Silver, "Playing atari with deep reinforcement learning", 2013. arXiv: 1312.5602.

[12] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, *et al.*, "Continuous control with deep reinforcement learning", 2015. arXiv: 1509.02971.

[13] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun, "Carla: An open urban driving simulator", *Proceedings of the 1st Annual Conference on Robot Learning*, pp. 1–16, 2017.

[14] *Supplementary video file.* https://www.dropbox.com/s/082ujeqsvsyukac/learning_path_tracking.mpg?dl=0.