Q-learning based Reinforcement Learning Approach for Lane Keeping

Arpad Feher[1], Szilard Aradi[1] and Tamas Becsi[1]

Abstract— **The paper presents the application of a Q-learning reinforcement learning method in the area of vehicle control. The purpose of the research presented is to design an end-to-end behavior control of a kinematic vehicle model placed in a simulated race track environment, by using reinforcement learning approach. The varied trajectory of the track to provide different situations for the agent. The environment sensing model is based on high-level sensor information. Track curvature, lateral position, and relative yaw angle can be reached from the current automotive sensors, such as camera or IMU based systems. The objectives were reached through the definition of a rewarding system, with subrewards and penalties enforcing lane keeping. After the description of the theoretical basis the environment model with the reward function is detailed. Finally the experiments with the learning process are presented and the results of the evaluation are given from some aspects of control quality and safety.**

I. INTRODUCTION AND MOTIVATION

One key motivation of the automotive industry related researches is to achieve higher level of automation. [1]. Although many manufacturers often refer to their products as nearly self-driving car, currently these products only reach Level 2. Series production of the first Level 3 vehicle is likely to begin this year by Audi. From the aspect of control design and complexity, one of the best-defined and best-maintained environments are the highways and motorways. Therefore the first Level 3 control systems will be able to operate on highways as well.

When a functionality works well on highway scenario the next step is to extend it to more complex environments such as major road, where there are narrower curvatures, unseparated lanes and level crossings. The large number of varied and complex traffic situations need new approaches from which one possibility is the utilization of artificial intelligence. However, it is still questionable today which methods and architectures will be able to provide the appropriate level of safety. Meanwhile the IT giants devote huge resources to the AI developments and some of them has started to develop autonomous vehicles and/or control systems.

One common approach is supervised learning using artificial neural networks (ANN). In automotive control systems it can be used for image processing and the so-called "end-to-end learning" where the control signals can be calculated directly from the image sequence recorded by the camera. See e.g. [2]. These solutions assume the presence of huge-amount of training data containing corresponding control signal for each frame. Another considerable approach could be Reinforcement Learning (RL) which is an intensive research area today. Its theoretical background has long been researched and it can be combined with the state-of-art deep learning methods. In the recent years, several remarkable results were published by Google's Deepmind, see [3], [4] and [5]. In the field of reinforcement learning the techniques are demonstrated and benchmarked by video and board games, furthermore basic robotic and control tasks. These methods can also be applied to automotive research areas and vehicle control problems as well. The researchers primarily focus on very complex environments which can be sensed and controlled only by humans. Typical examples are the self-driving cars however RL can be used efficiently in mechatronics systems where the environment is more straightforward, but the control design is more difficult because of the system's non-linearity.

In the field of vehicle control and machine learning, the simulation environments are essential for R&D and testing. On one hand the vehicle dynamics simulators can be used for control function development, which is often extended with the detailed environment and sensor models. On the other hand for complex ADAS functions, it is necessary to provide realistic traffic flow, which can be realized by microscopic traffic simulators. These software environments are typically capable of interfacing with each other. Thus a complex simulations framework can be built-up for self-driving car development. Nevertheless, these setups are unnecessarily complex in many cases, in the field of education or research a partial simulation could be enough as well. Several RL benchmark environments exist, providing video and board games, classic control and robotic tasks, but the vehicle control researchers lack RL environments.

Our motivation is to make a complex RL environment with traffic generation and customized road sections and crossings. We have started to apply the state-of-art reinforcement learning based control methods on a simplified rural road environment model. In this paper the development of a lane keeping function with Q-learning is presented.

The paper is organized as follows: Section 2 describes the used Q-Learning reinforcement learning method. Section 3 introduces the environment model which implemented as an extension of Open AI Gym [6], then in Section 4 the simulation results are exposed. Finally in Section 5 the

*The research reported in this paper was supported by the Higher Education Excellence Program of the Ministry of Human Capacities in the frame of Artificial Intelligence research area of Budapest University of Technology and Economics (BME FIKPMI/FM).
EFOP-3.6.3-VEKOP-16-2017-00001: Talent management in autonomous vehicle control technologies- The Project is supported by the Hungarian Government and co-financed by the European Social Fund.

[1] A. Fehér, Sz. Aradi and T. Bécsi are with the Department of Control for Transportation and Vehicle Systems, Budapest University of Technology and Economics, Stoczek u. 2, H-1111 Budapest, Hungary, [feher.arpad; aradi.szilard; becsi.tamas]@mail.bme.hu

978-1-7281-1117-9/18/$31.00 ©2018 IEEE

conclusion and the possible improvements are detailed.

II. CONTROLLER DESIGN WITH Q-LEARNING-BASED REINFORCEMENT LEARNING

A. Reinforcement Learning

In reinforcement learning (as in other areas of artificial intelligence) the learner, the decision maker and the actor is called the *agent*. It receives information from the so-called *environment*, which includes everything outside the agent. Furthermore the agent chooses and action, which is sent to the environment causing a state transition, i.e. these interact continually with each other, the agent selecting actions and the environment responding to these actions and presenting new situations to the agent (see Fig. 1). In reinforcement

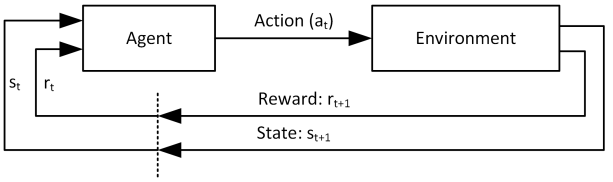

Fig. 1. Agent-environment interaction in reinforcement learning

learning the environment can be typically formulated as a finite-state Markov Decision Process (MDP). It is described with state s_t (where $s_t \in S$ and S represents the system's state space), action a_t changes the environment state from s_t to s_{t+1} with state transition probability $P(s_t, a_t, s_{t+1})$. The agent and environment interact at each of a sequence of discrete time steps, $t = 0, 1, 2, 3, ...$ Finally a numerical reward $r_{t+1} \in R$ indicates how well the agent is doing. The agent's job is to learn how can be the reward maximized, i.e. to find a policy $\pi : S \rightarrow A$ that maximizes the cumulative future reward $R = r_0 + r_1 + ... + r_n$, which is discounted representing the uncertainty of the future $R = \sum_0^n \gamma^t r_t$. The discount factor ($0 \leq \gamma \leq 1$) represents the uncertainty of the future, i.e. how much the future rewards depend on the actions in the past. A prediction of cumulative future reward $V_\pi(s) = \mathbf{E}_\pi[R|S_t = s]$ is defined as the "value" of the state which can be used to evaluate the badness or goodness of the state and therefore to select between actions. There exists an optimal policy π^* for which $V^{\pi*}$ is optimal for every state. In a finite MDP, the sets of states, actions, and rewards (S, A, R) all have a finite number of elements. The Markov property of MDP requires that the probability distribution of random variables (transitions and rewards) depends on the current state and action only, and on the past actions and states.

Depending on the selected method, the RL agent may include policy function, value function and model. The latter is the agent's representation of the environment, according to its presence in the solution we can categorize RL methods as model-free or model-based solutions.

B. Policy-based Methods

The other approach is based on the approximation of the policy directly. From this group of RL solutions the policy-gradient methods have gained interest in recent years to solve control problems (see e.g. [7]). These algorithms modify the parameters of the function approximator (neural network) in the direction that maximizes the expected reward. Its advantage over the value-based methods is the guaranteed convergence, but typically to a local rather than global optimum [8].

Policy gradient assumes that the actions are given by a stochastic policy $\pi_\theta(a_t, s_t)$ with parameters θ. Policy-base RL is an optimization problem, where the goal is to find θ that maximizes $J(\theta) = J(\pi_\theta)$ where

$$J(\pi_\theta) = \mathbf{E}\left[\sum_{t=0}^{T} r_t\right] \tag{1}$$

in episodic environments. The policy gradient algorithms search for a local maximum in $J(\theta)$ by continuously refining a given parametrization vector. It follows the steepest ascent of the expected reward, which can be formulated by the gradient update rule:

$$\Delta\theta = \alpha \nabla_\theta J(\theta) \tag{2}$$

where $\nabla_\theta J(\theta)$ is the policy gradient vector and α is the learning rate.

C. Value-based Methods

The value function-based reinforcement learning is a heavily researched area thanks to Deepmind's improvements in deep Q-learning. The main idea is to learn a Q-function instead of the value function. Q-function is a predictor function which outputs a Q-value for each actions in a given state. The prediction can be updated according to the Bellmann equation:

$$Q(s_t, a_t) = r_t + \gamma \max Q(s_{t+1}, a) \tag{3}$$

The predictor function is typically a deep neural network where the training goal is the more accurate estimate of the Q-function according to the Bellmann equation. It should be noted that the usage of function approximation in RL algorithms do not guarantee the convergence, but these methods usually show good results in practice, in particular with regard to the improved versions, see e.g. [9] and [10].

D. Q-Learning method

Q-Learning has been used for the lane keeping function which presented in this paper. The applied Q-learning algorithm works as follows.

- At the beginning of learning, the agent tries to get the reward with random actions.
- The positive rewards or the penalties of the trials are stored in the array.
- An attempt series is called an episode. An element contains the action, the reward, the pre-action vehicle state, and the state of the vehicle after the action.
- At the end of each episodes the agent select 35 random sampled element from the array.

The Q-learning method (which trains the neural network) was implemented according to the following steps:

1) $Q_{\text{value}} = reward + \gamma \times$ *(highest probability from the neural network based on the state after the action)*
2) *target_f = actions array from neural network based on the pre-action vehicle state*
3) *target_f[action] = Q_value*
4) *train the neural network with the pre-action vehicle state and the modified actions array with Q value*

After each training decrease the randomness of agent's actions ($\epsilon - greedy$ method).

III. TRAINING ENVIRONMENT

As previously mentioned the agent needs an environment where it can act and learn to drive. Such environment must consist at least the following subsystems:

- Vehicle model, including vehicle dynamics, sensor model and actuation:
- Simplified major road model, describing topology, lane geometry etc.
- Reward function, where a positive value is given for the expected behavior and a negative value is given for undesired decisions.

A. Vehicle model

The model of the agent's vehicle is a rigid kinematic single-track vehicle model, where tire slip is neglected. Thus, lateral motion is only affected by the geometric parameters. More details about the model can be found in [11], [12]. This model seems quite simplified, and as stated in [13], such model can behave significantly different from an actual vehicle, though for the many control situations, the accuracy is suitable [12]. The main parameters of the model are: L-axle length. State parameters are x, y vehicle position, v longitudinal speed and ψ yaw angle. Control parameters are the acceleration demand, and the δ steering angle. By neglecting tire slip, the path radius R can be calculated as:

$$\tan \delta = \frac{L}{R} \tag{4}$$

Thus, the yaw rate is:

$$\dot{\psi} = \frac{v}{R} = \tan \delta \frac{v}{L} \tag{5}$$

B. Track generation

In the reinforcement learning-based systems for a more universal solution important is the generation of varied scenarios for the agent. To ensure this the environment contains a parametrizable track generator module with the basic concepts taken from [14].

As a first step, the algorithm generates 10 to 20 randomly spaced points to a previousy defined area, then covers it with a convex polygon getting the main holder points of the track. See Fig. 2.

In the next step, new points will be added to the existing points, in case there is enough distance between the two points altering intermediate points on a normal direction,

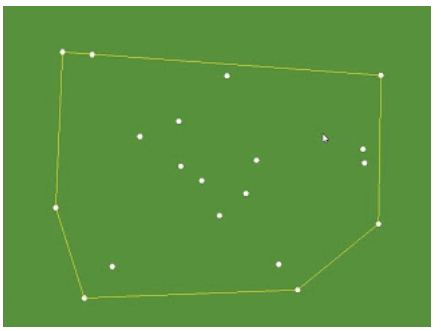

Fig. 2. Convex polygon

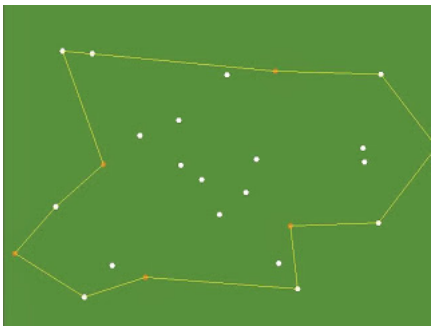

Fig. 3. Supplemented polygon

randomly as parametrized , which parameter will determine the difficulty of the track. See Fig. 3.

Finally, the algorithm interpolates the B-spline from the resulting points. The resolution of the curve can be parameterized. The more detailed are more accurate, but then the calculation of distances and angular deviations are more resource-intensive. The Fig. 4 shows tracks were generated by the algorithm. On the left is an easy track and the difficulty level increases to the right.

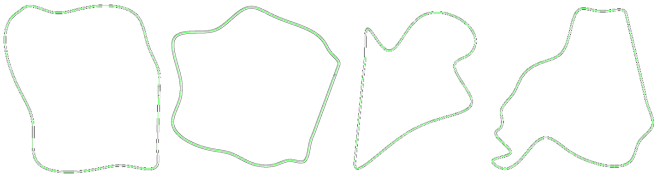

Fig. 4. Generated tracks

C. Environment sensing

The environment sensing model of the agent assumes high level output from automotive sensors, such as camera with IMU and odometry based systems. This way the environment information that is available for the agent is its position and heading relative to the lane (Yaw angle). See Fig. 5 These information provides the state input of the agent.

D. Simulation and Reward Calculation

At the beginning of the learning process, the environment generates easy tracks and with the evolution of the agent,

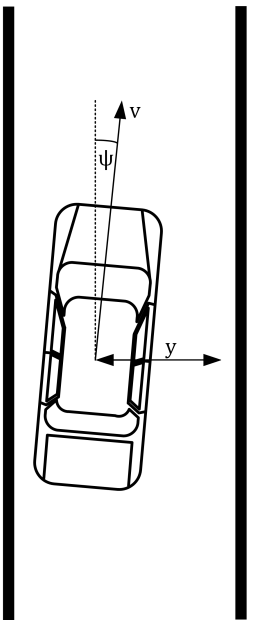

Fig. 5. Environment state on the track

the tracks are getting harder. In each training step, the agent receives the state vector, and determines its action, the steering demand. Then, the simulation actuates these inputs to the agent's vehicle. Each episode of the training process lasts as long as the vehicle doesn't reach the end of the track unless a terminating condition stops it.

To define the reward function for the agent, the following requirements were considered, from which terminating conditions are:

- The vehicle leaves the track.
- The yaw angle is too high.
- The agent passes by an appointed gate

Besides terminating conditions the angular and distance deviation requirements describe the quality features of the performance of the agent.

When a terminating condition rises, the episode is stopped, and the agent is given a negative reward ($R \approx -10$). The reward system was set up as follows: The environment defines 9 gates equally distributed on the track. The agent will only receive performance-based reward in case it passes through a gate.

$$R_{\text{gate}} = \begin{cases} 1 - \frac{dx}{4} - \frac{\psi}{5}, & \text{if } dx < 4 \text{ m and } \psi < 5^\circ \\ 0, & \text{if } dx \geq 4 \text{ m or } \psi \geq 5^\circ \end{cases} \quad (6)$$

Once the vehicle has passed through a gate then the next doorway will be + 10 % on the track (ie, if it has passed through the 10 % gate, it will receive a reward of 20 % gate). This way you can earn more rewards during a series of actions. In case of an incident the episode is terminated a negative value. The environment includes a reset method to restore the vehicle to its initial position.

IV. RESULTS

The machine learning algorithms usually use a lots of iteration. The success of the training process is highly affected by the hyperparameters of the algorithm. Hence, compared to supervised learning the reinforcement learning process may be more complicated.

In the present case the most significant hyperparameters are the learning rate (α), the discount factor (γ), the exploration decay factor (ξ) and the minimum exploration factor (ν). Further parameters are the maximum episodes number is a track and the penalty constant (μ) of a rewarding system. The hyperparameters of the neural network remained constant during the iterations. During the development process it became clear how strongly the reward function shape and parameters affect the learning and the results. After several iterations the chosen hyperparameters and reward function parameters are summarized in Table I.

TABLE I
HYPERPARAMETERS AND REWARD FUNCTION PARAMETERS

Parameter	Value
Learning rate (α)	0.001
Discount factor (γ)	0.95
Exploration decay (ξ)	0.995
Maximum episodes in a same track	500
Penalty constant (μ)	-1
Num. of hidden layers	1
Num. of neurons	30
Track width [m]	8
Gate width [m]	4

The following figures show the result after 3000 episodes training. After approx. 2500 episodes the vehicle circled 10 times in succession on a first track. In the training phase the action is selected by random choice according to the probabilities given by the exploration rate, which ensures the exploration explicitly. While in the testing phase the action is determined by greedy algorithm.

Fig. 6 shows the trend of the training reward, smoothed by moving average using window length of 10 episodes. The diagram shows that the maximum reward is set to a fixed value.

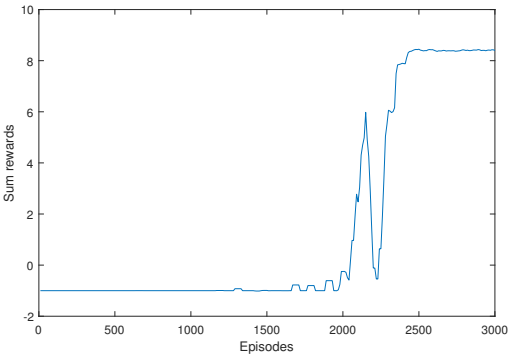

Fig. 6. Training rewards

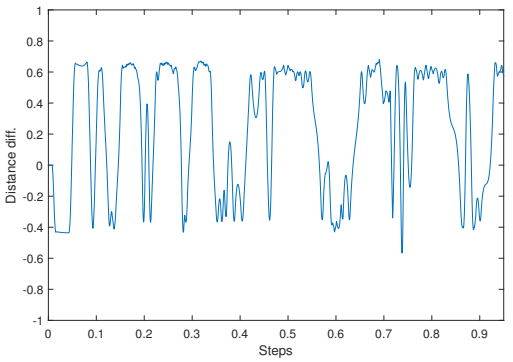

Fig. 7. Distance difference

The latter distance errors during the evaluation periods can be seen in Fig. 7. As it can be seen the error varies between -0.4 and 0.6.

V. CONCLUSIONS

The paper presented a possible end-to-end vehicle control based on high level sensor information, with the application of Q-Learning based reinforcement learning. The research showed that reinforcement learning is a good tool for designing basic autonomous functions, such as lane keeping.

As the initial step of the research, the algorithm shows convergence during the learning hence, the trained agent is basically capable to control a vehicle in a simplified environment. The visual inspection of the situations shows that the overall behavior of the agent fulfills the requirements.

Further research will focus on extending the environment with a traffic simulator, a realistic vehicle model and a continuous realistic road generator.

REFERENCES

[1] SAE International, *Taxonomy and Definitions for Terms Related to Driving Automation Systems for On-Road Motor Vehicles*, 2016.
[2] Z. Chen and X. Huang, "End-to-end learning for lane keeping of self-driving cars," in *2017 IEEE Intelligent Vehicles Symposium (IV)*, June 2017, pp. 1856–1860.
[3] V. e. a. Mnih, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, pp. 529 EP –, Feb 2015.
[4] D. e. a. Silver, "Mastering the game of go with deep neural networks and tree search," *Nature*, vol. 529, pp. 484 EP –, Jan 2016, article.
[5] ——, "Mastering the game of go without human knowledge," *Nature*, vol. 550, pp. 354 EP –, Oct 2017.
[6] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym," 2016.
[7] S. Aradi, T. Bécsi, and P. Gáspár, "Policy gradient based reinforcement learning approach for autonomous highway driving," in *2nd IEEE Conference on Control Technology and Applications*, 2018.
[8] R. S. Sutton, D. McAllester, S. Singh, and Y. Mansour, "Policy gradient methods for reinforcement learning with function approximation," *Advances in Neural Information Processing Systems*, vol. 12, pp. 1057–1063, 2000.
[9] H. van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double q-learning," *CoRR*, vol. abs/1509.06461, 2015.
[10] Z. Wang, N. de Freitas, and M. Lanctot, "Dueling network architectures for deep reinforcement learning," *CoRR*, vol. abs/1511.06581, 2015.
[11] O. Töro, T. Bécsi, and S. Aradi, "Design of lane keeping algorithm of autonomous vehicle," *Periodica Polytechnica. Transportation Engineering*, vol. 44, no. 1, p. 60, 2016.
[12] J. Kong, M. Pfeiffer, G. Schildbach, and F. Borrelli, "Kinematic and dynamic vehicle models for autonomous driving control design," in *2015 IEEE Intelligent Vehicles Symposium (IV)*, June 2015, pp. 1094–1099.
[13] P. Polack, F. Altché, B. D'Andréa-Novel, and A. De La Fortelle, "The kinematic bicycle model: a consistent model for planning feasible trajectories for autonomous vehicles?" in *IEEE Intelligent Vehicles Symposium (IV)*, 2017.
[14] G. Maciel. (2013) How to generate procedural racetracks. [Online]. Available: http://blog.meltinglogic.com/2013/12/how-to-generate-procedural-racetracks/