Available online at www.sciencedirect.com

ScienceDirect

IFAC PapersOnline CONFERENCE PAPER ARCHIVE

IFAC PapersOnLine 51-22 (2018) 429–434

Highway Environment Model for Reinforcement Learning ⋆

Tamás Bécsi * **Szilárd Aradi** * **Árpád Fehér** * **János Szalay** * **Péter Gáspár** *

* *Department of Control for Transportation and Vehicle Systems, Budapest University of Technology and Economics (e-mail: {becsi.tamas, aradi.szilard, feher.arpad, gaspar.peter}@mail.bme.hu, szalay.janos93@gmail.com)*

Abstract: The paper presents a microscopic highway simulation model, built as an environment for the development of different machine learning based autonomous vehicle controllers. The environment is based on the popular OpenAI Gym framework, hence it can be easily integrated into multiple projects. The traffic flow is operated by classic microscopic models, while the agent's vehicle uses a rigid kinematic single-track model, with either continuous or discrete action spaces. The environment also provides a simple high-level sensor model, where the state of the agent and its surroundings are part of the observation. To aid the learning process, multiple reward functions are also provided.

Keywords: Machine Learning, Reinforcement Learning Control, Autonomous Vehicles, Road traffic

1. INTRODUCTION

Reinforcement Learning (RL) has become an exciting area of machine learning. 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 Mnih et al. (2015), Silver et al. (2016) and Silver et al. (2017). 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, such 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 essentials for R&D and testing. On one side 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 realistic simulations framework can be built-up for self-driving car development. Nevertheless, these setups are unnecessarily complex in many cases furthermore it has costs which limit their usage in education and research. Several RL benchmark environments (see Section 2) exist, which provides video and board games and classic control and robotic tasks, but the vehicle control researchers lack RL environments.

Our motivation is to make simplified models for the most important traffic environments available, that provide the necessary interfaces for the RL software agents. The framework presented in this paper simulates the highway environment with configurable traffic flow and ideal sensor models.

Section 2 introduces the available RL benchmarks and environments. Section 3 describes the environment model in detail, then in Section 4 some RL experiments with our model are exposed. Finally, in Section 5 the conclusion, the possible improvements and the accessibility of the project are detailed.

2. REINFORCEMENT LEARNING SANDBOXES

If one would like to develop an RL environment model, we need to know the basic idea behind reinforcement learning. In artificial intelligence, the learner and decision maker algorithm is called the *agent*. Everything outside the agent is called the *environment*. The environment shall provide the following signals to the agent:

- state (output)
- action (input)
- reward (output)

⋆ 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

10.1016/j.ifacol.2018.11.596

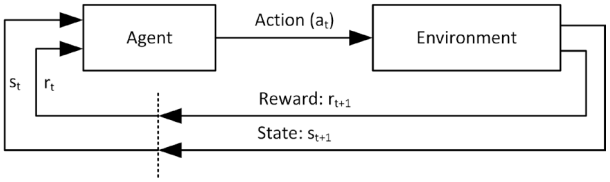

Fig. 1. Agent-environment interaction in reinforcement learning

The agent interacts with the environment continuously (see Fig. 1), the agent selects actions, and the environment sends back its new state representing the new situation. Furthermore, the environment provides information about how well the agent does its job. It is a scalar value called reward signal. In the following subsections, the most significant general and automotive RL training and benchmark environments will be introduced.

2.1 OpenAI Gym and Roboschool

OpenAI Gym of Brockman et al. (2016) is the most popular RL benchmark collection toolkit developed in Python by a non-profit AI research company. [1] It uses an episodic approach, which means that the agent's activity is broken into a number of timesteps. An episode may end (the environment reaches a terminal state) if the agent has reached the goal or the agent has made a mistake, or a predefined number of timesteps has been reached. When starting an episode, the environment's state is randomly sampled from a distribution.

Gym comes with a diverse collection of RL test problems from the areas of classic control, algorithmic problems, Atari games and 2D/3D robots. The robot simulations use the MuJoCo physics engine which requires a paid license. Thus OpenAI published Roboschool [2] in July 2017 as a free alternative to MuJoCo.

Gym provides only the environment, not the agent with a shared interface allowing the researchers to use general RL algorithms. OpenAI offers a well-defined and freely usable abstraction for developing new models, which (besides its popularity) makes Gym particularly suitable to create and share new RL environments.

2.2 Automotive Environments

In the automotive industry, one can choose from the several physics-based simulation platforms such as PreScan, CarSim, CarMaker, etc. These tools excellently support the development process of ADAS and V2V applications, from control design to real-time tests, offering many types of interfaces for third-party scenarios, sensor, vehicle and traffic flow modeling. These softwares primarily try to serve the requirements of the industrial sector, and they need paid licenses for academic applications as well.

In the open source world, one can choose a game engine with physics simulation and 3D graphics capability and develop its environment model from scratch. This solution can be advantageous in case of vision-based sensing because these engines provide very realistic 3D rendering.

[1] https://openai.com/about/
[2] https://blog.openai.com/roboschool/

The other approach is choosing a ready-to-use open source vehicle simulator. One of the most significant simulators is The Open Racing Car Simulator (TORCS) [3] which is an open source 3D car racing simulator. Its initial release was published in 1997. Besides its the ordinary car racing capability it can be used as a research platform. It provides a well-defined and detailed interface for sensing and actuating, but it offers only race track environment.

A very new and remarkable project is CARLA (Car Learn To Act) simulator (see Dosovitskiy et al. (2017)) developed by Intel Labs, Toyota Research Institute and Computer Vision Center Barcelona. It is an open simulator platform for urban driving. It has been built on Unreal Engine 4 as an open-source layer, which provides state-of-art rendering quality and detailed physics simulation. CARLA provides an interface for agents as a server-client system. The API is implemented in Python, thus it can be easily integrated with the existing machine learning systems. Its sensor system consists of different cameras and vehicle dynamics sensors, but the developers promise new sensor types (such as lidar) soon. This concept looks very promising for the academic community.

3. HIGHWAY ENVIRONMENT MODEL

The environment presented in this paper provides a highway traffic scenario for the learning agent. Since the reinforcement learning process takes a long time and requires multiple attempts from the agent, the underlying environment needs to be lightweight in terms of computational requirements. By keeping these restrictions in mind, such environment must consist at least the following subsystems:

- Vehicle model, including vehicle dynamics, sensor model and actuation
- Highway model, describing topology, lane geometry
- Traffic generation and behavior for environmental vehicles etc.

Naturally, the reward function should encourage the agent for the proper behavior, just like speed and lane choice with respect to the surroundings.

3.1 Vehicle model

For the model of the agent's vehicle, a rigid kinematic single-track bicycle model was chosen, where tire slip is neglected, thus lateral motion is only affected by the geometric parameters, and only longitudinal behavior considers the dynamics. The more detailed description of the model can be found in Törő et al. (2016) and Kong et al. (2015). This model seems quite simplified, and as stated in Polack et al. (2017), such model can behave significantly different from an actual vehicle, though for the many control situations, the accuracy is suitable as shown by Kong et al. (2015). The main parameters of the model are: L-axle length and m weight. 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:

[3] http://torcs.sourceforge.net/

$$tan\delta = \frac{L}{R} \tag{1}$$

Thus, the Yaw rate is:

$$\dot{\psi} = \frac{v}{R} = tan\delta \frac{v}{L} \tag{2}$$

3.2 Environment sensing

The environment sensing model of the agent assumes high-level output from common automotive sensors, such as camera and radar-based systems. This way the environment information that is available for the agent is its position and heading relative to the highway lane, and also the relative sates of the vehicles in the ego and adjacent lanes. This information together with the vehicle state provides the state input of the agent. Table 1 summarizes the state vector, while Fig. 2 gives an overview for better understanding. IDs $[0-9]$ provides relative distance and relative speed of the surrounding vehicles. In case, the lane is unoccupied or not exists (e.g., when the agent is in the rightmost lane) the $[dx, dv]$ distance and relative speed values are filled with $[500, 0]$. The IDs $[12, 13]$ show the occupancy of the lane on the two sides of the agent while $[14-16]$ give information about the lateral position relative to the mid-line of the rightmost lane, the vehicle's heading relative to the orientation of the highway and the vehicle speed.

Table 1. Environment state vector description

ID	Meaning	Elements
0,1	Front Left Lane	dx,dv
2,3	Front Ego Lane	dx,dv
4,5	Front Right Lane	dx,dv
6,7	Rear Left Lane	dx,dv
8,9	Rear Ego Lane	dx,dv
10,11	Rear Right Lane	dx,dv
12	Left Safe Zone	Occup [0,1]
13	Right Safe Zone	Occup [0,1]
14	Vehicle lateral position	y [m]
15	Vehicle heading	Ψ[rad]
16	Vehicle speed	v[m/s]

3.3 Traffic generation

The model considers a previously initialized constant lane count through the simulation, where traffic generation is also based on the previously determined density. The parameters are the mean and deviation of the generated traffic density on the lane, and also the mean and deviation of the desired speed of the generated vehicles on these lanes $[m(k_i), \sigma(k_i), m(v_i), \sigma(v_i)]$.

Traffic generation starts with an empty highway section, and for each lane, vehicles are generated to reach the desired traffic density, though afterwards the simulation keeps on going to produce a random situation. After this warm-up stage, a vehicle in the middle of the highway in a random lane is chosen to be the agent for learning.

Throughout the simulation the environmental vehicles follow multiple goals:

- Maintaining desired speed
- Avoiding collisions
- Overtaking if necessary

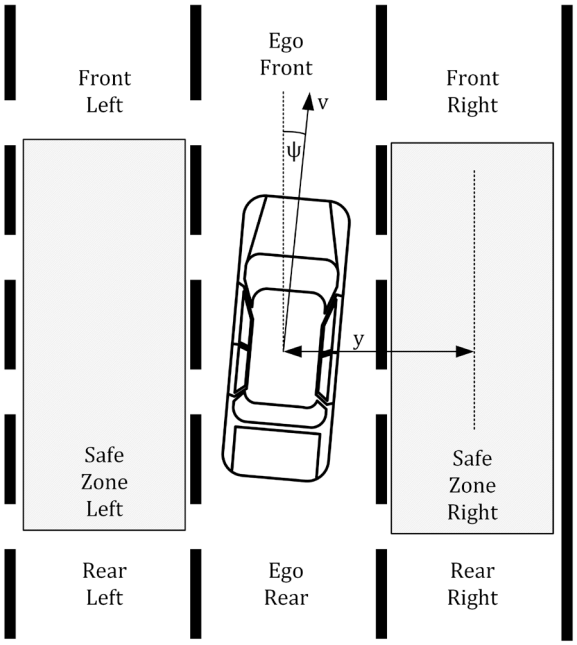

Fig. 2. Environment state on the highway

- Keeping right if possible

These goals are realized with a two-layer hierarchical control. On the first layer, a state machine chooses the high-level strategy, and for each state, a low-level controller is responsible for the actions. Though to reduce computational efforts, the behavior of the environmental vehicles lacks vehicle-dynamics and utilizes microscopic traffic simulation principles. The following four states describe the strategic level's choices:

- Stay in lane while maintaining desired speed and applying car-following rules
- Acceleration before overtaking to gain up to the speed of the target lane if necessary
- Switch lane to left in case of overtaking
- Switch lane to right in case it's possible

The car-following model is similar to the one described in Fang et al. (2001), where first, the desired following distance of the j_{th} vehicle on the i_{th} lane $d^j_{des,i}$ is determined based on the speed of the leading vehicle:

$$d^j_{des,i} = \alpha^j_i v^{j+1}_i, \tag{3}$$

where v^{j+1}_i is the speed of the leading vehicle in m/s and α^j_i is the sensitivity parameter with random values from $\mathcal{N}(1.3, 0.02)$. Afterward, a PD controller calculates the acceleration command of the car-following based on the following distance and relative speed:

$$a^j_{cf,i} = K_p(x^{j+1}_i - x^j_i) + K_d(v^{j+1}_i - v^j_i), \tag{4}$$

where K_p and K_d are the gains of the car-following controller. The final acceleration command of the environmental vehicles is the minimum of the car-following and the desired speed based accelerations, saturated by the physical constraints.

More complex behaviors, like an overtaking maneuver, builds up from these states, where the states can fall back

to the safe lane keeping behavior. When a vehicle wants to overtake, it merely needs to change to the right, though in a dense traffic situation, the safe lane changing has its prerequisites. The target lane section needs to be empty, the distance and relative speed of the following vehicle and also of the leading vehicle in that lane need to allow this maneuver.

In most of the cases under dense flow, the speed difference of the neighboring lanes prevents the lane change. In order to eliminate this disadvantage, the vehicle applies the acceleration state, to reduce the gap of the vehicle in front, and to keep up with the speed of the vehicles in the rightmost lane. This gap closing behavior enables the vehicle to approach the leading vehicle much closer than in the car following situation. Naturally, this behavior is only applied when the leader's speed is lower than the desired speed of the follower, and the gap is eligible for the acceleration.

Changing lane to the right has similar logic, the eligibility of the target gap is evaluated with respect to the surrounding vehicles' states. The only difference is that the reasonableness of the maneuver also depends on the time that the vehicle can spend in the rightmost lane without the need of deceleration or changing back to its current lane.

3.4 Interacting with the environment

The environment's step function takes the ego-vehicle action as an input command, which can be configured as discrete action space for value-based methods, or continuous for generic usages. The function, after conducting one simulation step, returns the following data:

- Agent state, as described above;
- The immediate reward for the current step;
- Boolean information about the termination of the run and
- Informal details about the termination (e.g. front-collision)

For each step, the simulation actuates the actions to the agent's vehicle and also simulates one step for each environmental vehicle on the examined highway section. Since this way vehicles can leave the examined highway section, the actual traffic density is compared to its prescribed value, and new vehicles are generated on the edges of the highway. The maximal episode length of the simulation can be configured, though an episode can be terminated in case an undesired event occurs.

The early terminating conditions are not permitted events, from which the simulation cannot be continued. These conditions are the following:

- Leaving the highway;
- Causing collision;
- Stopping or reaching a previously defined low speed.

When a terminating condition rises, the episode is stopped, and the agent is given a high negative reward.

Most reinforcement learning problems can be described as episodic, where the result and thus the reward is determined at the end of the game, or the reward is given eventually when a subtask is achieved. Contrary, in case of highway driving, the actual quality of the actions are also crucial in each step. The quality indicators are the followings:

- The vehicle should not travel between the lanes for a long time.
- The agent should travel with its desired speed, when possible.
- The vehicle should keep right.
- The vehicle should not approach other vehicles over a safe margin.

For these quality measurements, the environment provides some previously defined reward functions.

Keeping right is checked in a way that examines the adjacent right lane, based on the values of "Right Safe Zone occupied" $ID[13]$ and Right Lane distance dx, $ID[4]$ from the state vector. In case, the lane is occupied in the previously defined close range d_l, the rule's reward is $(R_l = 1)$, in case there is no vehicle in far range d_h, $R_l = 0$, otherwise the reward gets a value based on the distance of the vehicle:

$$R_l = \begin{cases} 1, & \text{if } dx < d_l \\ 0, & \text{if } dx > d_h \\ 1 - \dfrac{dx - d_l}{d_h - d_l}, & \text{otherwise} \end{cases} \tag{5}$$

Staying on the highway is rewarded considering the lateral distance from either side of the highway y_d, with thresholds $[y_l, y_h]$:

$$R_y = \begin{cases} 0, & \text{if } y_d < y_l \\ 1, & \text{if } y_d > y_h \\ 1 - \dfrac{dx - d_c}{d_f - d_c}, & \text{otherwise} \end{cases} \tag{6}$$

Keeping the desired speed is rewarded by the trapezoid shaped function, considering the speed of the agent v, the desired speed v_d with thresholds $[v_l, v_h]$:

$$R_v = \begin{cases} 0, & \text{if } |v - v_d| > v_h \\ 1, & \text{if } |v - v_d| < v_l \\ 1 - \dfrac{|v - v_d| - v_l}{v_h - v_l}, & \text{otherwise} \end{cases} \tag{7}$$

Finally approaching neighboring vehicles is also considered, in this case, critical approaching of the leading vehicle, or the vehicles in the neighboring lanes are penalized, resulting in a reward for keeping safe distance from the others: R_c

The final reward is calculated as the weighted average of the $[R_y, R_l, R_v, R_c]$ with parameters $[\alpha_y, \alpha_l, \alpha_v, \alpha_c]$:

$$R = \alpha_y R_y + \alpha_l R_l + \alpha_v R_v + \alpha_c R_c, \text{ where} \tag{8}$$

$$\alpha_y + \alpha_l + \alpha_v + \alpha_c = 1 \tag{9}$$

4. REINFORCEMENT LEARNING EXPERIMENTS

The focus was on test learnings in order to gain experience after the development of the highway environment. Following the agent-based approach, a learning agent has to

learn the successful and effective behavior in our Python-based highway environment. The first goal of the learning is the collision-free driving and the second one is striving for regularity and efficiency.

4.1 Reinforcement learning method

High-dimensional or continuous action spaces are preferred for the appropriate control of the ego-vehicle. The policy-based reinforcement learning methods effectively meet this requirement, furthermore they have good convergence properties. We used 'Vanilla' policy gradient method for learning experiments (see Peters and Schaal (2006)) with discrete actions. The aim of the training is to minimize the loss function to calculate policy weights. A multi-layer neural network represents the policy function. The learning process consists of a series of episodes. The episodes consist of steps. Each result of steps provides a reward value. The rewards determine the goodness of the actions. At the end of each episode, the algorithm accumulates the gradients using the bellow loss function:

$$L(\theta) = -\sum r \log \pi(a|s;\theta) \tag{10}$$

The neural network is updated using accumulated gradients. The gradients are cumulated to the number of episodes according to the frequency parameter. The reward values have been discounted and normalized per episodes in order to reduce the policy gradient variance since noisy gradient updates might destabilize the policy's convergence.

4.2 Training

During the training, the maximum step size limited the maximum length of the episode. As a consequence, we figured out how many steps were needed for an evaluable maneuver. Every step started with an action, ended with a reward value and set a new state. The action space consisted of 25 elements. All combinations of [-0.003,-0.0005,0,0.0005,0.003] steering and [-6.0,-2.0,0.0,2.0,3.5] accelerating values arrayed. The reward calculation uses the following weights: $[\alpha_y = 0.2, \alpha_l = 0, \alpha_v = 0.5, \alpha_c = 0.3]$. Therefore, the total reward equal the maximum step size limit in an episode. In order to make the training faster and reduce the chances of getting stuck in the local optima, the states are normalized. The table 2 summarizes the training parameters of our experimental learning.

Table 2. Training setup parameters

Parameter	Value
Number of states	17
Number of actions	25
Number of hidden layers	2
Neurons / layer	512
Learning rate	0.0001
Max. steps / episode	500
Total episodes	250000
Gradients apply frequency	5 [episode]
Discount factor	0.95
Reward normalization	True
State normalization	True
Training device	CPU

The teaching was performed on CPU and it took 44 hours.

The chart 3 shows the evolution process of the 250000 episodes training. It presents averaged total reward per 100 episodes. At the beginning of the training process, (in the first 50000 episodes) the agent found a good survival strategy, hence the number of large negative rewards decreases rapidly. After this section, the agent seeks to maximize total reward values. At the end of the learning phase, average values of the total rewards converge to approximately 480.

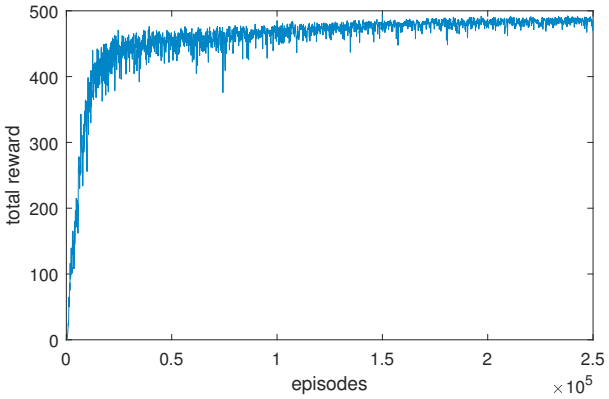

Fig. 3. 250000 episode training

4.3 Evaluate

The exploration was reduced to zero in the evaluation process. The neural network strictly determines the agent's response to a state. This phase aims to evaluate the operation of the learned agent that has two parts. Firstly, we observed the agent's driving style with using model rendering. Secondly, we have produced statistics about unexpected events.

Evaluation has been integrated with a rule-based decision check that helps to prevent unexpected collisions.

4.4 Results

The results of the evaluation are the following:

Table 3. Results of the evaluation of 10000 episodes

Cause	How many times
No collision	9978
Low speed	0
Left highway	0
Front collision	9
Rear collision	13

The average reward value of the 10000 episodes was 478.9 and there took place 9 front and 13 rear collisions. The agent kept the desired 130 km/h speed effectively. In order to keep this speed, the agent overtakes if it is necessary. Therefore, keeping the desired speed causes lot of overtakes. In case the lanes are crowded and overtaking is impossible, the agent keeps a safe tracking distance. At the close to the lane changing states, it goes too close to the vehicles in the other lane. It can cause collisions.

Sometimes the agent has the opportunity to change the lane, but it does not try, just follows the front vehicle slower than the required speed.

5. CONCLUSIONS

In our paper, an open-source highway simulator has been presented. It can be used as an OpenAI Gym extension, which makes it suitable for easy interfacing with different state-of-the-art reinforcement learning algorithms. The simulations and the reinforcement learning tests showed that the framework provides a powerful environment for AI-based learning of highway behavior on different levels. The initial release will be upgraded with several useful features (traffic signs, curves, ramps, etc.) in the future. The tool provided can aid university lectures with simple examples, though it can be a base for high level research projects also. We provide the source code of the environment for the research community as is, free to use. The code is released open-source at *https://github.com/szilu7/gym-highway*.

REFERENCES

Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., and Zaremba, W. (2016). Openai gym.

Dosovitskiy, A., Ros, G., Codevilla, F., Lopez, A., and Koltun, V. (2017). CARLA: An open urban driving simulator. In *Proceedings of the 1st Annual Conference on Robot Learning*, 1–16.

Fang, X., Pham, H.A., and Kobayashi, M. (2001). Pd controller for car-following models based on real data. In *Proceedings of 1st Human-Centered Transportation Simulation Conference*, volume 2001.

Kong, J., Pfeiffer, M., Schildbach, G., and Borrelli, F. (2015). Kinematic and dynamic vehicle models for autonomous driving control design. In *2015 IEEE Intelligent Vehicles Symposium (IV)*, 1094–1099. doi: 10.1109/IVS.2015.7225830.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A.A., Veness, J., Bellemare, M.G., Graves, A., Riedmiller, M., Fidjeland, A.K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., and Hassabis, D. (2015). Human-level control through deep reinforcement learning. *Nature*, 518, 529 EP –. URL `http://dx.doi.org/10.1038/nature14236`.

Peters, J. and Schaal, S. (2006). Policy gradient methods for robotics. In *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2219–2225. doi: 10.1109/IROS.2006.282564.

Polack, P., Altché, F., D'Andréa-Novel, B., and De La Fortelle, A. (2017). The kinematic bicycle model: a consistent model for planning feasible trajectories for autonomous vehicles? In *IEEE Intelligent Vehicles Symposium (IV)*.

Silver, D., Huang, A., Maddison, C.J., Guez, A., Sifre, L., van den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., Dieleman, S., Grewe, D., Nham, J., Kalchbrenner, N., Sutskever, I., Lillicrap, T., Leach, M., Kavukcuoglu, K., Graepel, T., and Hassabis, D. (2016). Mastering the game of go with deep neural networks and tree search. *Nature*, 529, 484 EP –. URL `http://dx.doi.org/10.1038/nature16961`. Article.

Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., Hubert, T., Baker, L., Lai, M., Bolton, A., Chen, Y., Lillicrap, T., Hui, F., Sifre, L., van den Driessche, G., Graepel, T., and Hassabis, D. (2017). Mastering the game of go without human knowledge. *Nature*, 550, 354 EP –. URL `http://dx.doi.org/10.1038/nature24270`. Article.

Törő, O., Bécsi, T., and Aradi, S. (2016). Design of lane keeping algorithm of autonomous vehicle. *Periodica Polytechnica. Transportation Engineering*, 44(1), 60.