Learning to Drive with Deep Reinforcement Learning

Nut Chukamphaeng
Faculty of Information Technology
King Mongkut's Institute of Technology Ladkrabang
Bangkok, Thailand
60070134@kmitl.ac.th

Kitsuchart Pasupa*
Faculty of Information Technology
King Mongkut's Institute of Technology Ladkrabang
Bangkok, Thailand
kitsuchart@it.kmitl.ac.th

Martin Antenreiter
Department of Mathematics and Information Technology
Montanuniversität Leoben
Leoben, Austria
martin.antenreiter@unileoben.ac.at

Peter Auer
Department of Mathematics and Information Technology
Montanuniversität Leoben
Leoben, Austria
auer@unileoben.ac.at

***Abstract*—Autonomous driving cars are important due to improved safety and fuel efficiency. Various techniques have been described to consider only a single task, for example, recognition, prediction, and planning with supervised learning techniques. Some limitations of previous studies are: (1) human bias from human demonstration; (2) the need for multiple components such as localization, road mapping *etc.* with a complicated fusion logic; (3) in reinforcement learning, the focus was mostly on the learning algorithms but less on the evaluation of different sensors and reward functions. We describe end-to-end reinforcement learning for an autonomous car, which used only a single reinforcement learning model to create the autonomous car. Further, we designed a new efficient reward function to make the agent learn faster (18% improvement for all settings compared to the baseline reward function) and build the car with only the necessary perceptions and sensors. We show that it performed better with state-of-the-art off-policy reinforcement learning for continuous action (SAC, TD3).**

I. Introduction

Autonomous driving cars will significantly change urban mobility. They have not only high impact on the environment [1–3] but also to society, for example, good productivity gains while commuting [1, 4, 5], reducing drivers' stress and road accidents [6], and decreasing requirements for parking space [7]. Also, it can decrease 38 hours of commuting time per individual per year [8]. Due to advances in computer hardware technology, the capability of artificial intelligence has improved over recent years, especially deep learning. Deep learning is a subset of machine learning, that enables the computer to learn and improve performance automatically. It uses a layered structure of algorithms—an artificial neural network—inspired by the neural network of the human brain. Most current autonomous driving cars used multiple machine learning algorithms to learn how to drive [9].

Machine learning algorithms can be divided into three major types: supervised, unsupervised and reinforcement learning [10]. A supervised learning algorithm trains a model on labelled data, for example, localization [11, 12] and road mapping [13]. This approach is laborious as data collection is required. It can lead to a biased model, due to the innate nature of biases, within humans when labelling the data. Unsupervised learning trains with unlabeled data. It requires many samples to understand the patterns and properties of the data, *e.g.* to learn to group samples. Unsupervised learning techniques are used in autonomous driving cars, *e.g.* anomaly detection [14], generating synthetic data with generative adversarial networks [15]. Reinforcement learning involves taking decisions to perform suitable actions, by maximizing reward in a specific situation [16].

Many researchers have tried to use reinforcement learning to control an autonomous driving car [3, 9, 17–19]. They showed that using reinforcement learning in an autonomous car is simpler than supervised learning, because supervised learning requires multiple models to make the car work. Reinforcement learning has shown great potential in various challenging scenarios, that require both dynamic modelling and long term planning, *e.g.* game playing [20], real-time advertisement bidding [21, 22] and neural network structure searching [23, 24]. Reinforcement learning falls into two types by policy learning—on-policy learning and off-policy learning. An on-policy agent learns from observed rewards, generated from some policy. Usually on-policy learning is more efficient. Reinforcement learning is considered for discrete and continuous domains. For driving cars, a continuous action space seems appropriate. Some researchers reduced the continuous action space for car driving to a discrete one [25]. However, we cannot clearly divide all the actions into discrete ones. It might lead to a sub-optimal sequence of actions.

Some previous work focused on developing reinforcement learning for autonomous cars in computer games [9, 18, 26], which had no focus on the perception and sensor aspect. In contrast, we developed reinforcement learning for an autonomous car with sensory input in a simulator. Here, we focused on off-policy learning, because we wanted to enable the agent to be quickly trained. Also, a continuous action space was used as it is more realistic. Our contributions are:

i) We implemented end-to-end deep reinforcement learning

to drive the model, instead of supervised learning with multiple models in a custom environment.

ii) We compared two state-of-the-art off-policy continuous control algorithms, one with stochastic and one with deterministic polices.

iii) A new reward function for learning to drive was used. It yielded better results than the baseline reward function.

iv) We investigated two types of perception, which were distance sensors and a front camera: both perceptions were useful for autonomous driving cars.

II. Related work

Autonomous cars have been an extremely active research area in robotics, artificial intelligence and control. Many papers have discussed methods to control the car automatically, but they require many components for full control. Creating an autonomous agent requires three main tasks—recognition, prediction and planning [18, 26].

Recognition identifies the surrounding environment, *e.g.* localization [11, 12], offline obstacle mapping [27], road mapping [13], *etc*. This task is relatively well understood, thanks to advances in machine learning algorithms, which have reached human-level recognition in some detection and classification problems [28, 29]. Next, prediction is required. After we recognized the environment, we still need to build internal models, that can predict the future state of the environment. This is followed by planning, that enables the car to navigate successfully. It aims to generate an efficient sequence of driving actions.

As the traditional approach required many components, it is necessary to have a well-designed framework to combine all parts. The end-to-end concept tackled this issue, by replacing the handcrafted processes, with an automatically learnt feature representation module, using deep learning techniques. Recently, Bojarski *et al.* [30] used end-to-end learning for an autonomous car. They trained a convolutional neural network (CNN) [31] to map raw pixels from a single front-facing camera directly to steering commands. Surprisingly, this approach was powerful, but it needed a lot of effort in collecting the data from human demonstration. The limitation of this approach was that not every car has a front camera and can collect the steering commands data. Chopra and Roy [9] used an end-to-end reinforcement learning to learn how to drive. They used the Deep Q-network algorithm, with discrete action to train an agent from images. Chishti *et al.* [25] used CNN to classify objects in the environment, *e.g.* road signs and traffic lights. They combined them to define a reward function. Then they trained a model with reinforcement learning in conjunction with their reward function in discrete settings. Kadam *et al.* [19] compared imitation learning [32] with a deep deterministic policy gradient (DDPG) method, a reinforcement learning algorithm for continuous settings.

Thus, some works have discussed end-to-end reinforcement learning for autonomous cars, but they only considered images as input, without any additional perception or sensors. Moreover, they did not pay much attention to the reward function.

III. Methodology

This section explains an off-policy reinforcement learning for continuous actions to learn how to drive in the simulator, that we built (see Subsection III-A), and corresponding reward (see Subsection III-B).

A. Reinforcement Learning

The reinforcement learning agent learns to achieve a goal in an uncertain environment without a supervisor. It interacts with the environment and used trial and error to learn an optimum policy for sequential decisions, which, in turn, maximizes cumulative reward. The essential underlying model of reinforcement learning is a Markov Decision Process (MDP) [33]. An MDP is defined as a tuple (S, A, P, R, γ), where S is a state-space, A is an action space and P is a state transition probability function, that define the probability to move to the next state, $s' \in S$, when we observe state $s \in S$ and choose the action $a \in A$. For continuous actions or state spaces the mathematical definition is more complicated. R is a reward—$S \times A \times S \mapsto \mathbb{R}$ and $\gamma \in [0, 1]$ is a discount rate.

The objective of an agent in MDP is to find an optimal (probabilistic) policy, π_θ: $S \times A \mapsto [0, 1]$, that maximizes the expected cumulative rewards from any state s:

$$V^*(s) = \max_{\pi_\theta} \mathbb{E}[\sum_{k=0}^{\infty} \gamma^k r_{t+k} | s_t = s]. \quad (1)$$

It is equivalent to maximizing the expected cumulative rewards from any state-action pair, $s \in S$ and $a \in A$:

$$Q^*(s, a) = \max_{\pi_\theta} \mathbb{E}[\sum_{k=0}^{\infty} \gamma^k r_{t+k} | s_t = s, a_t = a] \quad (2)$$

In our problem, a state s represents a perception of car sensor, *e.g.* a front camera or distance sensors. An action, $a \in \mathbb{R}$, is a continuous parameter, that represents the steering angle. Transition probability P is a probability of interaction between state and action of the car. Based on action a and the perception s, we provided feedback to the algorithm with reward function R. Discount rate γ is a factor measuring the present value of long-term rewards. If $\gamma = 0$, the agent considered only immediate rewards, but ignores long-term rewards, and when $\gamma = 1$, the agent considered immediate and long-term rewards to be equal. Fig. 1 illustrates the agent interaction with the environment.

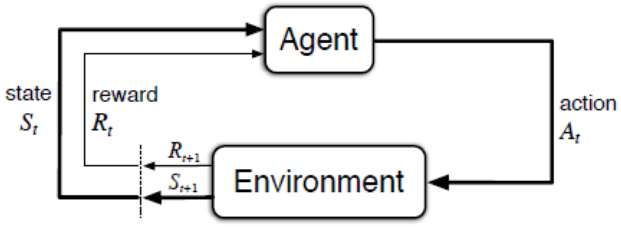

Fig. 1: Markov Decision Process

There are many reinforcement learning algorithms, for example, Q-learning [20], Policy Gradient [34] and Black-box Optimization [35]. We focused on two state-of-the-art off-policy algorithms, that were designed for continuous action.

1) Twin-delayed deep deterministic policy gradient (TD3): TD3 is a model-free, online algorithm, that uses a deterministic policy to choose the actions [36]. It is an extension of DDPG [37], and it explores by adding some noise to the policy.

2) Soft Actor-Critic (SAC): It uses stochastic policy optimization and a DDPG-style technique. It uses entropy regularization to trade exploration against exploitation. The action is sampled from the stochastic distribution.

B. Reward function

We used the 'Collision Penalty', $Reward_C$, as a baseline reward function:

$$Reward_C = -50C, \tag{3}$$

where C is a binary: if it is "1", a collision was detected.

Here, we describe the "Direction Guided" reward function, that enables the agent to learn to drive faster than the baseline in our environment. If the target direction of the car is known, this can be used in the reward function. If the car goes in that direction, a high reward is received. In contrast, if the car goes in the opposite direction, it would get a low reward or penalty.

Fig. 2 explains the fundamental of the proposed reward function. Assuming that a car is driving with a velocity of μ. It is heading toward a direction at $\Delta\theta$ angle to the direction in which the car should head in the environment. We can convert from the magnitude/angle way of specifying a vector to the coordinate way of expression. Therefore, the x-axis equals $\mu\cos(\Delta\theta)$ and the y-axis equals $\mu\sin(\Delta\theta)$. The car should not head toward y-axis, thus $\mu\sin(\Delta\theta)$ should be as low as possible. On the other hand, the car should head toward x-axis, therefore $\mu\cos(\Delta\theta)$ should be as high as possible. This idea led to this reward function,

$$Reward_D = \mu\cos(\Delta\theta) - |\mu\sin(\Delta\theta)|. \tag{4}$$

The direction guided reward values are represented in Fig. 3. The circumference indicates the angle of the car in respect to the desired direction, $\Delta\theta$. The distance from the centre of the circle represents the speed, μ. Red implies a negative reward, whereas blue implies a positive reward.

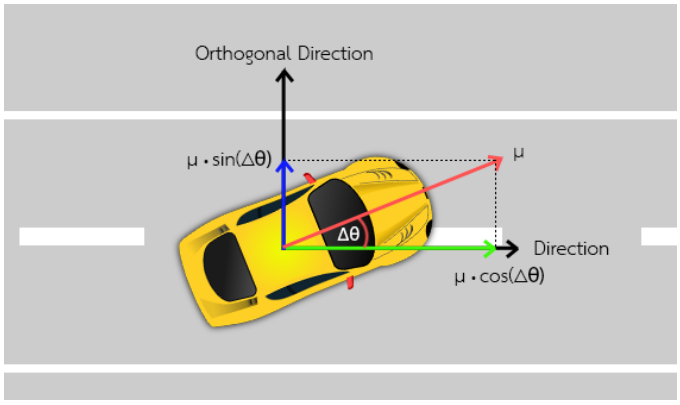

Fig. 2: The fundamental of the proposed reward function

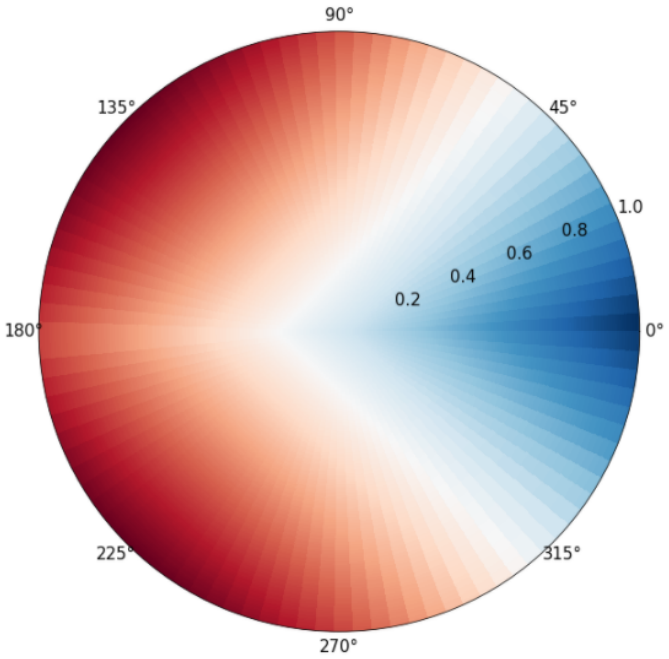

Fig. 3: Direction guided reward function

IV. Experiments

We compared performances of the two state-of-the-art off-policy continuous control algorithms, which were based on a stochastic policy, SAC, and a deterministic policy, TD3. We also compared the performance of the model with different reward functions and different types of perception.

A. Simulator

We created a custom environment from scratch with PyBullet [38]. Fig. 4 shows the simple race track used here. We investigate the model performance when additional perception was added. Thus, we created three different agents, labelled:

i) Distance sensor car, with seven sensors $-90°$, $-60°$, $-30°$, $0°$, $30°$, $60°$, and $90°$ as shown in Fig. 5. Each sensor returns a value between 0 and 1.
ii) Front camera car: We used a single 20×20 pixel 'image'. Example images are shown in Fig. 6).
iii) Combined perception car: We simply used all the seven sensors and the front camera.

Note: image from the front camera is a 20×20 pixel. It is fed into a CNN to be feature extracted, implemented in IMPALA [39].

B. Experiment Settings

We split the experiment into two parts:

a) Evaluation of algorithms and reward functions: We first focus on the distance sensor car. Agents, trained by SAC and TD3, were compared. Both algorithms were used in conjunction with the collision penalty and direction guided reward functions. We set the maximum number of step of the environment allowed in each epoch of the algorithm to 10,000 steps.

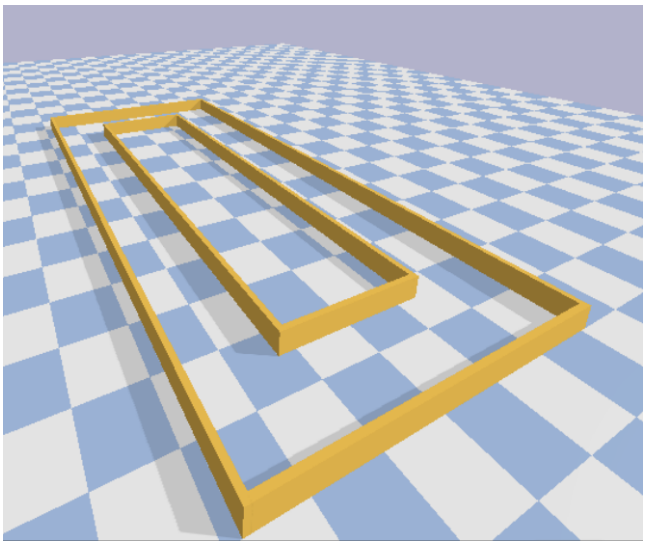

Fig. 4: Simple race track

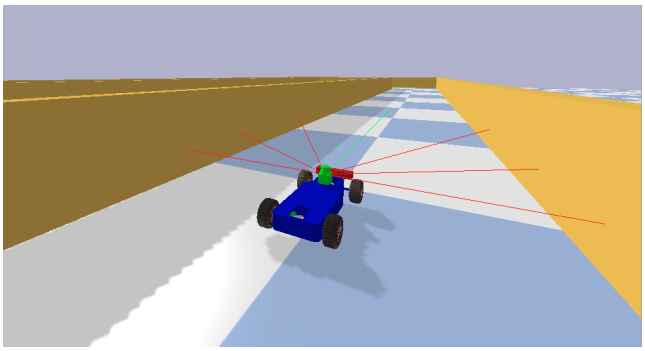

Fig. 5: Seven distance sensors

b) Comparing the perception of the agent: We compared the three different agents, discussed in Section IV-A, trained with SAC and TD3 in conjunction with the direction guided reward function.

We set the learning rate for each algorithm to 0.0003. $\gamma = 0.9$. Replay buffer size was set to 10^6 and the target network update, $\tau = 0.05$. The experiments were implemented in Python. The source code can be downloaded from https://github.com/nutorbit/crazycar [40].

V. Results and Discussions

To compare the reward functions, we monitored the speed of convergence of each reward function because two rewards were on a different scale. Therefore, we observed the time-step achieved in one epoch, as shown in Fig 7. Both SAC and TD3, in conjunction with the direction reward function, converged faster than with the collision penalty reward function. This is because our new reward function considered the direction of the car for penalty the agent. On the other hand, the collision penalty did not consider the direction, but only added a cost when a collision was detected. Also, SAC converged faster than TD3 with both penalty functions. This may be because TD3 did not explore widely, because it used a deterministic

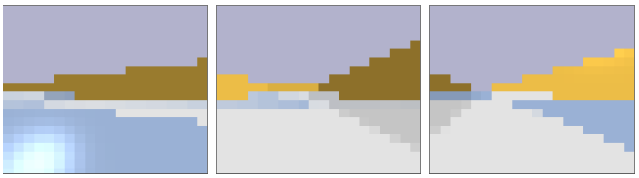

Fig. 6: Example images from the front camera of the car

policy. To explore widely, we normally add some noise to the policy to encourage new actions. However, adding too much noise might lead the algorithm more difficult to exploit. On the other hand, SAC uses a stochastic policy. It balances between exploration and exploitation with entropy regularization.

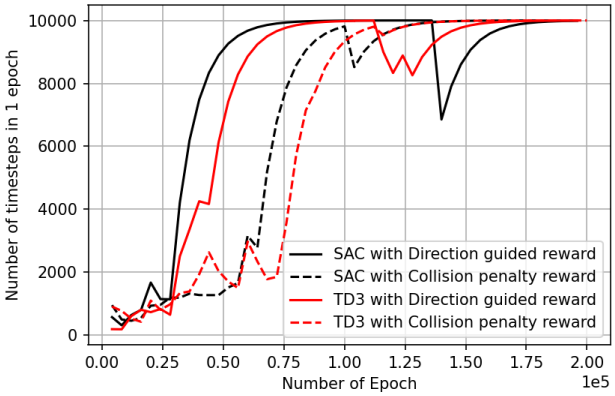

Fig. 7: Number of time steps in one epoch achieved by SAC and TD3 in conjunction with the baseline and our new reward functions. One epoch indicates 1 simulation. When the number of epoch increases, the car can spend more time step to run in a circuit. The drops indicate that the agents tried to explore.

Then, we studied three different agents—distance sensor, front-camera, combined perception cars—with TD3 and SAC in conjunction with the new reward function. Fig. 8 shows the reward achieved in one similar for SAC (top) and TD3 (bottom). The agent with distance sensors performed better in both SAC and TD3. The worst agent used a front camera alone. Also, the agent with the front camera, trained with TD3, failed to perform as it could not learn anything—the perception from the front camera may be attributed to its complexity, *i.e.* it was too complicated to learn. Combining distance sensors with a front camera slowly converged to about the same level as the distance sensor agent only with SAC algorithm. Training a combined perception agent with TD3 failed to learn, similar to TD3 with a front camera.

TABLE I: Average number of time steps required to finish one lap. "N/A" indicates that the algorithm did not finish the lap.

Algorithm	Reward	Perception		
		Distance Sensor	Front Camera	Combine
SAC	$Reward_C$	517.5	531.3	540.1
	$Reward_D$	**436.4**	450.1	456.8
TD3	$Reward_C$	538.7	N/A	N/A
	$Reward_D$	456.8	N/A	N/A

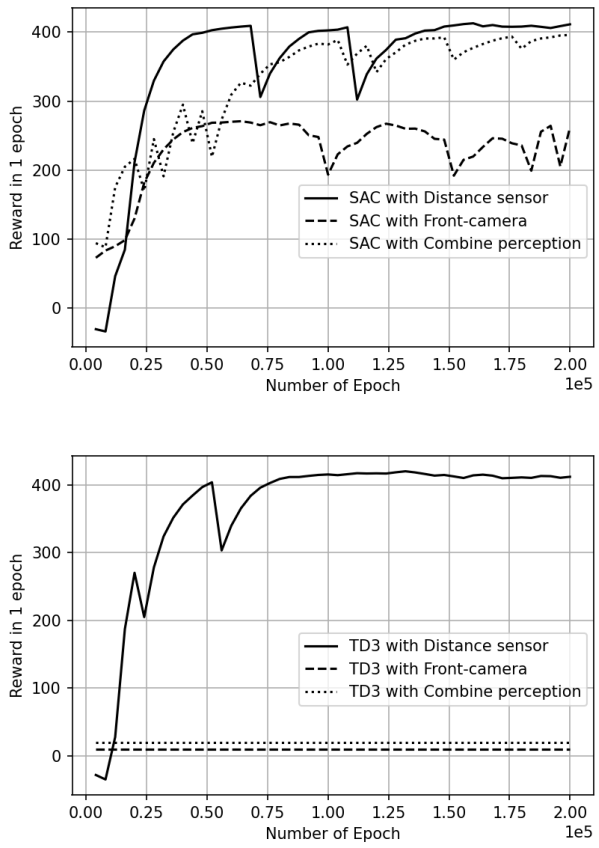

Fig. 8: Rewards from agents with different perceptions trained with SAC (top) and TD3 (bottom)

Table I shows the number of time steps required to finish one lap. Our new reward function performed better in all cases. Also, the number of time steps was proportional to the dimension (i.e. the number of sensor inputs) of perception: the dimension of an agent with distance sensors only was 7 and with the front camera it was 400—the number of 'pixels'. We found that using only the distance sensors was adequate to drive in our environment, as it is simple. In addition, the stochastic policy was better than the deterministic one in our settings.

VI. Conclusion

We described end-to-end deep reinforcement learning to drive a car in a simulator. The stochastic policy, SAC, achieved better performance than deterministic policy, TD3. Also, our new reward function was better than the baseline in our settings. We also showed that using only distance sensors was enough to train an agent in our settings. In future work, we plan to extend to a multi-agent system. Therefore, adding more perception or front cameras to the car might be necessary.

Acknowledgment

This work was supported by ASEAN-European Academic University Network [grant number ICM-2018-11794].

References

[1] W. Payre, J. Cestac, and P. Delhomme, "Intention to use a fully automated car: Attitudes and a priori acceptability," *Transportation Research Part F: Traffic Psychology and Behaviour*, vol. 27, pp. 252–263, 2014.

[2] T. Luettel, M. Himmelsbach, and H.-J. Wuensche, "Autonomous ground vehicles—concepts and a path to the future," *Proceedings of the IEEE*, vol. 100, no. Special Centennial Issue, pp. 1831–1839, 2012.

[3] B. R. Kiran, I. Sobh, V. Talpaert, P. Mannion, A. A. A. Sallab, S. Yogamani, and P. Pérez, "Deep reinforcement learning for autonomous driving: A survey," 2020.

[4] S. L. Vine, A. Zolfaghari, and J. Polak, "Autonomous cars: The tension between occupant experience and intersection capacity," *Transportation Research Part C: Emerging Technologies*, vol. 52, pp. 1–14, 2015.

[5] A. H. Jamson, N. Merat, O. M. Carsten, and F. C. Lai, "Behavioural changes in drivers experiencing highly-automated vehicle control in varying traffic conditions," *Transportation Research Part C: Emerging Technologies*, vol. 30, pp. 116–125, 2013.

[6] C. M. Rudin-Brown and H. A. Parker, "Behavioural adaptation to adaptive cruise control (acc): implications for preventive strategies," *Transportation Research Part F: Traffic Psychology and Behaviour*, vol. 7, no. 2, pp. 59–76, 2004.

[7] A. Alessandrini, A. Campagna, P. D. Site, F. Filippi, and L. Persia, "Automated vehicles and the rethinking of mobility and cities," *Transportation Research Procedia*, vol. 5, pp. 145–160, 2015, sIDT Scientific Seminar 2013.

[8] M. Bartl, "The future of autonomous driving—introducing the foresight matrix to support strategic planning," *The Making-of Innovation*, April 2015.

[9] R. Chopra and S. S. Roy, "End-to-end reinforcement learning for self-driving car," in *Advanced Computing and Intelligent Engineering*, B. Pati, C. R. Panigrahi, R. Buyya, and K.-C. Li, Eds. Singapore: Springer Singapore, 2020, pp. 53–61.

[10] G. Bonaccorso, *Machine Learning Algorithms: A Reference Guide to Popular Algorithms for Data Science and Machine Learning*. Packt Publishing, 2017.

[11] J. Levinson and S. Thrun, "Robust vehicle localization in urban environments using probabilistic maps," in *Proceedings of the International Conference on Robotics and Automation (ICRA)*, 2010, pp. 4372–4378.

[12] R. W. Wolcott and R. M. Eustice, "Robust LIDAR localization using multiresolution gaussian mixture maps for autonomous driving," *International Journal of Robotics Research*, vol. 36, no. 3, pp. 292–319, 2017.

[13] U. Lee, J. Jung, S. Jung, and D. H. Shim, "Development of a self-driving car that can handle the adverse weather," *International Journal of Automotive Technology*, vol. 19, no. 1, pp. 191–197, 2017.

[14] K. Cheng, Y. Bai, Y. Zhou, C. Yu, and Y. Liu, "Multi-IF: An approach to anomaly detection in self-driving systems," 2020.

[15] M. Uřičář, P. Křížek, D. Hurych, I. Sobh, S. Yogamani, and P. Denny, "Yes, we GAN: Applying adversarial techniques for autonomous driving," *Electronic Imaging*, vol. 2019, no. 15, pp. 48–1–48–17, 2019.

[16] A. Greenwald and K. Hall, "Correlated-q learning," in *Proceedings of the 20th International Conference on Machine Learning (ICML)*. AAAI Press, 2003, pp. 242–249.

[17] C. Badue, R. Guidolini, R. V. Carneiro, P. Azevedo, V. B. Cardoso, A. Forechi, L. Jesus, R. Berriel, T. Paixão, F. Mutz, L. Veronese, T. Oliveira-Santos, and A. F. D. Souza, "Self-driving cars: A survey," 2019.

[18] C. Kondapalli, D. Roy, and N. Srishankar, "Deep reinforcement learning applied to a racing game," Worcester Polytechnic Institute, Tech. Rep., 2017, artificial Intelligence Course Project. [Online]. Available: https://nsrishankar.github.io/files/docs/projects/AI_DDPG_Outrun.pdf

[19] R. Kadam, V. Vidhani, B. Valecha, A. Bane, and N. Giri, "Autonomous vehicle simulation using deep reinforcement learning," in *Machine Learning for Predictive Analysis*, A. Joshi, M. Khosravy, and N. Gupta, Eds. Singapore: Springer Singapore, 2021, pp. 541–550.

[20] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis, "Mastering the game of go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, pp. 484–489, 2016.

[21] J. Jin, C. Song, H. Li, K. Gai, J. Wang, and W. Zhang, "Real-time bidding with multi-agent reinforcement learning in display advertising," *Proceedings of the 27th ACM International Conference on Information and Knowledge Management (CIKM)*, 2018.

[22] H. Cai, K. Ren, W. Zhang, K. Malialis, J. Wang, Y. Yu, and D. Guo, "Real-time bidding by reinforcement learning in display advertising," in *Proceedings of the 10th ACM International Conference on Web Search and Data Mining (WSDM)*. ACM Press, 2017.

[23] H. Cai, T. Chen, W. Zhang, Y. Yu, and J. Wang, "Efficient architecture search by network transformation," 2017.

[24] B. Zoph and Q. V. Le, "Neural architecture search with reinforcement learning," 2017.

[25] S. OwaisAli Chishti, S. Riaz, M. BilalZaib, and M. Nauman, "Self-driving cars using CNN and Q-learning," in *Proceedings of the 21st International Multi-Topic Conference (INMIC)*, 2018, pp. 1–7.

[26] A. Sallab, M. Abdou, E. Perot, and S. Yogamani, "Deep reinforcement learning framework for autonomous driving," *Electronic Imaging*, vol. 2017, no. 19, pp. 70–76, 2017.

[27] C. Yin, Z. Xiao, X. Cao, X. Xi, P. Yang, and D. Wu, "Offline and online search: UAV multiobjective path planning under dynamic urban environment," *IEEE Internet of Things Journal*, vol. 5, no. 2, pp. 546–558, 2018.

[28] V. Badrinarayanan, A. Kendall, and R. Cipolla, "Segnet: A deep convolutional encoder-decoder architecture for image segmentation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 39, no. 12, pp. 2481–2495, 2017.

[29] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Communications of the ACM*, vol. 60, no. 6, pp. 84–90, 2017.

[30] M. Bojarski, D. D. Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. Monfort, U. Muller, J. Zhang, X. Zhang, J. Zhao, and K. Zieba, "End to end learning for self-driving cars," 2016.

[31] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[32] A. Billard and D. Grollman, *Imitation Learning in Robots*. Boston, MA: Springer US, 2012, pp. 1494–1496.

[33] M. L. Puterman, *Markov Decision Processes: Discrete Stochastic Dynamic Programming*, 1st ed. USA: John Wiley & Sons, Inc., 1994.

[34] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Machine Learning*, vol. 8, no. 3-4, pp. 229–256, 1992.

[35] N. Hansen and A. Ostermeier, "Completely derandomized self-adaptation in evolution strategies," *Evolutionary Computation*, vol. 9, no. 2, pp. 159–195, 2001.

[36] S. Fujimoto, H. van Hoof, and D. Meger, "Addressing function approximation error in actor-critic methods," 2018.

[37] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller, "Deterministic policy gradient algorithms," in *Proceedings of the 31st International Conference on International Conference on Machine Learning (ICML)*. JMLR.org, 2014, p. I–387–I–395.

[38] "Pybullet: physics simulation for games, visual effects, robotics and reinforcement learning," https://pybullet.org/.

[39] L. Espeholt, H. Soyer, R. Munos, K. Simonyan, V. Mnih, T. Ward, Y. Doron, V. Firoiu, T. Harley, I. Dunning, S. Legg, and K. Kavukcuoglu, "Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures," 2018.

[40] N. Chukamphaeng, "Crazycar," https://github.com/nutorbit/crazycar, 2020.