A3C for drone autonomous driving using Airsim

David Villota
davillot@unirioja.es

Montserrat Gil-Martínez, Javier Rico Azagra
montse.gil@unirioja.es, javier.rico@unirioja.es

Summary

In this work, we apply artificial intelligence to guide a drone to a certain point autonomously. Unreal engine creates a virtual environment where the drone can fly, and the algorithm is trained simulating the drone dynamics thanks to Airsim plugin. The implemented algorithm is Asynchronous Actor-Critic Advantage (A3C), which trains a neural network with less computing resources than standard reinforcement learning algorithms that normally needs costly GPUs. To prove these advantages, several experiments are run using a different number of parallel simulations (threads). The drone should reach a point randomly generated each episode. The reward, the value and the advantage function are used to evaluate the performance.
As expected, these experiments show that a higher number of threads helps the leaning process improve and become more stable. These learning results are of interest to optimize the computing resources in future applications.

Key words: A3C, Actor critic, Reinforcement learning, Autonomous driving, Airsim, Multithread.

1 INTRODUCTION

Artificial intelligence nowadays is at the leading edge in almost all the industry fields, from data analytics to self-driving cars or drones [1][15]. It is evolving at a breakneck speed and enterprises are more and more interested in taking advantage of this technology, which pretty means that more researching efforts are being intended to develop new utilities.

In the framework of autonomously guiding a drone to a specific point, the main objective of this work is the use of reinforcement learning to achieve the best guiding performance using the least possible resources. We deem efficiency very important as it yields flexibility to run different trainings at the same time, easing the comparison of different algorithms as well as it also optimizes the time to get final results.

Unreal Engine and Airsim [17][18] will be used to simulate the drone's behaviour due to the simplicity and flexibility this API offers over other graphic engines' APIs (Gazebo, Unity) to be integrated in the code along IA libraries. In addition, they are not as user friendly, as they required a deep knowledge and specialization in their software to achieve the same results.

In this case, the control of the drone will be focused on reaching a specific point, randomly generated each episode of the training. The "brain" that will determine the variation of the speed vector searching for the objective point will be a traditional recurrent neural network with one LSTM (Long Short-Term Memory) cell to provide it with some memory. In the future, more architectures will be explored such as some variations of ResNet [24], kvCORE or DRNN (Deep Recurrent Neural Network).

For the training strategy, we have deemed several algorithms but selected A3C (Asynchronous Actor-Critic Advantage) [20] as one of our biggest concerns is the efficiency and the ability to run this code in a low performance server. A3C algorithm does not need GPUs and yields the opportunity of squeezing the CPU's performance to the max achieving great results. Also it has a better relation simplicity/results over others such as DQN (Deep Q Learning) [11], A2C (Advantage Actor Critic), PPO (Proximal Policy Optimization)[8], ACKTR (Actor–Critic using Kronecker–Factored Trust Region) [26], GAIL (Generative Adversarial Imitation Learning) [9].

A3C algorithm needs several instances of neural networks, also called models: those that will interact with the environment at the same time (local models); and another from where the local models will update their own weights when starting the episode and also to which the local models will update the weights when the episode has finished (shared model). A3C uses the leverage of multithreading to perform asynchronous updates to the shared model. By continuously updating the shared model, it is expected that the more threads used the steadier and the faster the model should learn. In our set up, we will reproduce this algorithm being able to validate that the learning performance improves when increasing the

https://doi.org/10.17979/spudc.9788497498043.203

number of threads applied to our specific purpose, guiding the drone towards a random point.

In the following text, we will explain how we have solved some of the difficulties faced. In the methodology section, we will firstly sum up the reinforcement learning algorithm that has been implemented, defining the return function, actor, critic, advantage and loss. Then, we will explain the set up developed with the different functions such as reward, actions codification, random point generation, multithreading in Airsim and the structure of the neural network. Finally, we will show our results and state some conclusions and future work to do.

2 METHODOLOGY

2.1 REINFORCEMENT LEARNING

This work will follow the standards reinforcement learning [19][20][21][11] settings where we have an agent interacting with an environment. In this case, the agent will be the NN and the environment will be the Unreal engine graphic engine alongside Airsim.

Each discrete step will have its own state s_t and for this state, the agent will output a policy π that will indicate which action is best to take. Once the action has been carried out, the reward r_t will measure how good the performance has been. This process is repeated continuously until a final state is achieved, for example a collision or distance too large to the objective point.

The process just mentioned only measures the behaviour in a single discrete point. As the drone is moving in a continuous state, it is also important to consider other discrete points. For this, we use the Return function, defined as in (1)

$$R_t = r_t + \gamma r_{t+1} + \cdots = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \quad (1)$$

, which allows considering several discrete points with a discount factor $\gamma \in (0,1]$.

The goal of the NN will be to maximize R_t. The neural network will output the policy π; the policy, as mentioned above, will tell us the expected return for R_t when applying that specific action in that specific state. This is also called the Action Value or the Actor, defined as in (2),

$$Q^{\pi}(s,a) = \mathbb{E}[R_t | s_t = s, a] \quad (2)$$

Thus, given a state s_t, the neural network outputs a policy π that tell us the expected returns to choose an action a (2). With this action, the reward is calculated to tell us how good the performance has been in this specific discrete point. The fact of evaluating only a discrete point is that sometimes, some reward must be sacrificed to achieve a major reward in the future. For this, we use the Value function, also called Critic and defined as in (3),

$$V^{\pi}(s) = \mathbb{E}[R_t | s_t = s] \quad (3)$$

This means, given a state and a policy, which return should we expect. With this function, we are in condition of evaluating how good is the state we are in to achieve the best possible reward. Thus, this is a reference to control whether the action-value function (2) output by the NN is good or not.

While learning the model needs to be continuously updating by backpropagating the loss. The loss function will allow us to measure if the Critic is correct or also if the Critic has the correct criteria to judge the Actor. The Critic (3) asses the Actor (2) but it somehow needs to be assessed as well, for this we have the loss function, defined as in (4),

$$V_{loss} = V_{loss} + 0.5 \cdot A^2 \quad (4)$$

,where A is the Advantage function. This is defined as in (5).

$$A(a_t, s_t) = Q(a_t, s_t) - V(s_t) \quad (5)$$

and means that, given a state s_t whose Critic value is V (3), what extra reward can be achieved if an action a_t is taken.

2.2 MULTITHREADING IN AIRSIM

In this work, Unreal Engine and Airsim [17] will be used to implement the simulation environment. For A3C implementation, several Airsim instances will be used at the same time; each one of them will be one of the environments in which the agent will interact with. As there will be several threads, several instances of Airsim will have to be implemented at the same time. This is one of the main difficulties, as the configuration of Airsim does not allow multithreading [18]. The only way we have figured out to solve that is to configure different Local Host IPs to each Airsim instance. This way, each thread will have its environment with its single IP. The configured IPs are from 127.0.0.1 to 127.0.0.*n*, being *n* the number of threads used in the simulation (see Figure 1).

https://doi.org/10.17979/spudc.9788497498043.203

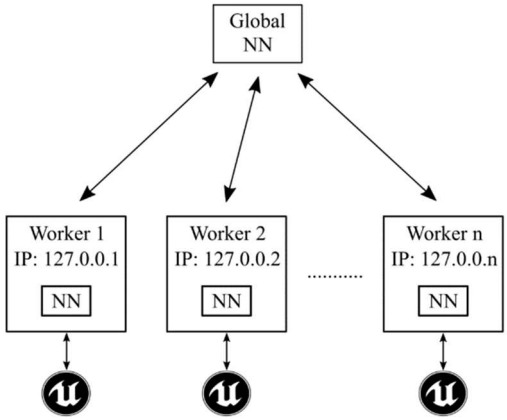

Figure 1: A3C Structure

2.3 RANDOM POINT GENERATION

The objective of the drone is to reach a specific point in a 3D virtual environment. The random point will be calculated using trigonometry in XY plane (α), and a random Z coordinate (z) in a constant spherical radius d, as Figure 2 illustrates.

$$\begin{cases} \alpha = random\ (360) \\ z_{obj} = random(\max\ (Z)) \\ x_{obj} = d \cdot \cos\ (\alpha) \\ y_{obj} = d \cdot \sin(\alpha) \end{cases} \quad (6)$$

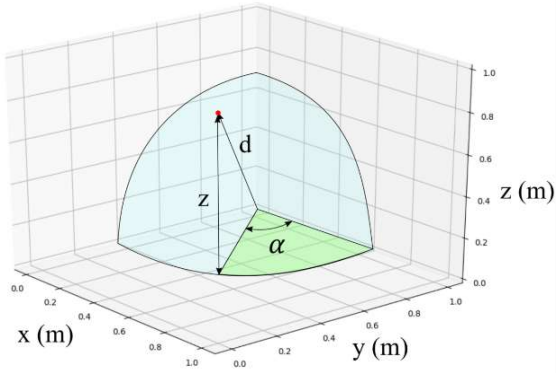

Figure 2: Parameters used for the objective point calculation

Note that the virtual environment has objects in which the drone could collide, preventing it from reaching the objective point. For this work, the object avoidance is not yet implemented, and the only drone information used as input for the NN will be the absolute drone´s coordinates. If the drone collides with an object, the experiment will have finished, and another episode will start. The object will have reached the objective point when the distance between the drone and the point is less than 1 meter.

2.4 NEURAL NETWORK ARCHITECTURE

The Neural Network for the A3C algorithm is in Figure 3. It receives the relative distance to the objective point $[\Delta x, \Delta y, \Delta z]$ as well as the hidden state $[h_{t-1}]$ and the cell state $[c_{t-1}]$[21]. On the other side, it outputs the Actor $[Q(s,a)]$, the Critic $[V(s)]$, the hidden state $[h_t]$ and the cell state $[c_t]$. The internal structure is very simple. The network has an initial linear transformation with its linear rectification function (ReLU). Afterwards, the LSTM cell, whose hidden cell $[h_t]$ will be used as input for both the Actor and Critic linear transformations. Matrix dimensions are detailed in Figure 3; let us remind that the Actor dimension *n* refers to the number of threads.

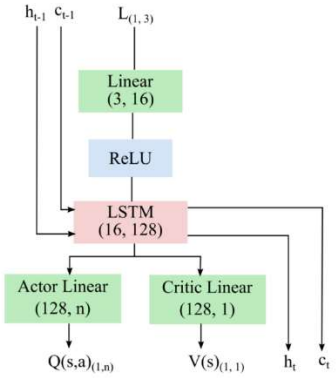

Figure 3: Neural Network Architecture

2.5 ACTION CODIFICATION

The action will define how the drone interacts with the environment. In this case, the actions will modify linear velocities. Taking into account the output of the NN, the current linear velocity will be increased or decreased by a certain constant value that is previously defined. The constant value is set to 0.6 m/s, i.e. the linear velocity of the axe will be incremented or decremented by 0.6 m/s. The image in Figure 4 tries to clarify the concept.

Thus, for a given Action a and a constant value k_v, the speed velocity will be incremented or decremented as

$$v(a) = [v_{x_{k-1}} + k_{v_x}, v_{y_{k-1}} + k_{v_y}, v_{z_{k-1}} + k_{v_z}] \quad (7)$$

The action will be gotten after selecting a random sample of a multinomial distribution created using as input:

$$softmax(Q^{\pi}(s,a)) \quad (8)$$

After that, the action will be decodified as shown in Figure 4 obtaining the variation of linear velocity in the chosen axe, also called $k_{v_x}, k_{v_y}, k_{v_z}$. Note that in

https://doi.org/10.17979/spudc.9788497498043.203

each episode only one of them will have a deviation value, the rest of them will be zero.

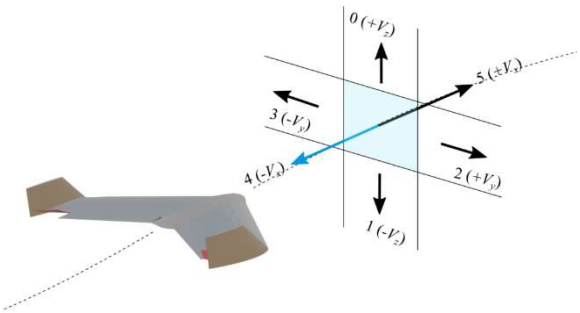

Figure 4: Codification of the actions

3.5 REWARD FUNCTION

The reward function will determine how good is the action yielded by the neural network once it is carried out. In other words, it is the way we can tell the algorithm how good it is performing.

The objective of the drone is to reach a given point in the environment. Thus, the variable used to compute the reward function will be the distance between the absolute position of the drone $[x_d, y_d, z_d]$ and the absolute coordinates $[x_o, y_o, z_0]$ of the objective point (6)

$$L = \sqrt{(x_d - x_o)^2 + (y_d - y_o)^2 + (z_d - z_o)^2} \qquad (9)$$

Then, the reward function

$$r(L) = \begin{cases} -1 & L > 80 \text{ or Collision} \\ -L/40 + 1 & 80 > L > 40 \\ 0.5^{(0.15 \cdot L)} & 40 > L > 1 \\ 1 & L < 1 \end{cases} \qquad (10)$$

, which is represented in Figure 5, has been created following the tips in [6]. It is constrained between (-1,1) to prevent the appearance of large numbers when backpropagating. For the intermediate values, an exponential function has been chosen to clarify that the desired behaviour is to minimize variable *L*.

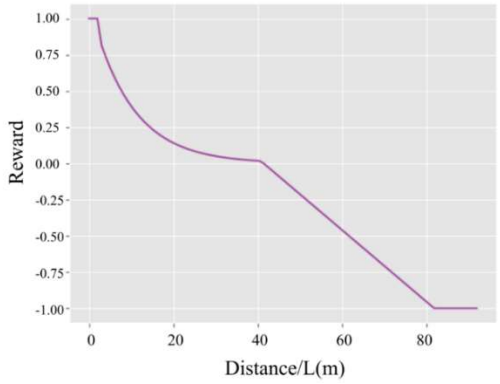

Figure 5: Reward function

3.5 EXPERIMENT SET UP

The experiments have been carried out in a Windows 10 Server with Visual Studio 2019, Unreal Engine 21, Airsim 1.5.0 and python 3.7.

To compare the impact the number of threads have in the learning process, several simulations will be carried out with different number of threads. We will simulate with 1,2,3,6 threads [2].

Each simulation will have 3000 episodes and each episode will last 15 seconds maximum, though it can finish before in case the point has been reached, there is a collision, or the drone gets stuck in the same position for three steps.

4 RESULTS

4.1 INITIAL EPISODE

In the initial episodes we can observe (Figure 6) that the relation between the Remaining Length and the Reward is the one stated in 3.5 while the value shows a behaviour that is not yet properly defined. We can see that in the first steps, the value grows rapidly but then it starts decreasing and never reverts this tendency even though the Remaining Length is decreasing.

A good behaviour would be that while the Remaining Length decreases, the Value function grows, as each step we are nearer to achieve our objective.

In the first epochs, weights are not yet properly updated. The Loss is too large and if the backpropagation of the Loss function is done, it could cause exploding gradients. To avoid this the gradients will be clipped to ± 20. This is a reason it takes several epochs to be able to see a proper behaviour

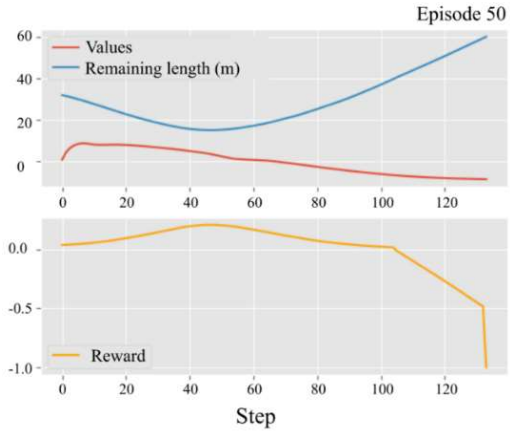

Figure 6: Parameters of an initial episode

https://doi.org/10.17979/spudc.9788497498043.203

4.2 FINAL EPISODE

If we pay attention to the behaviour in the final episodes (2400), we can see (Figure 7) that it behaves properly.
In the initial steps the Value function increases as the remaining length decreases. This means that at the time we are getting nearer to the objective point, the state we are in is better than before, and given the policy output by the network and the state, the expected Reward is larger. When the Remaining length starts growing in the middle steps, the Value function stops increasing and starts decreasing as the drone is getting further to the objective point.

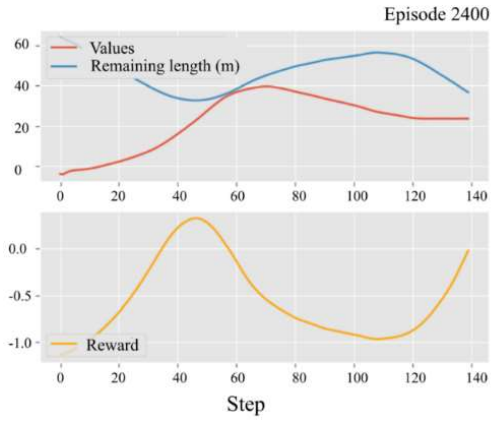

Figure 7: Parameters of a final episode

4.3 SUCCESSFUL EPISODE

Finally, as an example, we show the behaviour of a successful episode (Figure **8**) where we can see the synchronized fluctuation of the Remaining length and the Reward as well as the Value function continuously growing until the last steps.

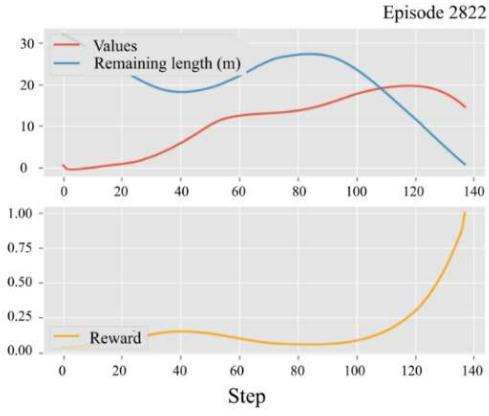

Figure 8: Parameters of a successful episode

In Figure 9, we show the 3D trajectory of the different threads in the same episode (2822). Only Thread number 1 reaches the objective point. Note that since the update to the global network is asynchronous, it is not certain that all the local networks have the same weights, indeed it is highly improbable.

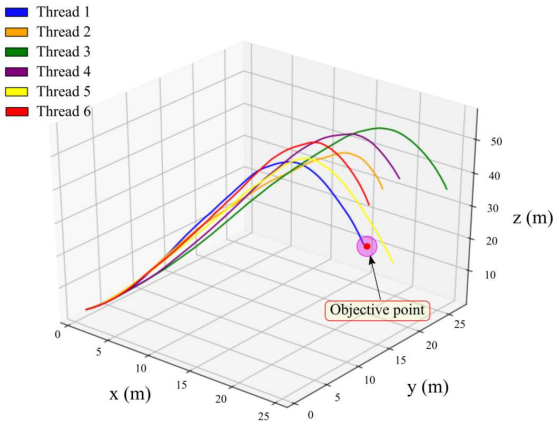

Figure 9: Behaviour of different threads in the same episode

4.4 LEARNING RESULTS

To evaluate how a simulation has learned, we will use the Value function as it is the one that tells us how good the states along the simulation have been. Since each single step of each episode has its own value, we will calculate the mean function of each episode and afterwards a line will be fitted in the points to ease the understanding. The graphs in Figure 10 contains the values for the different simulations with different number of threads.

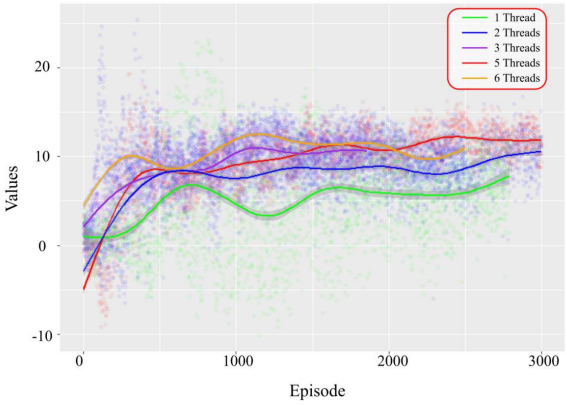

Figure 10: Values comparation for different simulations

The simulation with one single thread presents a stable behaviour during the training and it also reaches the lowest values. However, the simulation with 6 threads is the fastest one to reach the highest values. It only needs some hundreds of episodes to reach the same value than the while simulation with 2 threads.
For the simulations with 2, 3 and 5 threads, we can see that each one surpasses the values of the previous one. This is the expected behaviour as mentioned in [20]. Thus, for our implementation, we can also conclude

https://doi.org/10.17979/spudc.9788497498043.203

that to have many threads directly affects the learning stability. The simulation with one single thread behaves significantly worse than those with more than one thread. The value oscillates in the first two thousand steps. Also, the more threads used the bigger values attained in the simulation and it also learns faster in the initial epochs.

5 CONCLUSIONS

In this work, we have successfully implemented the graphic engine and the A3C algorithm applied to the UAV field in a low performance server. This allows to launch simulations with great flexibility without the need of costly GPUs that sometimes are hard to set up. With these simulations, we have been able to demonstrate the influence of different number of threads in the training, observing that the behaviour is more stable and achieve better values with more threads. This sets the basis for starting a research line in which Asynchronous Actor-Critic Advantage will be the main training strategy for future experiments.

We are aware that the control carried out is very limited, since it only uses absolute positions. In future works, we will put more effort in implementing a more robust control of the drone. Our intention is to perform a control receiving as input the image, GPS coordinates and current angular and linear velocity. For the image processing we intend to use depth estimation as in [13][10] integrated in the neural network along with the sensors data as in [12]. Also we will study different modifications on the training strategy such as the adaptative reward described in [5].

References

[1] Antonio Loquercio, Ana I. Maqueda, Carlos R. del-Blanco, and Davide Scaramuzza (2018). DroNet: Learning to Fly by Driving. IEEE Robotics and Automation Letters, 3(2).

[2] Arun Nair, Praveen Srinivasan, Sam Blackwell, Cagdas Alcicek, Rory Fearon, Alessandro De Maria, Vedavyas Panneershelvam, Mustafa Suleyman, Charles Beattie, Stig Petersen, Shane Legg, Volodymyr Mnih, Koray Kavukcuoglu, David Silver (2015) Massively Parallel Methods for Deep Reinforcement Learning arXiv:1507.04296v2

[3] David Silver (2015) Introduction to Reinforcement Learning with David Silver, DeepMind x UCL.

[4] Danijar Hafner, Timothy Lillicrap, Jummy Ba, Mohammad Norouzi (2020) Dream to Control:Learning Behaviors by latent imagination.

[5] Emilie Kaufmann , Pierre Ménard , Omar Darwiche Domingues, Anders Jonsson, Edouard Leurent, Michal Valko (2021) Adaptive Reward-Free Exploration

[6] Felix Gers (2001) Long Short-Term Memory in Recurrent Neural Networks Thesis nº 2366

[7] Jean-Bastien Grill, Florent Altche, Yunhao Tang, Thomas Hubert, Michal Valko, Ioannis Antonoglou, Remi Munos (2020) Monte-Carlo tree search as regularized policy optimization.

[8] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, Oleg Klimov (2017) Proximal Policy Optimization Algorithms

[9] Jonathan Ho, Stefano Ermon (2016) Generative Adversarial Imitation Learning

[10] Ke Xian, Chunhua Shen , Zhiguo Cao, Hao Lu , Yang Xiao , Ruibo Li , Zhenbo Luo, Huazhong University of Science and Technology, China The University of Adelaide, Australia Samsung Research Beijing, China (2018) Monocular Relative Depth Perception with Web Stereo Data Supervision

[11] Matteo Hessel, Joseph Modayil, Hado van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Dan Horgan, Bilal Piot, Mohammad Azar, David Silver (2017) Rainbow: Combining Improvements in Deep Reinforcement Learning

[12] Ratnesh Madaan, Nicholas Gyde, Sai Vemprala, Matthew Brown, Keiko Nagami, Tim Taubner, Eric Cristofalo, Davide Scaramuzza, Mac Schwager, Ashish Kapoor (2020) AirSim Drone Racing Lab. Proceedings of the NeurIPS 2019 Competition and Demonstration Track, PMLR 123:177-191, 2020

[13]Rene Ranftl, Katrin Lasinger, David Hafner, Konrad Schindler, and Vladlen Koltun (2020) Towards Robust Monocular Depth Estimation: Mixing Datasets for Zero-shot Cross-dataset Transfer

[14] Richard S. Sutton and Andrew G. Barto (2018) Reinforcement Learning: An Introduction, MIT Press, Cambridge, MA, 2018

[15] Shantanu Ingle, Madhuri Phute (2016) Tesla Autopilot : Semi Autonomous Driving, an Uptick for Future Autonomy

https://doi.org/10.17979/spudc.9788497498043.203

[16] Sorg, Jonathan Daniel (2011) The Optimal Reward Problem: Designing Effective Reward for Bounded Agents

[17] Shital Shah and Debadeepta Dey and Chris Lovett and Ashish Kapoor (2017). Field and Service Robotics, AirSim: High-Fidelity Visual and Physical Simulation for Autonomous Vehicles. arXiv:1705.05065

[18] Shital Shah (2018). Airsim Documentation

[19] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu , Joel Veness , Marc G. Bellemare , Alex Graves , Martin Riedmiller , Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen , Charles Beattie , Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumara , Daan Wierstra, Shane Legg & Demis Hassabis (2015) Human-level control through deep reinforcement learning, Macmillan Publishers Limited

[20] Volodymyr Mnih1, Adriá Puigdom`enech Badia1, Mehdi Mirza, Alex Grave, Tim Harley, Timothy P. Lillicrap, David Silver, Koray Kavukcuoglu1 (2016) Asynchronous Methods for Deep Reinforcement Learning

[21] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, Martin Riedmiller (2013) Playing Atari with Deep Reinforcement Learning arXiv:1312.5602v1

[22] Xingyu Sha, Jiaqi Zhang, Keyou You, Kaiqing Zhang , Tamer Başar (2015) Fully Asynchronous Policy Evaluation in Distributed Reinforcement Learning over Networks, arXiv:2003.00433v3

[23] Yuhuai Wu, Elman Mansimov, Shun Liao, Roger Grosse, Jimmy Ba (2017)Scalable trust-region method for deep reinforcement learning using Kronecker-factored approximation.

[24] Zifeng Wu, Chunhua Shen, Antonvan den Hengel (2019). Wider or Deeper: Revisiting the ResNet Model for Visual Recognition. ELSEVIER Pattern Recognition,Pages 119-133.

https://doi.org/10.17979/spudc.9788497498043.203