Scientific Paper

1st
Student Artificial Intelligence
Johannes Kepler University, AT

2nd
Student Artificial Intelligence
Johannes Kepler University, AT

3rd
Student Computer Science
Universidad de Sevilla, ES

***Abstract*—In this paper the use of machine learning algorithms for autonomous cars and its behaviour for different traffic situations will be discussed. Also, it will be explained from our own experience using Unity Hub for the environment and ML Agents for the machine learning process. Finally, it will be compared with other technical papers and discussed if our project reaches our goal or not.**

***Index Terms*—artificial intelligence, autonomous vehicles, simulation framework, intelligent agents, unity**

I. Introduction

Nowadays autonomous driving is one of the most studied technologies and this technology presents an important challenge, driving in urban environments. The reason for this, is that beyond the driving rules you have to predict what the other cars are going to do. If the car in front of our agent brakes our agent must react accordingly. Also, the autonomous car needs a "brain", so it knows what to do when an obstacle appears? In this part act the algorithms and the machine learning process that after it is going to be explained better in this paper.

II. Scientific relevance of the research idea compared to related published work

The softwares used for this paper are: CityEngine, Unity and ML Agents that is based on the technique deep reinforcement learning (RL).

CityEngine has done well in game industry since its commercial arrival in 2008. This software was often used to create a very detailed 3D model of urban and fictional landscapes. CityEngine cities contains the ability to import various types of architectural and geographic datasets such as CAD and GIS, and quite promising options for modeling of real landscapes. [4][5]

Unity's historical focus on developing a general-purpose engine to support a variety of platforms, developer experience levels, and game types makes the Unity engine an ideal candidate simulation platform for AI research. Furthermore, the Unity Editor enables rapid prototyping and development of games and simulated environments. [3]

ML-Agents toolkit [8] is a package from the Unity team for training Machine Learning agents. The package includes a Python API that communicates with the Unity engine and allows for a simulation when training. The ML-Agent package includes documentation and really well explained examples on how to set up the API. By default, the API uses Proximal Policy Optimization (PPO) [9], an RL technique that is implemented in TensorFlow [6][7] which runs in Python. PPO uses the Agent's observation to map the best action it can take using a neural network. [10]

Deep learning is a particular type of machine learning, normally for supervised or unsupervised learning, and can be integrated with reinforcement learning, usually as a function approximator. Supervised and unsupervised learning are usually so directly and only focused on the instant reward while reinforcement learning is sequential and considers long-term accumulative reward. [2]

A machine learning algorithm is formed by a dataset, a cost/loss function, an optimization procedure, and a model [1]. A dataset is divided into non-overlapping training, validation, and testing subsets. Training error measures the error on the training data, trying to minimize the optimization problem. Generalization error measures the error of the new data, that differentiates machine learning from optimization. A machine learning

algorithm tries to minimize the gap between training error and testing error. A model is under-fitting if it cannot achieve a low training error; a model is over-fitting if the gap between training error and test error is large. [2]

III. Research Objective

The research objective is focused on the theoretical part for the development of the simulation correctly, based on the technical knowledge from other papers. We will also only cover surface level research, as this is the first time working on such a project.

IV. Goal of the work

The goal of our work is to create a functioning driving agent. The main objective of the agent should be driving from a set point A to a set point B without either colliding with traffic or any other obstacles such as the sidewalk, should not run red lights on intersection and also stay on the right side of the road without crossing the middle-line.

V. Own research questions and hypotheses

- Q: How will the agent react to traffic lights?

  H: The agent will stop when the lights shows red and continue driving when it turns green.

- Q: How will the agent react to stops signs?

  H: The agent will stop at the sign for a short time and only continue driving if it is safe to do so.

- Q: Will the agent react to other cars?

  H: Yes, the agent will abide the traffic rules.

- Q: What happens in case of an accident that happened in front of the agent?

  H: The agent will come to a full stop to avoid crashing into the accident.

VI. Planned method

We aim to implement two kind of sensors at the front and the back, so that our agent can react to traffic and obstacles ahead of it, while also paying attention to what's happening behind it. For the model we intend to use a simple CNN. For training we will use reinforcement learning, as it allows us to reward and penalize the agent based on its actions. It would be rewarded for passing way-points, avoiding crashes and stopping at traffic lights. It will be penalized for crashing into obstacles, running red lights, or driving into the other side of the road.

VII. Agent

A. General implementation and functionality

For our study purpose we took a prefab car as an agent given by the lecturer with the package name "carDemo19" and modified the belonging behavior script "CarAgent.cs". Our choice for the prefab car was for design reasons. We also implemented ourselves a car agent with the same functionalities but it hadn't such a professional design.
The agent is able to drive and steer by user/machine input. Furthermore, we implemented the option to make usage of a brake with the public variable "breakZeroOne". Here the user can determine whether the car shall be able to drive backward (breakZeroOne=0) or whether it shall be able to break (breakZeroOne=1). We have chosen breakZeroOne=1 for the learning process, else the car would always drive backwards when it is not needed in the beginning of the learning phase.

B. Sensors

- Camera sensors: We implemented two camera sensors. Both are directly placed in the car filming the street like it would be in real world the case. One camera is filming straight forward in driving direction. The other one is working like a rear view mirror. Therefore, the car can also learn to avoid abrupt breaks when other cars are close behind it (Fig. 1).

- LIDAR: The implementation was inspired by a internet video tutorial [11]. The five sensor lines are placed in front of the car, three straight forward looking, the remaining two at the pig launcher with angle of ca. 45 degrees to the longitudinal car axis

(Fig. 1). Aim of the LIDAR is it to measure the distance between a car and a surrounding object. As soon as some object is located within the sensor line length ”sensorLength” (public variable) of 8, the distance to it is measured and fed into the function ”vectorSensor.AddObservation()”. When nothing reaches a line the default distance of 1 is fed into the neural network. We have chosen that distance because a neural network can easily handle it. Besides that, we always need the same amount of observations (five for five vector lines).

C. Bonus and penalties

To enable the learning process of our car agent, it is important to tell it for which kind of actions it will be rewarded or punished. This information is contained in the agent behavior script ”CarAgent.cs”. With help of tagging the city objects, our car shall be able to learn from interaction with the environment.

- Bonus: The agents get rewarded when reaching subgoals, objects with tag=check, by value 1. This objects are ”AI Traffic Waypoints” which are invisible marks for non-player characters which automatically drive to these way points as mark of their path (package ”Simple-AI-Traffic”) (Fig. 3). The car agent shall learn to drive forward in the direction of these way points which are always placed in the street. An city object with tag=goal gives a reward of 10 and with reaching it the learning episode ends successfully. We marked one way point at the end of the learning route with tag=goal.

- Penalty: To avoid that our agent learns to simply not to drive to avoid all penalties it is also punished for each time step with -10/MaxStep, where ”MaxStep” is the maximal number of steps for each episode. It is motivating to end each episode as fast as possible. In the function ”OnCollisionEnter(Collision col)” of ”CarAgent.cs” we describe that when the agent touches an object with tag=wall/Player/Untagged then the episode ends and the reward is set to -1.

VIII. City Map

A. City planning

To simulate our experiments, we first need an active testing environment which should include roads and some buildings. Using ArcGis CityEngine, we can

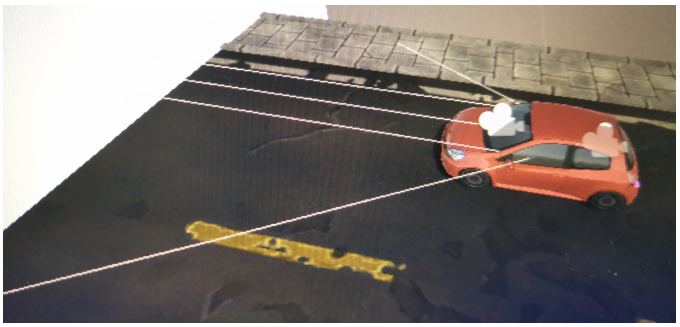

Fig. 1: Car agent with LIDAR lines reacting to a wall and right kerbstone. In addition, two camera sensors for front and back view.

easily create such a city. For our project, we used the city of New York, Manhattan to be exact. After selecting the city and generating the model, manual cleanups need to be done, as CityEngine has some trouble identifying the correct street markings, often leaving us with unnecessary one-way roads which were changed to more useful two-way roads. Since elevation is also taken into account, we brought the entire city to the same level, as elevation would severely complicate things later on. Furthermore, some of the generated buildings tended to overlap with the streets, which were also changed accordingly. After cleaning up the city, the file is exported and then imported to Unity3d, where we are left with a 3d-model of our city. Adding a plane will give our city a ground to reside on and adding some trees gives it a nice touch (Fig. 2).

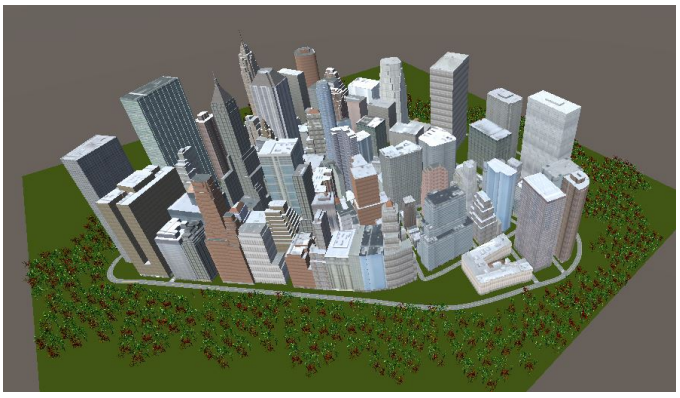

Fig. 2: 3D city in Unity.

B. Traffic

Since we want to test our autonomous vehicle in a realistic environment, we need to implement some

type of traffic. This includes driving vehicles, street signs and traffic lights. Using a Unity package called ”Simple-AI-Traffic” we can implement self-driving vehicles. The package includes tools that allow us to manually place way-points that the vehicles will follow (Fig. 3). We can also adjust a few parameters that determine vehicle speed and the amount of spawned vehicles on a spawning point. Additionally, traffic lights are placed at intersections for a controlled traffic flow (Fig. 4). Furthermore, invisible traffic lights were placed on stop signs, as the vehicles cannot detect those.

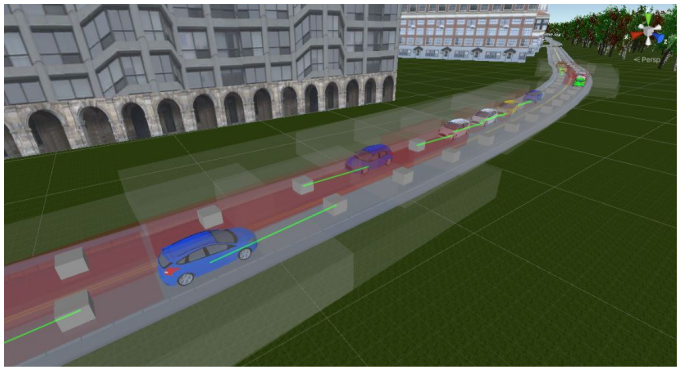

Fig. 3: Traffic example in our city. The grey boxes indicate the way-points of the self driving vehicles. The green line indicates the next way-point they are trying to reach and the red box serve as a collision detector.

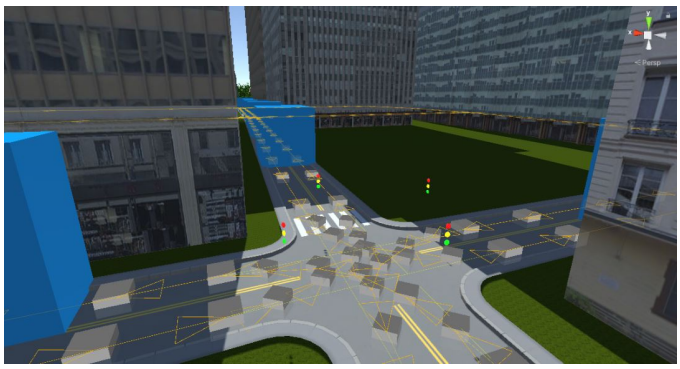

Fig. 4: Intersection complete with working traffic lights to regulate traffic. The blue boxes indicate spawn points for the self-driving vehicles that have been scattered across the city roads.

IX. Results

A. Results without traffic

To make things less complicated, we first trained the model without any traffic on the road, so that things are less complicated for now. The agent was trained on a prerecorded 20-30 minute demo for 1 million iterations, which took roughly 4 hours. By the end of the training, the agent was driving rather ”wobbly”, meaning constantly turning left and right instead of driving straight. Although, turns were included in the demo, the agent had a really hard time turning left or right on an intersection. There were some visible attempts at turning, but only very vague and not nearly enough to make a full turn.

B. Results with traffic

Due to complications with our sensors, we had to disable them, resulting in our agent not having ”eyes” to recognize obstacles. Therefore, our agent was not able to react to traffic, resulting in the agent constantly running into other cars. The results from the above tests also apply to this.

C. Results using a demo where we just turn left/right

Here we tried to focus more on the car actually turning in corners on intersections. We recorded a demo where we start just in front of an intersection and turn left. We repeated this process for about 10 minutes, crashing into an obstacles after each turn to end the current episode and start from the beginning. This demo is then again used to train the model. We trained it for 500 thousand iterations. The result was rather expected as our previous tests did not yield good results either. While there are visible attempts to turn, yet the agent still fails to do a proper turn.

D. Note on results

The results might have been better if we trained the model for much longer than 2-4 hours. We spent some time on changing and just experimenting with some of the parameters in general, but the results weren’t much better than before.

X. Discussion: Possible improvements

- Drastic increase in training time. Due to time constraints training was cut rather short. 2-4 hours of training is not nearly enough for the agent to learn everything properly. This would allow us to observe more specific situations and adjust the agent accordingly.

- Properly implement sensors so our agent can actually react and adjust to whats happening around it. Additionally reward the agent for successfully avoiding obstacles.

- Experiment more with the parameters. Experimenting more with the parameters so that we significantly improve our results.

- Training power and intensity. More powerful machines would allow us to train multiple agents parallel to each other, increasing the amount of information it gains while also reducing training time. Ideally, spawn the agents at random points across the city, so it does not always drive on the same road, so that we don't run into over-fitting.

- Rework the roads of the city. After exporting our city from ArcGis CityEngine and importing it into Unity, it is not possible to change the models at all. While implementing the traffic, it turned out that our streets were rather tight including the turns at intersections. This was an issue, as the self-driving cars were not able to properly turn in some cases. It was also too tight for our agent, as sometimes minor mistakes immediately lead to failure.

- Further improve on the way-points for the self-driving vehicles, as they sometimes run into the sidewalk, or just randomly turn into the other lane of the road.

- Invisible collision mesh at intersections. This would let our agent know if its about to drive into an intersection. This would allow us to implement additional parameters to tell the agent that it is able to take left or right turns.

- Further introduce more variables into the agent such as its position, velocity or turn radius. This would allow the agent to have more information on its location, how fast it is going and whether it is turning or not.

XI. Acknowledgments

We thank Morales-Alvares Walter for giving us an insight into the area of autonomous driving and introducing us to programs like Unity and CityEngine, aswell as showing us the ground work of machine learning agents in Unity. Furthermore, the course was well designed and easy to follow, aswell as the great support that has been provided in the lectures.

References

[1] Goodfellow, I., Bengio, Y. and Courville, A., 2016. Deep Learning. MIT Press.

[2] Li, Y., 2017. Deep reinforcement learning: An overview. [online] Available at: https://arxiv.org/pdf/1701.07274.pdf [Accessed 3 June 2022].

[3] Juliani, A., Berges, V., Teng, E., Cohen, A., Harper, J., Elion, C. and Lange, D., 2018. Unity: A general platform for intelligent agents.

[4] Edvardsson, K.N., 3D GIS modeling using ESRI's CityEngine: a case study from the University Jaume I in Castellon de la Plana Spain. 2013.

[5] Ghorbanian, M. and Shariatpour, F., 2019. Procedural modeling as a practical technique for 3D assessment in urban design via CityEngine. Iran University of Science Technology, 29(2), pp.255-267.

[6] Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M., Kudlur, M., Levenberg, J., Monga, R., Moore, S., Murray, D., Steiner, B., Tucker, P., Vasudevan, V., Warden, P., Wicke, M. and Yu, Y., 2016. TensorFlow: a system for large-scale machine learning. OSDI'16: Proceedings of the 12th USENIX conference on Operating Systems Design and Implementation, [online] pp.265–283. Available at: https://dl.acm.org/doi/10.5555/3026877.3026899 [Accessed 3 June 2022].

[7] Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M., Kudlur, M., Levenberg, J., Monga, R., Moore, S., Murray, D., Steiner, B., Tucker, P., Vasudevan, V., Warden, P. and Wicke, M., 2016. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. [online] pp.1-19. Available at: https://arxiv.org/pdf/1603.04467.pdf [Accessed 3 June 2022].

[8] Juliani, A., Berges, V., Vckay, E., Gao, Y., Henry, H., Mattar, M. and Lange, D., 2018. Unity: A general platform for intelligent agents. [online] Available at: https://arxiv.org/pdf/1809.02627v1.pdf [Accessed 3 June 2022].

[9] Prafulla Dhariwal Alec Radford Oleg Klimov. John Schulman, Filip Wolski. Proximal policy optimization algorithms. August, 2017.

[10] Johnsson, T. and Jafar, A., 2020. Efficiency Comparison Between Curriculum Reinforcement Learning Reinforcement Learning Using ML-Agents.

[11] EYEmaginary, 2017. Car AI Tutorial 8 (Unity 5) - Adding Sensors. [online] Youtube.com. Available at: https://www.youtube.com/watch?v=CqcvfxOxNPk [Accessed 3 June 2022].