Bournemouth University

Faculty of Science and Technology

BSc (Hons) Games Software Engineering

June 2020

Implementing a one-way self-driving car framework using Unity ML-Agents and deep learning

by

Maciej Legas

CONTENTS

FIGURES

1 Abstract

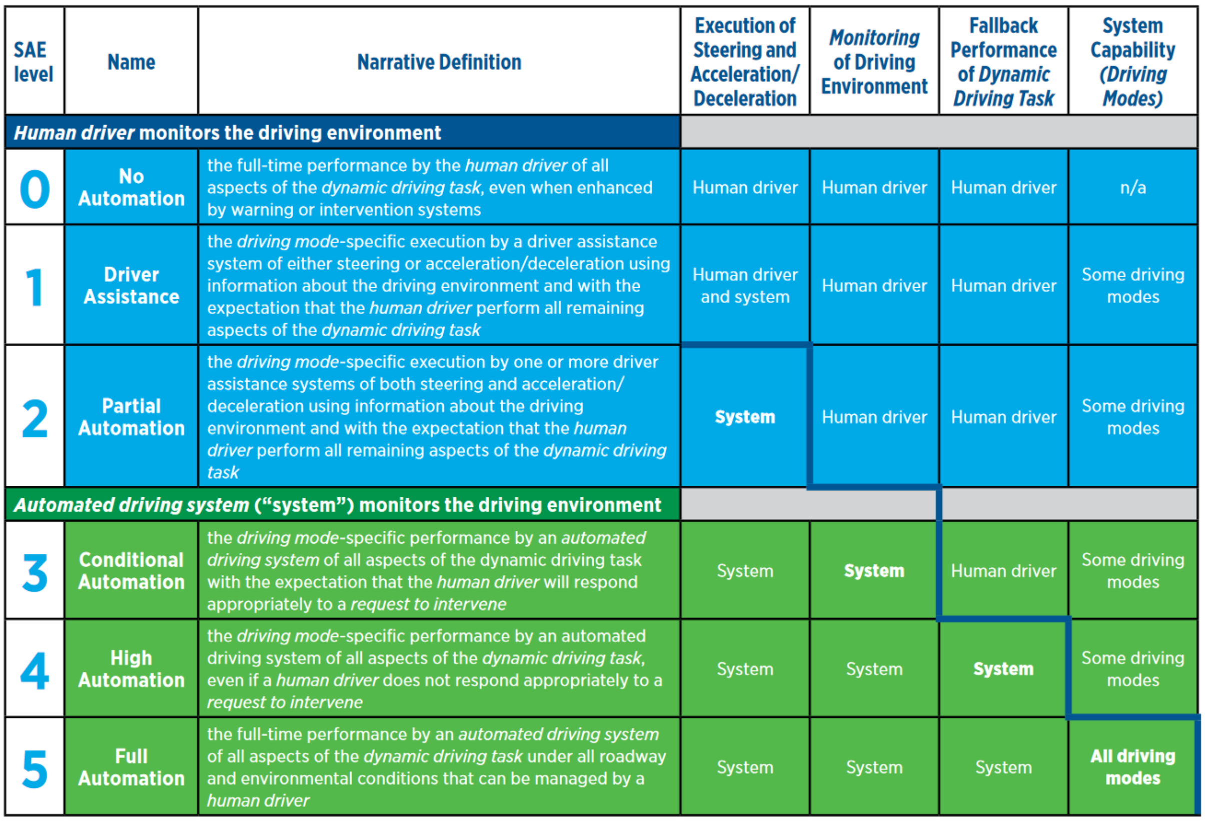
As the world's automotive research searches for new, comfortable conveniences, the evolvement of artificial intelligence that brought technologies such as computer vision or machine learning greatly boosted the research of automotive vehicles. As games developers, a question should be raised whether it would be possible to use AI for an in-game framework. With the release of the Unity ML-Agents toolkit, allowing for easy access of using machine learning in Unity, this question becomes easier to answer. This research aims to study the use of computer vision and reinforcement learning in a virtual environment, combining the use of both to create a simplified autonomous car framework that uses traffic sign detection to adjust its behaviour. After the initial training success of a straight road environment, two traffic sign detection approaches were evaluated and a number of learning environments was created for further training. The environments were then built to be wrapped into a gym interface to be externally trained using the PPO2 algorithm and TensorFlow in Python. Albeit the development progress was interrupted, and although the ML-Agents toolkit is still early in development, this research shows potential in the usage of reinforcement learning and computer vision in games development.

2 INTRODUCTION

2.1 AUTONOMOUS VEHICLE

Over the years of research & development in the automobile industry around the world, a significant shift in the primary research path has been noticed, moving away from high-performance vehicles and engines to safe and comfortable vehicles. This shift has led to the rapid development of various intelligent vehicle technologies, eventually leading to a belief that maximizing safety and comfort in vehicles can be achieved by the innovation of autonomous cars, as 93 percent of all accidents are said to be caused by human error (Pettigrew 2016, p. 5). To develop such functionality, Artificial Intelligence, a subset of computer science, is used, which focuses on the development of programs such that they will cope not worse than a human in an arbitrary world (Dobrev 2004, p. 2).

By definition, an autonomous car is a self-driving vehicle that has the capability to perceive the surrounding environment and navigate itself without human intervention (Jo et al. 2014, p. 7131). The Society of Automotive Engineers International suggests a number of six possible automation levels, beginning from level 0 being a standard utility vehicle, and level 5 being a fully autonomous vehicle (Figure 1).



SAE level	Name	Narrative Definition	Execution of Steering and Acceleration/ Deceleration	*Monitoring* of Driving Environment	Fallback Performance of *Dynamic Driving Task*	System Capability *(Driving Modes)*
***Human driver* monitors the driving environment**						
0	No Automation	the full-time performance by the *human driver* of all aspects of the *dynamic driving task*, even when enhanced by warning or intervention systems	Human driver	Human driver	Human driver	n/a
1	Driver Assistance	the *driving mode*-specific execution by a driver assistance system of either steering or acceleration/deceleration using information about the driving environment and with the expectation that the *human driver* perform all remaining aspects of the *dynamic driving task*	Human driver and system	Human driver	Human driver	Some driving modes
2	Partial Automation	the *driving mode*-specific execution by one or more driver assistance systems of both steering and acceleration/ deceleration using information about the driving environment and with the expectation that the *human driver* perform all remaining aspects of the *dynamic driving task*	**System**	Human driver	Human driver	Some driving modes
***Automated driving system* ("system") monitors the driving environment**						
3	Conditional Automation	the *driving mode*-specific performance by an *automated driving system* of all aspects of the dynamic driving task with the expectation that the *human driver* will respond appropriately to a *request to intervene*	System	**System**	Human driver	Some driving modes
4	High Automation	the *driving mode*-specific performance by an automated driving system of all aspects of the *dynamic driving task*, even if a *human driver* does not respond appropriately to a *request to intervene*	System	System	**System**	Some driving modes
5	Full Automation	the full-time performance by an *automated driving system* of all aspects of the *dynamic driving task* under all roadway and environmental conditions that can be managed by a *human driver*	System	System	System	**All driving modes**

Figure 1: SAE levels of automation (SAE International 2014).

To ensure that the scope of this research stays manageable, this research will base to create an automation level 4 vehicle, meaning that it will be able to drive autonomously without the interference of a human driver (player), but only in defined scenarios, such as driving on one-way roads and on roads without traffic.

In order to create an autonomous car, it needs to be clearly listed what makes an autonomous car. Jo et al. (2014, p. 7132) in their research propose a number of five essential functions for an autonomous car to be able to drive by itself, as shown in Figure 2 with conceptual descriptions:

- Localization
- System management
- Perception
- Planning
- Vehicle control

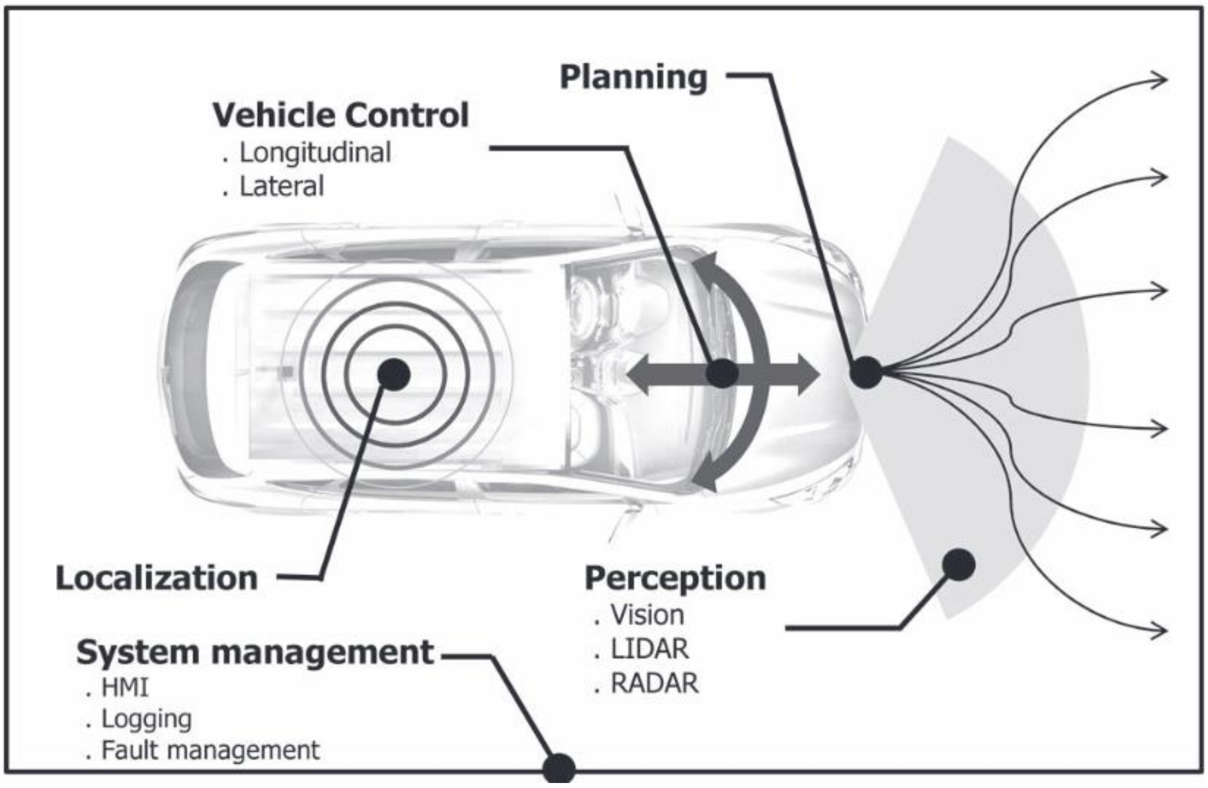


Figure 2: Basic functions of autonomous cars (Jo et al. 2014).

However, these functions have been thought of in terms of real-life application, and therefore it needs to be reviewed how many of them are required in a virtual environment. Two of these functions have been determined to be optional for a level 4 automation vehicle, which are localization and system management.

The proposed functionality of the localization function is to find the position of the vehicle, via the means of components such as the Global Positioning System, dead reckoning, and roadway maps (Jo et al. 2014, p. 7132). While this functionality is useful in a scenario where the car might drive on a network of roads to a defined goal, such as in realistic simulation games, it is not a necessary component, although with the directly accessible coordination system in the 3D environments, it can be easily implementable if needed.

As for system management, the proposed components of it are the fault management system, logging system, and a human-machine interface (Jo et al. 2014, p. 7132). Again, while this functionality is useful in a realistic simulation, in the case of this research the vehicle will not be able to receive any damage and internal wear, rendering this component as not required.

Therefore, this research will focus on using the three main functions: perception, planning and control.

2.2 Perception

Perception is defined as the "awareness of the elements of environment through physical sensation" (Merriam-Webster 2020). Joe et al. (2014, p. 7132) propose that perception in autonomous cars should be implemented as the ability to sense the surrounding environment using various types of sensor techniques, such as RADAR (Radio Detection and Ranging) or LIDAR (Light Imaging Detection and Ranging), and computer vision. As this research initially bases on a level 4 automation vehicle with no road traffic, the sensor techniques can be omitted from implementation.

Computer vision is defined as the "science of endowing computers or other machines with vision, or the ability to see" (Learned-Miller 2011, p. 2). However, it needs to be discussed what exactly should the car agent see. In the view of this research, the agent should be able to see and detect the properties of the road it is driving on, as well as the passed traffic signs to adjust its behaviour to the currently enforced driving law. As the research will be based in a virtual environment, the agent's vision of the road can be simplified and the agent can be directly provided with knowledge about the road edges using vectorized observations, placing the main focus on the traffic sign detection and additionally saving GPU resources in the place of realism.

The research on traffic sign recognition is extensive, with multiple techniques used in it in the past, such as Support Vector Machines or AdaBoost (Liu et al. 2019, p. 86592). Currently, however, computer vision research bases on convolutional neural networks (CNNs), as they "do not need manually designed features and can handle a larger amount of training samples to generate a detector with better generalization ability" (Liu et al. 2019, p. 86592).

CNNs have greatly evolved with the use of deep learning, which is a subset of machine learning. The field "of deep learning is primarily concerned with how to build computer systems that are able to successfully solve tasks requiring intelligence" (Goodfellow et al. 2016, p. 15), able to provide accurate recognition or prediction (Goodfellow et al. 2016, p. 22). Deep learning bases on the use of deep neural networks with multiple layers of neurons, as shown in Figure 3.

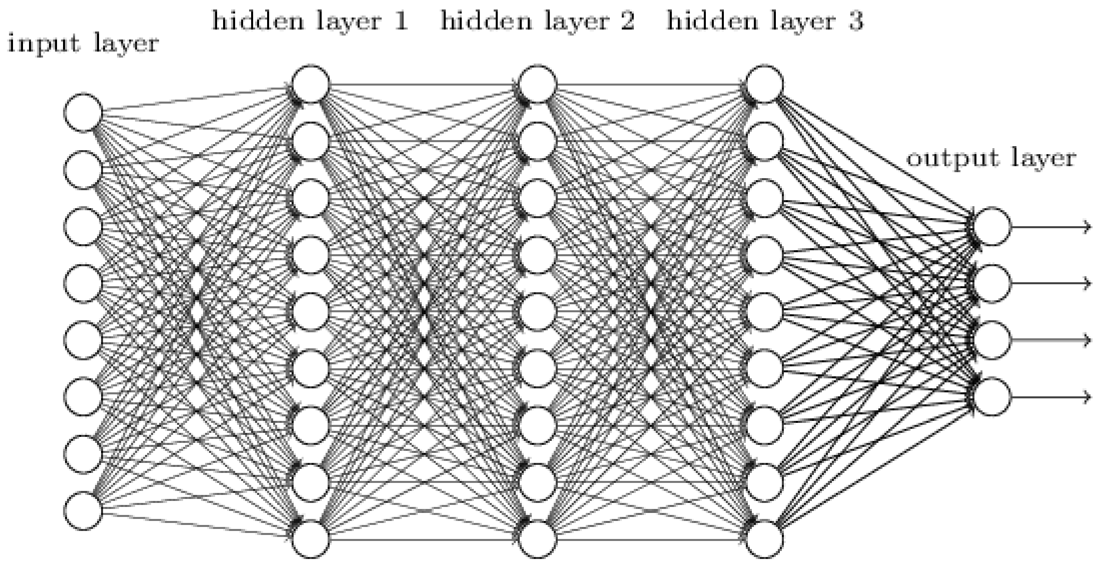


Figure 3: Example of a deep neural network (Nielsen 2019).

By the use of CNNs, models are able to be trained using supervised learning with a labelled dataset to identify commonalities amongst objects of the same type (Figure 4), as CNNs have specifically designed types of layers to identify parts of the image needed for recognition and classifying the input image to a correct label (Cao et al. 2019, p. 4027).

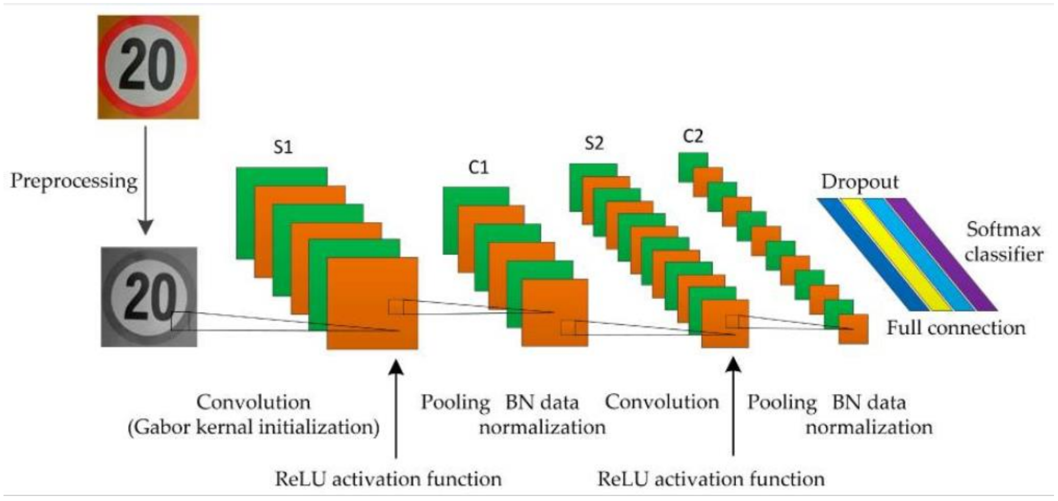


Figure 4: An example model of a Convolutional Neural Network used in traffic sign recognition (Cao et al. 2019).

However, performing object detection and recognition using solely CNNs was deemed as “too slow and computationally very expensive” (Sachan 2017). That led to the research of object detection models such as Region-based Convolutional Neural Networks (R-CNNs), which although were based on CNNs, used additional layers for performing object detection on parts of the image, before performing recognition (Sachan 2017).

An example of research successfully using an object detection model for traffic sign detection in virtual environments is one made by Zhu et al., who managed to successfully implement a RetinaNet model for traffic sign detection (2019). However, their research based on using a dataset composed of annotated front-view images made in a virtual simulation environment, as shown in the example on Figure 5, instead of real-time usage in a virtual environment.

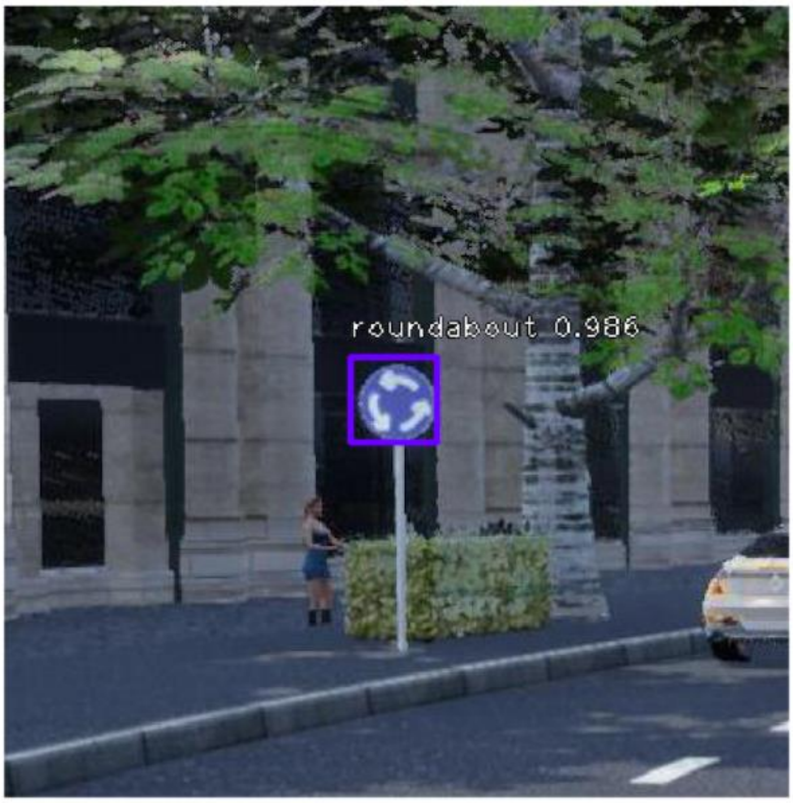


Figure 5: A successful detection of a roundabout sign in a virtual environment (Zhu et al. 2019).

As each country has its own traffic sign design, a decision was made to use the German Traffic Sign Detection Benchmark (GTSDB) dataset (Houben et al. 2013) in this research for traffic sign recognition and traffic signs used in the virtual environment, as the dataset is one of the most widely used datasets in the research of computer vision, ensuring the validity of this dataset. Example image from the dataset has been shown as Figure 6. The German traffic sign design will be used in this research as well.

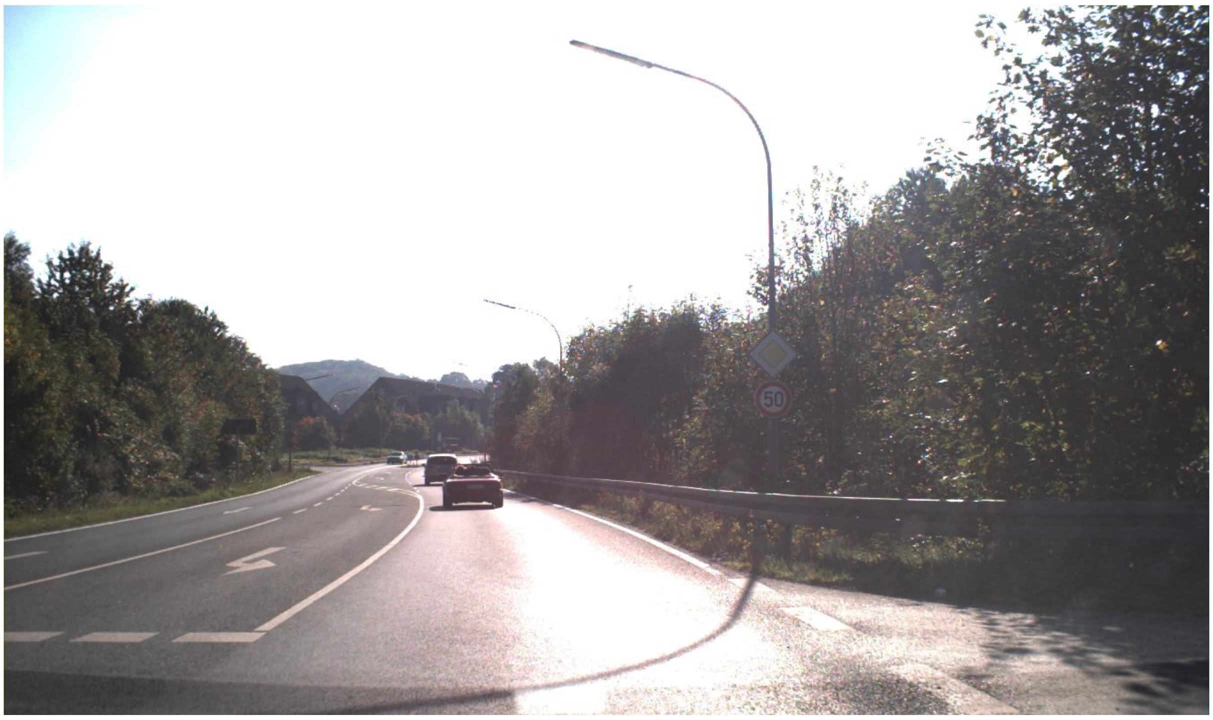


Figure 6: Example image from the German Traffic Sign Detection Benchmark dataset (Institut für Neuroinformatik 2013).

2.3 Planning

The planning function takes the information from perception and localization and determines the behaviour and motion of the autonomous car (Jo et al. 2014, p. 7132). Therefore, it is needed to allow the car agent to make a decision based on taken observations about the surrounding environment from its perception functionality. For this case, deep learning is used yet again as it is able to predict an outcome by providing it with input.

Although CNNs have been researched and already used for autonomous driving, where Bojarski et al. managed to train a CNN to "map raw pixels from a single front facing camera directly to steering controls" (Bojarski et al. 2016, p. 1) with a small training dataset created from just 72 recorded hours of driving (Bojarski et al. 2016, p. 4), they can operate only in cases when there are visual observations provided. Moreover, the trained CNN model was only able to ensure the car stayed on its lane on the road, without taking into consideration the current speed of the vehicle.

Another example of a successful implementation of the planning functionality in a virtual environment is Udacity's simulation for their Self-Driving Car Nanodegree (Udacity 2019), though it also bases on the use of a CNN to train the car agent. The notable improvement was taking as observations in the training dataset not only the visual input, but the speed, throttle and brake pressure as well, which allowed the car to drive autonomously on a one-way road, lacking the use of traffic signs, however.

Both of these researches used behavioural cloning for collecting the training dataset as well. In behavioural cloning, the agent trains itself by imitation (Torabi et al. 2018, p. 2), learning from a dataset of recorded demonstrations, which in the previous cases was a user driving a car. While this is usually a quicker learning method as it allows to indicate the correct behaviour of the agent without the need to interact with the environment first (Torabi et al. 2018, p. 2), the importance of the quality of the dataset with demonstrations becomes heavily leveraged in behavioural cloning and more often than not the training data needs to be augmented and cleaned to be usable. To help with these issues, and taking into consideration that this research aims to train the car agent from scratch as well, reinforcement learning will be used instead, which is one of the three types of machine learning, as shown in Figure 7.

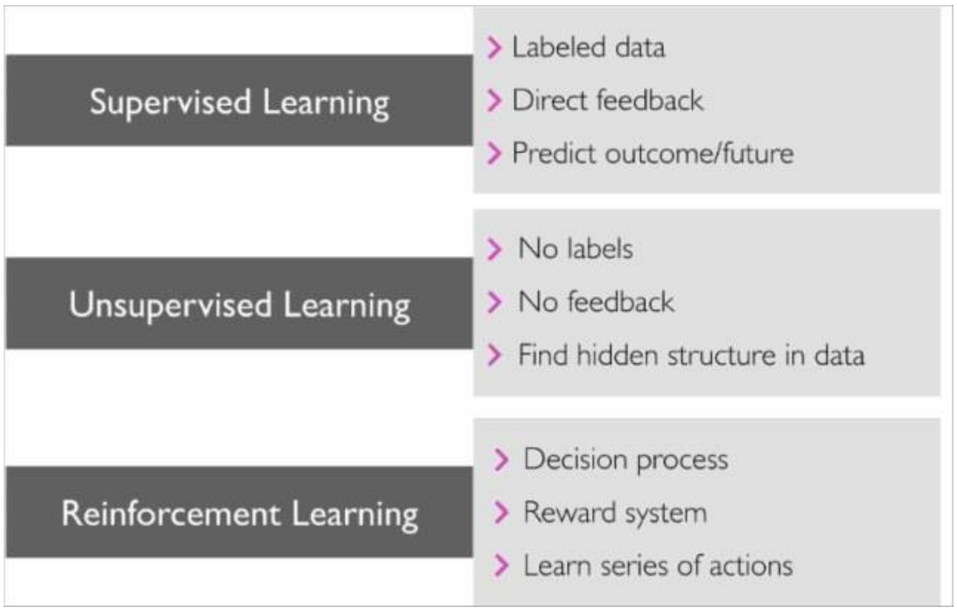


Figure 7: Types of Machine Learning (Raschka 2017).

Reinforcement learning is “about an agent interacting with the environment, learning an optimal policy, by trial and error, for sequential decision making problems“ (Li 2018, p. 5). In reinforcement learning, an agent representing the model of a neural network interacts with its environment and “upon observing the consequences of its actions, can learn to alter its own behaviour in response to rewards received” (Arulkumaran et al. 2017, p. 27), therefore teaching itself in a similar manner to a trial-and-error strategy, as shown in Figure 8. As the research is performed in a virtual environment, the agent can interact with the car in order to learn how to drive properly, without the serious consequences that would happen in a real-life scenario in cases of negative performance of the agent.

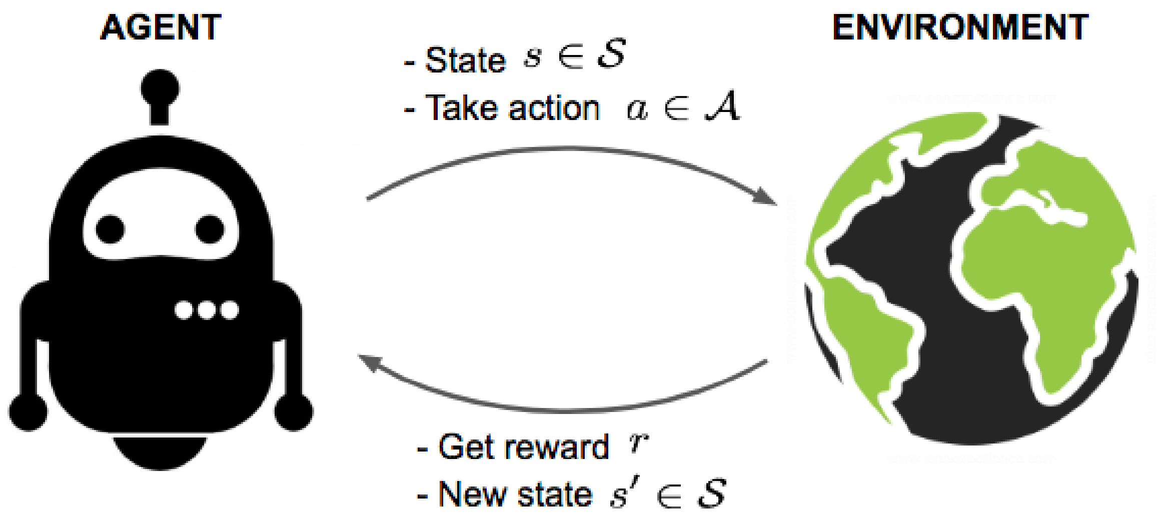

AGENT
ENVIRONMENT
- State $s \in \mathcal{S}$
- Take action $a \in \mathcal{A}$
- Get reward r
- New state $s' \in \mathcal{S}$

Figure 8: Agent interacting with the environment in order to maximize the cumulative reward (Weng 2018).

Besides the agent and the environment, there are “four identifiable main subelements of a reinforcement learning system: a policy, a reward signal, a value function, and, optionally, a model of the environment” (Sutton and Barto 2014, p. 7).

A policy “defines the learning agent’s way of behaving at a given time” (Sutton and Barto 2014, p. 7), mapping the received observations about the environment from the agent to an action and providing the action back to the agent as it follows the policy (Figure 9).

$$a = \pi(s)$$

Figure 9: An agent receives an action to perform by sending its state (observations) to the policy function (Simonini 2018).

A reward signal “defines the desired goal in a reinforcement learning problem” (Sutton and Barto 2014, p. 7). By the use of the reward signal, the agent can accomplish its sole objective of maximizing the total reward over the long-run, by receiving the reward signal in each time step, even if it is zero (Sutton and Barto 2014, p. 7).

Whereas the reward signal is used to estimate short-term rewards, a value function sets out a long run strategy by determining the best course of action to maximize the reward received by an agent (Sutton and Barto 2014, p. 8). A mathematical definition for a state-value function for policy *π*, denoted as *v*$_π$, has been shown as Figure 10. A state-value function gives the expected return for a value of state *s* following policy *π*, when starting from state *s* at time *t* and following policy *π* after that (Deeplizard 2018). The value of a state in the value function is defined as "the total amount of reward an agent can expect to accumulate over the future, starting from that state" (Sutton and Barto 2014, p. 8).

$$v_\pi(s) = E_\pi\left[G_t \mid S_t = s\right] = E_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \mid S_t = s\right].$$

Figure 10: Mathematical definition of a state-value function v$_π$*(s) (Deeplizard 2018).*

Taking into consideration the above, it clearly shows that it is important to consider how high should the reward be for each goal specified in the environment, as it has a direct effect on the learning of the agent, as well as whether it learns to follow the long-run rewards instead of the immediate ones.

An optional element of some reinforcement learning systems is a model of the environment. An agent given a state and action and having full knowledge about the environment, therefore having a model of the environment, can predict its next state and reward before taking an action (Sutton and Barto 2014, p. 8). Model-based approaches are used for planning, where it is needed to "consider possible future situations before they are actually experienced" (Sutton and Barto 2014, p. 8), when deciding on a course of action. Reinforcement learning algorithms that use models are called of the model-based type, and the ones that focus solely on the trial-and-error learning approach are named model-free (Sutton and Barto 2014, p. 8). However, model-based algorithms require the agent to try and at least approximate a representation of the surrounding world in the environment, which is usually unavailable. Therefore, a higher usage of on model-free algorithms is notable, as is the case with this research.

The usage of the policy differs between algorithms as well, as "on-policy algorithms attempt to evaluate or improve the policy that is used to make decisions, whereas off-policy methods evaluate or improve a policy different from that used to generate the data" (Sutton and Barto 2014, p. 124). Nowadays, there are numerous ready to use reinforcement learning algorithms in the research field, as shown in Figure 11, with the most commonly now used and researched ones being the model-free, off-policy algorithms such as Proximal Policy Optimization (PPO), Asynchronous Actor-Critic (A3C), or Deep Deterministic Policy Gradient (DDPG).

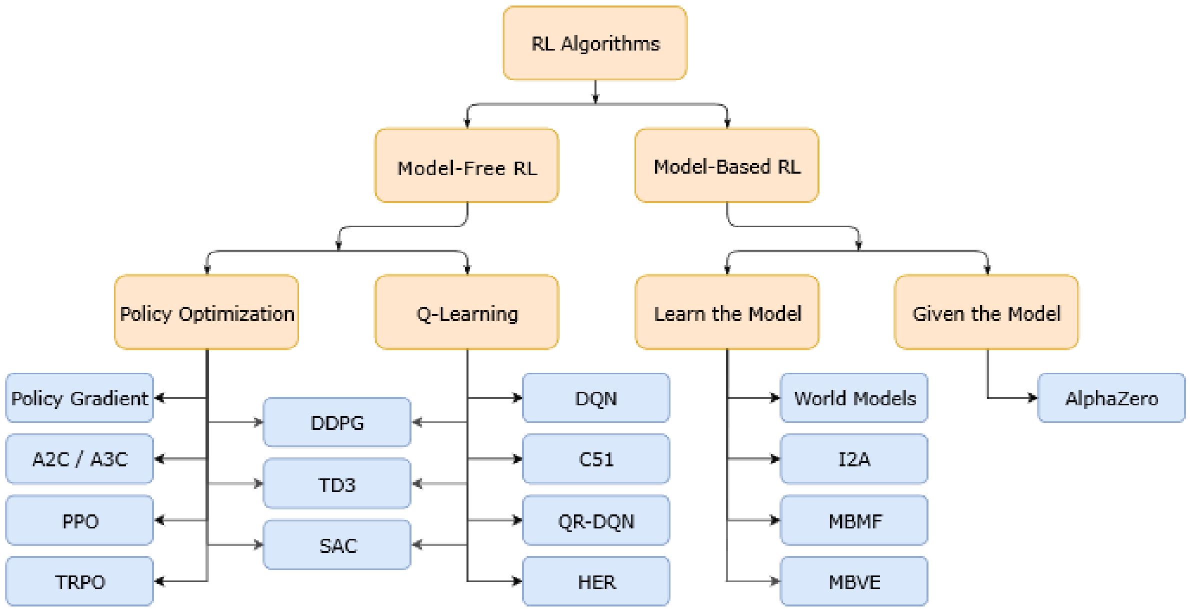


Figure 11: Taxonomy of reinforcement learning algorithms (OpenAI 2018).

Some of these algorithms have been already used in autonomous driving. Pan et al. in their research (2017) that focused on converting a model trained in a virtual environment to be usable in real life successfully trained the model using the A3C algorithm, stating that “the learnt reinforcement learning model on realistic frames can be easily applied to real-world environment” (Pan et al. 2017, p. 10). Previous research under the name of Autocar by a user named kwea123 (2018), made in Unity ML-Agents as well, successfully used both computer vision and an own implementation of PPO to train a car agent how to detect the road edges (Figure 12) and stay on track while driving. Therefore, this research will focus on using this algorithm first, although an attempt will be made to evaluate multiple algorithms such as the previously mentioned A3C, as research has been made to show that training agents with the use of reinforcement learning may differ in behaviour between algorithms. Such behaviour has been found in the research made by Danijar Hafner (2016), who trained agents the basic mechanics of Doom, 1998, using the raw visual observation of the game as the input for the agents. The findings were that his Long Short-Term Memory – Asynchronous Actor Critic (LSTM-A3C) trained agents tended to only attack when facing an enemy, in contrast to the agents trained by a Deep Q-Network, which shot regularly and more often than LSTM-A3C (Hafner 2016, p. 26).

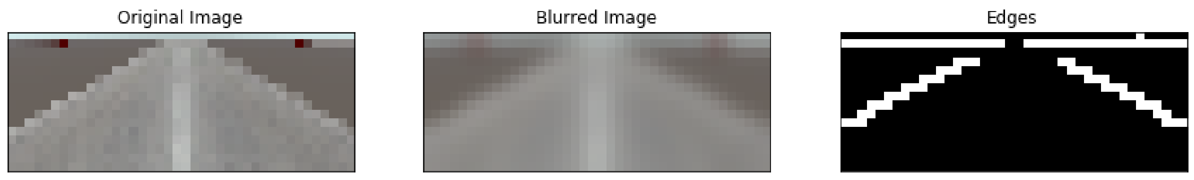


Figure 12: Computer vision augmentation to detect road edges (kwea123 2018).

2.4 CONTROL

The control function executes the commands given by the planning function, by performing the steering, accelerating and braking in the autonomous car (Jo et al. 2014, p. 7132). To allow the control function to perform, an interactable and realistic environment for the agent to operate in needs to be provided. In line with what Juliani et al. (2020) in their research propose, modern game engines such as Unity are the solution to this problem, as they allow for an easy creation of environments for deep learning, by also allowing for complex interactions between agents due to the sophisticated physics systems most of them contain. Due to their purpose of being development tools for game developers, game engines provide an easy and direct access to variables, allowing the agent to collect observations "using a variety of possible sensors corresponding to different forms of information such as rendered images, ray-cast results, or arbitrary length vectors" (Juliani et al. 2020, p. 12).

Juliani et al. have therefore released an open source toolkit named Unity ML-Agents, which although is a quite early project that is continuing to be in development, allows to integrate creating simulation environments using the Unity Editor with the usage of deep learning algorithms for the environments via a Python API (Juliani et al. 2020, p. 11), as shown in Figure 13. It provides an easy way of directly adjusting the reward system with in-game events, such as collision detection, via the use of C# scripts that act as a middle-man communicator in the training pipeline, leaving the training mechanism to the Python API (Juliani et al. 2020, p. 11) that bases heavily on the usage of TensorFlow, which is an open source machine learning library.

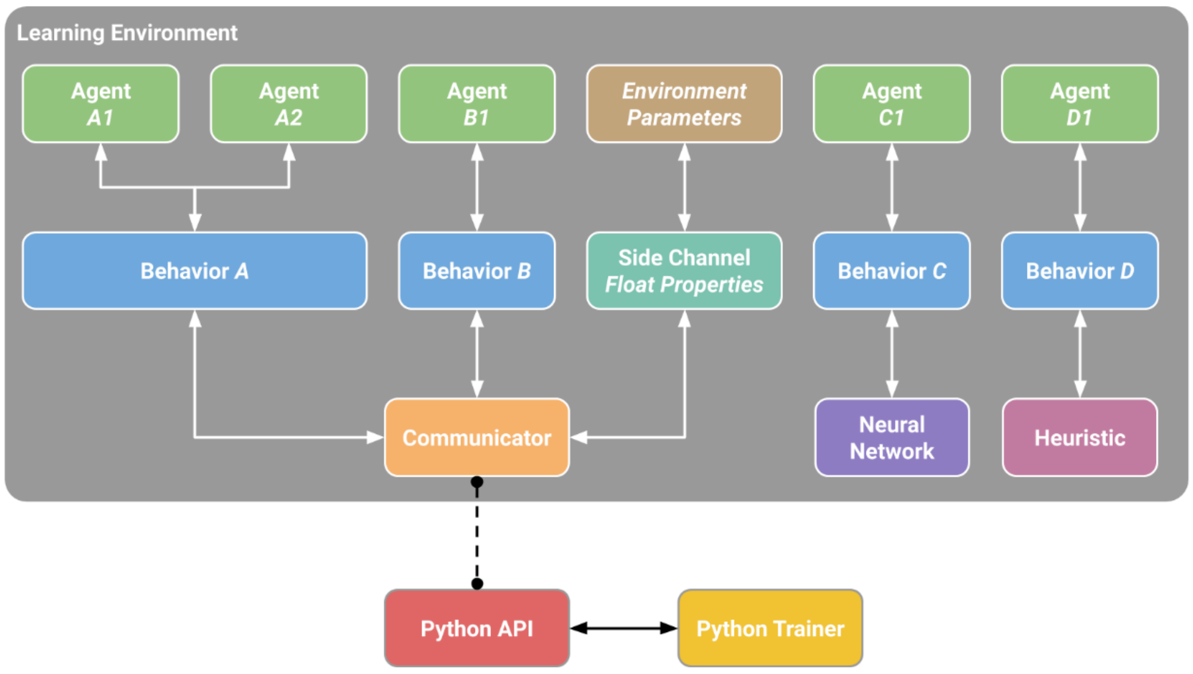


Figure 13: Unity ML-Agents Learning Environment (Juliani et al. 2020).

Training via the use of Unity ML-Agents is performed by either using one of the already internally implemented algorithms such as, for example, PPO, Soft Actor-Critic (SAC), or using an external Python trainer by using the provided Unity's gym wrapper (Juliani et al. 2020, p. 13), which allow to abstract and control the Unity learning environment by a widely used in research gym interface, provided by OpenAI (Brockman et al. 2016, p. 2). As this research bases on using a CNN for predicting the passed traffic signs, and therefore being one of the inputs of the agent, the latter will have more usage.

In a Unity learning environment, the agents are represented by in-game GameObject objects, which are able to "collect observations, take actions and receive rewards" (Juliani et al. 2020, p. 12), therefore performing the three key functions of autonomous cars: perception, planning and control. The finish of a simulation can be controlled by calling a Unity C# script or by exceeding a predefined max step value (Juliani et al. 2020, p. 12).

2.5 Conclusion

Although there is wide research both on training an agent how to drive on the road, and on detection and recognition of signs in virtual environments, there has been no publicly available research on the connection on both, which is what this research aims to achieve. As the Unity ML-Agents is a recently released toolkit for Unity, the completion of this research should result in widening the toolkit's usage and providing game developers with more tools available to use.

3 Development and Implementation Report

3.1 Car mechanics

To provide the car mechanics to the agent, a realistic car controller was imported to the project from Unity's Asset Store, as well as a car prefab. This car controller was then used as a component in the car prefab and had its settings set to be as close to a standard vehicle as possible, with some of the characteristics listed below:

- The wheel drive was set to a front-wheel-drive (FWD), as it is the most commonly sold drive system, at least in the United States (Gareffa 2018).
- The number of gears was set to 6 in total to help with the power distribution, as the higher gear ratio helps to improve acceleration (Roy 2011).
- The gearbox has been set to be automatic, as otherwise the agent would have to learn how to change gears, which would possibly be an obstacle in this research.
- The car has been limited to a speed limit of 140 kilometres per hour. This has been based on the highest motorway speed limit in the world, which is 140 kilometres per hour in Poland (Rhino Car Hire 2020). Although a higher speed would cause the agent to arrive at the goal faster, it would also leave little room for the agent to react in case, for example, turning, and therefore possibly highly interfering in training.
- The handbrake function has been removed, as it would have to be set as a possible action of the agent, which might have resulted in the agent learning to use it for braking, which would be improper to any correct driving techniques.

The car controller script has then been modified so that it did not require a first person controller dependency and could be directly controlled by an agent.

3.2 Initial test of reinforcement learning

To provide the perception of the road for the agent, eight boxes acting as ray-cast origins were placed on the front of the car (Figure 14).

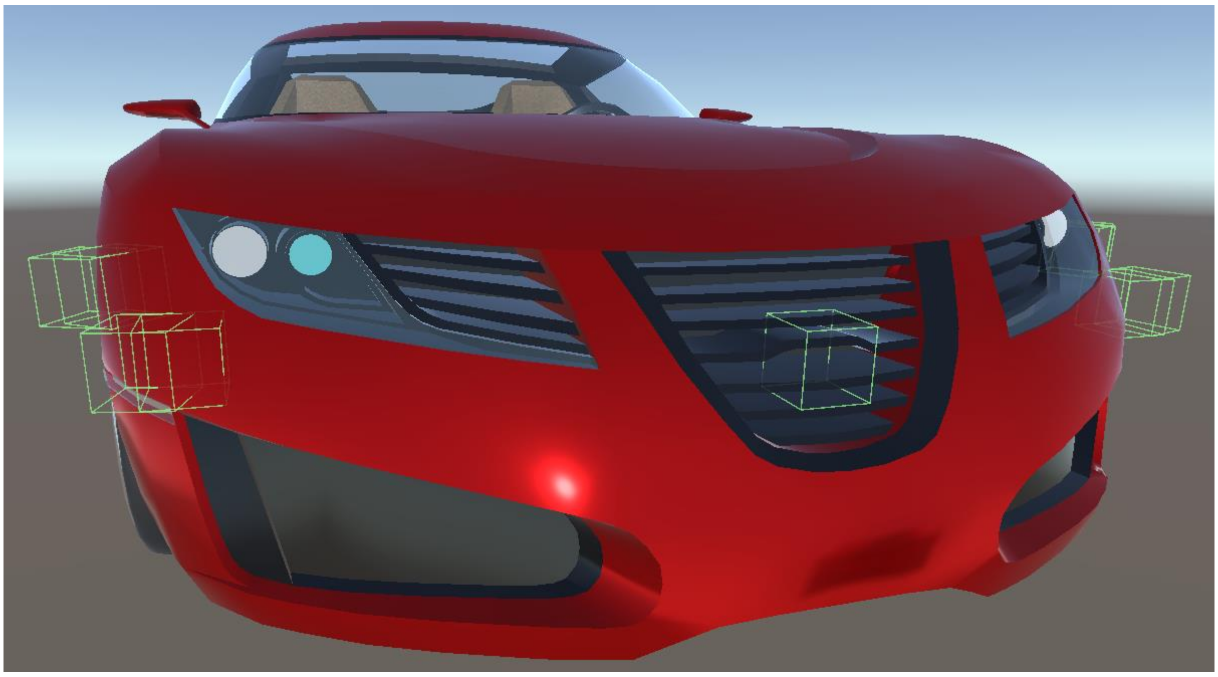


Figure 14: Boxes with ray-cast origins located on the agent's vehicle.

Using the position of these boxes, ray-casts are casted in the forward direction, with their collision results to be used as vector observations, as shown in Figure 15. These ray-casts are only visible in the Scene mode in the Unity Editor, having a yellow colour when they are colliding with the road, blue if colliding with nothing at all, and red if colliding with anything else (e.g. traffic signs).

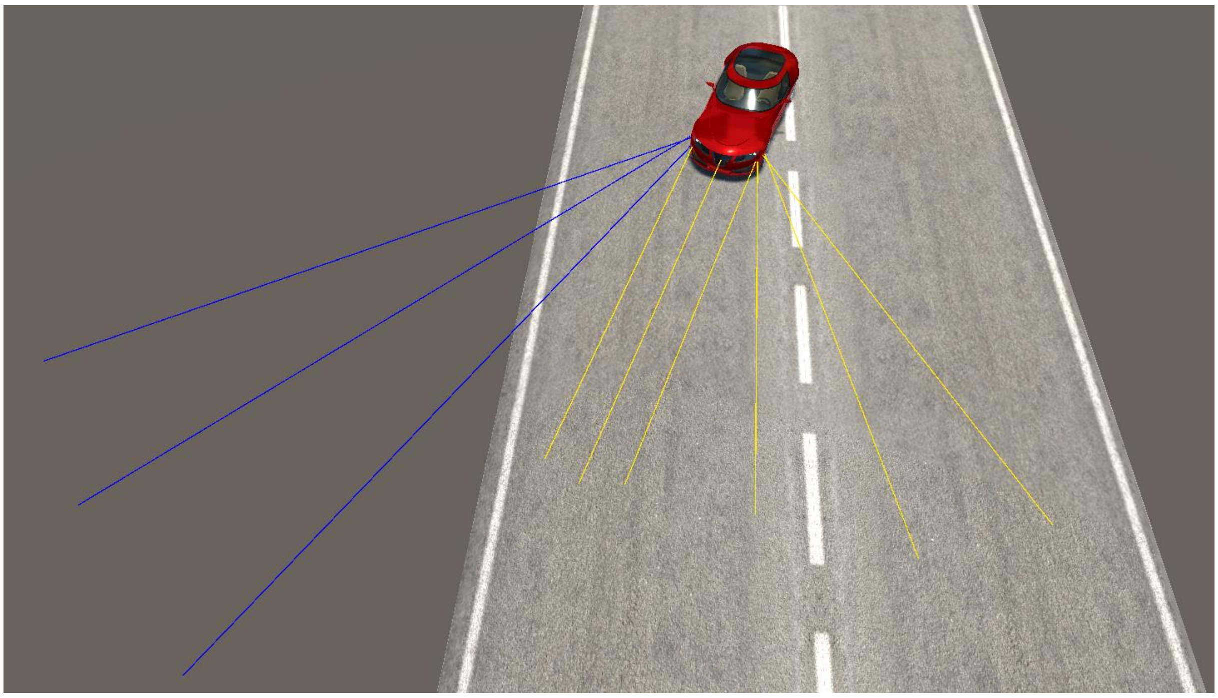


Figure 15: Results of the ray-cast collisions.

These ray-cast collisions were then added as observations for the agent by being represented as Boolean variables, having a true value when colliding with the road and a false one when not. Besides the ray-cast observations, the agent is also provided with its current speed in kilometres per hour, as well as the X-axis rotation values of the front left and right wheels.

As for the rewards, the main reward provided at all time is the speed of the car divided by 20 to keep the agent's main focus on accelerating. If the agent hits a goal at the end of the environment, a reward of 50 units is provided to ensure the agent focuses on finishing the environment.

The first penalty is provided for going reverse, which is the speed of car divided by 5. This is to ensure the car agent would always go forwards and never reverse. If the car falls off the road, a penalty of minus 50 units is provided to ensure the car agent learns to stay on the road using the ray-cast observations.

To firstly test whether the proposed system of observations, rewards and penalties worked correctly, a test environment was made with six agents in total, driving on a straight road, as shown in Figure 16. The max step value was set to 15,000 to give agents enough time to accelerate to the end goal.

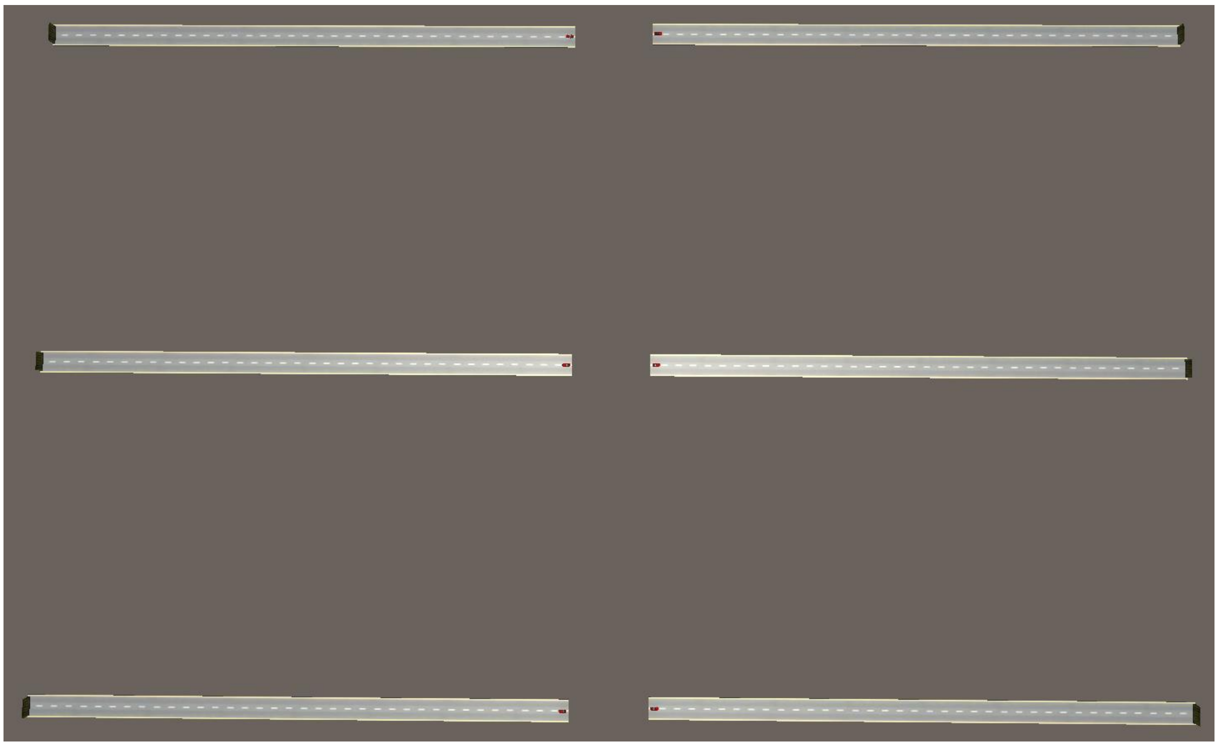


Figure 16: The initial test environment.

A virtual Python environment has been created using Anaconda. The ML-Agents toolkit was then installed using the PIP package manager, with all its other dependencies, although

TensorFlow version 1.15 had to be installed beforehand, as TensorFlow's object detection API, used later for traffic sign detection, does not fully support the higher version.

Using the ML-Agents toolkit, the model was then successfully trained using the internal implementation of the PPO algorithm. As shown in Figure 17 on the Tensorboard graph, although the max reward of around 750 has been hit at approximately the 500,000th time step, the training was left to continue to see whether it could improve the maximal reward, albeit unsuccessfully.

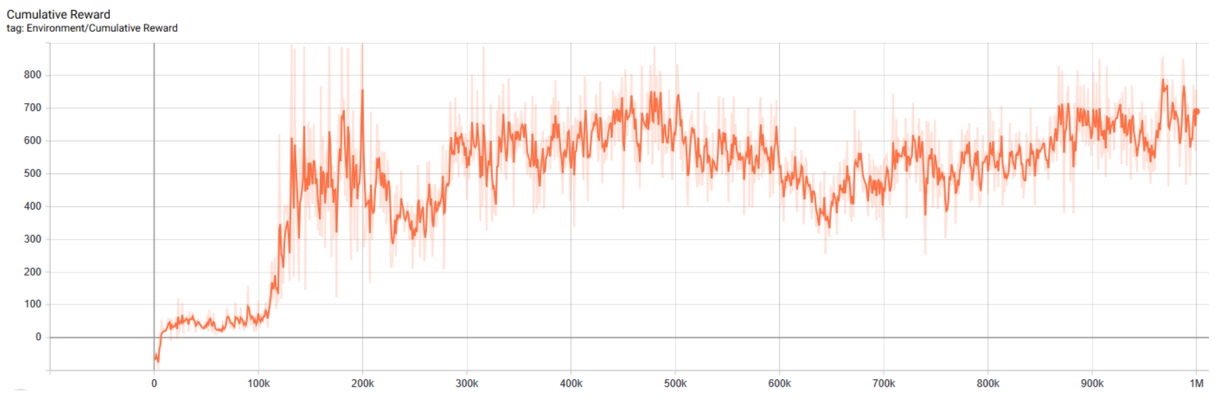


Figure 17: Cumulative reward of the test environment.

3.3 Traffic Sign Detection and Recognition

To allow for traffic sign detection in the virtual environments, the traffic signs needed to be available for use in the Unity Editor first. As no publicly available German traffic signs models were found, a package of traffic sign models was downloaded, as well as a stop sign model, and had their textures amended to be in line with the German traffic sign design, as shown in Figure 18.

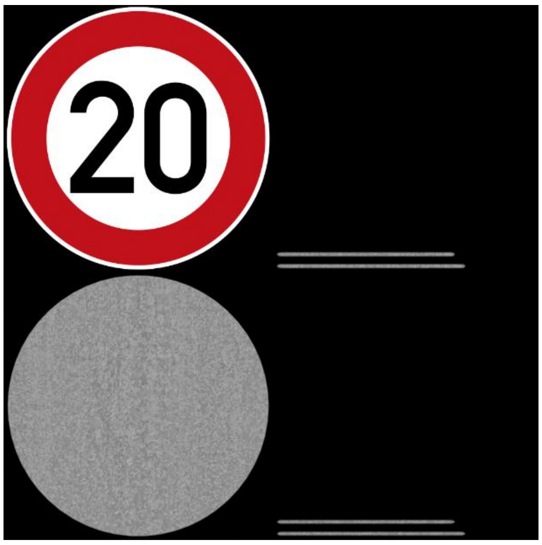
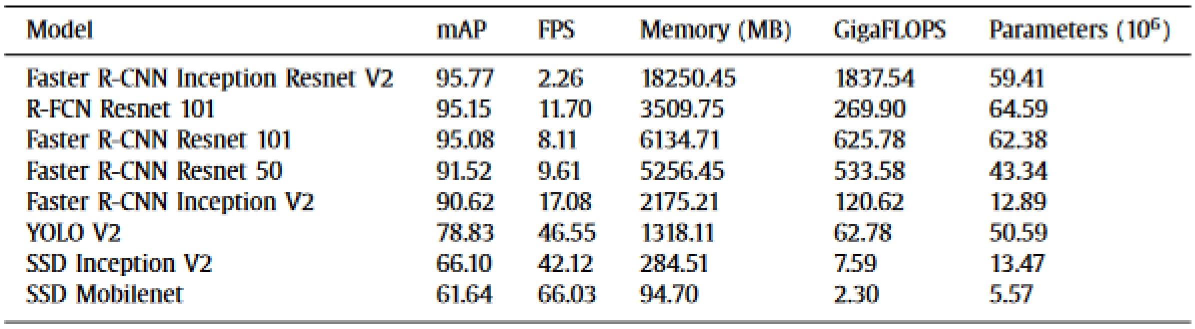


Figure 18: An example of the modified traffic sign textures.

The initial traffic signs chosen for training were the speed limits of 20, 30, 50, 60, 70, 80, 100 and 120 kilometres per hour, as well as the turn left, turn right, go straight ahead, stop and end of restrictions signs.

For the first traffic sign detection experiment, after following the research made by Arcos-García et al. (2018), which compared a number of object detection models with the use of the GTSDB dataset (Figure 19), a decision was made to use the Single Shot Detection (SSD) Mobilenet model due to its high speed and low usage of resources, which allows for more resources to be used by the game engine. However, as their research based on the detection of only the three original benchmark sign classes: prohibitory, mandatory and danger signs, the direct usage of their trained model is unavailable in this research, meaning that the model had to be trained again.



Model	mAP	FPS	Memory (MB)	GigaFLOPS	Parameters (10^6)
Faster R-CNN Inception Resnet V2	95.77	2.26	18250.45	1837.54	59.41
R-FCN Resnet 101	95.15	11.70	3509.75	269.90	64.59
Faster R-CNN Resnet 101	95.08	8.11	6134.71	625.78	62.38
Faster R-CNN Resnet 50	91.52	9.61	5256.45	533.58	43.34
Faster R-CNN Inception V2	90.62	17.08	2175.21	120.62	12.89
YOLO V2	78.83	46.55	1318.11	62.78	50.59
SSD Inception V2	66.10	42.12	284.51	7.59	13.47
SSD Mobilenet	61.64	66.03	94.70	2.30	5.57

Figure 19: Models' properties ordered by mean average precision (Arcos-García et al. 2018).

Moreover, another issue was found as the GTSDB dataset had a low amount of images for each separate traffic sign class. To solve this issue, 128 new images were created and manually annotated, meaning that each traffic sign class received approximately 10 new images. TFRecords files were created using the annotated dataset, which are files used by object detection models for training.

SSD Mobilenet V1 model from the TensorFlow object detection model zoo was acquired. As the model was already pre-trained on the COCO dataset, it had to be re-trained on the amended GTSDB dataset using the technique of transfer learning, which allows using the already acquired knowledge from one domain to another (Moltzau 2019). The training was then performed using Google Colab. Unfortunately, after 20,000 timesteps, the mean average precision, which is the mean of the average precision of each class detection, did not improve (Figure 20) even though the loss function had apparently reached its minimum of 1 unit (Figure 21). This was probably due to the large amount of similar signs, especially as is the case with speed limits, which although did not have an impact in the research of Arcos-García et al. (2018) when the model was detecting just 3 separate classes, however it brought a negative impact when detecting multiple separate classes of the dataset. The low input resolution of

300 by 300 pixels of the SSD Mobilenet model could have a negative impact as well, making it troublesome for the model to find signs of a small size.

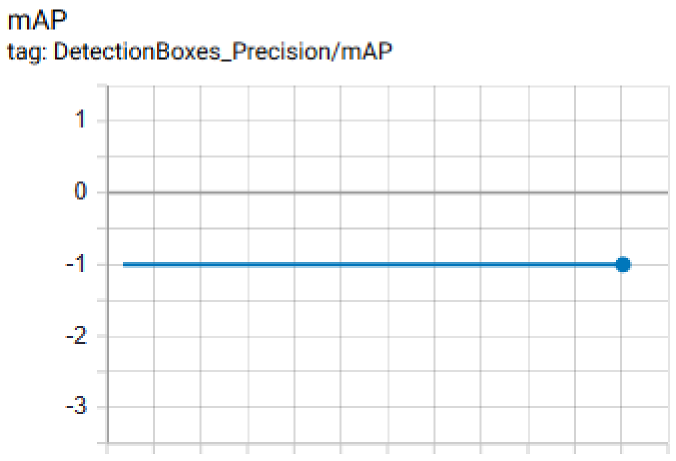


Figure 20: Mean average precision.

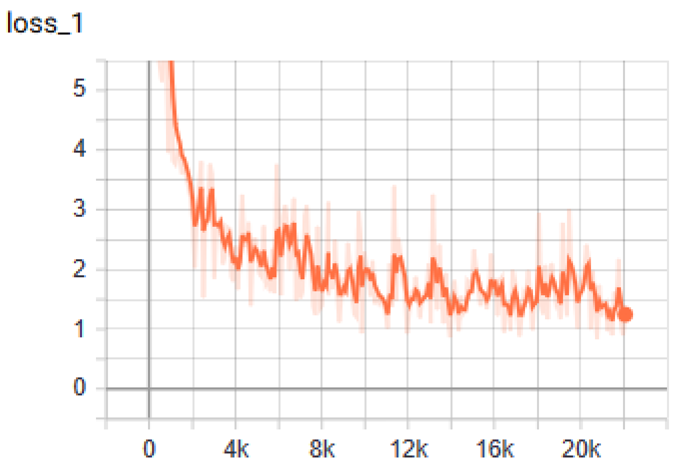


Figure 21: Loss function.

After this initial experiment, another approach was tried to combine detection with recognition by using separate neural networks for localization and classification. A ready framework by a user of the name helloyide (2018) was acquired, which used a U-Net model for localization, and a SqueezeNet model for classification, both of which are capable of achieving high performance with a low usage of resources.

As the models were already trained for the detection of separate sign classes of the GTSDB dataset, the model was evaluated on a number of the previously created Unity images. The results were that this combination was suitable for this research, although the U-Net model had trouble detecting traffic signs with a lot of distance, and the SqueezeNet model tended to mistake the speed limit classifications for one another, as shown in Figure 22.

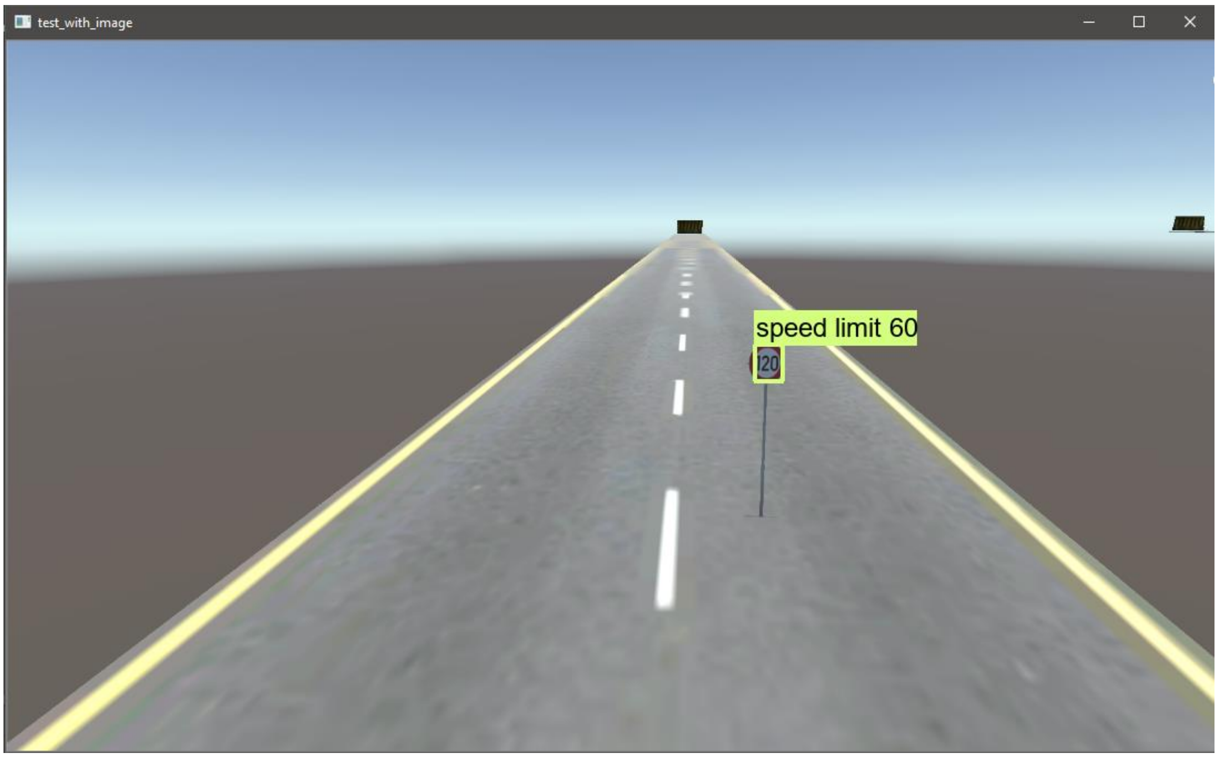


Figure 22: Example of the framework evaluation.

Although Unity provides a framework of the name Barracuda, which is an inference engine allowing to convert trained TensorFlow models for direct use in the Unity Editor, as of now it is unable to directly use the output of one model as input in another one. Due to this reason, any use of an object detection model has to be performed using Unity's gym wrapper, which has a limitation of using only one agent. The following environments had to be therefore created with this limitation in mind.

3.4 CREATION OF ENVIRONMENTS

The approach taken was to create three separate environments, so that instead of having the agent trying to learn everything in a single trial, which could lead to the minimal amount of timesteps required for the agent to successfully train end up being in millions, the agent would train on the next environment, using transfer learning to use the knowledge from the previous environment, which would allow to have the agent simply try to learn the new knowledge.

3.4.1 Turns

The first environment consists solely of turns without any speed limits, as seen in Figure 23. This environment was created in order to have the agent learn how to perform on turns. The straight road lengths were made so that the agent can gain some speed before turning, in order to learn that braking before turning is important for the long-term reward strategy.

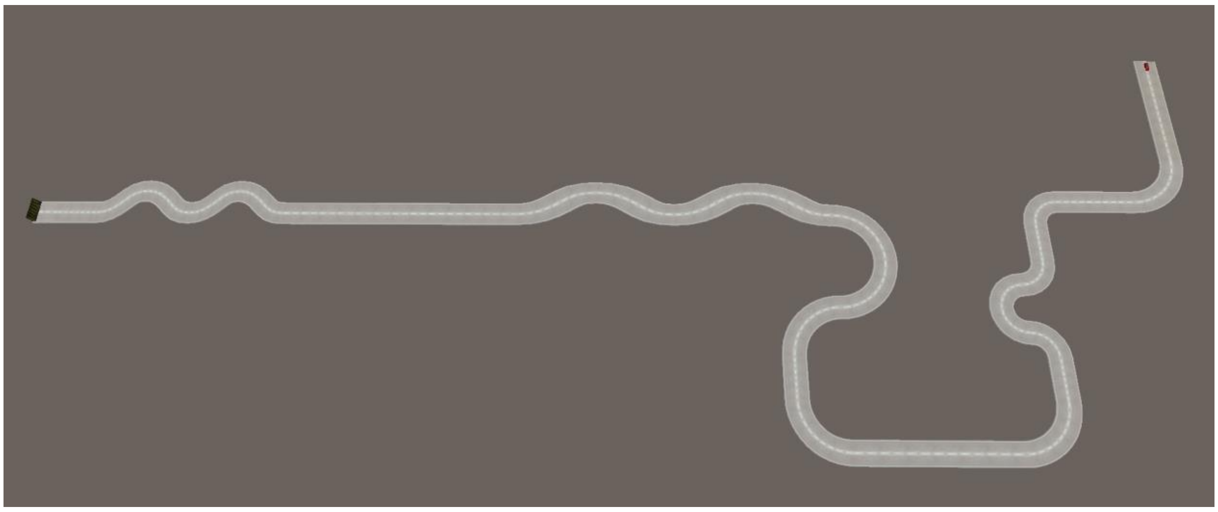


Figure 23: The first environment.

3.4.2 Speed limits

The second environment consists of a straight road, with solely speed limit traffic signs set. The traffic signs were set in this order:

- 60 km/h
- End of restrictions
- 120 km/h
- 100 km/h
- End of restrictions
- 80 km/h
- 70 km/h
- 50 km/h
- 30 km/h
- 20 km/h

The environment begins with a lot of distance to the first sign to allow the agent to gain up a large amount of speed before approaching the first sign. As shown in Figure 24, once the agent drives past the trigger collision box, the global speed limit is set to the value of the passed speed limit traffic sign. This is set in order to have the agent receive an adequate penalty for excessing the speed limit. The end of restrictions sign has been set twice, in small distance to the speed limit traffic signs in order to ensure that the agent learns the purpose of these signs.

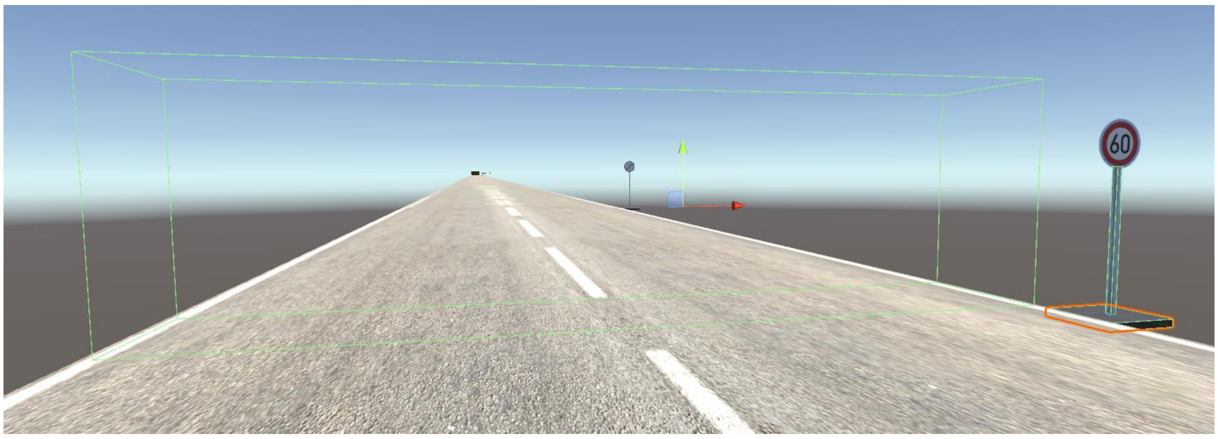


Figure 24 Collision trigger box for the reward-penalty system.

3.4.3 Forced turns

The third environment consists of two turns, with only traffic signs of the mandatory type set (Figure 25).

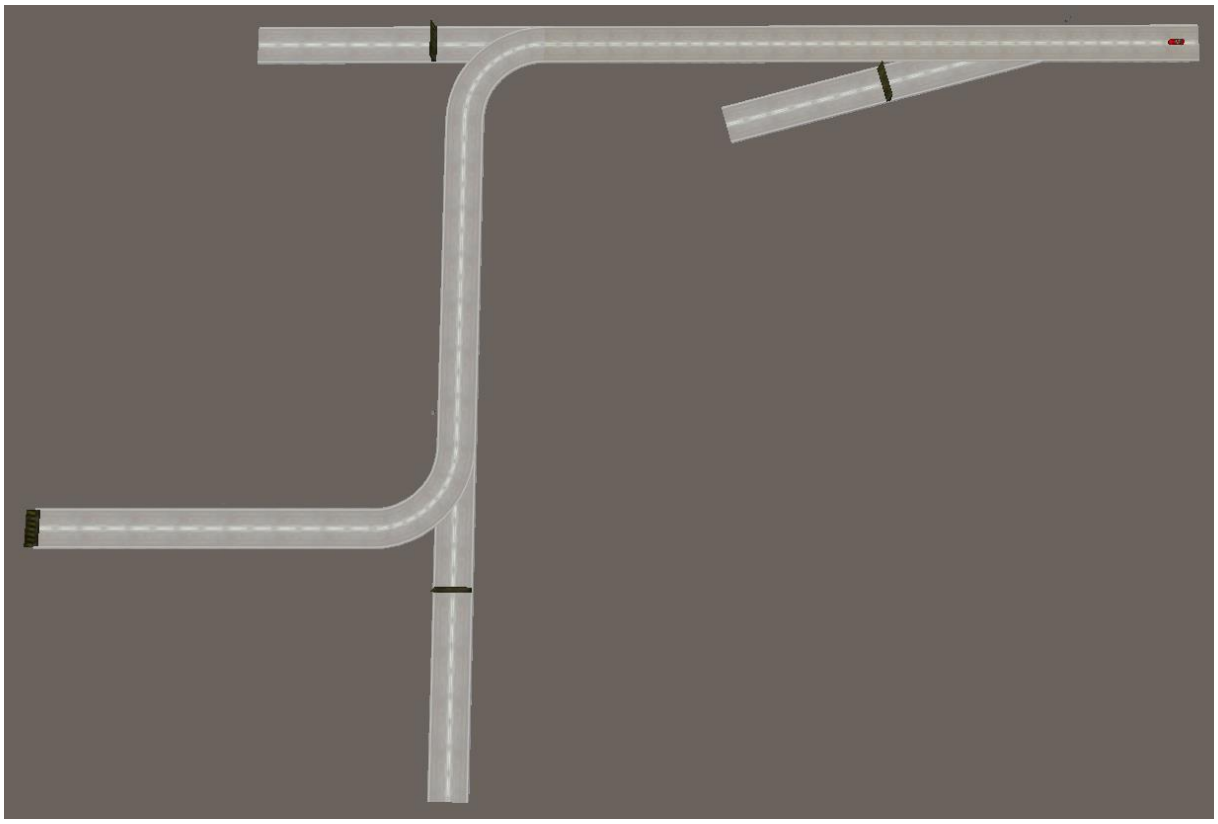


Figure 25: The third environment.

The traffic signs were set in this order:

- Go straight ahead
- Turn left
- Turn right

- Stop

If the agent does not follow the forced turn, he collides with a road blocker and receives a large penalty, resetting back to the initial position (Figure 26).

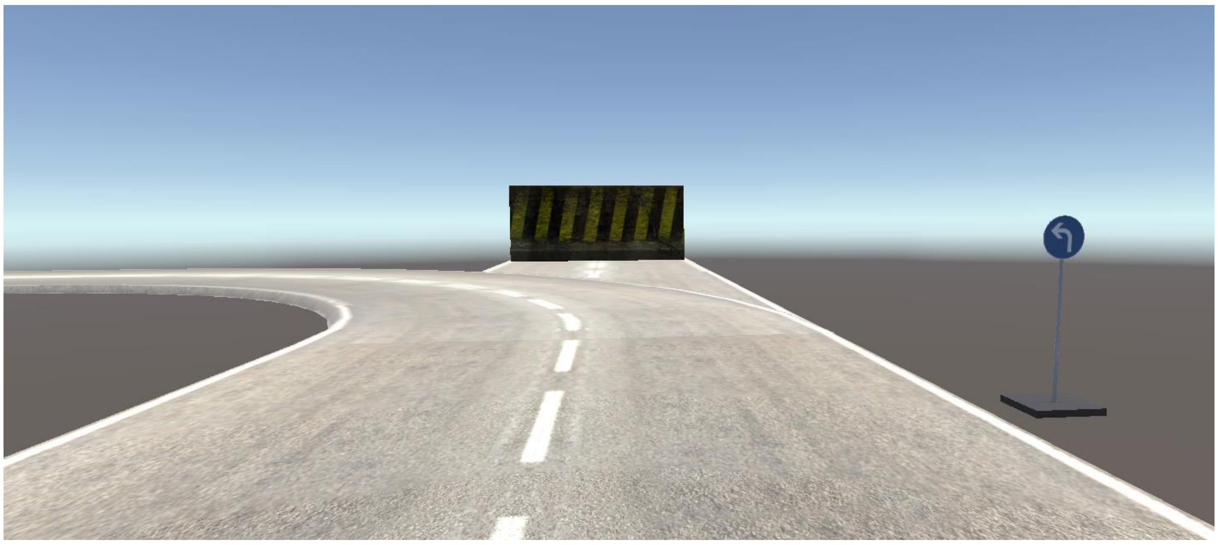


Figure 26: Example of the forced turn mechanic.

If the agent drops his speed below 5 km/h when inside the trigger collision box of the stop sign, he gets a reward, otherwise he receives an adequate penalty (Figure 27).

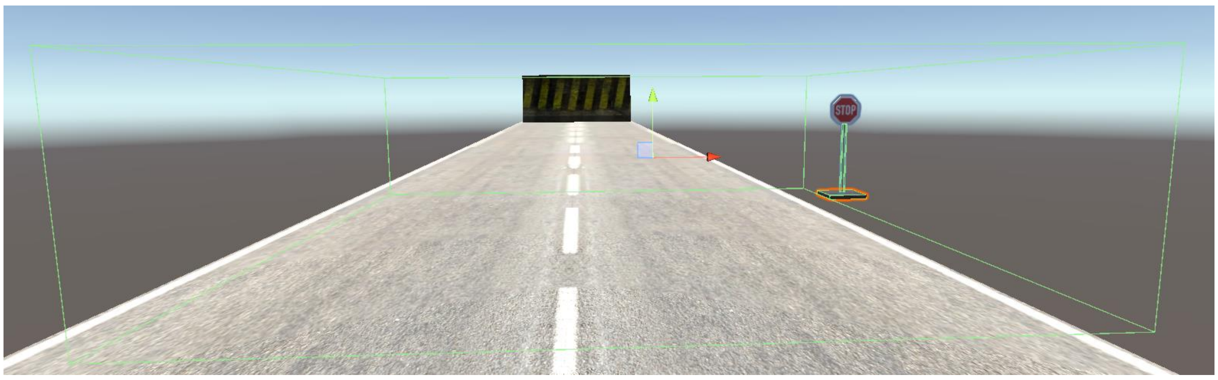


Figure 27: Stop detection mechanic.

3.4.4 Evaluation test

A test environment has been made to evaluate the trained model. The traffic signs were set in this order:

- Turn right
- 50 km/h
- Stop

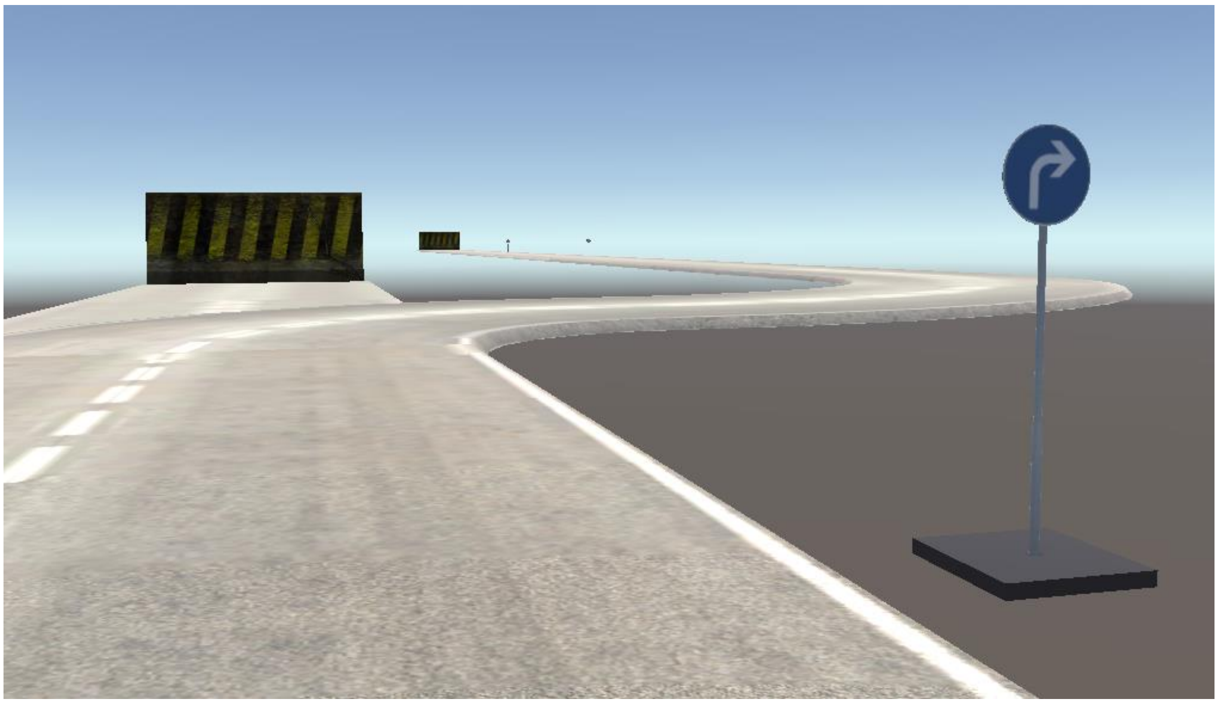


Figure 28: A turn in the evaluation test.

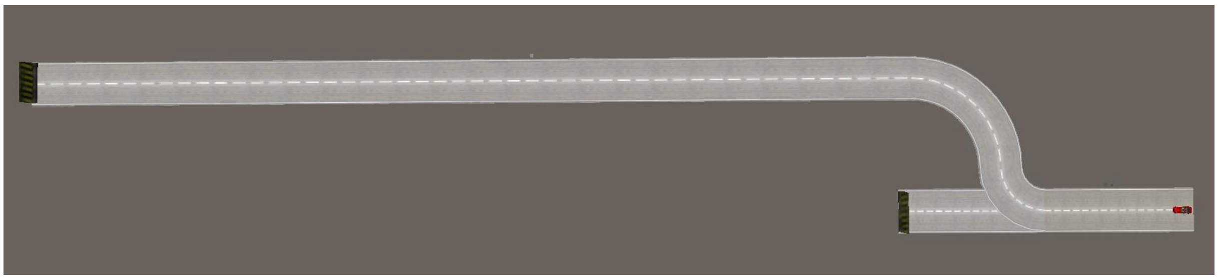


Figure 29: View of the evaluation test.

3.5 Combining the sign detection model with Unity ML-Agents

A camera has been placed at the front of the agent to collect the traffic sign visual input as an observation. To be compatible with the training environments, new penalties and a reward have been added to the agent's script.

For hitting a road sign or a "wrong way" road blocker, the agent receives a penalty of 50 units. If the agent exceeds the speed limit, he receives the difference between his speed and the speed limit divided by 5 as the penalty. Similar behaviour happens when the agent does not stop before the stop sign, although in that case he receives his whole speed divided by 5 as penalty.

As for the reward, the agent receives a reward of 50 units for successfully stopping at the stop sign.

To combine the sign detection model with the ML-Agents toolkit, the PPO2 algorithm firstly was acquired from Stable Baselines, which is a fork of OpenAI's collection of algorithms. The learning code of this algorithm has been modified to take the observation from the camera, process it in the object detection model and output the prediction class with the highest score, and a class of 0 if none detected.

The code has been modified to account for performance as well. Instead of processing the observation all the time, the model processes the observation from 1 up to 4 times per second depending on the speed the car agent is driving with, as shown below (Figure 30).

$$time\ delay\ between\ sign\ detections = \left(-\frac{3.0}{560.0}\right) * \frac{km}{h} + 1$$

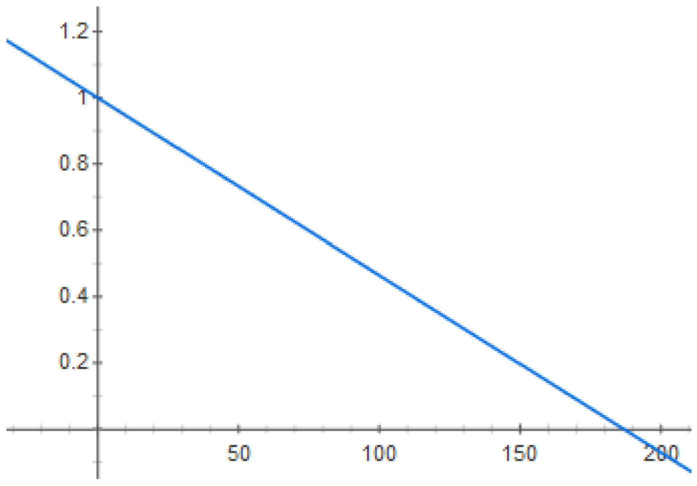


Figure 30: Graph of the speed function.

4 SELF-ASSESSMENT

Considering the extensiveness of the three computer science research fields that were aimed to be used together in this research, which are reinforcement learning, computer vision and games development, the progress made is thought as satisfactory. The first reinforcement learning model of the car agent has been successfully trained not to fall off the road, and to systemically accelerate forwards, meaning that it made ground for learning further steering behaviour.

The original aims and objectives were set too broadly, in comparison to both the time constraint and amount of technologies used, however. Although eventually the main objectives in the research were cut to only learn the agent model how to drive and adjust to the detected traffic signs, even taking a reasonably small group of 13 traffic signs meant that there are 13 different behaviours for which an appropriate learning scenario must have been planned. As it has been learned, in machine learning research, the environment and observation setup needs to be planned and well evaluated before training. As training takes long periods of time on a regular PC, there are lengthy waits between training runs to investigate if something has been setup wrongly.

Wide research into object detection has been made, with a lot of recently researched and past models personally discovered, as well as reinforcement learning algorithms, such as PPO2. With more time available, the PPO2 and the used U-Net object detection model combination could be tested on the specially created environments and evaluated whether the algorithms would be suitable for this research. Training and using new object detection models, such as TinyYOLOv3, could be then tested.

In retrospective, training the SSD Mobilenet model should have been left after firstly testing the already trained U-Net object detection model, and then seeing whether there is any room for improvement, such as a possibility of using less resources.

Using a framework in beta version was found to be a hard task and a lesson for life, as was the case with the ML-Agents toolkit. Lacking documentation and the tear between C# and Python was a challenge in this research. The inability of accessing direct communication between TensorFlow and Unity forced to find a workaround in order to combine both computer vision with reinforcement learning.

In the end, the amassed personal computer vision and machine learning knowledge was found to be a major advantage of this research even in the light of the previously mentioned difficulties.

5 BIBLIOGRAPHY

Atienza, R., 2018. *Advanced Deep Learning with Keras*. [online] Available from: https://subscription.packtpub.com/book/big_data_and_business_intelligence/9781788629416 [Accessed 10 Jun. 2020] Packt Publishing Limited.

Goodfellow, I., Bengio, Y. and Courville, A., 2016. *Deep Learning* [online]. Available from: http://www.deeplearningbook.org [Accessed 10 Jun. 2020]. MIT Press.

Maurer, M., Gerdes, J., Lenz, B. and Winner, H., 2016. *Autonomous Driving*. [online] Available from: https://link.springer.com/book/10.1007/978-3-662-48847-8 [Accessed 10 Jun. 2020] Berlin: Springer Berlin.

Sutton, R. and Barto, A., 2014. *Reinforcement Learning: An Introduction*. [online] Available from: https://ieeexplore.ieee.org/book/6267343 [Accessed 10 Jun. 2020] Cambridge, Massachusetts: The MIT Press.

6 References

Arcos-García, Á., Álvarez-García, J. and Soria-Morillo, L., 2018. Evaluation of deep neural networks for traffic sign detection systems. *Neurocomputing* [online], 316, 332-344. Available from: https://www.sciencedirect.com/science/article/pii/S092523121830924X?via%3Dihub [Accessed 11 Jun 2020].

Arcos-García, Á., Álvarez-García, J. and Soria-Morillo, L., 2018. *Table 5 Models' properties ordered by mAP.* [image] Available from: https://www.sciencedirect.com/science/article/pii/S092523121830924X?via%3Dihub [Accessed 11 Jun 2020].

Arulkumaran, K., Deisenroth, M., Brundage, M. and Bharath, A., 2017. Deep Reinforcement Learning: A Brief Survey. *IEEE Signal Processing Magazine* [online], 34 (6), 27. Available from: https://arxiv.org/pdf/1708.05866.pdf [Accessed 10 Jun. 2020].

Bojarski, M., Del Testa, D., Dworakowski, D., Firner, B., Flepp, B., Goyal, P., Jackel, L., Monfort, M., Muller, U., Zhang, J., Zhang, X., Zhao, J. and Zieba, K., 2016. *End to End Learning for Self-Driving Cars* [online], p. 1. NVIDIA Corporation. Available from: https://arxiv.org/pdf/1604.07316.pdf [Accessed 10 Jun. 2020].

Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J. and Zaremba, W., 2016. *OpenAI Gym* [online], p. 2. OpenAI. Available from: https://arxiv.org/pdf/1606.01540.pdf [Accessed 10 Jun. 2020].

Bojarski, M., Del Testa, D., Dworakowski, D., Firner, B., Flepp, B., Goyal, P., Jackel, L., Monfort, M., Muller, U., Zhang, J., Zhang, X., Zhao, J. and Zieba, K., 2016. *End to End Learning for Self-Driving Cars* [online], p. 4. NVIDIA Corporation. Available from: https://arxiv.org/pdf/1604.07316.pdf [Accessed 10 Jun. 2020].

Cao, J., Song, C., Peng, S., Xiao, F. and Song, S., 2019. Improved Traffic Sign Detection and Recognition Algorithm for Intelligent Vehicles. *Sensors* [online], 19 (18), 4027. Available from: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC6767627/pdf/sensors-19-04021.pdf [Accessed 10 Jun. 2020].

Cao, J., Song, C., Peng, S., Xiao, F. and Song, S., 2019. *Figure 6. The improved LeNet-5 network model structure*. [image] Available from: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC6767627/pdf/sensors-19-04021.pdf [Accessed 10 Jun. 2020].

Deeplizard, 2018. *Policies and Value Functions - Good Actions for a Reinforcement Learning Agent* [online]. Available from: https://deeplizard.com/learn/video/eMxOGwbdqKY [Accessed 10 Jun. 2020].

Deeplizard, 2018. *Mathematically we define vπ(s) as.* [image] Available from: https://deeplizard.com/learn/video/eMxOGwbdqKY [Accessed 10 Jun. 2020].

Dobrev, D., 2004. *A Definition of Artificial Intelligence* [online], p. 2. Available from: https://arxiv.org/abs/1210.1568 [Accessed 9 Jun. 2020].

Gareffa, P., 2018. *All About Front-, Rear-, Four- and All-Wheel Drive* [online]. Edmunds. Available from https://www.edmunds.com/car-technology/what-wheel-drive.html [Accessed 11 Jun. 2020].

Hafner, D., 2016. *Deep Reinforcement Learning From Raw Pixels in Doom* [online], p. 26. Haso Plattner Institute. Available from: https://arxiv.org/pdf/1610.02164.pdf [Accessed 10 Jun. 2020].

helloyide, 2018. *real time German traffic sign recognition* [online]. Available from: https://github.com/helloyide/real-time-German-traffic-sign-recognition [Accessed 11 Jun. 2020].

Houben, S., Stallkamp, J., Salmen, J., Schlipsing, M. and Igel, C., 2013. Detection of traffic signs in real-world images: The German traffic sign detection benchmark. *The 2013 International Joint Conference on Neural Networks (IJCNN)* [online]. Available from: https://ieeexplore.ieee.org/document/6706807 [Accessed 10 Jun. 2020].

Institut für Neuroinformatik, 2013. *00009.ppm.* [image] Available from: http://benchmark.ini.rub.de/?section=gtsdb&subsection=dataset [Accessed 10 Jun. 2020].

Jo, K., Kim, J., Kim, D., Jang, C. and Sunwoo, M., 2014. Development of Autonomous Car—Part I: Distributed System Architecture and Development Process. *IEEE Transactions on Industrial Electronics*, 61 (12), p. 7131-7132. [online] Available from: https://ieeexplore.ieee.org/abstract/document/6809196 [Accessed 9 Jun. 2020].

Jo, K., Kim, J., Kim, D., Jang, C. and Sunwoo, M., 2014. Development of Autonomous Car—Part I: Distributed System Architecture and Development Process. *IEEE Transactions on Industrial Electronics*, 61 (12), p. 7141. [online] Available from: https://ieeexplore.ieee.org/abstract/document/6809196 [Accessed 9 Jun. 2020].

Jo, K., Kim, J., Kim, D., Jang, C. and Sunwoo, M., 2014. *Fig. 1. Basic functions of autonomous cars.* [image] IEEE. Available from: https://ieeexplore.ieee.org/abstract/document/6809196 [Accessed 9 Jun. 2020].

Juliani, A., Berges, V., Tang, E., Cohen, A., Harper, J., Elion, C., Goy, C., Gao, Y., Henry, H., Mattar, M. and Lange, D., 2020. *Unity: A General Platform for Intelligent Agents* [online], p. 11-13. San Francisco: Unity Technologies. Available from: https://arxiv.org/pdf/1809.02627.pdf [Accessed 10 Jun. 2020].

Juliani, A., Berges, V., Tang, E., Cohen, A., Harper, J., Elion, C., Goy, C., Gao, Y., Henry, H., Mattar, M. and Lange, D., 2020. *Figure 2: A Learning Environment (as of version 1.0) created using the Unity Editor contains Agents and an Academy. The Agents are responsible for collecting observations and executing actions. The Academy is responsible for global coordination of the environment simulation.* [image] Unity Technologies. Available from: https://arxiv.org/pdf/1809.02627.pdf [Accessed 10 Jun. 2020].

kwea123, 2018. *Autocar* [online]. kwea123. Available from: https://github.com/kwea123/RL/tree/master/ai/unity_test/autocar [Accessed 10 Jun. 2020].

kwea123, 2018. *Test the camera.* [image] Available from: https://github.com/kwea123/RL/blob/master/ai/unity_test/autocar/autocar.ipynb [Accessed 10 Jun. 2020].

Learned-Miller, E., 2011. *Introduction to Computer Vision* [online], p. 2. Department of Computer Science, University of Massachusetts, Amherst. Available from: https://people.cs.umass.edu/~elm/Teaching/Docs/IntroCV_1_19_11.pdf [Accessed 10 Jun. 2020].

Li, Y., 2018. *Deep Reinforcement Learning* [online], p. 5. Available from: https://arxiv.org/pdf/1810.06339.pdf [Accessed 10 Jun. 2020].

Merriam-Webster, 2020. Perception. *Merriam-Webster Dictionary* [online]. Available from: https://www.merriam-webster.com/dictionary/perception [Accessed 10 Jun. 2020].

Moltzau, A., 2019. *What is Transfer Learning?* [online]. Avaliable from: https://medium.com/@alexmoltzau/what-is-transfer-learning-6ebb03be77ee [Accessed 11 Jun. 2020].

Nielsen, M., 2015. *A neural network with many hidden layers*. [image] Available from: http://neuralnetworksanddeeplearning.com/chap5.html [Accessed 9 Jun. 2020].

OpenAI, 2018. *A non-exhaustive, but useful taxonomy of algorithms in modern RL*. [image] Available from: https://spinningup.openai.com/en/latest/spinningup/rl_intro2.html [Accessed 10 Jun. 2020].

Pan, X., You, Y., Wang, Z. and Lu, C., 2017. *Virtual to Real Reinforcement Learning for Autonomous Driving* [online], p. 10. Available from: https://arxiv.org/pdf/1704.03952.pdf [Accessed 11 Jun 2020].

Pettigrew, S., 2016. Why public health should embrace the autonomous car. *Australian and New Zealand Journal of Public Health* [online], 41 (1), p. 5. Available from: https://espace.curtin.edu.au/bitstream/handle/20.500.11937/36130/246964.pdf [Accessed 10 Jun. 2020].

Raschka, S., 2017. *3 different types of machine learning.* [image] Michigan State University. Available from: https://www.kdnuggets.com/2017/11/3-different-types-machine-learning.html [Accessed 10 Jun. 2020].

Rhino Car Hire, 2020. *Speed Limits by Country* [online]. Available from: https://www.rhinocarhire.com/Drive-Smart-Blog/Speed-Limits-by-Country.aspx#/ [Accessed 11 Jun. 2020].

Roy, R., 2011. *Gears Galore: How Many Speeds Is Too Many?* [online]. PopularMechanics. Available from: https://www.popularmechanics.com/cars/a7243/gears-galore-how-many-speeds-is-too-many/ [Accessed 11 Jun. 2020].

Sachan, A., 2017. *Zero to Hero: Guide to Object Detection using Deep Learning: Faster R-CNN,YOLO,SSD* [online]. CV-Tricks. Available from: https://cv-tricks.com/object-detection/faster-r-cnn-yolo-ssd/ [Accessed 11 Jun. 2020].

SAE International, 2014. *Levels of Driving Automation.* [image] Inverse. Available from: https://www.inverse.com/article/26661-understanding-autonomous-cars [Accessed 10 Jun. 2020].

Simonini, T., 2018. *action = policy(state)*. [image] Available from: https://www.freecodecamp.org/news/an-introduction-to-reinforcement-learning-4339519de419/ [Accessed 10 Jun. 2020].

Torabi, F., Warnell, G. and Stone, P., 2018. *Behavioral Cloning from Observation* [online], p. 2. The University of Texas at Austin. Available from: https://arxiv.org/pdf/1805.01954.pdf [Accessed 10 Jun. 2020].

Udacity, 2019. *Udacity's Self-Driving Car Simulator* [online]. Udacity. Available from: https://github.com/udacity/self-driving-car-sim [Accessed 10 Jun. 2020].

Weng, L., 2018. *Fig. 1. An agent interacts with the environment, trying to take smart actions to maximize cumulative rewards.* [image] Available from: https://lilianweng.github.io/lil-log/2018/02/19/a-long-peek-into-reinforcement-learning.html [Accessed 10 Jun. 2020].

Zhu, M., Hu, J., Pu, Z., Cui, Z., Yan, L. and Wang, Y., 2019. *Traffic Sign Detection and Recognition for Autonomous Driving in Virtual Simulation Environment* [online]. University of Washington. Available from: https://arxiv.org/pdf/1911.05626.pdf [Accessed 10 Jun 2020].

Zhu, M., Hu, J., Pu, Z., Cui, Z., Yan, L. and Wang, Y., 2019. *Figure 9: Roundabout detection.* [image] University of Washington. Available from: https://arxiv.org/pdf/1911.05626.pdf [Accessed 10 Jun 2020].