Published by :
http://www.ijert.org

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 9 Issue 11, November-2020

Machine Learning based Self Driving Escort Vehicle

A. Rama Chetan
B.Tech. CSE Student,
Indian Institute of Technology,
Bhubaneswar

Dr. Manoranjan Satpathy
Associate Professor of CSE
School of Electrical Sciences
IIT Bhubaneswar, India

Dr. A. Arjuna Rao
Professor & Principal
Miracle Educational Society Group of
Institutions, Bhogapuram, AP, India

***Abstract*— In this era, the vehicles are focused to be automated to offer human driver relaxed driving. Various aspects are considered in the field of automobile which makes a vehicle automated. Google has started research on self-driving cars since 2010 and is still developing new changes to offer an entirely new level to the automated vehicles. Present paper focused on applications of an automated car, one in which two vehicles have the same destination, and one knows the route, where others do not. The following vehicle will follow the guiding vehicle automatically. The various applications are automated driving during the heavy holdup, hence relaxing the driver from continuously pushing brake, accelerator, or clutch. The idea described during this project has been taken from the Google car, defining the one aspect here into account is making the destination dynamic. This can be done by a vehicle automatically following the destination of another vehicle. Since taking intelligent decisions within the traffic is additionally a problem for the automated vehicle, this aspect has also been considered during this project.**

Keywords*— *Autonomous Vehicles, Machine Learning, Sensorimotor, Carla, Reinforcement Learning, Deep Learning, Modular Pipeline, Driving Simulator, Traffic Constraints, Navigation, A3C Algorithm, Artificial Intelligence.

1. INTRODUCTION

Sensorimotor control in three-dimensional environments remains a major challenge in machine learning and robotics. The development of autonomous ground vehicles is a long-studied instantiation of this problem. Its most difficult form is navigation in densely populated urban environments. This setting is particularly challenging due to complex multi-agent dynamics at traffic intersections; the necessity to track and respond to the motion of tens or hundreds of other actors that may be in view at any given time; prescriptive traffic rules that necessitate recognizing street signs, street lights, and road markings and distinguishing between multiple types of other vehicles; the long tail of rare events – road construction, a child running onto the road, an accident ahead, a rogue driver barreling on the wrong side; and the necessity to rapidly reconcile conflicting objectives, such as applying appropriate deceleration when an absent-minded pedestrian strays onto the road ahead but another car is rapidly approaching from behind and may rear-end if one brakes too hard. Research in autonomous urban driving is hindered by infrastructure costs and the logistical difficulties of training and testing systems in the physical world. Instrumenting and operating even one robotic car require significant funds and manpower. And a single vehicle is far from enough for collecting the requisite data that cover the multitude of corner cases that must be processed for both training and validation.

This is true for classic modular pipelines and even more so for data hungry deep learning techniques. Training and validation of sensorimotor control models for urban driving in the physical world is beyond the reach of most research groups. An alternative is to train and validate driving strategies in simulation. Simulation can democratize research in autonomous urban driving. It is also necessary for system verification, since some scenarios are too dangerous to be staged in the physical world (e.g., a child running onto the road ahead of the car). Simulation has been used for training driving models since the early days of autonomous driving research. More recently, racing simulators have been used to evaluate new approaches to autonomous driving. Custom simulations setups are commonly used to train and benchmark robotic vision systems. And commercial games have been used to acquire high-fidelity data for training and benchmarking visual perception systems. While ad-hoc use of simulation in autonomous driving research is widespread, existing simulation platforms are limited. Open-source racing simulators such as TORCS do not present the complexity of urban driving: they lack pedestrians, intersections, cross traffic, traffic rules, and other complications that distinguish urban driving from track racing. And commercial games that simulate urban environments at high fidelity, such as Grand Theft Auto V , do not support detailed benchmarking of driving policies: they have little customization and control over the environment, limited scripting and scenario specification, severely limited sensor suite specification, no detailed feedback upon violation of traffic rules, and other limitations due to their closed-source commercial nature and fundamentally different objectives during their development.

In this project, we introduce CARLA (Car Learning to Act) – an open simulator for urban driving. CARLA has been developed **[1]** from the ground up to support training, prototyping, and validation of autonomous driving models, including both perception and control. CARLA is an open platform. Uniquely, the content of urban environments provided with CARLA is also free. It includes urban layouts, a multitude of vehicle models, buildings, pedestrians, street signs, etc. The simulation platform supports flexible setup of sensor suites and provides signals that can be used to train driving strategies, such as GPS coordinates, speed,

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 9 Issue 11, November-2020

acceleration, and detailed data on collisions and other infractions. A wide range of environmental conditions can be specified, including weather and time of day. Several such environmental conditions are illustrated in Figure 1. We use CARLA to study the performance of three approaches to autonomous driving. The first is a classic modular pipeline that comprises a vision-based perception module, a rule-based planner, and a maneuver controller. The second is a deep network that maps sensory input to driving commands, trained end-to-end via imitation learning. The third is also a deep network, trained end-to-end via reinforcement learning.

Figure 1: A street in Town 2, shown from a third-person view in four weather conditions. Clockwise from top left: clear day, daytime rain, daytime shortly after rain, and clear sunset.

We use CARLA to stage-controlled goal-directed navigation scenarios of increasing difficulty. We manipulate the complexity of the route that must be traversed, the presence of traffic, and the environmental conditions. The experimental results shed light on the performance characteristics of the three approaches.

2. IMPLEMENTATION

Reinforcement Learning allows machines (known as agents) to learn by experimentation **[2]**. Imagine learning how to walk: At first, you move your legs in a certain pattern but fall. You fall down a bunch of times, but eventually, after many different trials, you will slowly learn how to move your legs to walk. Reinforcement Learning uses the same principles. More formally, an agent in an environment with a specific state has a set of actions that it can perform. After performing those actions, it receives a reward which tells it how good that action was. Of course, we want to receive the highest rewards which correspond with our objective. The Bellman Equation is used to account for future rewards as it is normally a series of actions that leads to a positive outcome **[3]**. In Q-Learning, we use these rewards to update Q-Values that tell us how good/desirable a certain state **[4]** is. In Deep Q-Learning, instead of storing Q-Values, we instead use a Deep Neural Network which allows us to approximate Q-Values, given our state as input **[5]**. Next time our agent moves through our environment, it will use the Deep Q-Network to generate Q-Values for each action and take the action with the highest Q-Value. After initiating the self-driving sequence, we then approach the "lead vehicle following" scenario. Our vehicle is initially a self-driving vehicle and then we execute the "lead vehicle following" scenario whenever necessary.

3.1 Simulation Engine

CARLA has been built for flexibility and realism in the rendering and physics simulation. It is implemented as an open-source layer over Unreal Engine 4, enabling future extensions by 2 the community. The engine provides state-of-the-art rendering quality, realistic physics, basic NPC logic, and an ecosystem of interoperable plugins. The engine itself is free for non-commercial use. CARLA simulates a dynamic world and provides a simple interface between the world and an agent that interacts with the world. To support this functionality, CARLA is designed as a server-client system, where the server runs the simulation and renders the scene. The client API is implemented in Python and is responsible for the interaction between the autonomous agent and the server via sockets. The client sends commands and meta-commands to the server and receives sensor readings in return. Commands control the vehicle and include steering, accelerating, and braking. Meta Commands control the behavior of the server and are used for resetting the simulation, changing the properties of the environment, and modifying the sensor suite. Environmental properties include weather conditions, illumination, and density of cars and pedestrians. When the server is reset, the agent is re-initialized at a new location specified by the client 2.

3.1.1 Environment

The environment is composed of 3D models of static objects such as buildings, vegetation, traffic signs, and infrastructure, as well as dynamic objects such as vehicles and pedestrians. All models are carefully designed to reconcile visual quality and rendering speed: we use low-weight geometric models and textures but maintain visual realism by carefully crafting the materials and making use of variable level of detail. All 3D models share a common scale, and their sizes reflect those of real objects. At the time of writing, our asset library includes 40 different buildings, 16 animated vehicle models, and 50 animated pedestrian models. We used these assets to build urban environments via the following steps: (a) laying out roads and sidewalks; (b) manually placing houses, vegetation, terrain, and traffic infrastructure; and (c) specifying locations where dynamic objects can appear (spawn). This way we have designed two towns: Town 1 with a total of 2.9 km of drivable roads, used for training, and Town 2 with 1.4 km of drivable roads, used for testing. The two towns are shown in the supplement. One of the challenges in the development of CARLA was the configuration of the behavior of non-player characters, which is important for realism. We based the non-player vehicles on the standard UE4 vehicle model (PhysX Vehicles). Kinematic parameters were adjusted for realism. We also implemented a basic controller that governs non-player vehicle behavior: lane following, respecting traffic lights, speed limits, and decision making at intersections. Vehicles and pedestrians can detect and avoid each other. More advanced non-player vehicle controllers can be integrated in the future. Pedestrians navigate the streets according to a town-specific navigation map, which conveys a location-based cost. This cost is designed to encourage pedestrians to walk along sidewalks and marked road crossings but allows them to crossroads at any point. Pedestrians wander around town in accordance

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 9 Issue 11, November-2020

with this map, avoiding each other and trying to avoid vehicles. If a car collides with a pedestrian, the pedestrian is deleted from the simulation and a new pedestrian is spawned at a different location after a brief time interval. To increase visual diversity, we randomize the appearance of non-player characters when they are added to the simulation. Each pedestrian is clothed in a random outfit sampled from a pre-specified wardrobe and is optionally equipped with one or more of the following: a smartphone, shopping bags, a guitar case, a suitcase, a rolling bag, or an umbrella. Each vehicle is painted at random according to a model-specific set of materials. We have also implemented a variety of atmospheric conditions and illumination regimes. These differ in the position and color of the sun, the intensity and color of diffuse sky radiation, as well as ambient occlusion, atmospheric fog, cloudiness, and precipitation. Currently, the simulator supports two lighting conditions – midday and sunset – as well as nine weather conditions, differing in cloud cover, level of precipitation, and the presence of puddles in the streets. This results in a total of 18 illumination weather combinations. (In what follows we refer to these as weather, for brevity.) Four of these are illustrated in Figure 1.

3.1.2 Sensors

CARLA allows for flexible configuration of the agent's sensor suite. At the time of writing, sensors are limited to RGB cameras and to pseudo-sensors that provide ground-truth depth and semantic segmentation. These are illustrated in Figure 2. The number of cameras and their type and position can be specified by the client. Camera parameters include 3D location, 3D orientation with respect to the car's coordinate system, field of view, and depth of field.

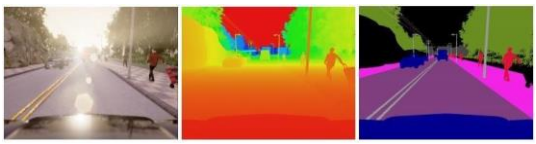

Figure 2: Three of the sensing modalities provided by CARLA. From left to right: normal vision camera, ground-truth depth, and ground-truth semantic segmentation.

The semantic segmentation pseudo-sensor provides 12 semantic classes: road, lane marking, traffic sign, sidewalk, fence, pole, wall, building, vegetation, vehicle, pedestrian, and other. In addition to sensor and pseudo-sensor readings, CARLA provides a range of measurements associated with the state of the agent and compliance with traffic rules. Measurements of the agent's state include vehicle location and orientation with respect to the world coordinate system (akin to GPS and compass), speed, acceleration vector, and accumulated impact from collisions. Measurements concerning traffic rules include the percentage of the vehicle's footprint that impinges on wrong-way lanes or sidewalks, as well as states of the traffic lights and the speed limit at the current location of the vehicle.

Finally, CARLA provides access to exact locations and bounding boxes of all dynamic objects in the environment. These signals play an important role in training and evaluating driving policies. The below Figure 3 shows the positions of sensors.

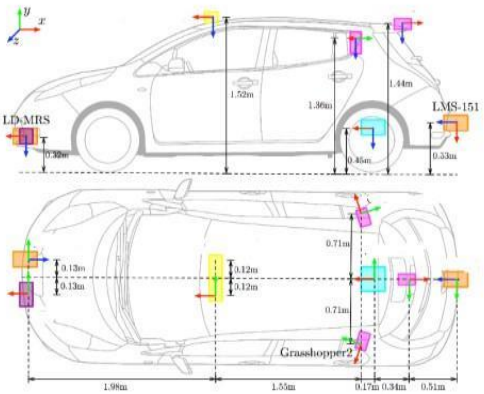

Figure 3: Picture showing the sensors positions on the simulated car

3.2 Autonomous Driving

CARLA supports development, training, and detailed performance analysis of autonomous driving systems. We have used CARLA to evaluate three approaches to autonomous driving. The first is a modular pipeline that relies on dedicated subsystems for visual perception, planning, and control **[6]**. This architecture is in line with most existing autonomous driving systems. The second approach is based on a deep network trained end-to-end via imitation learning **[7]**. This approach represents a long line of investigation that has recently attracted renewed interest. The third approach is based on a deep network trained end-to-end via reinforcement learning **[8]**. We begin by introducing notation that is common to all methods and then proceed to describe each in turn. Consider an agent that interacts with the environment over discrete time steps. At each time step, the agent gets an observation of and must produce an action at. The action is a three-dimensional vector that represents the steering, throttle, and brake. The observation is a tuple of sensory inputs. This can include high-dimensional sensory observations, such as color images and depth maps, and lower dimensional measurements, such as speed and GPS readings. In addition to momentary observations, all approaches also make use of a plan provided by a high-level topological planner. This planner takes the current position of the agent and the location of the goal as input and uses an algorithm to provide a high-level plan that the agent needs to follow to reach the goal. This plan advises the agent to turn left, turn right, or keep straight at intersections. The plan does not provide a trajectory and does not contain geometric information. It is thus a weaker form of the plan that is given by common GPS navigation applications which guide human drivers and autonomous vehicles in the physical world. We do not use metric maps.

Figure 4: Picture taken during CARLA Simulation in Town-1.

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 9 Issue 11, November-2020

Figure 5: CARLA Simulation in Town-2 - Modular pipeline.

As seen in Figure 6, First method used in present work is a modular pipeline that decomposes the driving task among the following subsystems: (i) perception, (ii) planning, and (iii) continuous control. Since no metric map is provided as input, visual perception becomes a critical task. Local planning is completely dependent on the scene layout estimated by the perception module. The perception stack uses semantic segmentation to estimate lanes, road limits, and dynamic objects and other hazards. In addition, a classification model is used to determine proximity to intersections. 4 The local planner uses a rule-based state machine that implements simple predefined polices tuned for urban environments. Continuous control is performed by a PID controller that actuates the steering, throttle, and brake. We now describe the modules in more detail.

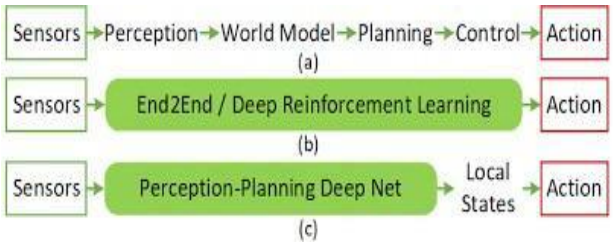

Figure 6: Working of Modular Pipeline

3.2.1 Perception

The perception stack we describe here is built upon a semantic segmentation network based on Refine Net. The network is trained to classify each pixel in the image into one of the following semantic categories: C = road, sidewalk, lane marking, dynamic object, miscellaneous static. The network is trained on 2,500 labelled images produced in the training environment using CARLA. The probability distributions provided by the network are used to estimate the ego-lane based on the road area and the lane markings. The network output is also used to compute an obstacle mask that aims to encompass pedestrians, vehicles, and other hazards. In addition, we estimate the likelihood of being at an intersection by using a binary scene classifier (intersection/no intersection) based on Alex Net. This network is trained on 500 images balanced between the two classes as shown in Fig.7.

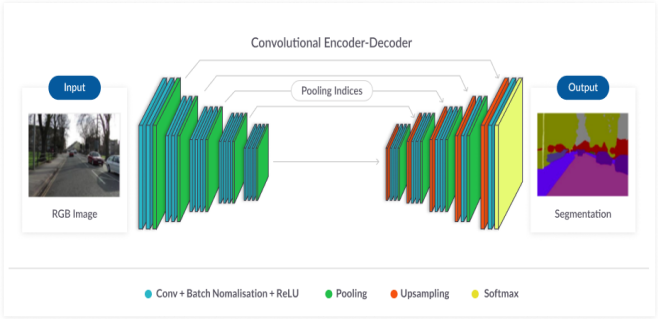

Figure 7: Picture describing how the network is trained

3.2.2 Local planner

The local planner coordinates low-level navigation by generating a set of waypoints: near-term goal states that represent the desired position and orientation of the car soon. The goal of the planner is to synthesize waypoints that keep the car on the road and prevent collisions. The local planner is based on a state machine with the following states: (i) road-following, (ii) left-turn, (iii) right-turn, (iv) intersection forward, and (v) hazard-stop. Transitions between states are performed based on estimates provided by the perception module and on topological information provided by the global planner. Fig 8 shows simulation of the local planner. The local plan in the form of waypoints is delivered to the controller, along with the vehicle's current pose and speed.

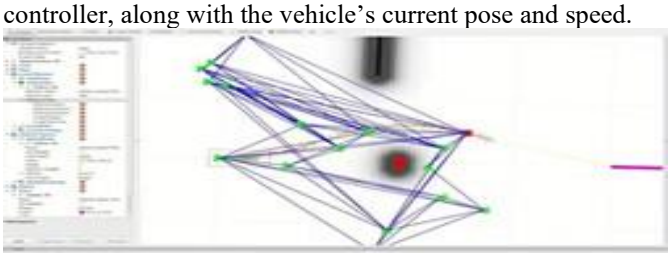

Figure 8: Picture represents simulation of the local planner.

3.2.3 Continuous Controller

Proportional-integral-derivative (PID) controller is used due to its simplicity, flexibility, and relative robustness to slow response times. Each controller receives the current pose, speed, and a list of waypoints, and actuates the steering, throttle, and brake, respectively. A cruise speed target is 20 km/h. Controller parameters were tuned in the training town.
A proportional-integral-derivate (PID) controller is a generic control loop feedback mechanism(controller) widely used in industrial control systems. A PID controller calculates an "error" value as the difference between a measured process variable and a desired set point. Fig 9 shows the working of PID controller.

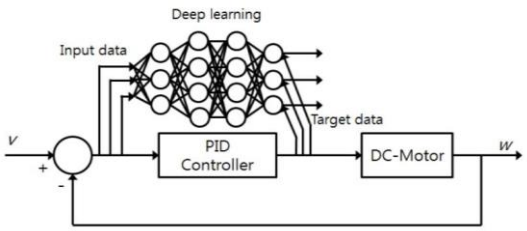

Figure 9: Picture shows working model of PID controller

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 9 Issue 11, November-2020

3.3 Reinforcement learning

Our third method is deep reinforcement learning, which trains a deep network based on a reward signal provided by the environment, with no human driving traces. We use the asynchronous advantage actor-critic (A3C) algorithm **[9]**. This algorithm has been shown to perform well in simulated three-dimensional environments on tasks such as racing and navigation in three-dimensional mazes. The asynchronous nature of the method enables running multiple simulation threads in parallel, which is important given the high sample complexity of deep reinforcement learning. We train A3C on goal-directed navigation. In each training episode the vehicle must reach a goal, guided by high-level commands from the topological planner. The episode is terminated when the vehicle reaches the goal, when the vehicle collides with an obstacle, or when a time budget is exhausted. The reward is a weighted sum of five terms: positively weighted speed and distance travelled towards the goal, and negatively weighted collision damage, overlap with the sidewalk, and overlap with the opposite lane. The network was trained with 10 parallel actor threads, for a total of 10 million simulation steps. We limit training to 10 million simulation steps because of computational costs imposed by the realistic simulation. This correspond to roughly 12 days of non-stop driving at 10 frames per second. This is considered limited training data by deep reinforcement learning standards, where it is common to train for hundreds of millions of steps, corresponding to months of subjective experience. To ensure that our setup is fair and that 10 million simulation steps are enough for learning to act in a complex environment, we trained a copy of our A3C agent to navigate in a three-dimensional maze (taskD2 from Dosovitskiy and Koltun **[10]**). The agent reached a score of 65 out of 100 after 10 million simulation steps – a good result compared to60 out of 100 reported by Dosovitskiy and Koltun after 50 million simulation steps for A3C with less optimized hyperparameters **[11]**.

.

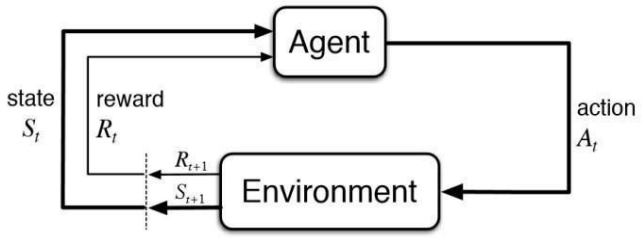

Figure 10: Picture shows reinforcement learning cycle in the present project

Reinforcement learning is the training of machine learning models to make a sequence of decisions. The agent learns to achieve a goal in an uncertain, potentially complex environment. In reinforcement learning, an artificial intelligence faces a game-like situation. The computer employs trial and error to come up with a solution to the problem. To get the machine to do what the programmer wants, the artificial intelligence gets either rewards or penalties for the actions it performs. Its goal is to maximize the total reward. Although the designer sets the reward policy–that is, the rules of the game–he gives the model no hints or suggestions for how to solve the game. It is up to the model to figure out how to perform the task to maximize the reward, starting from totally random trials and finishing with sophisticated tactics and superhuman skills. By leveraging the power of search and many trials, reinforcement learning is currently the most effective way to hint machine's creativity. In contrast to human beings, artificial intelligence can gather experience from thousands of parallel gameplays if a reinforcement learning algorithm **[12]** is run on a sufficiently powerful computer infrastructure.

4. RESULTS AND DISCUSSIONS

The Present work is completely implemented using CARLA Simulator where several simulations are created using scenario runner module in CARLA.

The project has been simulated and found working smoothly as shown in below YouTube link in Fig 13. The front vehicle is moving on its way to some destination, while the following vehicle (at back) is getting GPS location of the front vehicle and moving towards it by getting directions and instructions from Maps using Maps API. By testing the vehicle in simulation, it is also observed that even if the target vehicle takes the wrong route, the following vehicle will follow the right route because it is connected to Maps. As it is a prototype and conditions given to the vehicle are very small in number, hence it is very slow but if the system is implemented in real vehicles then this could help in solving the discussed issues in real time. Fig 11 and Fig 12 are the taken during simulation. One can watch these simulations from the link pasted below.

Figure 11: Two cars generated during simulation, the back car is following the front car

Figure 13: QR code to watch the videos referred

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 9 Issue 11, November-2020

Link for the videos to the above-mentioned simulations:
https://qrgo.page.link/3RBi8

5. CONCLUSION

This is an advanced step for autonomous driving vehicles. With the help of this algorithm, vehicles can be set to automatically navigate to the destination location by continuously receiving the direction from another vehicle moving ahead to the same destination. The vehicle routes itself with the guidance of another vehicle moving ahead to the same destination, therefore, deviations in time can occur. The goal of the navigation process for a vehicle is to move the robot to a known destination in an unknown environment. Navigation planning is one of the vital aspects of autonomous systems. When the robotic vehicle starts to move towards the planned route it may find unknown obstacles from the existing location to the destined location, hence the robotic vehicle must avoid the obstacles and follow an optimal route to reach the destined position. The potential applications of this robotic vehicle are to use these types of an autonomous vehicle on highways or heavy traffic roads. These types of autonomous vehicles can also be used when a driver travels to new areas. It is an improved navigation system for autonomous vehicles.

6. REFERENCES

[1] https://github.com/carla-simulator/carla
[2] Russell, S.J. &Norvig, P., 2016. Artificial intelligence: a modern approach, Malaysia; Pearson Education Limited,
[3] (2011) Bellman Equation. In: Sammut C., Webb G.I. (eds) Encyclopedia of Machine Learning. Springer, Boston, MA. https://doi.org/10.1007/978-0-387-30164-8_71
[4] Watkins, C.J.C.H., Dayan, P. *Q*-learning. *Mach Learn* **8,** 279–292 (1992). https://doi.org/10.1007/BF00992698
[5] Ian Goodfellow, YoshuaBengio, and Aaron Courville. 2016. Deep Learning. The MIT Press.
[6] T. Stolker1, M. J. Bonse1, S. P. QuanzPynPoint: a modular pipeline architecture for processing and analysis of high-contrast imaging data.
[7] Pan, Y. et al. (2020) 'Imitation learning for agile autonomous driving', The International Journal of Robotics Research, 39(2–3), pp. 286–302. doi: 10.1177/0278364919880273.
[8] Ahmad El Sallab, Mohammed Abdou1, Etienne Perot and Senthil YogamaniEnd-to-End Deep Reinforcement Learning for Lane Keeping Assist
[9] Volodymyr Mnih, AdriàPuigdomènechBadia, Mehdi Mirza, Asynchronous Methods for Deep Reinforcement Learning
[10] Alexey Dosovitskiy, German Ros, CARLA: An Open Urban Driving Simulator.
[11] Mark A. Haidekker. *The PID controller*, pages 253–273. 01 2018.
[12] TommiJaakkola, Satinder P. Singh, Michael I. Jordan. Reinforcement Learning Algorithm for Partially Observable Markov Decision Problems.