An Evaluation of Autonomous Car Simulators and their applicability for Supervised and Reinforcement Learning

Martin Holen[1], Kristian Knausgård[2], and Morten Goodwin[1]

[1] Centre for Artificial Intelligence Research, UiA, Norway
[2] Top Research Centre Mechatronics, UiA, Norway

Abstract. Recent advancements in the field of Machine Learning have sprouted a renewed interest in the area of autonomous cars. Companies use different techniques to develop autonomous cars, including buying several vehicles for the development of Advanced Driver Assistance Systems (ADAS), while others use car simulators. Simulators for autonomous cars include a wide variety of different sensors. Some simulators come free or even open-source, while others require to pay for the simulator itself, server time, or each sensor. The quality and focus of each of these vary, with some having LIDAR scanned roads for highly realistic roads, while others have entirely natural and realistic vehicle dynamics. The focus of this paper is to give an overview of available simulators for supervised & reinforcement learning, and their use cases.

Keywords: Autonomous cars · Machine Learning · Simulation.

1 Introduction

Autonomous vehicle is a field of research, in which the first attempts occurred in the 1920's [16, 2]. Though these were more simplistic implementations, more complex Neural Networks (NN) based implementations occurred in the late 1980s, ALVINN [12] in 1989, predicted the right action within 2 of its 45 units to the correct answer 90% of the time.

More recent papers include both Reinforcement Learning (RL) implementations [8, 17, 15] as well as Supervised learning (SL) implementations [18, 7]. When implementing on physical vehicles, the SL method uses data collected with a human driver, and RL methods face the issue of additional costs and safety associated with vehicle collisions during training. Data from human drivers, on the other hand, may not include unwanted situations during training such as the car veering off course or colliding in SL implementation. The existing disadvantages of using SL and RL on physical vehicle models for implementations lead us to simulators that support both methods of learning. For example, if there is a lack of data for a car about to collide or drive off the road, using a simulator you can collect this data safely. Looking at RL implementations specifically, simulations are a great tool, as the vehicle can crash as many times as required for the vehicle to train.

Though one of the issues in this field of research is which simulator to use, and which systems the researcher has to create. Some simulators may include a large variety of sensors and agents such as pedestrians[6]; other simulators focus on the vehicle dynamics, and realism[20, 13]. Though overall there is a lack of an overview for each simulators' use case which we will go into more detail about; with simulators focusing on SL and RL as the focus.

2 Simulators

Among the variety of simulators, we will look at the available ones. This includes a variety of simulators, while excluding simulators such as Nvidia drive sim, which is currently only available to those in their early access program[11].

We also exclude simulators such as summit which is an extension of Carla, the same goes for the Donkey simulator which is an extension to the self-driving car sandbox; Flow which focuses on traffic flow and does not have autonomy as its main goal; OpenAI's carracing-v0 which is a 2d top down simulator; TORCS as it does not include a simple method for interacting with the simulator; AirSim which was made as a drone simulator, which has been reworked to work for self-driving cars.

Each simulator has a set of sensors, some include the most common sensors widely used in research. The level of realism for each of these implementations varies as well. One example of this is the difference between rFpro's implementations versus that of the Udacity simulator. rFpro's sensors are highly realistic with companies selling sensor models to the user, this is in stark contrast to the idealized sensors of the Udacity simulator or its modified version.

There is also a difference in how the maps look and how realistic they are. Carla includes a variety of maps with flat or sloped roads; while the rFpro implementation includes LIDAR scans of the roads, which adds potholes and bumps on the road surface increasing the realism. Some of the simulators also allow for the creation of roads based on real-world maps[10], which may suit the researchers' need compared to the pay-per-map approach used in some simulators.

2.1 Carla

CARLA is an open-source simulator for autonomous driving research made in cooperation between Intel Labs, Toyota Research Institute, and the Computer Vision Center in Barcelona[6]. It is made using the Unreal Game engine and includes a variety of actors as well as sensors when training an autonomous car. The simulator includes eight different urban maps, as well as the ability to simulate different weather conditions. This variation helps test the validity of an algorithm by changing the map or weather and seeing that the algorithm still performs the same.

The complexity can also be varied quite a lot, by controlling one or more vehicles at a time while having other actors in the environment. These other actors include traffic lights, traffic signs, vehicles, and pedestrians. The traffic

lights will change according to how many vehicles are driving from each direction, while pedestrians walk around and there are traffic signs giving information to the researcher-controlled autonomous car in the environment.

In the original paper, there was a RL implementation of carla, which can be found in their github. Though it excludes the ability to train and get feedback from the environment[3].

Actors The actors include a few sensors, traffic signs and lights, vehicles, and walkers. The sensors are placeable objects which the researcher can move; to simulate a security camera, or simply gather specific data such as segmented images. These sensors can be placed on an actor controlled by either CARLA or the researcher.

There are also spectators, which is an actor placed to give an in-game point of view over a specified location and rotation.

Then there are the traffic signs and traffic lights, the traffic signs currently include stop and yield traffic signs, while the traffic lights are an actor which the vehicle is only aware of if the vehicle is by a red light. Both are spawned using the OpenDRIVE file, and cannot be placed when the simulation has started, and the light can be changed using the API.

Unlike most other simulators CARLA includes vehicles and pedestrians. This allows the autonomous car to act in a vehicle & pedestrian-free environment until it becomes proficient at driving. The autonomous car is then tested using multiple pedestrians and vehicles. These actors can be controlled by either the simulator itself or through an algorithm programmed by the researcher.

Sensors Carla includes a variety of sensors, which are categorized into 3 major groups[4]; namely Cameras, Detectors, and Other. For the cameras, we have a few different types, meant to handle different tasks. There is a depth-sensing camera, an RGB camera, a semantic segmentation camera as well as a DVS camera. The depth-sensing camera senses the distance to each pixel on the screen and sends it out of its object. While the RGB camera takes an image in RGB colors in its current location and rotation. The semantic segmentation camera works similarly to an RGB camera, but instead changes the color of each object it sees, so a car is one color while trees are another color. Then there is the DVS camera, which finds the change in brightness between frames and outputs those. This information can be used with an RGB camera to drive on the road and find objects which are moving, which the car may want to avoid.

We also have detectors, these detect when an event happens. The possible events are collision, lane invasion, and obstacles on the road. The collision detector detects when a vehicle crashes, which can be very useful in RL algorithms and such. Then there are the lane invasion detectors which can sense when the autonomous vehicle is changing lanes so that if the lane change is expected it can

[3] `https://github.com/carla-simulator/reinforcement-learning`

be ignored otherwise an emergency stop can occur. Lastly, we have the obstacle detector, which detects if there are any possible obstacles ahead of the parent.

We also have a variety of other sensors; like GNSS, IMU, LIDAR, Radar, RSS, Semantic LIDAR. The GNSS is a Global Navigation Satellite System, which gives the altitude latitude and longitude of the object it is placed at. An IMU or Inertial Measurement Unit, measures the linear acceleration, orientation, and includes a gyroscope that gives the angular velocity. The RADAR is a simulated radar that sends out a wave and gets back the range, angle, and velocity of the object it hits. LIDAR works quite similar to the RADAR but sends a ray outwards, sending multiple rays allows for measuring the distance to multiple objects as well as their shape. A responsibility-sensitive safety (RSS) sensor is a sensor that modifies the control safely applied to the vehicle. While the semantic LIDAR sensor which is a rotating lidar generates a 3D point cloud but with information of the object hit by the rays.

Focus Carla is a very open simulator with a large variety of sensors, with the focus of this simulator seemingly being on creating SL based algorithms.

2.2 DYNA4

Dyna4 is a software made by Vector, their "...physical models include vehicle dynamics, power train, combustion engine, electrical motors, sensors, and traffic"[20]. These systems can be used for tasks such as creating ADAS systems, which are relevant in the development of an autonomous vehicle. To aid in the creation of an ADAS system, Dyna4 has a variety of included sensors and can use traffic simulators such as SUMO.

DYNA4 bases its physics engine on Matlab and Simulink while utilizing the graphics engine from Unity. The physical simulation of mechanics, electronics, thermodynamics, control logics, and traffic is done with MatLab and Simulink. While Unity is used for more detailed sensor simulation, Simulink includes idealized versions of the sensors. DYNA4 is split into two parts, being able to only use the MatLab/Simulink parts, with multiple OSs, while the Unity part runs on Windows only. Changing the simulation to add features is done in Simulink, and modification is easily done as DYNA4 gives access to the model states and signals.

When looking at what makes DYNA4 different, is their ability to scale the level of detail of the simulation is their main selling point. For component models there are different levels of detail, for vehicle chassis models we have a mass point, extended bicycle model, and a full vehicle dynamics model, these can be combined with other components such as a drivetrain. Choosing a simpler model indicates that there is less effort required for parameterization and computational power. When looking at the level of detail for their drivetrain architecture, we see that it starts with being able to control the torque of the wheels, up to a full simulation of the combustion engine and powertrain. They also use a variety of open standards such as ASAMs OpenDRIVE, OSI, XiL API, and MDF. The

environment can also be auto-generated from OpenDRIVE files, including the road, traffic signs, lane markers, roadside objects. Their traffic model is also able to control traffic objects, these objects can be controlled with DNN or stochastic traffic simulation via SUMO. DYNA4 also includes a tire model, TMeasy, which adapts to different friction conditions. The user can set the friction coefficient of the road or the tires if it is wanted. They are also open to use any tire model, such as MFTyre or FTire and more.

As Dyna4 is meant to be used by the industry there is a cost to purchasing a license. The monetary cost of the simulation is very low for educational licenses, with industrial licensing being significantly more expensive. Educational licenses are on a per computer basis, meaning that if multiple users are developing the software at once, they need a license each.

DYNA4 also uses a continuous action space, which allows for fine control of the vehicle. While its state space can be set to continuous or discrete state space, depending on the task.

Given DYNA4's set of sensors; makes them a great tool for developing and prototyping new ADAS systems. When combining these sensors with the vehicle dynamics package vehicle control can be achieved.

Sensors DYNA4 includes a range of available sensors like RADAR, LIDAR, Ultra-sonic, and Camera sensors. With these sensors, the vehicle can sense its surroundings, as well as the distance and speed of an object. The detail among the sensors is also varying, with both idealized as well as more detailed versions of sensors. Given their sensors, they can get both bounding boxes and features such as detected lanes as an output from their camera and radar. Their camera is not a simplistic model, taking into account the dirt and lens distortion.

Focus DYNA4's focus seems to be on the creation of SL algorithms.

2.3 rFpro

rFpro is a simulator made with realistic physics and high visual realism. rFpro's high visual fidelity is done through ray-tracing, when using it or one of the purchased sensor models at their most realistic setting of the simulation resolution, the simulation can easily become slower than real-time. This means that there is a balance between realism and computational time [13]. The realistic physics includes a highly realistic road surface with potholes, the suspension is also simulated so that the bumps are realistic. These realistic road surfaces, which are created using extremely detailed LIDAR scans, make the simulation currently very costly. The maps also have to be purchased at a significant price, with at least one of the maps costing more than the base simulator itself [13]. As their maps are LIDAR survey scans of real roads, the validity of an algorithm can be tested on the real road while being created and trained on the simulated road.

The OS support for rFpro is currently only in Windows; though it aims for a Linux distribution sometime in the future. In regards to their API, it supports C++ for interfacing with it.

Sensors There are a variety of sensors available for rFpro, though most of them have to be purchased at an extra cost. Purchasing the base simulator the customer gets a camera sensor included, while the rest of the sensor models have to be created or purchased from other vendors. The purchasable sensor models are simulated to high degree of accuracy. The user can create the sensor model themselves, to be as realistic as possible, simulating the sensor down to the smallest detail including error models. One example of their realistic simulation is a LIDAR which is simulated down to its UDP packets sent between the sensor and the computer.

Focus rFpro focuses on physical modeling of the vehicle, as well as very high degree of accuracy for their sensors. And so their aim seems to be on sensor fusion as well SL based algorithms which can be moved directly from simulation to real-world.

2.4 Deepdrive

Deepdrive is an open-source autonomous car simulator, with an MIT license. It has some sensor data which can be used for the creation of path following using SL or RL-based autonomous cars. They have compatibility with Gym, docker, and more[5].

Compatibility The Deep drive simulator works with both gym and docker. Their docker implementation uses a docker container for the environment, as well as a docker container per Deepdrive client. This allows for simple setups of clients. Their compatibility with gym, also makes them good for testing the validity of changes to their environment, such as changing the reward function for the system as any person making such changes can run a test with and without those changes with state of the art (SOA) methods such as Rainbow, PPO, DDPG and more[5].

Sensors Deepdrive has a few sensors, namely an RGB camera, a GPS, an IMU, and some location sensors for staying in the center of the lane, and following a path[5].

Focus Their focus seems to be on both SL, and RL, for performing tasks such as lane changing, centering the vehicle in the lane, following a path, and generalized autonomous vehicle development[5].

2.5 PGDrive

PGDrive is an open-source car simulator using RL. It has compatibility with gym, making it easy to test with the maps being procedurally generated with a few premade maps[9].

Compatibility PGDrive is compatible with gym, which allows for testing the SOA methods such as PPO and more. Improving the reproducibility of results.

Sensors In regards to sensors, PGDrive includes LIDAR, RGB Camera, Depth camera, birds-eye view. The sensors are adjustable with the LIDAR, RGB, and Depth Camera having "a field of view, size of film, focal length, aspect ratio, and the number of lasers"[9].

Focus Reinforcement Learning, with procedural generation of the maps and interaction with gym environments.

2.6 Duckie Town Simulator

Unlike most other simulators, the Duckie Town simulator is a python and OpenGL-based simulator, made originally for lane keeping, but now with realistic camera distortion. Their focus is on Supervised learning with imitation learning as the currently available version of Reinforcement learning.

Sensors It has a single sensor; an RGB camera of shape 640x480.

Focus Duckie Town simulators focus is on SL algorithms, and lane detection more specifically.

2.7 SVL Simulator

SVL simulator is a Unity-based open simulator by LG Electronics. It allows for digital twins, making testing between the simulator and the real world simple. The simulator allows the researcher to run hardware in the loop by having two machines, one controlling the vehicle using chassis commands, and the other one hosting the environment.

Sensors SVL includes a few sensors namely LIDAR, RADAR, Camera, IMU, GPS, and Segmentation sensors [14].

Focus This simulator focuses on creating an environment and a set of sensors, allowing for the training of algorithms for tasks around perception, planning, and control of the autonomous vehicle[14].

2.8 MATLAB Automated Driving Toolbox

MATLAB Automated Driving Toolbox is a toolkit that helps in the design, development, and testing of ADAS systems.

With the simulator being able to control the vehicle through checkpoints, and using this to collect data. The simulator shows a cuboid simulation to the user when running the simulator, though the data collected from the environment is highly realistic.

Sensors The simulator comes with a couple of different sensors, the camera, and Lidar. These sensors are made in a way where the labeling of their data is made simple, through a built-in toolkit. With this toolkit, the researcher can create bounding boxes and label them directly in the software. This makes the data collection process easier than manually gather the images.

There is also the implementation of real sensors, such as a Velodyne LIDAR[10].

Maps The MATLAB software does not include a set of maps, instead having a toolkit that allows for the creation of a map using HERE HD Live Map, where the researcher marks a region of road, and then gets the corresponding map to this road. This allows for easy creation of different testing scenarios, though may lack some detail compared to other maps[10].

Focus MATLAB Automated Drive Toolbox is a set of tools, focusing on the creation of a simulator for SL.

2.9 Udacity simulator

Udacity is an educational organization, making courses to teach specific subjects [19]. For one of their courses, Udacity made an open simulator for an autonomous vehicle made for supervised learning algorithms. This simulator was created in Unity with two different rural maps, and a singular vehicle [4]. A modified version of the simulator support was added for RL-based autonomous vehicles. The modification is done using C#, writing the scripts and attaching them to the corresponding objects [5].

Sensors For the modified version both the RGB camera as well as the segmentation cameras are available. The RGB camera is a simple camera attached to the vehicle, collecting photos as the vehicle drives, the segmented camera, on the other hand, is quite different, as it segments objects in the world through shaders. Lastly, there is an ideal GNSS or GPS if you wish, with which the position and rotation of the vehicle are given. These sensors are ideal sensors, with the segmented cameras giving pixel-perfect data.

Reinforcement Learning based systems When training RL-based algorithms in simulations, certain things are very important to standardize. This includes a standardized method of giving rewards and penalties, which the Udacity simulator has. Another important system is the one that gives feedback for how far the RL-based algorithm has driven, achieved through checkpoints. For the modified Udacity simulation checkpoints are based on the track, some are

[4] `https://github.com/udacity/self-driving-car-sim`
[5] `https://github.com/marho13/udacityReinforcement`

straight and give a smaller reward, while other tracks require the vehicle to steer, granting a higher reward. To give a feedback for being off the road, a small penalty is given if the vehicle has any of its tires off the road. These systems are aimed to make the feedback standardized, making recreating results easier.

Other than the feedback, resetting the vehicle is another very important tool.

The reset is done by checking if the vehicle moves over a few hundred steps or if it is consistently off the road, if either is true the vehicle is reset and the AI algorithm is informed. These resets aim to improve the training speed of the RL algorithm steering the vehicle.

Focus The Udacity simulator focuses on SL, while the modified version focuses on RL.

2.10 AWS DeepRacer

AWS DeepRacer is a 3D racing simulator, for training RL-based autonomous cars. The simulator focuses on performing Deep RL racing and moving these algorithms from simulation to the real-world[3]. As the simulator is very similar to that of their physical DeepRacer robot; it becomes significantly easier to move from a simulation to the real world. This allows for training in the simulator to transfer learned quite easily in the real world which is a significant advantage. The training also occurs on the AWS servers; where Amazon has a lot of available codes which aim at helping people understand not only how to run on their systems, but also how to perform RL.

The goal of the simulator map is to get around the track as fast as possible, while creating waypoints, to estimate how far the vehicle has come. After the vehicle is trained on the simulator, you can move to the real world where you can mark the road in a similar way to create a race track using tape. There is also a competition online in which each person can test the validity of their autonomous car[1].

The only sensors currently in use are either one or two 4MP cameras, which they downsize to 160x160 greyscale for performance reasons.

Reinforcement Learning The DeepRacer Autonomous cars get feedback based on how close to the center of the track the car is. With the center of the track being a reward of 1, being off-center giving a reward of 0.5, and driving off-road giving a reward of 0. Rewarding the vehicle for driving in the middle of the track allows the car to continuously[3].

Focus AWS DeepRacer focuses on the creation of an RL algorithm which can be transferred from their simulator to their robot.

3 Discussions

Discussing the use case of each simulator, we group them into a few different types with the main categories being Supervised learning and Reinforcement learning.

Based on our understanding we have categorized the different simulators according to their focus and compatibility Table 1.

Table 1. An overview of the simulators, their focus and their compatibility with SL and RL. *The training code is not published yet, only the validation without rewards; **There is a small number of sensors, and so it is more suited for RL.

Simulator	RL & SL	Focus
rFpro	SL	Realisic sensors and physics
Dyna4	SL	Realistic sensors and physics
Carla	Both*	Many sensors, maps, agents & some RL
AWS DeepRacer	RL	Teaching how to perform simple RL & easily transfer it to robot
MATLAB	SL	Tools for the creation of a simulator fitting the researchers needs
Udacity	Both**	Supervised Learning and Reinforcement learning
SVL	SL	Creating env & sensors for developing perception, planning and control algorithms
Donkey	SL	Racing algorithms
Duckie Town	SL	Developing lane detection algorithms
PGDrive	RL	Generate maps procedurally, standardized testing using GYM
DeepDrive	RL & SL	Developing Lane changing, path following algorithms

3.1 Supervised learning

When looking at supervised learning there a couple of main focus points for each simulation software. The first focus is to prototype a system and move into the real world later with less focus on the realism of the sensors. The next being on creating a system that can be directly translated from the simulator to a physical vehicle.

Prototyping ADAS systems When prototyping ADAS systems, the main focus is on testing the efficacy of algorithms, with less focus on realistic sensors and interactions with the sensors. This leads us to simulators with idealized sensors, and preferably a large amount of them. Therefore the simulators best aimed at such a task are that of Carla, Donkey, Duckie town, and MATLAB, which have some complexity, though use idealized systems. And though it is possible to increase the realism in these sensors by coding in extra functionality it is not a default sensor. Carla has a larger set of features, while not supporting MATLAB or gym environments. MATLABs' implementation is more of a toolbox for creating the simulator one might want. The Donkey simulator focuses on racing for Supervised Learning algorithms using Gym. While Duckie town is a more simple implementation, which runs pure python.

Realistic ADAS Few other implementations focus mainly on being able to move from a simulated vehicle to a real vehicle with ease. These systems may not have every sensor available at the start, and some use-cases are more difficult to train them in such as Reinforcement Learning, but their realism is significantly higher than other simulators. The three main simulators whose focus is on realism are that of rFpro and Dyna4. rFpro is more expensive than Dyna4 and SVL, though the realism in their sensor packages is higher. SVL has some features such as a digital clone which DYNA4 does not have, though DYNA4 has more vehicle dynamics in it than either of the other simulators.

3.2 Reinforcement learning

This is one of the newer implementations on autonomous vehicles, which attempts to utilize the increased performance seen in video games such as the Atari games and multiple new highly complex game titles. In this case, having a standardized system that gives the same feedback as any other system is an advantage, and these may aim at different implementations, one being on getting from point A to point B as fast as possible. Another focus may be on the systems for giving feedback to the RL algorithm.

Racing Focusing on driving as fast as possible on the road, is an intriguing topic, as it shows how good the vehicle is at steering and how its reaction time is. At which point does the algorithm see an obstacle and steer away from it, and is the vehicle able to not lose the grip of its tires. Here the simulator which fits best, are is AWS DeepRacer.

Generalized Autonomous Tasks As RL-based autonomous cars are a new field of study, the systems to test and create them are not in place yet. There are therefore simulators that aim at creating these standardized systems and standardized testing. For this task, we consider PGDrive, Deepdrive & Udacity. With the PGDrive and DeepDrive having gym support, which allows for standardized testing using State of the art Algorithms, such as the stable baseline algorithms from OpenAI. Deepdrive has some focus on the feedback which can tell the distance to the center of the lane in cm. The Udacity simulator also focuses on this feedback, with more of the focus being on the ease of implementation, at the cost of the standardized testing algorithms which are easily available to those with gym for the environment interaction.

4 Conclusion

Simulators are an easy way to test algorithms for autonomous vehicles, and within the field of autonomous cars, a few simulators exist with more coming in the future. Some simulators aim to give the most realistic interaction with the vehicle and the algorithms, where the sensors act similar to those in the physical

world. There are also ones that allow for prototyping and testing of the most common ADAS algorithms, focusing on more sensors, though not aiming to reach the same level of accuracy as physical sensors. Looking into the surging field of reinforcement learning-based autonomous cars, there are racers and prototyping simulators. With the racers giving feedback based solely on how quickly the car drives, while the prototyping simulator allows for the creation and testing of new feedbacks to the algorithm controlling the vehicle.

5 Acknowledgments

We would like to thank DYNA4 and rFpro for their time, and insights into their simulators. We would also like to thank Gabrielle Katsue Klein and Gulshan Noorsumar for their feedback on the paper.

References

1. Amazon: Developers, start your engines, https://aws.amazon.com/deepracer/#Getting_started_with_AWS_DeepRacer
2. Anschuetz, R.: A pioneer in navigation technology, https://www.raytheon-anschuetz.com/company/history/
3. Balaji, B., Mallya, S., Genc, S., Gupta, S., Dirac, L., Khare, V., Roy, G., Sun, T., Tao, Y., Townsend, B., et al.: Deepracer: Autonomous racing platform for experimentation with sim2real reinforcement learning. In: 2020 IEEE International Conference on Robotics and Automation (ICRA). pp. 2746–2754. IEEE (2020)
4. Community, C.: 4th. sensors and data, https://carla.readthedocs.io/en/latest/core_sensors/
5. Community, D.: Deepdrive, https://github.com/deepdrive/deepdrive
6. Dosovitskiy, A., Ros, G., Codevilla, F., Lopez, A., Koltun, V.: Carla: An open urban driving simulator. In: Conference on robot learning. pp. 1–16. PMLR (2017)
7. Karni, U., Ramachandran, S.S., Sivaraman, K., Veeraraghavan, A.: Development of autonomous downscaled model car using neural networks and machine learning. In: 2019 3rd International Conference on Computing Methodologies and Communication (ICCMC). pp. 1089–1094. IEEE (2019)
8. Kendall, A., Hawke, J., Janz, D., Mazur, P., Reda, D., Allen, J.M., Lam, V.D., Bewley, A., Shah, A.: Learning to drive in a day. In: 2019 International Conference on Robotics and Automation (ICRA). pp. 8248–8254. IEEE (2019)
9. Li, Q., Peng, Z., Zhang, Q., Qiu, C., Liu, C., Zhou, B.: Improving the generalization of end-to-end driving through procedural generation. arXiv preprint arXiv:2012.13681 (2020)
10. Mathworks: Automated driving toolbox, https://se.mathworks.com/products/automated-driving.html
11. Nvidia: Nvidia drive sim - powered by omniverse, https://developer.nvidia.com/drive/drive-sim
12. Pomerleau, D.A.: Alvinn: An autonomous land vehicle in a neural network. In: Touretzky, D. (ed.) Advances in Neural Information Processing Systems. vol. 1. Morgan-Kaufmann (1989), https://proceedings.neurips.cc/paper/1988/file/812b4ba287f5ee0bc9d43bbf5bbe87fb-Paper.pdf

13. rFpro: About rfpro, https://www.rfpro.com/about/
14. Rong, G., Shin, B.H., Tabatabaee, H., Lu, Q., Lemke, S., Možeiko, M., Boise, E., Uhm, G., Gerow, M., Mehta, S., et al.: Lgsvl simulator: A high fidelity simulator for autonomous driving. In: 2020 IEEE 23rd International Conference on Intelligent Transportation Systems (ITSC). pp. 1–6. IEEE (2020)
15. Sadigh, D., Sastry, S., Seshia, S.A., Dragan, A.D.: Planning for autonomous cars that leverage effects on human actions. In: Robotics: Science and Systems. vol. 2. Ann Arbor, MI, USA (2016)
16. Sentinel, T.M.: "phantom auto" will tour city (December 1926)
17. Shalev-Shwartz, S., Shammah, S., Shashua, A.: Safe, multi-agent, reinforcement learning for autonomous driving. arXiv preprint arXiv:1610.03295 (2016)
18. Stavens, D.M.: Learning to drive: Perception for autonomous cars. Stanford University (2011)
19. Udacity: About us, https://www.udacity.com/us
20. Vector: Dyna4 - function development in closed-loop system tests, https://www.vector.com/no/en/products/products-a-z/software/dyna4/