VIRTUAL BASED AUTONOMOUS VEHICLE

A PROJECT REPORT

(MINI PROJECT)

Submitted by

HARIHARAN P (18A112)

LITTHISH S (19A511)

SAM CLASTINE JESUMUTHU (19A515)

THIRUMURUGAN S (19A519)

in partial fulfillment for the award of the degree of

BACHELOR OF ENGINEERING

IN

AUTOMOBILE ENGINEERING

ANNA UNIVERSITY, CHENNAI
NOVEMBER 2020

DEPARTMENT OF AUTOMOBILE ENGINEERING

Project Approval Sheet

This is to certify that the project report titled

"VIRTUAL BASED AUTONOMOUS VEHICLE"

Bonafide record of work done by

HARIHARAN P (18A112)

LITTHISH S (19A511)

SAM CLASTINE JESUMUTHU (19A515)

THIRUMURUGAN S (19A519)

Report submitted in partial fulfillment for the

BACHELOR OF ENGINEERING IN AUTOMOBILE ENGINEERING

During the year 2020 – 2021

Mr.K. VEERAMANIKANDAN M.E.	Dr.J. GANESH MURALI M.E. Ph.D.
(FACULTY GUIDE)	(HEAD OF THE DEPARTMENT)

Certified that the candidate was examined in the viva-voce examination held on …….

INTERNAL EXAMINER CO-ORDINATOR

ACKNOWLEDGEMENT

We express our sincere thanks to Karpagam educational and charitable trust for providing necessary facilities to bring out the project successfully. We felt greatness to record our thanks to the chairman **Dr.R.VASANTHAKUMAR B.E.,(Hons),D.Sc.** for all his support and ray of strengthening hope extended.

It is the moment of immense pride for us to reveal our profound thanks to our respected principal, **Dr.P.VIJAYAKUMAR M.E.,Ph.D.** who happens to be striving force in all our endeavors.

We express our sincere thanks to our **Dr.J.GANESH MURALI M.E., Ph.D.** Head of the Department of Automobile Engineering for providing an opportunity to work on this project. His valuable suggestions helped us a lot to do in this project

A word of thanks would not be sufficient for the work of our project guide **Mr.K.VEERAMANIKANDAN M.E.,** Department of Automobile Engineering whose efforts and inspiration lead us through every trying circumstance.

We would like to recollect the courage and enthusiasm that was inculcated in us by our project coordinators, **Mr.V.KONDUSAMY M.TECH** Department of Automobile Engineering for valuable guidance and support through the tenure of our project.

We deeply express our gratitude to all the faculty members of the Department of Automobile Engineering for encouragement, which we received throughout the semester and we are indebted to our colleagues and friends who helped us along the way

CONTENTS

LIST OF FIGURES

1. ABSTRACT

Over the last years the automotive industry has advanced significantly towards a future without human drivers. Researchers are currently trying to overcome the technological, political and social challenges involved in making autonomous vehicles mainstream. These vehicles need to be safe, reliable and cost-efficient.

This project proposes a virtual self-driving car and a testing environment where it will be evaluated. The virtual environment fabricates the domain such like that it is the mimics of the activity of a genuine car. Distinctive driving situations were utilized to analyze how the sensors respond when they are connected to genuine circumstances and are also utilized to confirm the impacts of other parameters on the scenes. Options are available to choose flexible sensors, monitor the output and implement any autonomous driving, steering prediction, deep learning and end-to-end learning algorithm.

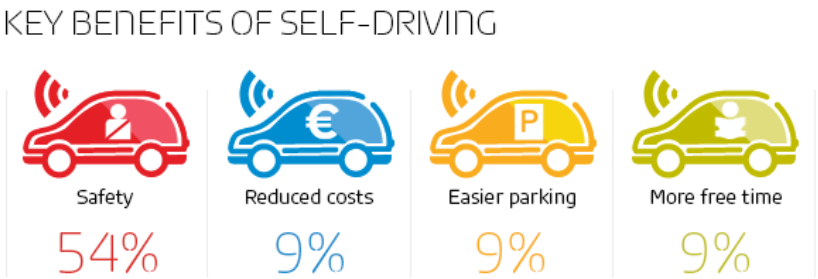
2. INTRODUCTION

2.1 Overview

A self-driving car, also known as an autonomous car or driverless car, is a vehicle that uses a combination of sensors, cameras, radars, and artificial intelligence (AI), to travel between destinations without the need of any human effort. In the past five years, autonomous driving has gone from "maybe possible" to "now commercially available" and hence become a concrete reality. Autonomous driving may pave the way for future systems where computers take over the art of driving. The global self-driving car market is segmented based on the type of vehicle, product type, application of the car, technology components (hardware and software), and geography. Automotive Innovators like Waymo and Tesla have been leading the self-driving car industry for long. On the other hand, legacy companies like GM, Ford, Toyota, Nissan, etc. have more recently joined the chase pumping billions of dollars into the research and development of autonomous vehicles.

2.2 Need for self-driving cars

Traffic fatalities claim more than a million lives a year around the world. This public safety crisis that is mainly caused by driver error has largely been overlooked and considered a necessary trade-off in our commuting economy. But with the development of advanced sensor technologies (e.g. cameras, radars, Lidars, etc.), and perception systems guided by big data, artificial intelligence, and increasing processing power, we are fast approaching the day when self-driving vehicles can do a better job than human drivers . Having self-driving cars could also give people a lot more free time. Commutes might be spent working on projects, talking to other passengers and doing many other productive tasks. Fully autonomous cars will be able to drive together in perfect.

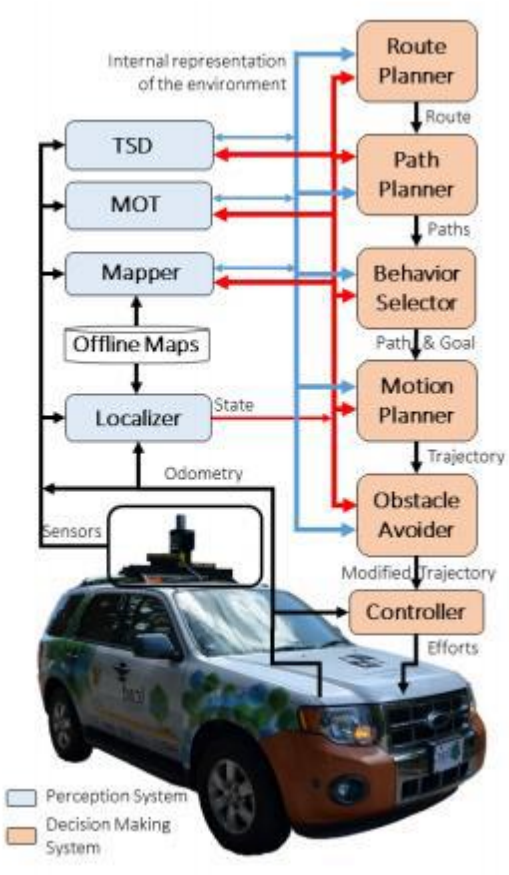


Fig(1):Benefits of self-driving vehicles

harmony like a swarm of bees, radically reducing the traffic jams and increasing the traffic flow. It is also projected that the advent of autonomous vehicles will cut down the insurance premiums, accident-related costs, driving-related fines and, increase fuel efficiency.

2.3 Architecture of Self-Driving Cars

The architecture of self-driving vehicles comprises of four main subsystems: Sensors, Perception, Planning and Control. These subsystems act together to perceive the environment around the autonomous vehicle, detect the paths, plan a route to the destination, predict the behavior of other traffic actors surrounding it, plan trajectories and finally execute the motion.



Fig(2):Self-Driving car Architecture

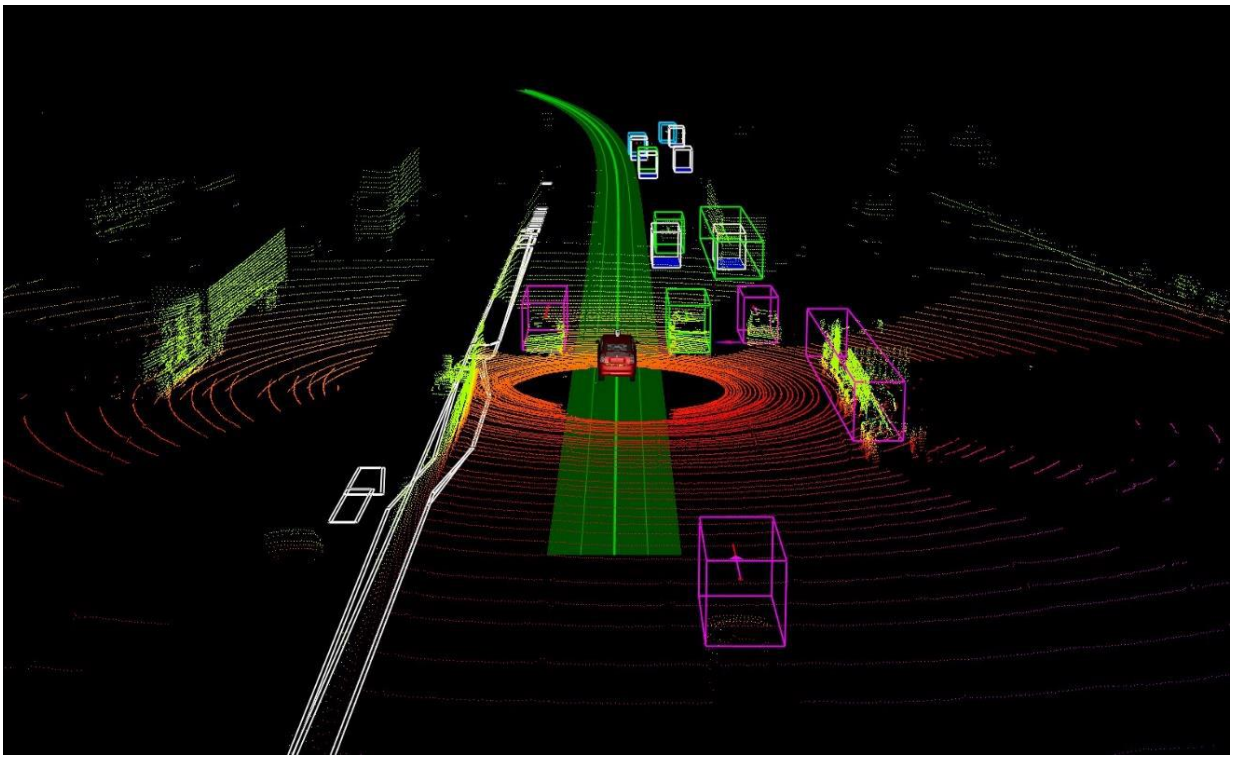
2.3.1 Sensors

The sensor subsystem consists of several sensors that gather data about the surroundings of an autonomous vehicle. Some of the most common sensors employed for this purpose are as follows:

Camera: Camera(image sensor) is certainly the most important sensor in an AV. Typically AVs have multiple cameras involved to provide a 360-degree view of the surrounding environment. Cameras have high resolution, are cheap, can collect a lot of data, therefore are useful for Deep Learning.

Radar: Radars are again very common automotive sensors for object tracking and detection. Radars are cheap, do well in poor weather as well, but have low resolution.

Lidar: Lidars are a bit expensive, but one of the most efficient sensors of an AV. Continuously rotating Lidar system sends thousands of laser pulses every second. These pulses collide with the surrounding objects and reflect, creating a 3D point cloud formation.



Fig(3):Object Detection using Lidar point clouds

GPS: GPS sensors are the common positioning sensors that give latitude and longitude information. Others: Apart from these, there are several other sensors like ultrasonic Sonars, IMUs, gyroscopes, etc. used in AVs.

2.3.2 Perception

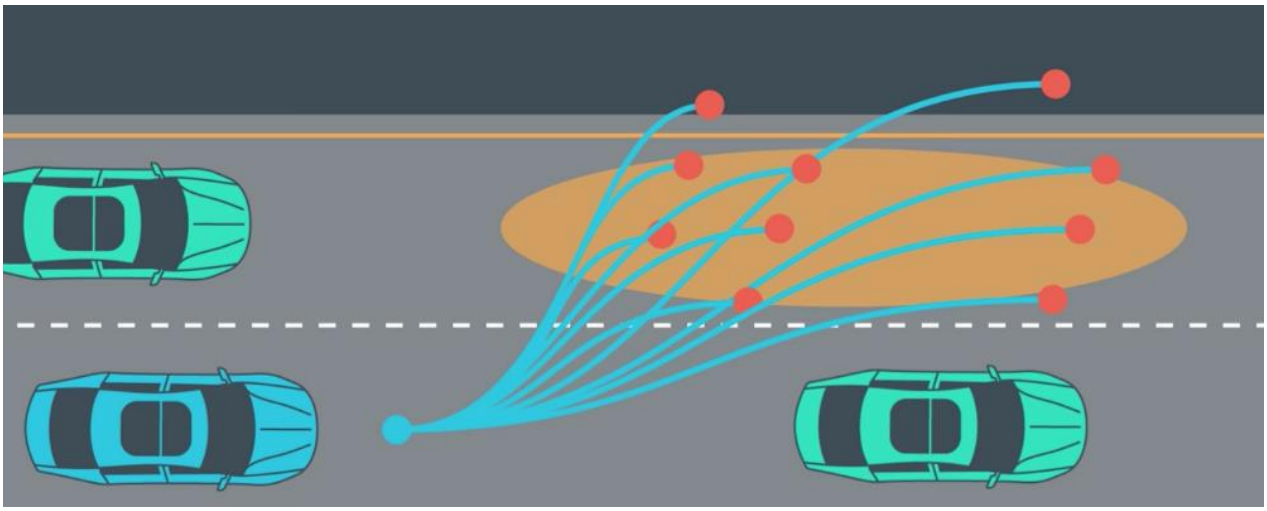
Perception in AVs is responsible for estimating the state of the car and for creating an internal representation of the environment, using data captured by on-board sensors as well as the prior information about the sensors' models, road networks, traffic rules, car dynamics, etc. As described in perception can be categorized into two components, as follows:

Localization: As the name suggests, it involves localizing an AV within the driving scenario. This system uses the data from GPS and other sensors to estimate the AV's pose relative to the driving scenario. Usually, this is one of the initial steps of the autonomous driving process.

Detection: This system gathers and processes the data from on-board sensors like camera, Lidar, radar, etc. to detect the static, variable, and dynamic objects in the driving scenario. However, the data from different on-board sensors need to be synchronized and processed accordingly to extract necessary information regarding the driving scenario. Sensor fusion is an approach for combining data delivered from disparate sources such that the coherent information is created. The resulting information is more certain than it would be possible when these sources were used individually. For example, on the AV, it is important to have a camera in order to clone a human vision, but the information about obstacle distance will be best gained through the sensors like radar or Lidar. For that reason, sensor fusion of camera with Lidar or radar data is very important since they are complementary.

2.3.3 Path Planning

The Path Planning functionality takes information from the perception system and uses it for long and short-range planning. There are several components of the Path Planning functionality such as the Route planner that plans the path that a vehicle should take between the given two points on a 6 map, the Prediction component that predicts behavior of other traffic actors, the Behavior planner that plans the behavior of the AV itself such as keeping the existing lane, change lanes, apply brakes or accelerate as needed, and finally the Trajectory planner that decides the ultimate trajectory that the AV must follow.



Fig(4):Path Planning in AVs

2.3.4 Control

The Controller module receives the trajectory generated by the Path Planner and sends effort commands to the actuators of the steering wheel, throttle, and brakes of the AV to make the car execute the trajectory as best as the physical world allows. Several controllers are used in AVs, depending on the problem to be solved. The PID (Proportional Integral Derivative) and MPC (Model Predictive Control) are two of the most commonly used controllers.

2.4 Testing and Validation of Autonomous Vehicles: A Challenge

As described in recent breakthroughs in deep learning have accelerated the development of autonomous vehicles: many research prototypes now operate on real roads alongside human drivers. While advances in computer vision techniques have made human-level performance possible on narrow perception tasks such as object recognition, several fatal accidents involving AVs underscore the importance of testing whether the perception and control pipeline – when considered a whole system, can safely interact with humans. Unfortunately, testing the AVs in real environments, the most straight forward validation framework for system-level input-output behavior requires prohibitive amounts of time due to the rare nature of serious accidents. Concretely, a recent study argues that AVs need to drive "hundreds of millions of miles and, under some scenarios, hundreds of billions of miles to create enough data to clearly demonstrate their safety". As a result, AV developers test extensively on public roads, potentially putting other road users at risk. In one such unfortunate event, Elaine Herzberg, 49, was killed by an Uber test vehicle on 18th March 2018 in Tempe, Arizona .Such incidents can infuse a sense of

insecurity in the minds of common road users making the social acceptance of AVs a complex process. Aside from safety concerns, costs pose an additional challenge to the testing and validation of AVs. Each new configuration of the AV requires re-calibration of a physical vehicle, which is labor-intensive. Furthermore, the vehicle can only be tested under conditions limited by either a testing track or current traffic conditions if a public road test is being performed. Apart from that, the Machine Learning techniques used for Autonomous Vehicle algorithms rely on substantial amounts of annotated data in regular, as well as dangerous scenarios. The dataset must encompass varied weather and lighting conditions. Gathering such data by physical tests can be expensive, difficult, and even dangerous, as discussed above.

2.5 Simulation: An Effective Solution

The testing problem of the AVs can be transferred over to the virtual world, i.e., Simulation Environments. Realistic simulation environments comprising of high level-extensible modules like Environment module, Vehicle module, Physics engine, Sensor module, etc. provide an efficient solution for testing and validation of Autonomous Vehicles. Modeling and simulation are well-established tools for analysis, design, acquisition, and training in the automotive domain. Despite the heterogeneity of subsystems and disciplines involved in the development of an Autonomous Vehicle, there are many simulation models that allow coverage of the entire development process . Modern and dedicated simulators cover many aspects of the Autonomous Vehicle development process such as scenario generation, data gathering, realistic physics laws, realistic traffic flow, and photo-realistic graphics.

Fig(5):Waymo's simulation platform

The Automotive and tech giants in the self-driving industry have widely adapted simulation environments for testing and validation of the AVs. However, this doesn't completely eradicate 9 the need for on-road testing. Alphabet subsidiary Waymo has accumulated the most virtual mileage of all self-

driving companies, with a February 2018 total of nine years and five million miles. Waymo simulations created over 2.5 billion self-driving miles in 2016 alone. The Waymo simulator (Carcraft) transforms real-world scenarios into virtual formats and runs 25,000 virtual cars simultaneously. The massive data flow from this process assists engineers in locating bugs and adjusting models efficiently [14]. Apart from that, Apollo (Baidu), XVIZ (Uber), AVS (GM Cruise), VIRTTEX (Ford), Nvidia Driveworks, etc. are some of the premium driving simulators used in the self-driving industry. Self-driving simulators can boost the speed of data collection to reach mileage accumulation targets while reducing fleet operating costs. Among all the rivals in the self-driving car industry, Waymo clearly appears to be leading the race. For this, a fair amount of credit goes Waymo's simulation platform as it has driven more virtual miles than any other competitor. Apart from that, CARLA, Microsoft AirSim, VisSim, CarSim, Gazebo, TORCS, Udacity simulator, Autono Vi-sim, etc. are some of the most widely used open-source simulators for Autonomous Driving research.

3. LITERATURE REVIEW

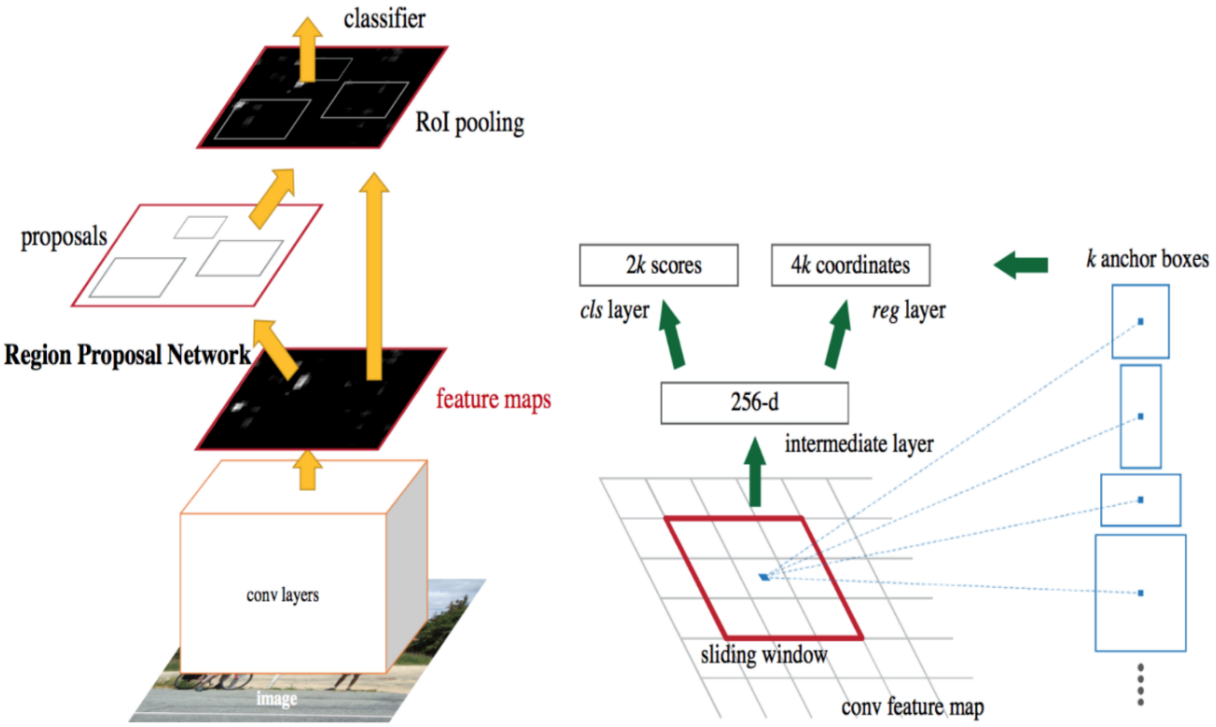
This chapter starts with a brief discussion about the Computer Vision techniques used for Autonomous Vehicle research, and the traditional datasets used to facilitate it. After that, we discuss the drawbacks and flaws associated with those traditional datasets. Then, we discuss how synthetic data has emerged as an option to fill this data void and the simulation environments used for that purpose in detail. Furthermore, we discuss how the synthetic data generated from basic simulators face the problem of the Reality gap and the ways to reduce this Reality gap.

3.1 Computer Vision in Autonomous Vehicles

Various Machine Learning and Deep Learning based techniques are used for several tasks like Object Detection, Object Recognition, Motion prediction, and Risk Assessment. Some of the most efficient techniques used for Autonomous Driving are briefly described as follows:

3.1.1 Faster R-CNN

- Faster R-CNN is one of the most widely used techniques for Object Detection in Autonomous Driving research. Faster R-CNN has displayed a superior performance over its predecessors, R-CNN and Fast R-CNN. At the conceptual level, Faster R-CNN is composed of 3 neural networks – Feature Network, Region Proposal Network (RPN), and Detection Network.
- The Feature Network is usually a well known pre-trained image classification network such as VGG minus a few top/last layers. The function of this network is to generate good features from images. The output of this network maintains the shape and structure of the original image (i.e., still rectangular, pixel size, etc.).
- The Region Proposal Network (RPN), is usually a simple network with three convolutional layers. There is one common layer that feeds into two layers – one for classification and the other for bounding box regression. The purpose of RPN is to generate a number of bounding boxes called Region of Interests (ROI) that have a high probability of containing any object. The output from this network is a number of bounding boxes identified by the pixel co-ordinates of two digital corners, and a value (1, 0 or -1) indicating whether an object is in the bounding box, or not in the bounding box or the box can be ignored respectively.

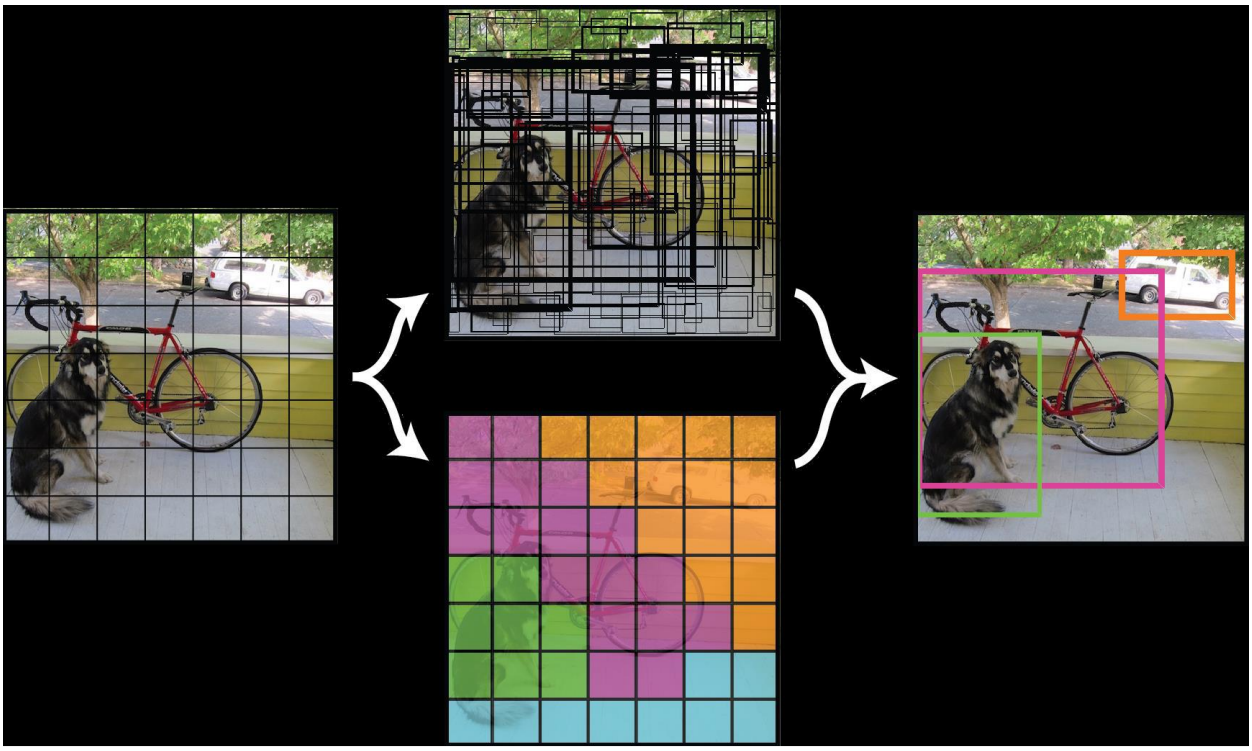


Fig(6):Faster R-CNN

- The Detection Network (sometimes also called the RCNN network) takes input from both of the above-discussed networks and generates the final class and bounding box. It is normally composed of 4 Fully Connected or Dense layers. There are two stacked common layers shared by a classification layer and a bounding box regression layer. To help it classify only the inside of bounding boxes, the features are cropped according to the bounding boxes.

3.1.2 You only look once (YOLO)

YOLO is an algorithm that utilizes a single convolutional network for object detection. Unlike other object detection algorithms that sweep the image bit by bit, the algorithm takes the whole image and reframes the object detection as a single regression problem, straight from pixels to bounding box co-ordinates and class probabilities. YOLO trains on full images and directly optimizes detection performance. YOLO divides up the image into a grid of 13 by 13 cells. Each of these cells is responsible for predicting five bounding boxes. YOLO outputs the confidence score that tells us how certain it is that the predicted bounding box actually encloses some object. After that, for each bounding box, the cell also predicts an object class.



Fig(7):YOLO: Object Detection with YOLO

3.2 Data Limitations for Computer Vision

Datasets are an integral part of contemporary object recognition research. They have been the chief reason for the considerable progress in this field, not just as the source of large amounts of training data, but also as a means of measuring and comparing the performance of competing algorithms. However, training deep neural networks for computer vision tasks typically require large amounts of labeled training data. A variety of approaches have been proposed to efficiently label large amounts of training data, including crowdsourcing, gamification, semi-supervised labeling, and Mechanical Turk. These approaches remain fundamentally bounded by the amount of human effort required for labeling or supervision.

3.2.1 Traditional Datasets Used in Computer Vision

Traditionally, datasets like KITTI , City Scapes , Imagenet , MS COCO, CIFAR-10, Open Images, etc. have been the most popular open-source datasets used in the computer vision field.

- The KITTI dataset: The KITTI dataset has been recorded from a moving platform while driving around Karlsruhe, Germany. It includes camera images, laser scans, high-precision GPS measurements, and IMU accelerations from a combined GPS/IMU system. The data in this dataset is calibrated, synchronized, and timestamped, along with rectified and raw image sequences. The

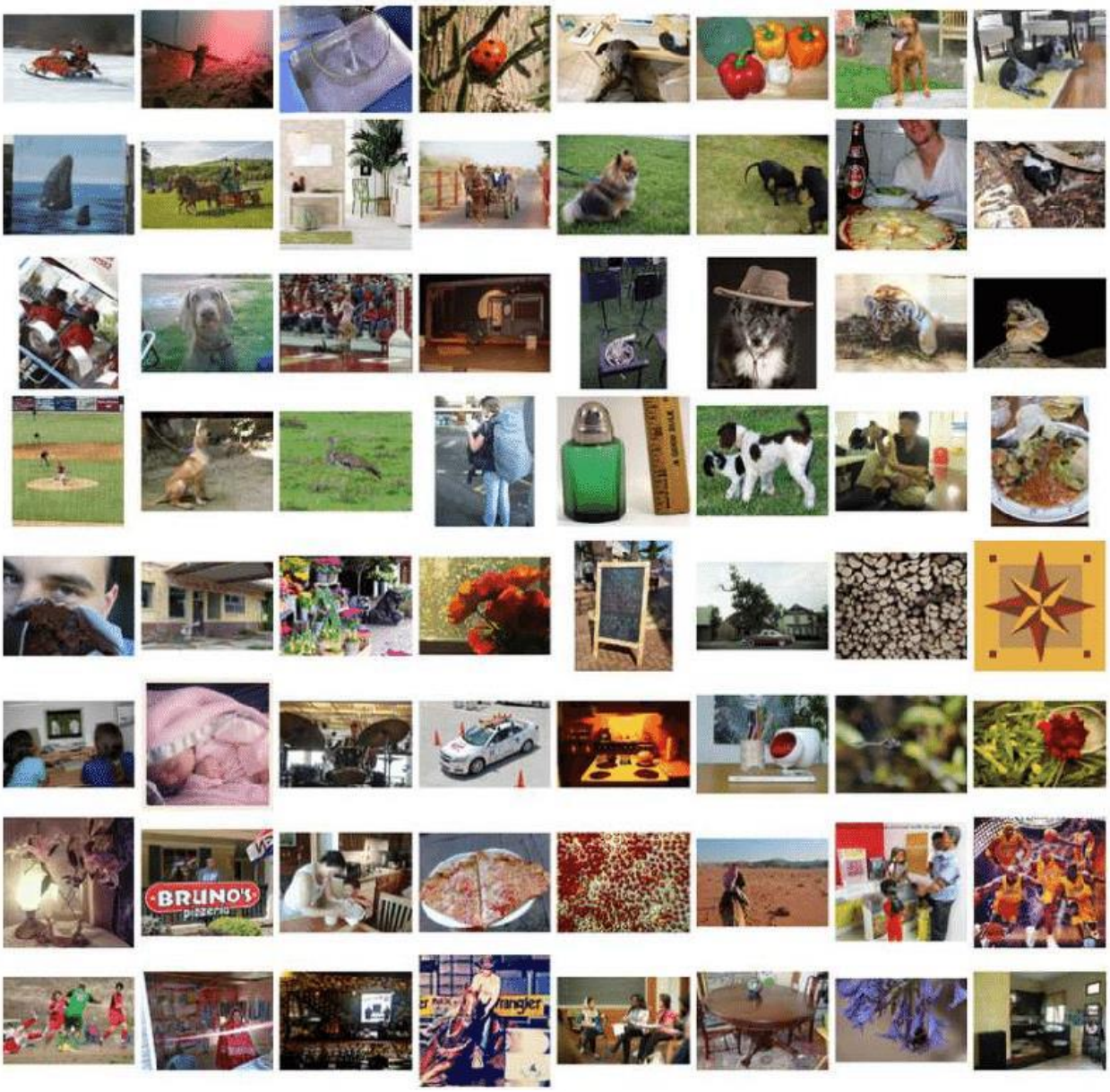
dataset also contains object labels along with online benchmarks for stereo, optical flow, object detection, and other tasks. For each dynamic object within the reference camera's field of view, the annotations are provided annotations in the form of 3D bounding boxes, represented in Velodyne coordinates. The object classes covered are 'car,' 'van,' 'Truck,' 'Pedestrian,' 'Person(sitting),' Cyclist,' Tram' and 'Misc'(e.g., Trailers, Segways, etc.).

Fig(8):Examples from KITTI dataset

- The CityScapes dataset: CityScapes is comprised of a large, diverse set of stereo video sequences recorded in streets from 50 different cities. 5000 of these images have high-quality pixel-level annotations; 20,000 additional images have coarse annotations to enable methods that leverage large volumes of weakly labeled data. However, recording in adverse weather conditions such as heavy rain or snow was avoided deliberately. The densely annotated data of 5000 images is split into separate training, validation, and test sets.

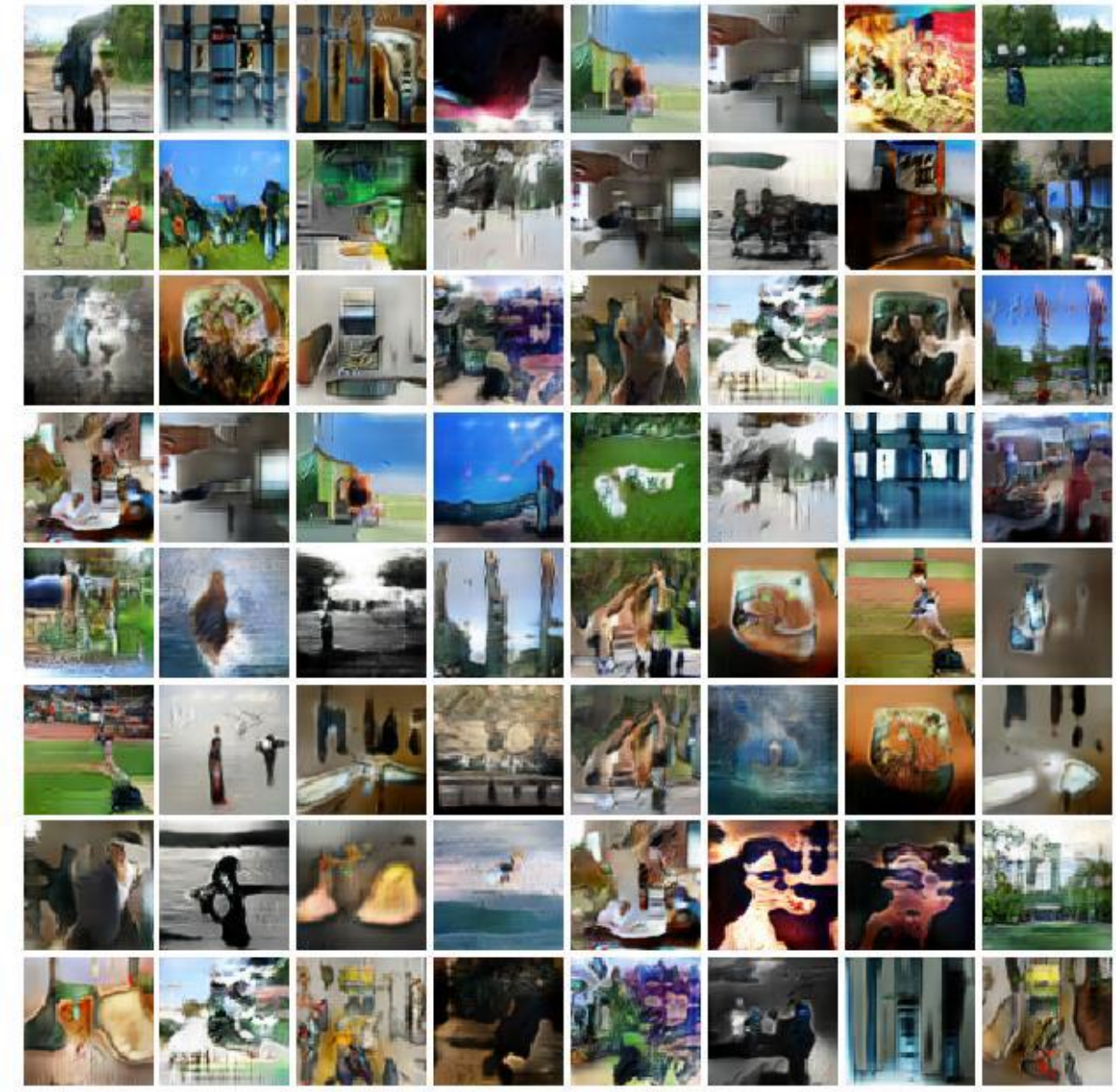
Fig(9):An example from CityScapes Dataset

- The ImageNet database: ImageNet database is a large visual database designed for use in visual object recognition software research. It contains more than 14 million images that have been hand-annotated to indicate what objects are pictured in at least one million pictures, and bounding boxes are also provided. ImageNet contains more than 20,000 categories with any typical category, such as “balloon” or “strawberry,” consisting of several hundred pictures. The database of annotations of third party URLs is freely available directly from ImageNet, though the actual images are not owned by ImageNet. Since 2010, the ImageNet project runs an annual software contest, the ImageNet Large
- Scale Visual Recognition Challenge (ILVRC), where software programs compete to classify and detect objects and scenes. ImageNet crowdsources its annotation process. Image level annotations indicate the presence or absence of an object class in an image such as “there are tigers in this image” or “there are no tigers in this image.” Object-level annotations provide a bounding box around the indicated object.



Fig(10):Examples from ImageNet dataset

- Microsoft COCO: MS COCO is another such large-scale dataset with the goal of advancing state-of-the-art in object recognition by placing the question of object recognition in the context of the broader question of scene understanding. This dataset contains photos of 91 object types and has a total of 2.5 million labeled instances in 328k images. Such annotation drew upon extensive crowd worker involvement via novel user interfaces for category detection, instance spotting, and instance segmentation.



Fig(11):Examples from MS COCO dataset

3.2.2 Need for Datasets Improvement

It would be safe to say that we are in the midst of a data revolution. Ubiquitous access to image datasets has been responsible for much of the recent progress in object recognition after decades of proverbial wandering in the desert. For instance, it was the availability of face training data, more than perceived advances in machine learning that produced the first breakthrough in face detection. And it is the dataset of millions of photographs of consumer products, as much as clever feature matching that allowed visual search engines like GOOGLE GOOGLES to become a reality. However, like any proper revolution, this one too has brought with it new problems to replace the old ones. It appears that this field is now getting

too obsessed with evaluation, spending more time staring at precision-recall curves than at pixels [21]. Some of the evident issues with current state-of-the-art datasets are as follows:

Lack of Diversity

Capturing sufficient diversity in a dataset is a challenge. We can often observe that datasets are restricted to a selected subset of cases, each dataset tackling one small part of the whole set of possible environments and conditions: for example, KITTY and CityScapes are collected only in Germany or even the Oxford dataset is only collected in Oxford, meaning that those datasets are geographically restricted.

Dataset Bias

The visual world is so complex and nuanced that any finite set of samples ends up describing just some of its aspects. Moreover, in case the samples are collected for a particular task, they will inevitably cover just some specific visual region. Hence, it is not surprising that pre-defined image collections like existing computer vision datasets, present such specific bias to be easily recognizable. The main types of bias found in existing computer vision datasets are as follows:

- **Selection Bias:** It is a known fact that datasets that are gathered automatically fare better than those collected manually. However, getting images from the internet does not guarantee a fair sampling, since keyword-based searches will return only particular types of images. Obtaining data from multiple sources (e.g. multiple search engines from multiple countries) can somewhat decrease selection bias.
- **Negative Bias:** Having a rich and unbiased negative set is important to classifier performance. Therefore, datasets that only collect the things they are interested in might be a disadvantage, because they are not modeling the rest of the visual world. An effective remedy would be to add negatives from other datasets.
- **Capture Bias:** Professional photographs, as well as photos collected using keyword search, appear to suffer considerably from the capture bias. The most well-known bias is that the object is almost always in the center of the image. For example, searching for a "mug" on Google Image search will mostly provide images with mugs situated at the center of the frame. Also, it will reveal another kind of capture bias: almost all the mugs have a right-facing handle.

Lack of challenging Weather and Lighting conditions

In computer vision tasks for applications like Autonomous Driving, it is very crucial that Computer vision algorithms are also trained in challenging and variety of weather and lighting conditions. Apart from a few datasets, all others only have images of driving scenarios in regular weather conditions. However, the landscape, roads, visibility, wind, etc. will differ for different parts of the world and with varying seasons as well.It is so important that Autonomous Vehicles are trained for every such scenario before finally deploying them on roads for public usage.

3.2.3 Synthetic Data: An Option

Amidst such complications with using real-world data, researchers have looked for efficient alternatives and Synthetic data (created using Virtual Reality and/or Augmented reality) has emerged as a worthy option. A promising approach to generate synthetic data is to use a graphic simulator to generate automatically annotated data. Several such simulated datasets have been created in recent years as found in etc.

SYNTHIA dataset: SYNTHIA stands for the SYNTHetic collection of Imagery and Annotations. This dataset is one of the best examples of datasets generated using simulated environments for Autonomous Driving research. SYNTHIA consists of photo-realistic frames rendered from a virtual city and comes with precise pixel-level semantic annotations for thirteen classes, i.e., sky, building, road, sidewalk, fence, vegetation, lane-marking, pole, car, traffic signs, pedestrians, cyclists and miscellaneous.

Fig(12):Examples from SYNTHIA Dataset

3.3 Simulation Environment

For dataset generation regarding Autonomous Driving research, there are various dedicated as well as general-purpose simulators available. Some of the dedicated simulators (Driving Simulator) even allow training the Autonomous Vehicles in the simulated environment, apart from data generation. Driving Simulators are usually a collection of high-level extensible modules that allow the rapid development and testing of vehicle configurations and facilitate the construction of complex traffic scenarios. They support multiple vehicles with unique steering or acceleration limits, as well as unique tire parameters and dynamic profiles to name from the vast features they provide. Engineers can specify the specific vehicle sensor systems and vary the time of day and weather conditions to generate robust data. Non-vehicle participants such as cyclists and pedestrians can be assigned specified routes or script scenarios that place the ego vehicle in dangerous reactive scenarios . Some of the main benefits of using driving scenarios are discussed as follows:

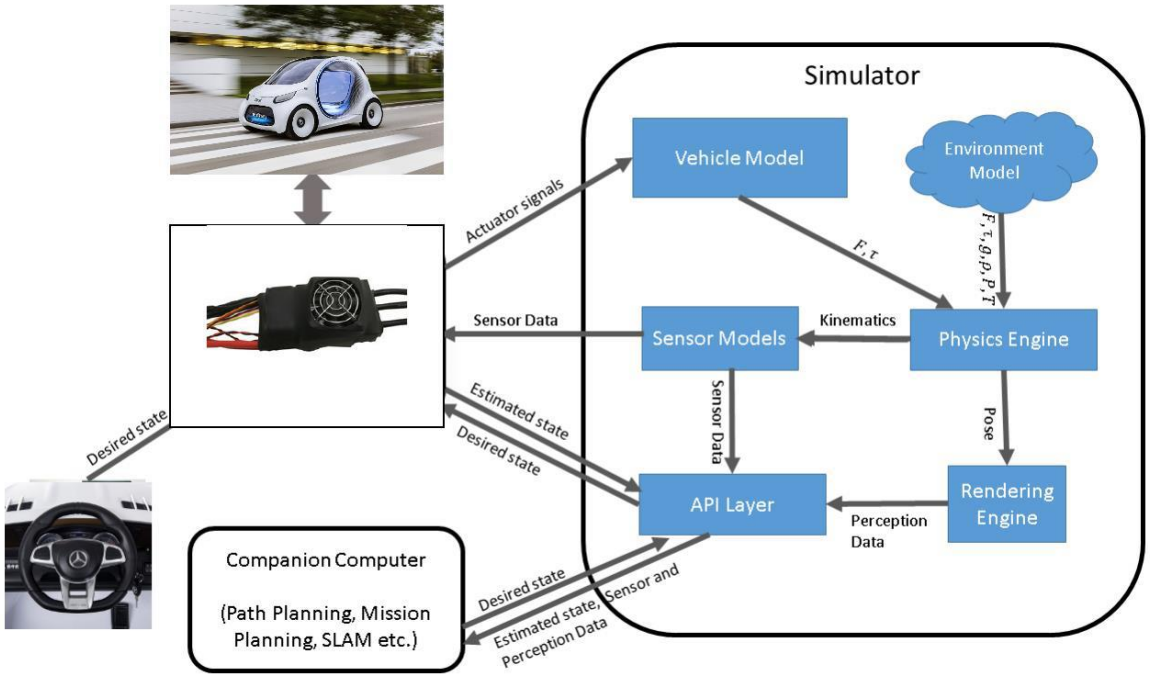
Data Generation: Driving simulators can generate virtually unlimited data for research and testing purposes. Apart from the main scene images, they also allow to export different vehicle configuration data and the data from virtual sensors attached to the ego vehicle. Furthermore, the data generated is automatically annotated as programmed, which eliminates the need for the labor-intensive manual annotating process. Varying vehicle, cyclist, pedestrian, and traffic conditions: The driving simulators include various vehicle and sensor models, pedestrians, and cyclists, as discussed above. The diversity of these traffic actors allows training for classification on different shapes, sizes, colors, and behaviors of cyclists, pedestrians, and other drivers.

- Dynamic Traffic, Weather, and Lighting Conditions: The driving simulators provide high fidelity traffic simulation, supporting dynamic changes in traffic density, time of day, lighting, and weather, including rain and fog.
- Rapid Scenario Construction: Typical road networks can be easily laid out using the in-built tools and are automatically connected for routing and navigation purposes.

3.3.1 Components of a Driving Simulator

An ideal simulator for autonomous driving research comprises a variety of inter-linked components covering the navigation, perception, and control modules. They are discussed as follows:

- **Game Engine (Rendering Engine):** A game engine is a part of a computer game that contains a 2D or 3D graphic representations (rendering engine), representations of physical laws (Physical Engine), or collision detection (and collision response), sound, scripting, animation, artificial intelligence, networking, streaming, memory management, threading, localization support, scene graph and may include video support for cinematics. The most modern game engines also include support for Virtual Reality (VR) simulation. However, for Driving Simulators, only the Rendering Engine is used among all the above-mentioned features of a game engine. This is due to the reason that the physics engine or collision response of a game engine may not be up to the standard required for a high-fidelity simulator. Some of the most popular game engines used for building driving simulators are Unreal Engine 4 , Unity 3D , Blender , CryEngine , etc. As of now, Unreal Engine 4, provided by Epic Games, has emerged as a favorite game engine for our purpose. At the same time, Unity 3D is also improving at an impressive rate.



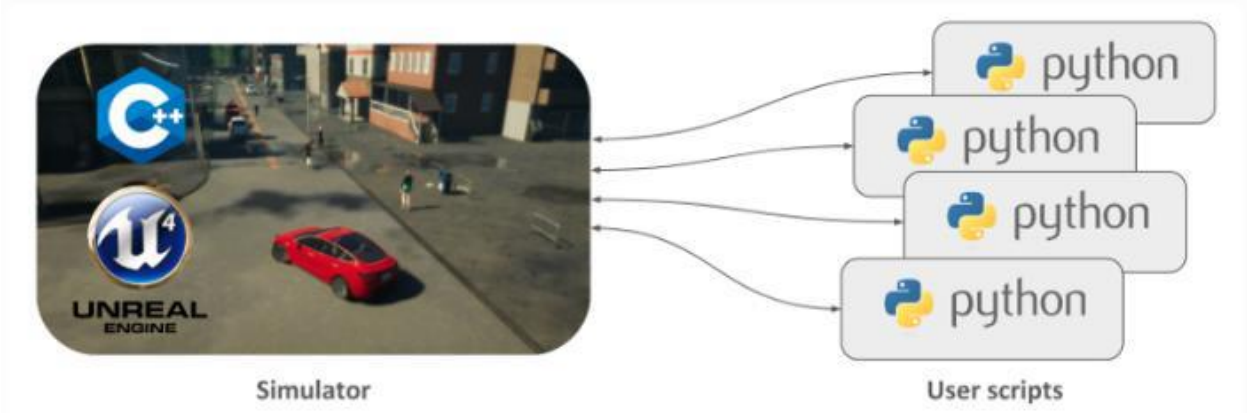
Fig(13): Components of a Driving Simulator

- **Physics Engine:** As the name suggests, a Physics Engine is a computer software that provides an approximate simulation of certain physical systems such as rigid body dynamics (including collision detection and response), soft body dynamics, and fluid dynamics. A Physics Processing Unit (PPU) is a dedicated microprocessor to host the physics engine.
- **Environment Model:** This module allows a user to specify a variety of lighting and weather conditions. The importance of data with varying weather and lighting conditions has already been discussed earlier.
- **Vehicle Model:** This module contains a variety of vehicle templates that can be simultaneously spawned in the simulation environment. Realistic vehicle control and behavior such as key steering assistance, braking assistance, support of traction control, flexible tires simulation, tire types, and customization, etc. are encoded in the vehicle templates.
- **Sensor Model:** This one of the most important and unique modules of a driving simulator. The sensor model comprises various virtual sensors that can be simultaneously equipped on the vehicles in the simulation environment. Though virtual, these sensors must be high fidelity and also behave like any other realistic sensors in the real world. The sensors are expected to provide realistic inputs to the ego vehicles, that can be extracted and stored with automatic

annotations for research purposes. Some of the common sensors provided in driving simulators are as follows:

- Camera
- Radar
- Lidar
- GPS
- Ultrasonic sensor

- **API Layer:** An API Layer is the component that makes the driving simulator extensible. Using simple but powerful scripting languages, a user can control traffic actors, weather and lighting conditions, change roadmaps, extract and process data, etc. Usually, this is



Fig(14): Client-Server architecture in Driving Simulators

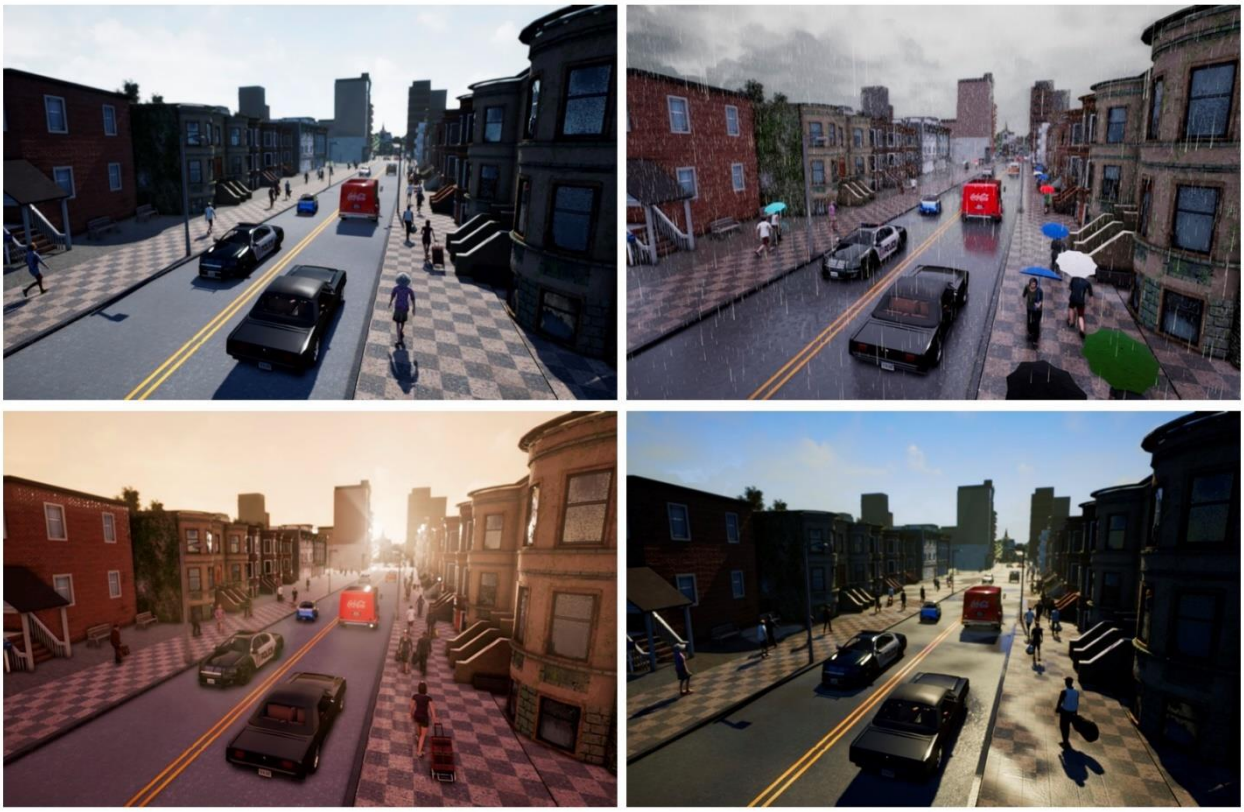
implemented as a client-server architecture where the game engine based simulator acts as a server, and various python/C++ scripts are on the client-side.

4. DRIVING SIMULATOR

4.1 CARLA: An Open Urban Driving Simulator

4.1.1 Introduction

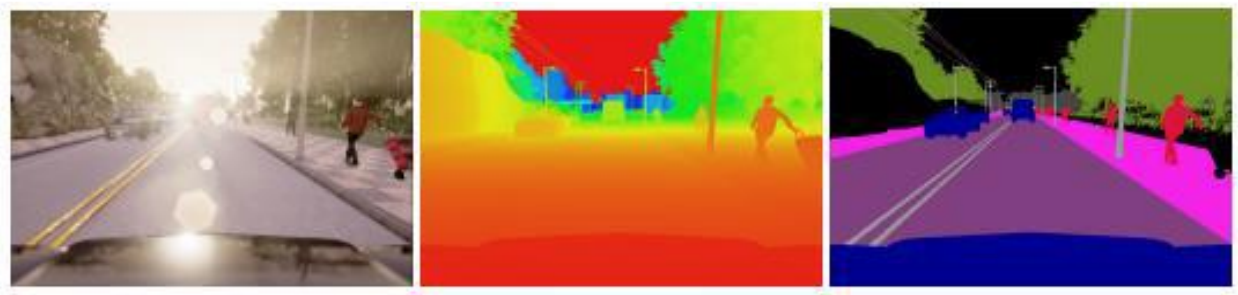
CARLA has been developed from the ground up to support the development, training, and validation of autonomous driving systems. In addition to open-source code, and protocols, CARLA provides open digital assets (urban layouts, buildings, vehicles) that were created for this purpose, and can be used freely. CARLA consists mainly of two modules, the CARLA Simulator and the CARLA Python API module. The simulator does most of the heavy work, controls the logic, physics, and rendering of all the actors and sensors in the scene; it requires a machine with a dedicated GPU to run.



Fig(15): A street in Town 2 in four weather conditions.

The CARLA Python API is a module that you can import into your python scripts, and it provides an interface for controlling the simulator and retrieving data. Most aspects of the simulation are accessible through the Python APIs, and the remaining will be covered in future releases. CARLA has been built on the Unreal Engine 4 (UE4). Apart from that, CARLA comprises almost all modern features and functionalities like impressive vehicle models, environmental models, photo-realistic 3D objects (static, variable, and dynamic), virtual sensors, etc. For implementation purposes, the necessary information and

basic steps to get started with this simulator can be found. Recently, various works have been implemented using the CARLA simulator for different purposes like vehicle testing, data generation, sensor validation, 28 object detection, semantic segmentation, etc. Implemented as a client-server architecture where the game engine based simulator acts as a server, and various python/C++ scripts are on the client-side.



Fig(16): Three of the sensing modalities provided by CARLA. From left to right: normal vision camera, ground-truth depth, and ground-truth semantic segmentation.

4.1.2 Highlighted features

- **Scalability via a server multi-client architecture**: multiple clients in the same or in different nodes can control different actors.
- **Flexible API**: CARLA exposes a powerful API that allows users to control all aspects related to the simulation, including traffic generation, pedestrian behaviors, weathers, sensors, and much more.
- **Autonomous Driving sensor suite**: users can configure diverse sensor suites including LIDARs, multiple cameras, depth sensors and GPS among others.
- **Fast simulation for planning and control**: this mode disables rendering to offer a fast execution of traffic simulation and road behaviors for which graphics are not required.
- **Maps generation**: users can easily create their own maps following the OpenDrive standard via tools like RoadRunner.
- **Traffic scenarios simulation**: our engine Scenario Runner allows users to define and execute different traffic situations based on modular behaviors.
- **ROS integration**: CARLA is provided with integration with ROS via our ROS-bridge.
- **Autonomous Driving baselines**: we provide Autonomous Driving baselines as runnable agents in CARLA, including an AutoWare agent and a Conditional Imitation Learning agent.

5. PROGRAMMING LANGUAGE AND IDE

5.1 PYTHON 3.6

Python is an easy to learn, powerful programming language. It has efficient high-level data structures and a simple but effective approach to object-oriented programming. Python's elegant syntax and dynamic typing, together with its interpreted nature, make it an ideal language for scripting and rapid application development in many areas on most platforms. The Python interpreter and the extensive standard library are freely available in source or binary form for all major platforms from the Python Web site, https://www.python.org/, and may be freely distributed. The same site also contains distributions of and pointers to many free third party Python modules, programs and tools, and additional documentation. The Python interpreter is easily extended with new functions and data types implemented in C or C++ (or other languages callable from C). Python is also suitable as an extension language for customizable applications. This tutorial introduces the reader informally to the basic concepts and features of the Python language and system. It helps to have a Python interpreter handy for hands-on experience, but all examples are self-contained, so the tutorial can be read off-line as well. For a description of standard objects and modules, see library-index. reference-index gives a more formal definition of the language. To write extensions in C or C++, read extending-index and c-api-index. There are also several books covering Python in depth. This tutorial does not attempt to be comprehensive and cover every single feature, or even every commonly used feature. Instead, it introduces many of Python's most noteworthy features, and will give you a good idea of the language's flavor and style. After reading it, you will be able to read and write Python modules and programs, and you will be ready to learn more about the various Python library modules described in library-index.

5.2 SPYDER IDE

Spyder is an open source cross-platform integrated development environment (IDE) for scientific programming in the Python language. Spyder integrates with a number of prominent packages in the scientific Python stack, including NumPy, SciPy, Matplotlib, pandas, IPython, SymPy and Cython, as well as other open source software. It is released under the MIT license.

Initially created and developed by Pierre Raybaut in 2009, since 2012 Spyder has been maintained and continuously improved by a team of scientific Python developers and the community.

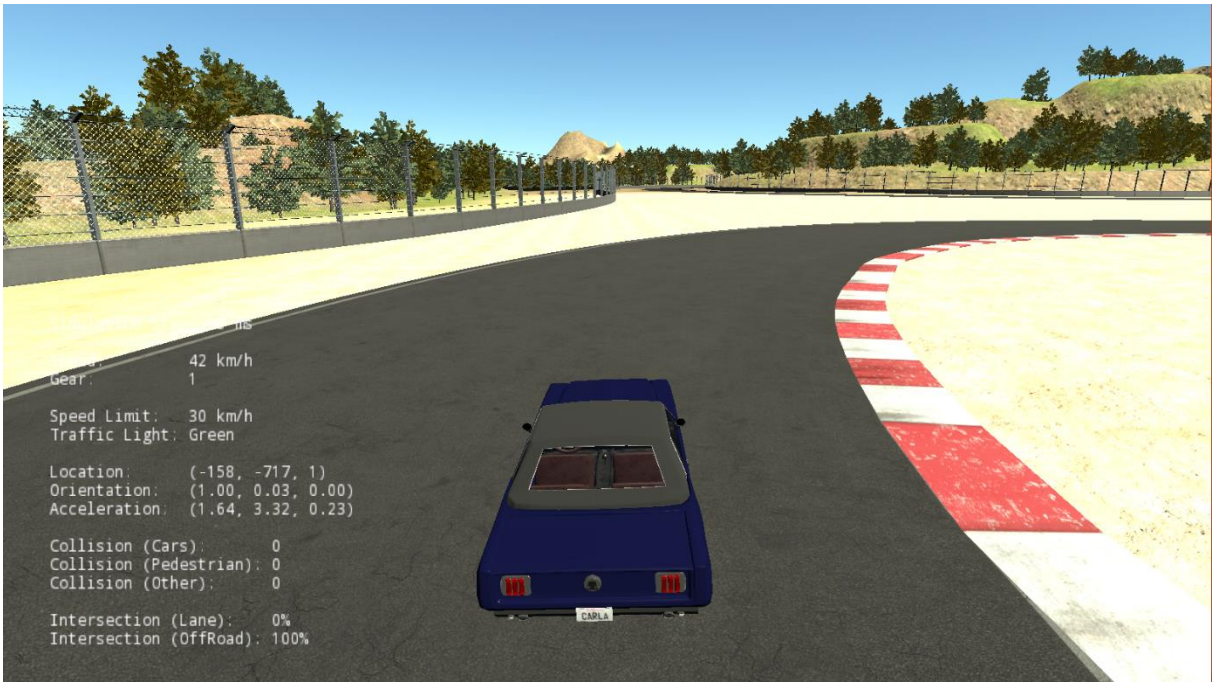
Spyder is extensible with first- and third-party plugins, includes support for interactive tools for data inspection and embeds Python-specific code quality assurance and introspection instruments, such as Pyflakes, Pylint and Rope. It is available cross-platform through Anaconda, on Windows, on macOS through MacPorts, and on major Linux distributions such as Arch Linux, Debian, Fedora, Gentoo Linux, openSUSE and Ubuntu.

Spyder uses Qt for its GUI, and is designed to use either of the PyQt or PySide Python bindings. PyOt, a thin abstraction layer developed by the Spyder project and later adopted by multiple other packages, provides the flexibility to use either backend.

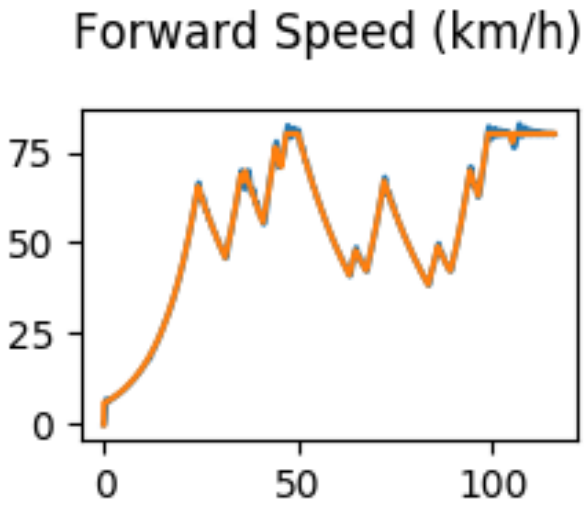
Features:

- An editor with syntax highlighting, introspection, code completion
- Support for multiple IPython consoles
- The ability to explore and edit variables from a GUI
- A Help pane able to retrieve and render rich text documentation on functions, classes and methods automatically or on-demand
- A debugger linked to IPdb, for step-by-step execution
- Static code analysis, powered by Pylint
- A run-time Profiler, to benchmark code
- Project support, allowing work on multiple development efforts simultaneously
- A built-in file explorer, for interacting with the filesystem and managing projects
- A "Find in Files" feature, allowing full regular expression search over a specified scope
- An online help browser, allowing users to search and view Python and package documentation inside the IDE
- A history log, recording every user command entered in each console
- An internal console, allowing for introspection and control over Spyder's own operation

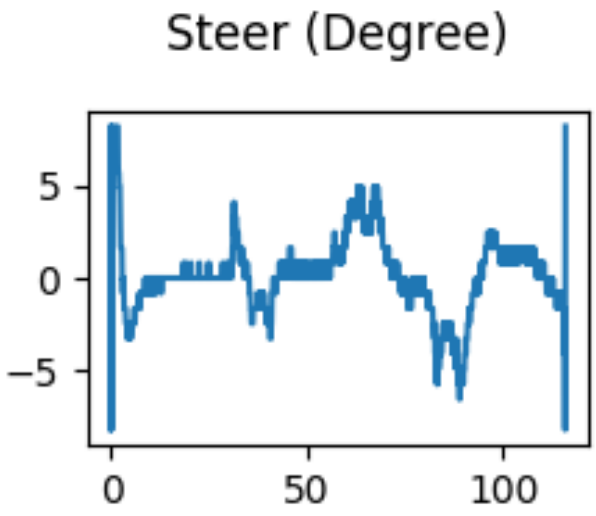
6. SCREENSHOTS



Figure(17): CARLA



Figure(18): Forward speed (km/h).



Figure(19): Steer (Degree).

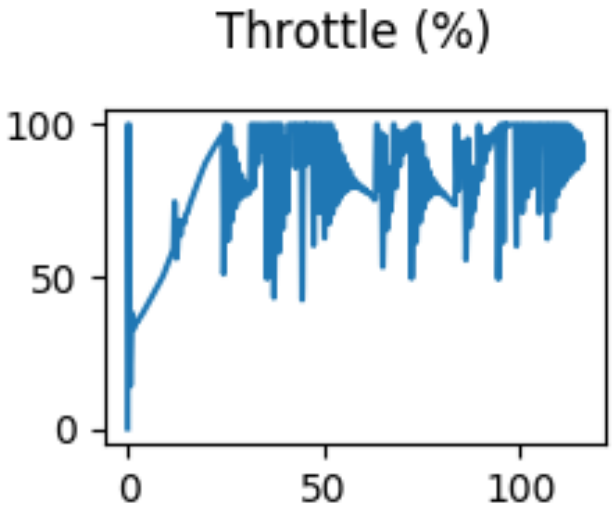


Figure (20): Throttle (%).

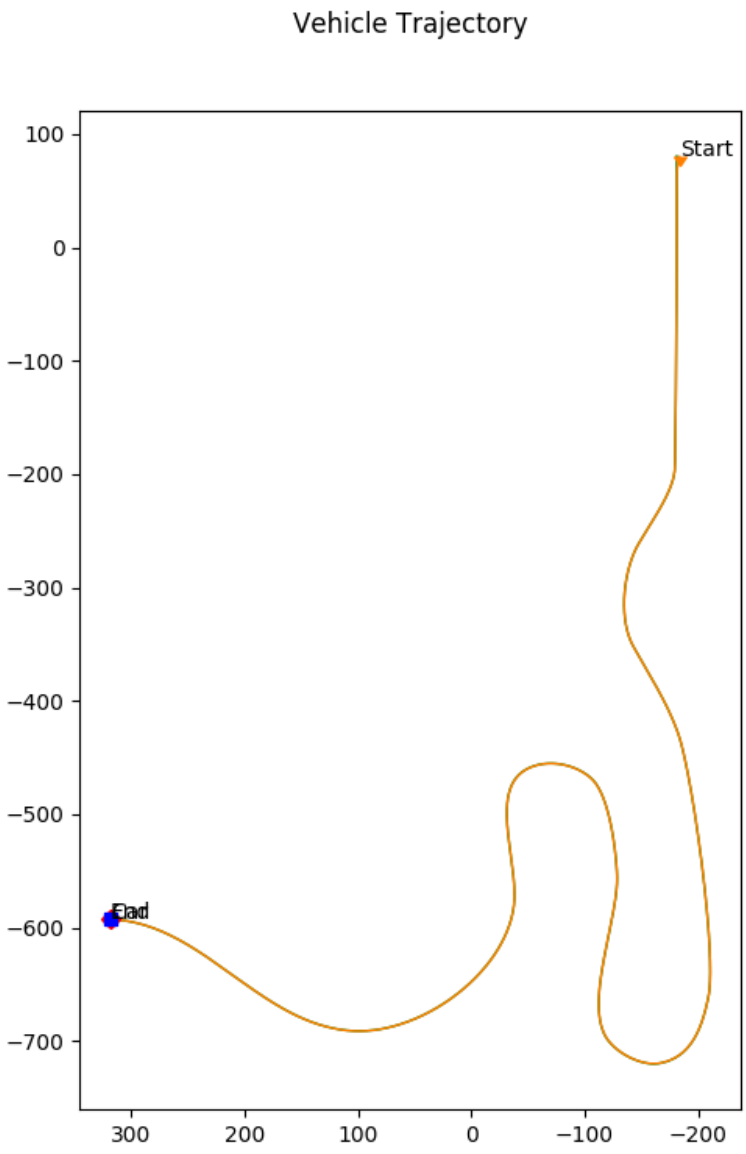


Figure (21): Vehicle Trajectory.

7. CONCLUSION

We conclude that we have presented a Simulation Environment with Reduced Reality Gap for data generation and testing Autonomous vehicle algorithms in a safe, fast, and cost-effective manner. We have provided a proof of concept of how real traffic scenarios can be replicated quickly in the simulation environment. We further demonstrated how those base scenarios can be edited and various complex test scenarios can be generated using the simulation environment set up in this work. It is safe to say that such developments in Computer Vision would be crucial steps towards fulfilling the qualitative and quantitive data requirements faced by Machine Learning algorithms. This would, in turn, solidify the Testing and Validation of Autonomous Vehicles in myriad scenarios, enabling the Autonomous Vehicles to learn more and get mature enough to replace human drivers and hence, change transportation forever.

FUTURE WORK

This section discusses the future work that can be carried out in this direction.

- As soon as the driving simulators get mature enough to allow the importing and usage of custom-built 3D object models while retaining all the functionalities, the landscapes, and other object models, the models used in this simulation environment can be replaced with actual custom ones, which would be a huge leap towards reducing the Reality gap.
- This simulation environment can be further used for Reinforcement Learning, which is usually the main purpose of driving simulators.
- This simulation environment can be customized further to facilitate more impressive Data Visualization.
- Higher levels of Domain Randomization can be practiced by including non-relevant object models in the traffic scenes.
- This simulation environment can be used to validate other Autonomous Vehicle algorithms like Object Recognition, Object Tracking, Semantic Segmentation, Scene Understanding, etc.
- Apart from the data generated in this work, this simulation environment can be used to implement a huge number of different test scenarios and generate large volumes of complex and realistic data (photographic).
- In this work, we have generated and analyzed just the photographic data. However, while implementing further test scenarios, sensors like radar and Lidar can be easily activated and 3D point cloud data can be generated.

9. REFERENCES

1. "Self-driving Car Market Global Industry Trends, Share, Size and Forecast Report By 2023|With CAGR of 36.2%," Market Watch, 3 September 2019. [Online]. Available: https://www.marketwatch.com/press-release/self-driving-car-market-global-industry-trends-share-size-and-forecast-report-by-2023with-cagr-of-362-2019-09-03. [Accessed 16 November 2019].
2. R. Spence, "Everything you need to know about the future of self-driving cars," Maclean's, 11 July 2019. [Online]. Available: https://www.macleans.ca/society/technology/the-future-of-self-driving-cars/. [Accessed 16 November 2019].
3. R. Spence, "Everything you need to know about the future of self-driving cars," Maclean's, 11 July 2019. [Online]. Available: https://www.macleans.ca/society/technology/the-future-of-self-driving-cars/. [Accessed 16 November 2019].
4. A. Acharya, 3 January 2017. [Online]. Available: https://atul.fyi/post/2017/01/03/how-self-driving-cars-work/. [Accessed 16 November 2019].
5. C. Badue, . R. Guidolini, R. V. Carneiro, P. Azevedo, V. B. Cardoso, A. Forechi, L. Jesus, R. Berriel, T. Paixao and F. Mutz, "Self-Driving Cars: A Survey," in *arXiv:1901.04407v2*, 2019.
6. "Simulations Pave the Road for Self-Driving Technologies," Synced, 8 April 2018. [Online]. Available: https://medium.com/syncedreview/simulations-pave-the-road-for-self-driving-technologies-78b696227383. [Accessed 17 November 2019].
7. M. O'Kelly, A. Sinha and H. Namkoong, "Scalable End-to-End Autonomous Vehicle," in *32nd Conference on Neural Information Processing Systems (NeurIPS 2018)*, Montreal, 2018.
8. D. Silver, "Self-Driving Path Planning, Brought to You by Udacity Students," Udacity, 25 August 2017. [Online]. Available: https://medium.com/udacity/self-driving-path-planning-brought-to-you-by-udacity-students-13c07bcd4f32. [Accessed 17 November 2019].