Lehrstuhl für Software Engineering II

Automatically Generating Driving Simulations from Videos to Address Safety Issues in Self-Driving Cars

Masterarbeit von

Qazi Mujahid

1. Prüfer	2. Prüfer	3. Advisor
Prof. Dr. Gordon Fraser	Prof. Dr. Sven Apel	Dr. Alessio Gambi

---

May 22, 2020

Contents

Abstract

Self-driving cars are an emerging part of automotive industry and a vital aspect of future. When it comes to automation of vehicles, that is transferring the control automobile to software, safety is the biggest concern as it can risk human life. In order to ensure safety in any driving conditions industry has to maintain some safety standards for the certification of self-driving cars. It is important to ensure that the software is intelligent enough to not only handle critical situations but also to predict or address any upcoming harmful event before deployment to prevent future mishaps. Discovery of test cases which can disclose the malfunction of an autonomous car is a thought-provoking task because the possibility of such test cases is infinite. Hence, one technique to analytically examine the autonomous cars safety is the simulations which are executed with no risk, no harmful condition and fast execution. Automatically generating driving simulations from real world driving videos could reduce time consumption of testers by manually creating different driving simulations. The intend to generate simulations from freely available videos will help testers to understand how the self-driving car would perform in similar situations. My thesis addresses this problem and define a new method to automatically generate driving simulations from commonly available geo-tagged videos recorded during driving. The process uses the GPS data to identify the roads in which the recorded driving took place, recreates those roads in the driving simulation, and configures the ego car to drive as the original car was driving in the videos. Next, the

videos are analysed using a machine learning classifier to identify leading cars by means of bounding boxes, their relative position and speed w.r.t. the ego car. Finally, a driving simulation reproducing the movement of the ego car and (one) leading car in front of it is generated. Evaluation results obtained by analysing randomly selected driving videos show that the proposed method is efficient and produce reasonably accurate simulations, suggesting that the proposed method is viable and can pave the way for future self-driving cars testing by providing an efficient tool to support testers in developing safe self-driving cars.

Acknowledgments

I would first thank to my thesis advisor Dr. Alessio Gambi, research assistant at Chair of the Software Engineering II at University of Passau. The door to Dr. Alessio Gambi office was always open whenever I stuck in difficult situation or had a question about my research. He consistently allowed this thesis to be my own work, but steered me in the right direction whenever he thought I needed it. His guidance played an important role in not only in fulfillment of research and implementation but also achieving results with greater accuracy. I would also like to acknowledge Professor Dr. Gordon Fraser, Chair of Software Engineering II at University of Passau for giving me this opportunity to work under his chair. Lastly, I would like to thank my parents, friends and siblings for being supportive in every step I took, which ultimately led me here.

Qazi Mujahid.

List of Figures

1 Introduction

Self-driving cars have been a vibrant component of automotive industry lately and are expected to gain more importance in future [Gar][PAP13]. When it comes to automation, safety is a big issue that cannot be compromised.

In self-driving vehicles the software is responsible to take rapid decisions and actions in possibly chaotic situations. This is where Artificial Intelligence (AI) enters; AI is all about intellectuality showed by the machines [Li+18][DAG17]. In the present, AI is expected to be a part of every aspect from smart home appliances to smart cars [Li+18]. In terms of self-driving cars one important question is what are the situations in which a self-driving car can crash? and how to prevent crashes? This question challenges the reliability of artificial intelligent applications. To address safety concerns in selfdriving cars the AI requires making quick and safe decisions because a human life could endangered otherwise.

The automotive industry has not yet established standards to certify self-driving cars safety [Li+18][Mar+17][DAG17], but research related to this aspect is in development by different organizations, out of which NHTSA (National Highway Traffic Safety Administration) [NHT18] is engage in saving lives, eliminating regulatory barriers to advanced

safety technology and promoting safe self-driving technology development. In order to prevent safety related mishaps, it is important to test autonomous vehicles in normal, problematic situations before deployment.

Virtual testing techniques have been introduced because the generic software testing techniques cannot manage the immense amount of situations in which a self-driving car may involve.
In some cases like Uber car crash[Abc][Theb], The object detection system was not able to identify a Women, when its sensors first detect as unknown object and then a bicycle. Therefore Testing in virtual environment can develop valuable results [Ber+13][PGE12]. After planting autonomous vehicles on test in real world scenarios the cars have shown appealing competency in avoidance of traffic accidents and interpreting there life threatening consequences [Mot16]. For instance, Google autonomous car ran into a bus while changing lane as the car anticipated that the bus will slow down and give space to the car, allowing it to join the lane. This indicated that self-driving vehicles can perform not accurately in less complex situations leading to unpreventable accidental situations. Searching for such faults in self-driving cars is a very difficult task. Since the amount of testable scenarios is not finite due to the fact that automated cars are expected to function safely on various roads, with continuously changing atmospheric and weather conditions that involves irregular movement of different objects (other vehicles, people and animals)[Zof+16].

Another incident was reported in March 2018 [usa][Thea][Abc][Theb], when an experimental Uber SUV, functioning in autonomous mode ran into and killed the pedestrian woman at night in Tempe, Arizona. After the investigation it was found that the self-driving system detect the two objects (the woman and her bicycle) 6 seconds before the

accident occurred but the engineers had deactivated the cars emergency braking system in order to decrease the strength of unpredictable behaviors [usa][Thea]. In addition, the system software was not programmed to prompt the human operator so that he/she could apply brakes manually. Such an accident could be prevent by reducing the speed to 90% or by enabling the car to apply brakes, horns as soon as it detects a static object in front of it or when the object comes under the radius of 10 meters. Another solution is to prompt the human operator, but this can also lead to human error related mishaps.

In this Thesis, I propose to address the above stated issue can be resolved by generating simulations from real dash-cam videos and test how self driving car will act under different situations. The video provide tester a clear view of the situation, for this virtual simulation using 3D Simulation software [Bea18] will be used to recreate the Scenario virtually and conclude if the cars performs better and takes more intelligent decision then humans to prevent damages thus the car can be termed better and safe as compared to a human driver in the same situation.

Amongst all the descriptive collected possibilities that includes textual information, official documents like police reports and sensorial data, I propose consider dash-cam videos with GPS data. I use the dash-cam videos as my main data source as the videos are widely available over the internet and has captured accidents occurred on various locations (highway, streets, flyovers, etc.). Another reason for selecting dash-cam videos is that they could provide an accurate, clear interpretation of the situations, which lead to the generation of better tests.

1.1 Research Questions

The automatically generated simulations from videos intent to not only reduce the testers efforts involved in the creation of simulations manually, but also this will let the testers build more effective and extensive test collections to evaluate the system ability of self driving car. Moreover the simulations, which replicate the story line of real scenarios could produce more test cases by changing situations, such as lighting conditions, by introducing more cars and the existence or nonexistence of prominent features from the original video. The objective to generate simulations is to construct several test situations for testers to try/evaluate the self-driving cars in situations where a tester can build more test cases through parameter exploration.

1.2 Thesis Organization

In this research the main data source will be dash-cam videos (video recorded during driving). The dataset of the video includes the GPS information which will be used in creation of roads using OSM (Open Street Map)[Ope17] then the simulation will be created with road and ego car. Then the next step is to detect the vehicle in video frames and estimate the relative speed, distance of the car facing in front of the ego car. The extracted parameters will be transferred to be executed as the final simulation.

2 Background

This section presents a explanation of the insights from the autonomous industry. Firstly, it introduce some of the terminology regarding the autonomy of self-driving cars, provided facilities and shared resources. The approach to elaborate the stages of AVs development as well as the advancement of technology in vehicles control at various stages is being discussed. Moreover this section discuss the stages of autonomy, as substantial fundamentals of the driving intellect. Furthermore, a comparative analysis in various light, weather and distance described would aid the purpose of research in my domain.

Autonomy in vehicles

Autonomy is being in the condition of self-determining; being decisive and self-dependent. Application of such concept on cars evolved vehicles which are capable to function with minimal or zero human interaction, through the implications of AI. Any vehicle with functionalities that allow it to start, break and function without any driver.

Autonomous vehicle

This describes car able to drive from location A to B with no driver. The autonomous car is able to travel due to the ability of sensing the surroundings, detection and identification of the objects and surroundings. By being proficient in constructing the location

frames as well as determination of exact location and by utilizing a global positioning system (GPS) autonomous vehicles would be able to function accurately. Along with the attributes of Sensing, Mapping and being aware of the Driving policies/rules, the potential to eliminate the drivers errors, could enhance performance of the an AV as it is less likely to experiences faced by a human while driving on a road. Yet, there are huge challenges that could not be over looked and needs to be catered in order to propose autonomous automobiles with minimal protection hazards to the community.

2.1 Autonomy Levels

The progress of self-driving in stages are presented, which is started from no automation to fully automated vehicles. These levels are characterized on the basis of functional ability associated with autonomous driving. Moreover in order to understand the AVs current market it is necessary to reflect over the types of automation. US National Highway Traffic Safety Administration (NHTSA) and Society of Automotive Engineers (SAE) [SAE]. To distinguish NHTSA has a scale of 5 levels for defining automated driving, while SAE has a 6 stages model. In contrast SAE standard was accepted by NHTSA. This thesis will describe the SAE standards and the six autonomy levels in detail.

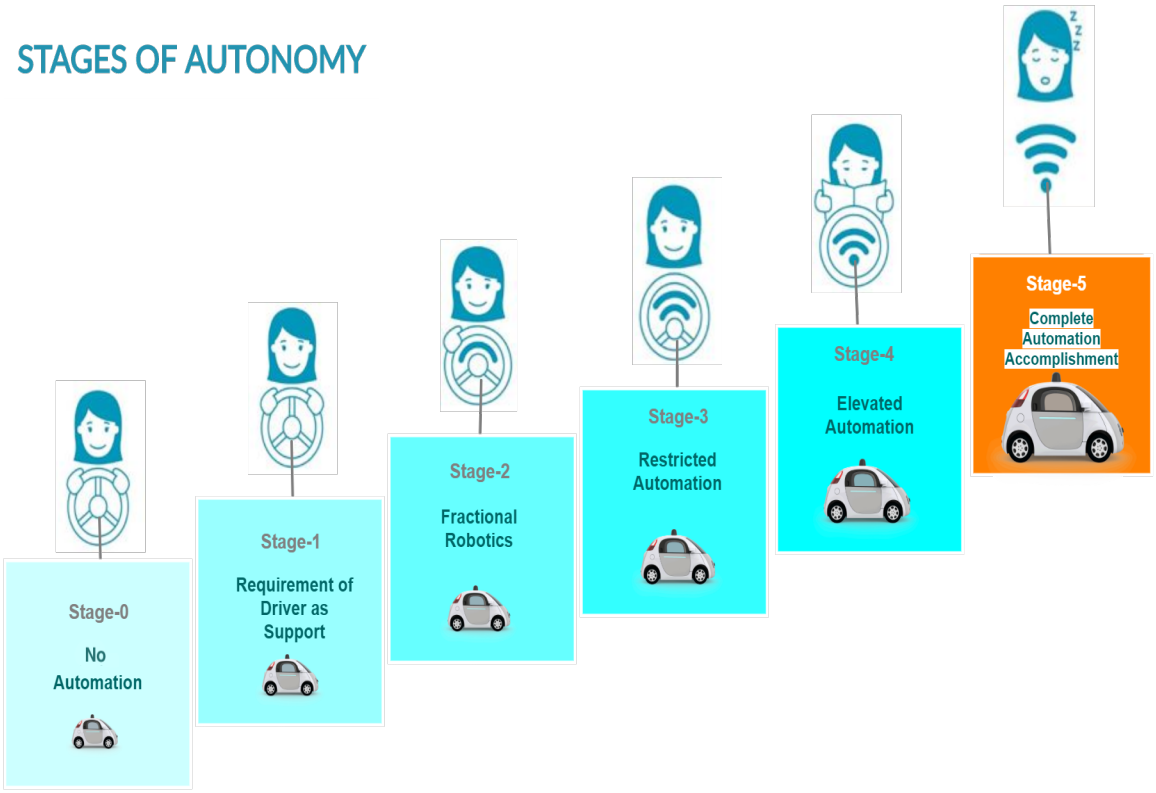

Figure 2.1: Stages of Autonomy

2.1.1 Stage 0 - No Automation

Level of automation: In this case, the car driver has complete control over the horizontal and longitudinal dynamics of an automobile. It means that human driver is accountable for an adequate performance in every feature of driving. Even though humans are responsible for each action taken in this mode there is no guarantee of accuracy or zero collision due to human error.

Drivers duty: The driver is liable for the safety procedures possessed by the automobile, has to observe and be conscious of the transportation nearby the car. There is no

automatic functionalities to transfer the mechanism of the automobile.

Time Period: still present, the orthodox cars were designed requisite human interference in order to regulate navigation, acceleration, and brake. In the interim, the car driver is blamable for checking the environments, steering and making preemptive decisions whenever it comes to turning pointers or adjusting tracks.

2.1.2 Stage 1 - Requirement of Driver as Support

Level of automation: This stage signifies a state where majority of the monitoring has been accomplished by the driver. Activities like acceleration, brakes could be automated. this is stated as a traffic flow supporter that retains the automobile in a flow, the car driver little influence over steering, correction, and braking. At a definite moment, the control can be transferred to driver assistance system.

Drivers duty: The human driver is considered to be in charge of controlling his vehicle with the safety procedures and operations but the control could be transferred to system software occasionally.

Time Period: Already existing. Multiple vehicles have a built-in traffic jam assistant fall in this category.

2.1.3 Stage 2 - Fractional Robotics

Level of automation: At Stage 2, there are numerous autonomous functions that are controlled by the system. Actions like lane keeping, automatic braking, and cruise control. Level 2 is the start of autonomy and it is restricted to particular functions,

mostly used in highway drives when there is no need to observe and recognize traffic lights. The core functionalities such as lane keeping and adjustment of steering are exhibited while driving on highways where applying brake, acceleration or navigation takes place in correspondence of the vehicles in the surroundings whenever lanes are being changed. Moreover, in level 2 the control is being transferred to the driver completely when the system detects object and the features which are not responding. Automobiles with stage 2 autonomy could be used in outlying regions wherever road infrastructures are slightly narrow, which makes easy for a system to detect objects easily.

Drivers duty: The car driver is accountable for general regulations applied on a car and safety processes. Certain parts of the main steering system could be assigned to auto-pilot mode, maintenance of a continuous speed and the identification traffic signs and landmarks on the road with respective adjustments could be delegated to the system at any point. The driver is required to take charge whenever the system faces a situation when navigation is not possible or complicated.

Time Period: Level 2 is publicly available and has been released for commercial usage. Tesla being a leading stakeholder in the automation industry has captured a huge market and is selling vehicles functioning in stage 2 automation.

2.1.4 Stage 3 - Restricted Automation

Level of automation: At level 3, a minimum two key steering systems are programmed and operated concurrently. For instance, parallel functioning of a lane centering composed with an adaptive cruise. Cars with stage 3 automation are skilled in leading a vehicle in complex settings in an urban area whenever associated with stage 2 which was commonly constrained to a freeway. The stage 3 cars are capable to identify and classify

the road traffic marks, traffic lights which means it can function in built-up regions but not with 100% accuracy. Similarly, a car with stage 3 autonomy has difficulties while detecting and distinguishing similar surroundings in changed weather conditions.

Drivers duty: Driver could rely on some of the controlling functions and delegate some controls to a vehicle. Even then the driver cannot leave the steering as danger can interrupt the safety protocol at any instance. At present such automation is seen on highways wherever certain vehicles are able to drive in the driver assistance mode where as some level 3 vehicles could be driven in inner-city and sub-urban areas. The navigation requires human interactions and decisiveness.

Time Period: Already available.

2.1.5 Stage 4 - Elevated Automation

Level of automation: At this stage the vehicle is intended to safely operate while being in the automated mode of driving. In a dangerous circumstance, a driver could control all safety related functions in order to cope up with the problem.

Drivers duty: The driver shifts to automatic driving only when there is no safety issues associated. After the delegation of the control to the system, the driver is not required to get involved in monitoring, observing and can rely on the system until it prompts an error. The driver can take over the control when the system is not able to respond properly.

Time Period: Tesla has already released vehicles with hardware currently functional at stage 4 automation. Such vehicles can confidently steer through the metropolitan areas and are capable to deal with certain situations without any driver on board as

well. Protocols and safety policies had not allowed Tesla to publish the revised software release online which could permit the Tesla to easily update the cars to this stage.

2.1.6 Stage 5 - Complete Automation Accomplishment

Level of automation: This level of automation comprises of a state where driving functionalities are completely automated and the vehicle is able to perform safely without human interaction. All the difficult conditions would be managed and detected by the system. Hardware like steering wheel and all pedals would be removed. Such cars would be able to navigate deprived of a driver completely.

Drivers duty: The driver would be a passenger, no surveillance and monitoring of the traffic would be conducted by the driver. The driver would be required to input a certain destination or some other specifications such as temperature into the system the passenger can also book or view the current car position through his phone. The vehicles can drive to any authorized target or pin point on itself showing the decisive intelligence during the drive.

Time Period: expected in upcoming years.

The removal of human in driving requires functioning with numerous complex technologies concurrently. Human capabilities for making decisions will be replaced by deep learning and machine learning. By connecting to a cloud can enable customized features and models paired with the extraction of continuous meaningful data and operations to distinguish the basic entities and hindrances close to the automobile. Self-driving car enactment can involve numerous segments. Subsequently the accomplishment of Stage 5, supervisory authorities will approve production of self-driving vehicles for commercial purposes.

Despite of all the shortcomings, the quality of data obtainable by Lidars makes them a perfect choice in the strategy of most of the autonomous driving industries (Waymo, Daimler, Audi, Bosch, General Motors). Tesla decided to eliminate the installation of Lidar sensor on cars. The Waymos self-driving minivans use three type of Lidars, five sensors and eight cameras while Tesla has mounted eight cameras, twelve ultrasonic sensors and one forward facing radar [ZF17]. CEO Tesla Elon Musk is firmly induced that Lidars are avoidable in the development of fully autonomous cars. According to Elon Musk [Bur19], "Lidars are just props that would drive companies to a naive deadlock where recovery would be daunting."

Tesla emphasis that camera sensors combined with radars and ultrasonic sensors could excel the current performance. This is an extreme choice the enterprise is following a completely different pathway as compared the competitors; the idea behind this approach is that enough data is provided by the camera sensors from the surroundings and consequently allows a car to drive all on its own. To challenge this strategy is difficult. Humans are perfectly able to drive cars deprived of the ultrasonic, radio or light sensor; they mostly fail because of the absence of attention, not of information. The hardware required for the computation and various costly sensors could be eliminated. In this thesis, image processing and machine learning techniques would be effectively used for the extraction of relative speed and distance.

2.2 Sensors

The reliance of an autonomous system is based on three blocks:

Perception: to sense the surroundings and defining an illustration of the location.

Decision Making: to analyze the setting and adopt an action.

Actuation: to execute the previously taken decision.

The observation systems of sensors are formally categorized in two forms: Proprioceptive sensors [Fin19] which are respnsible for distinguishing vehicles state like inertial measurement unit and wheel encoders etc. Exteroceptive sensors are responsible for identifying of ambience observing devices placed externally like cameras, Lidar, Radars, ultrasonic, etc. The exteroceptive sensors are significant in autonomous driving they provide the true definition of the position and setting that is require to aid in precise decision making. The main sensors possessed by this group are exemplified in the following sections.

2.2.1 Camera

Being the usual and inexpensive sensors, cameras are present in all the models of autonomous cars and used in the implementation of many ADAS features. Mono-vision or stereo-vision are the two common camera based systems which consist of multiple mono vision cameras that somehow resemble human eye vision. There is a possibility of having more than two cameras facing a similar measurements for the reconstruction of a 3D image. The development of various features, estimations and actions are dependent on the placement of cameras some common mounting places for cameras are the front grilles, side mirrors, and rear door and rear windscreen. Advanced camera systems and software integrated together could accomplish more than object detection that includes the accurate determination of object trajectories and enhanced feature classification. The modern addition in this field is the Mobileye [Mob18], presented by an Israeli subsidiary of Intel collaborating with automakers. The idea behind Mobileye is to make

computer able to take effective decisions like humans. Camera can have a great impact on AVs to enable them achieve perception/actuation similar to humans. Explicit and implicit information shows that if cameras are placed in a range with 360 degree coverage could comparatively collect meaningful data that is can become the main support for a motorized sensing suite.

2.2.2 Radar

Radars (Radio Detection And Ranging) utilize radio waves frequencies for the detection of objects and conclude their range, distance, angle, and/or velocity. Radars can obtain information of objects that are in the closer radius, the fetched information includes their distance, mass and speed. Advance driver assistance system (ADAS) can notify the driver in hazardous circumstances and reliant on the system, they can even trigger advanced navigation or braking. Additionally Long-ranged radars can provide coverage of approximately 200 meters at a higher speed whereas short-ranged radars are slower and are used for sensing the surroundings within or nearer location of the vehicle present in a range of 30 meters. Short-ranged radars are beneficial in the implementation of Blind Spot Detection, lane adjustments, collision prevention, parallel/adjacent traffic monitoring, distance control automation etc. Radars with high-precision and weather diagnostic are the must have features for any autonomous vehicle and prototype.

2.3 Coordinate System

The transmission of state information and entities existing in the real world is termed under DIS (Distributed Interactive Simulation) [12]. The most essential state of information is generally the information about where the entities are located and in which

direction they point. This arises question: which coordinate system would be helpful to define an entity's location. GPS uses geographically accurate coordinate system which are named as geodetic coordinate system, to account for the shape of the earth, while the simulator makes the simplifying assumption that the Earth is flat, so it uses a basic Cartesian coordinate system.

2.3.1 Game Coordinate system

The technology is used in military simulation application are similar to the once used in gaming engines, so the solution provided and applicable in games to illustrate real world direction can be useful to understand the above coordinate system problems. The graphics packages used as high coordinate system to position objects and then create draw them in Blender [Com18] 3D tool screen depicts a simpler situation, a cube that sets of a few unit from the origin of universal coordinate system, the general coordinate system origin depicted by intersection of red and blue lines as showed in figure 2.2, the cube has a set of local coordinate system at the very center.

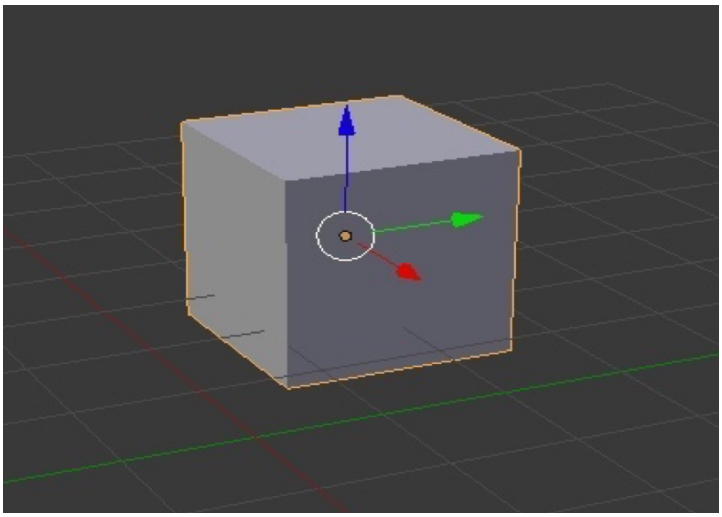

Figure 2.2: Coordinate system in Blender [Com18]

2.3.2 Coordinate Systems in Real World

When we are trying to describe the position of entities in the real world we have a less pleasant situation. Entities are placed on the earth's ellipsoid-shaped surface, their placement is worse than using a perfect round sphere. More realistically models used in naval warfare would not allow us to see a ship 100 km away from it would be below the horizon. Using a solo, cohesive, uniform coordinate system would also not be realistic to conduct across a large geographic scale. The gamers approach of using a flat coordinate system for a plane world could not function either because the actual world is not flat. The Army often uses Military Grid Reference System (MGRS) [dat] to designate the spot of units, as mentioned, a game graphics package is used to describe the position of entities and the graphics system coordinate system is Cartesian. For example a simulated automobile driving through a town, is being induced and rendered in a game engine that uses a conventional Cartesian coordinate system being characterized as flat and rectilinear, using the web as a foundation of place data for real objects in the world, like Google Maps can verify those points termed in latitude, longitude, and altitude.

In order to search that while being inside the game it should be properly simulated through satellite the position of that satellites orbit may be described using Keplerian orbital elements [Lho] for the arrival to designation specified in MGRS. Concurrently the curvature of earth could not be ignored. Cartesian coordinate system along with the origin at the center of the earth while using meters as the unit of measurement.

In this coordinate system the X-axis points out from center of the earth and crosses the surface of the earth at the prime meridian and equator. The Y-axis meets the earths surface at the equator at 90 degrees on east longitude and Z-axis points to the

North Pole. This coordinate system switches with the earth; it is "Earth-Centered, Earth-Fixed" (ECEF) [Pop14].

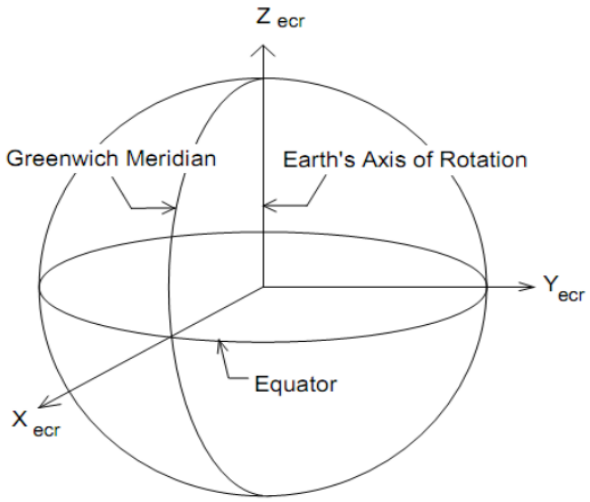

Figure 2.3: ECEF Coordinate System [Pop14]

The simulation presented in this research can use suitable coordinate and would transmit its entity positions to other simulations using the geocentric coordinate system [Pop14]. The necessity is a way to convert to and from our internal coordinate system to the global geocentric coordinate system. The gain of using a geocentric coordinate system is it could convert it to and from other general coordinate systems. There are equations to convert and from a position defined with latitude, longitude, and altitude to the geocentric coordinate system and vice versa.

2.3.3 Standard Problems in Coordinate System

Although the geocentric coordinate system basis is placed at the center of the earth, the coordinate system does not define itself where the surface of the earth is actually located. There are several mathematical models, called "datums" used to describe what

the shape of the earth is because the earth is somewhat flattened and an egg-shaped surface, usually in the form of an oblate spheroid. Model WGS-84 [Jan09], which is also the model for the shaping of the earth used in GPS which is not exact; the real world's mean sea level can differ from the geoid defined by WGS-84 by 0-5 meters but replications that are modeling kinetic weapons involve high precision for entity locations.
The usage of different datums also causes the models to differentiate in the shape of the earth. The latitude and longitude lines are fixed to the surface of the earth through models and maps are using different datums. The datums define two diverse 3D surfaces, entities labeled with the same latitude and longitude will be in two different settings in 3D space depending on the datum the map uses latitude and longitude.

There are computational problems, using units of geocentric coordinate system values could exceed six million meters that causes some numeric precision problems if using single precision floating point numbers. Thus it is required to convert coordinate systems discussed below could consequence in the termination of computational errors.

2.4 Open Street Map (OSM)

It is a utility that is helpful to generate an alterable map free of cost. This project helps in the generation of such data as primary output that aids in the navigation of satellite devices and provide prominent geographic information across the globe. The Open Street Map is inspired by the prevalence of patent map data amongst the world in order to outsource collected data in a usable format. Due to the availability to OSM replacement of default data with GPS has been comparatively favorable for this research.

2.4.1 Production of Maps:

Using OSM the collected map data from GPS units, cameras and sensors is added to the database of OpenStreetMap afterwards manual editing and automated imports are used for data preprocessing.

2.4.2 Street-level image data:

Similarly, numerous altered sets of satellite images are accessible to OSM editors, data from street-level image are available as map data photo covers, Bing Streetside 360° and the OpenStreetCam platforms as well as smartphone, mounted camera images. A traffic sign data layer can be enabled to enhance user-submitted images.

2.4.3 Data storage in OSM:

The OSM data is stored and managed in diverse formats. The main file of the OSM data is deposited in OSM's database this database is a PostgreSQL database with PostGIS extension, having a single table for every data primitive, with discrete objects stored as rows, updating and manipulation happens in this database after which other formats are generated from it. For transferring data, database dumps are made, which could be downloaded. The whole dump is called planet.osm. These dumps occur in two set-ups, one using XML and one using the Protocol Buffer Binary Format (PBF).

2.5 Histogram of Oriented Gradients (HOG)

HOG is a feature descriptor used for image processing, primarily for object detection and was developed by dallal et al.[DT05] and they use hog for human detection. HOG splits the image into tiny cells, HOG is calculated for each cell, the block pattern is used to normalize the HOGs and descriptor is provide for each cell. The features are used to detect objects in an image. The presence of local objects within image is define by the distribution of intensity gradients. This process involve measuring the frequency of gradient orientation and thus preserving photometric transformations and geometric invariances. The generation of descriptor consists of four steps: gradient computing, orientation binning, generation of descriptor blocks, and block normalization.

2.5.1 Linear Support Vector Classifier (SVC)

Linear-SVC [Ped+11] are classes that are able to performe a multi class dataset classification. The classifier is identical to SVC and by using kernel = 'linear' parameter and it is therefore flexible while choosing of penalties, loss functions and is well suited to a large number of samples. The class assist dense, sparse input and multi-class is handled by a one-on-the-rest scheme.

2.6 BeamNG

For the providence of organized inputs into a simulator, the system manages inputs and replicate steps accordingly. The BeamNG representations/setups of Road, Automobile or objects. The syntax of the BeamNG entity constructor has an appropriate and

graspable interface to manage and store the properties/attributes of the states and involving objects, as well as the moving paths which are necessary to recreate the visual scenarios. Using BeamNG software more enhanced, realistic atmospheric recreation is possible by including various objects that may include buildings, trees, sign boards and other moving or static bodies on particular positions inside simulation. Such simulations would help the testers in understanding the background of the scenarios and clarify the factors that contributed in the occurrence of the crash.

3 Literature Review

Road Event and Activity Detection: The most related research with my proposed method was publish in July 2018 in which a latest Road Event and Activity Detection (READ) [Fon+18] dataset was structure and generated from purely the perspective of an autonomous vehicle; to resolve action detection challenges in autonomous driving. READ gave researchers in computer visualization, automated cars and machine learning a huge the chance to conduct research in complications such as resolving complex (road) activities, understanding the behavior of conscious agents, forecasting both labels, their placement for future movements and events, in order to support self-governing decision making capability.

Simulation of real crashes: A system was developed by Erbsmehl [Erb09] for the simulation of behaviors of cars with or without a protective safety application, given as a crash scenario which had been retrieved from GIDAS (German In-Depth Accident Study) database. Similarly to my approach, the approach proposed by Erbsmehl aims to reconstruct realistic driving situations. However, different than my approach, it relies on the availability of sensor data which, in practice, are yet extremely hard to collect. Although, my system objectives is to generate test cases from actual car videos, and not from the actions of self-driving car in the simulated environment, so I would not include

the implementation of the simulations for the safety system, sensors of autonomous car in my Thesis.

From virtual to reality: Others proposed approaches [Zof+16][Gru+14][Ber+13] to test self-driving car in simulated/virtual environments. In these studies, the modules of autonomous cars/ADASs are demonstrated using 3D systems. Then various traffic setups were provided to the simulator, and the self-driving cars/ADASs performances and features were examined under the given situations. My system would also support this methodology, by introducing test settings for video recorded traffic accidents into the 3D simulation platform. After which, testers would be able to confirm whether the avoidance of the accidents involving autonomous car/ADAS models in similar crash scenarios.

Efficient scene understanding for intelligent vehicles: Another related approach published in 2013 was to use different high and low-level sensory cues to model uncertainties as well as contextual, spatial and semantic relations between objects [Spe+13]. Incorporating the use of spatial configuration of local vision features greatly improved the robustness of scene understanding approach. The authors also introduced a depth-first message passing scheme, which when applied to several difficult real world scenarios showed robust results and real-time efficiency [Spe+13].

Toward Driving Scene Understanding: Another applicable effort adjoining the possibility of READ was proposed by V.Ramanishka [Ram+18] in 2018, who also dealt with the behaviors and procedures of the traffic environment. The authors bound themselves to the performance of the driver besides detecting the events involving other cars.

Traffic situation assessment by recognizing interrelated road: In 2012 another research focused on sensitivity analysis on Bayesian Networks [PGE12] for controlling the recognition process on the basis of retrieve information. This led to a process in which a situation was observed in increments, concentrating on most significant sensor measurements. The proposed method was evaluated on a simulated inner-city scenario where it constantly recognized the affecting entities of each road user Instead of the classification of a whole situation; the individual vehicles were examined to a limit to which entities regulate their behavior. This approach decomposed a situation into multiple parts, each part consisting of a road user and all entities affecting its behavior. Specification of such structure was termed as a configuration defined by the participating entities and the affected entity tagged as reference entity[PE12].

A Novel Key Frame Extraction Approach: Gharbi et al. [GBZ16] proposed a basic procedure for video key frame extraction which is based on interest point description and repeatably measurement. The goal of the key frame extraction is to convert the entire video to a small number of representative images that preserve the content of the video while removing all redundancy.

Dynamic Scene Understanding for Behavior Analysis: The paper presented in 2014 determined two ways to identify the scene/state dynamically one using the abrupt movement analysis and second using the previously learned movements [BSV14]. The trajectories for training were represented as a sequence of symbols by considering crossed zones in the scene, speed, and shape. They proposed an efficient kernel-based clustering algorithm, for obtaining groups of normal routes. Investigations were conducted over three standard data sets, confirmed the effectiveness of the proposed approach.

A Deep Neural Network Architecture: Incredible work done by Paszke et al. [CKC16] on Semantic segmentation Using ENet architecture. The semantic segmentation algorithm is used to divide the image into meaningful parts while, at the same time linking each pixel in the input image to a class label road, car, person, bus etc. Paszke et al. [CKC16] trained the dataset on the Cityscapes Dataset [Cor+16], a semantic, instance-wise, dense pixel annotation of 20-30 classes.

A Survey of Current Trends in Autonomous Driving: Guillaume et al. [Bre+17] conduct a survey of the Simultaneous Localization And Mapping (SLAM) field while considering the recent evolution of autonomous driving. They present the limits of classical approaches for autonomous driving and discuss the essential criteria use for this kind of application.

Ontology based context awareness for driving assistance systems: Armand et al. [AFI14] implemented an Ontology to create an information base for ADAS for the identification of the setting of the roads, how the observed object on the road are expect to act, and what are the concerns/limitations of these actions; hence the ability of the ADAS system regarding context awareness had been improved. Also, Zhao et al. [Zha+15] designed an Ontology for ADAS that contains information about street map, traffic guidelines, vehicle characteristics and controls. The mentioned work is the knowledge bases which were involving the specification of the potential actions and vehicle properties which had been traveling/moving on the road could be applied on my system to specify the activities and attributes of actors involved in the video.

I believe that besides being fully aware of activities done by supplementary road users it is also required that an self-directed car navigates successfully even in multifaceted road situations. Thus my approach is to test the self driving car ability in virtual world by utilizing the simulations created from real world videos.

4 Methods

This section is divided in two steps, Step-1 includes the setup for creating the simulation using GPS information, the conversion of GPS coordinates into cartesian coordinates, extracting road geometry, normalization of car coordinates and the simulation with road and ego car. Step-2 starts by detecting vehicle in each video frame, creating bounding box to get the pixel information of each frame, finding the distance from the detected vehicle travelling in front of the ego car and speed estimation of detected vehicle.

4.1 Step-1

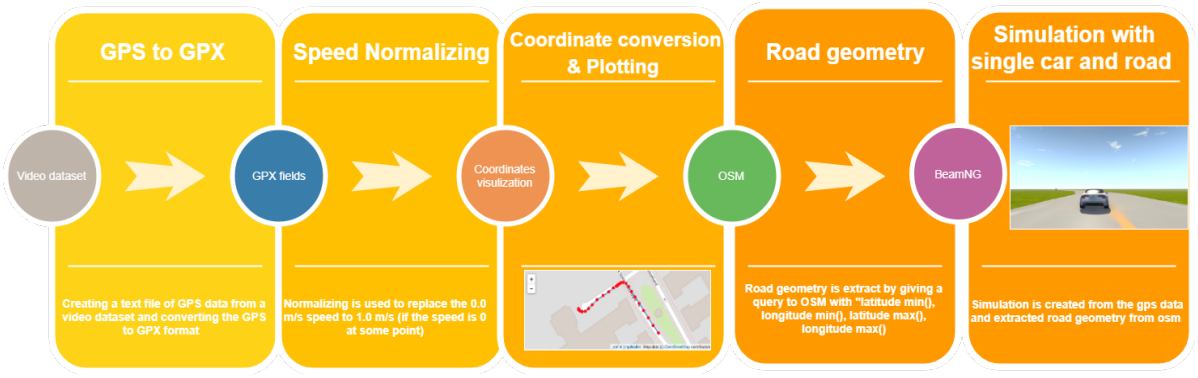

Figure 4.1: Overview of Step-1

4.1.1 Converting GPS format to GPX

In order to extract the selected information from raw dataset, I had created a text file to increase the usability of the records i.e. the GPS Coordinates. After which I uploaded the self-created text file on an online GPS visualizer that ensures customizability for the conversion of the provided data into the GPX format automatically. The uploading of the GPS Coordinate text file on the online visualizer is only effort that is needed.

GPS Visualizer: GPS Visualizer [Sch], an extremely customizable online and free utility that generates maps and their respective profiles using the geographic data. The input is the GPS data i.e. tracks and geographic points. Additionally the street landmarks, lane numbers and drive ways or coordinates. This tool can easily visualize geographic data (scientific interpretations, instances, business sites, consumers, real estate, geo labeled photographs, etc.).

4.1.2 Visualizing the GPX File

This imported file would be further divided into customized data frames for the better depiction of the retrieved and converted data file into GPX format. This data frame would consist of coordinate mainly latitude, longitude, altitude and time which is set as index.

Time	altitude	latitude	longitude
2017-08-24 18:41:05+00:00	3.681675	40.764163	-74.083704
2017-08-24 18:41:05.999000+00:00	3.719975	40.764156	-74.083700
2017-08-24 18:41:07+00:00	3.747196	40.764155	-74.083699
2017-08-24 18:41:08+00:00	3.770329	40.764155	-74.083699
2017-08-24 18:41:09+00:00	3.816257	40.764154	-74.083698

Figure 4.2: GPX segments converted to a pandas dataframe

4.1.3 Speed Normalizing

In the speed we have values with 0.00 m/s which means when the vehicle is at stationary point at signal or traffic congestion, normalizing is used to replace the 0.00 m/s values with 1.0 m/s because the simulation software BeamNG [Bea18] does not support the zero values.

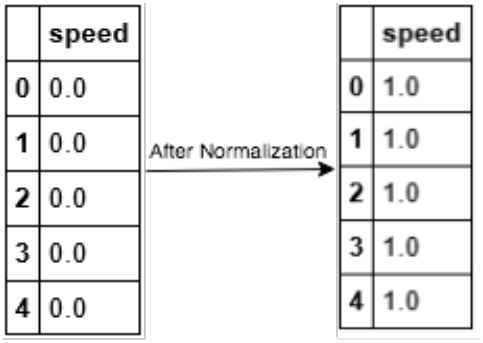

Figure 4.3: Speed Normalizing

To visualize speed at specific point I had used matplotlib plotting library

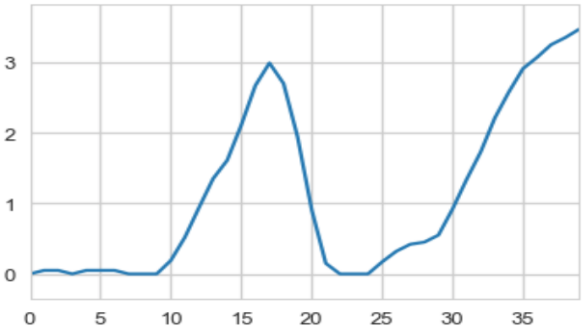

Figure 4.4: Speed at specific point

4.1.4 Plotting a Segment

I had checked visually that the extracted coordinate are correct by comparing them with the map as showed in figure 4.6 , A track on which a car travel can be seen in the following figure 4.5, on x axis is the longitude and y axis as latitudinal values extracted from GPS data.

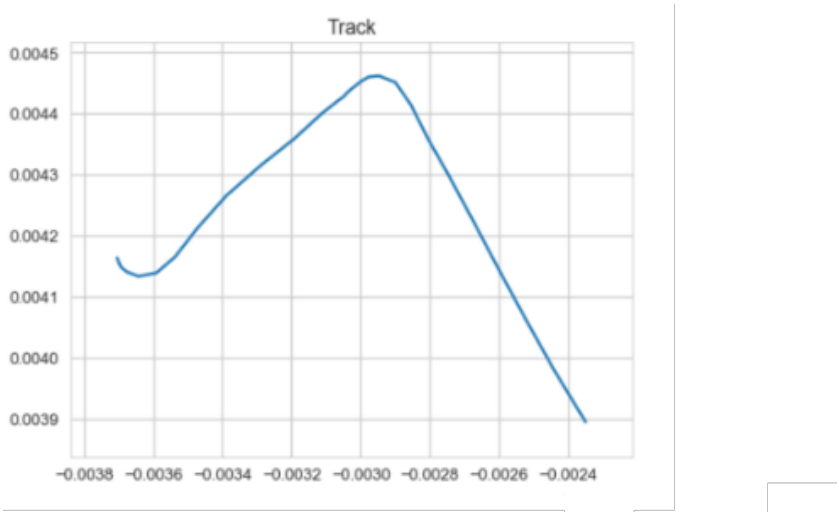

Figure 4.5: Visualizing the Track from GPS data

For plotting the track on Open street map [1] [Ope17] I had used mplleaflet which Converts Matplotlib plots into Leaflet web maps. mplleaflet is written and maintained by Jacob Wasserman [Was]. The figure 4.6 contains the red points which are nodes and the edges represent as ways.

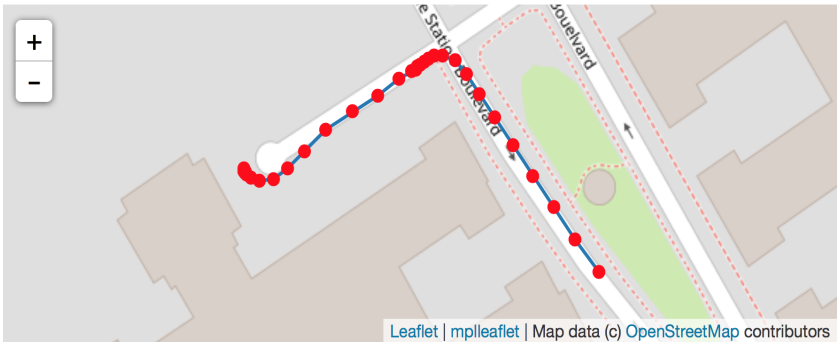

Figure 4.6: Track on OSM

4.1.5 Transformation of Geodetic Coordinate System to Geocentric Coordinate System

Using latitude, longitude, and altitude is quiet common to locate the objects and entities but we have to convert latitude, longitude to the geocentric coordinate system. For this complex conversion ("Earth-Centered, Earth Fixed") coordinate system technique in used. In equation, N is the radius of curvature in the prime vertical where φ is latitude, λ is the longitude and h represents the height. Equation 4.1 and 4.2 is described in GPS Satellite Surveying [15].

$$X = (N + h) \cos \varphi \cos \lambda \tag{4.1}$$

$$Y = (N + h) \cos \varphi \sin \lambda \tag{4.2}$$

[1] `www.openstreetmap.org`

Due to the large values of coordinates after conversion from geodetic coordinates, I scaled down the values by subtracting the minimum longitudinal value from the array of rest longitudinal Y values and same procedure goes for latitude X. The values are shown in

```
[(0.0, 1.0965067520737648),
 (0.5383472139947116, 0.6794684072956443),
 (0.6423491318710148, 0.6245012320578098),
 (0.6471538301557302, 0.6076524034142494),
 (0.7665542180184275, 0.4728237772360444),
 (0.813314181054011, 0.4378559850156307),
 (0.8834521437529474, 0.32121552247554064),
 (0.890893722884357, 0.29511984065175056),
 (0.8911345917731524, 0.2942751757800579),
 (0.8939679695758969, 0.2843392565846443),
 (1.431262553203851, 0.0),
 (2.637700537685305, 0.311376366764307),
 (5.554793786490336, 0.7146744867786765),
 (9.74481025338173, 2.0219713114202023),
 (13.668637813301757, 4.933605463244021),
 (17.961803536629304, 9.337083331309259),
 (23.69732824433595, 15.186157342977822),
 (30.7678296843078, 21.10240408591926),
 (37.687384702730924, 26.334857602603734),
 (43.57418724265881, 31.16287504415959),
 (46.73311477294192, 34.124518966302276),
 (47.745646241353825, 35.06441213656217),
 (47.850998394191265, 34.694985354319215),
```

Figure 4.7: Normalized Coordinates

4.1.6 Road Geometry

For extracting road coordinates I had used Overpass API [2] [Olb], The Overpass API is a non-customizable read-only API that aids conventional nominated fragments of the OSM map data. Performing as a database through the web where the user sends a query that enables to the API to fetch dataset that relates to the requested query. Comparatively to the standalone APIs which are boosted for editing, Overpass API is optimized for clients who requisite limited elements within a preview designated by searching conditions like location, type of objects, tag properties, vicinity, or their combinations. Overpass API aids as a backend database for many services. Overpass API consists of a stron query

[2] https://wiki.openstreetmap.org/wiki/Overpass_API

language with extensive features as compared to the former XAPI based tools. Overpass QL guide/language reference is suggested utilize features via overpass turbo, a usable Web-based frontend. For legacy apps, a compatibility layer is available that allows a leveled transition from XAPI. Apart from above mentioned features there are some constraints as well. Size of an Overpass API query outcome is only identified when the download is done. It is difficult to provide an ETA while downloading, vigorously produced files from Overpass API normally cause more time to generate and download as compared to downloading static abstracts of the same region, whenever to extract country-sized regions with data the usage of planet.osm is recommended. Overpass API is useful where amount of required data asserts the selection of data obtainable in that region.

4.1.6.1 Pre Processing:

Giving a query to Overpass API which includes (minimum latitude, minimum longitude, maximum latitude, maximum longitude) and the roads except path, track, cycleway and footway as shown in listing 4.1 from GPS data that creates a bounding box on OSM and fetch all the road properties e.g. name, type, width saved in a .txt file showed in figure 4.8.

```
result = api.query("""
    way("""+str(coords['lat'].min())+""" ,"""+str(coords['lon'].min())+
   """ ,"""+str(coords['lat'].max())+""",
    """+str(coords['lon'].max())+""")[highway~"."]
    [highway!~"path|track|cycleway|footway"];
   (._;>;);
    out geom;
```

```
7     """)
```

Listing 4.1: Query for extracting road geometry

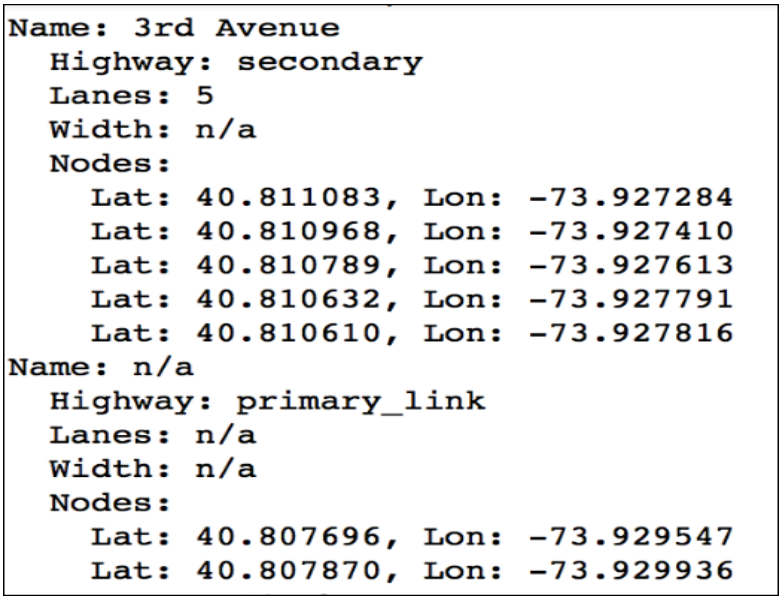

Figure 4.8: Road geometry

After extracting road properties, I convert the coordinates into cartesian coordinate as described earlier. If the width is not available in some roads than the width would be according to number of lanes on a road.

{'name': 'road_0', 'values': [(234.37463389127515, 0.0, 0, 8), (228.25871969386935, 2.018136058934033, 0, 8), (224.06
14053097088, 3.89734033215791, 0, 8), (219.60374281089753, 6.72019271645695, 0, 8), (188.52385493996553, 25.000111500
7326, 0, 8), (118.1174008352682, 66.6813445109874, 0, 8), (112.41021448839456, 70.0377518683672, 0, 8), (46.667867860
05832, 109.06968092080206, 0, 8), (13.080318236723542, 127.43561660684645, 0, 8), (10.06331306626089, 128.68653847184
032, 0, 8), (4.91699181497097, 130.81568646710366, 0, 8), (2.3932893816381693, 132.58430197648704, 0, 8), (0.0, 134.1
1111900303513, 0, 8)]}
{'name': 'road_1', 'values': [(0.0, 133.71113438531756, 0, 8), (5.230656830826774, 131.07065577898175, 0, 8), (8.6378
99504043162, 127.80465429183096, 0, 8), (12.559670817572623, 121.16409742925316, 0, 8), (21.121539307991043, 113.2152
0983427763, 0, 8), (32.89646622305736, 104.28248511068523, 0, 8), (62.70381537824869, 87.3194631151855, 0, 8), (111.7
5405022222549, 58.6680699409917, 0, 8), (119.93012480717152, 53.889443170279264, 0, 8), (138.0099885899108, 44.123751
78653747, 0, 8), (166.5728692195844, 30.757837735116482, 0, 8), (206.1010836919304, 15.980380951426923, 0, 8), (233.0
8631648495793, 5.746559513732791, 0, 8), (236.18379987054504, 4.367989110760391, 0, 8), (240.0039511404466, 2.8260724
246501923, 0, 8), (242.99623901792802, 0.0, 0, 8)]}
{'name': 'road_2', 'values': [(0.0, 0.0, 0, 8), (36.649742394685745, 27.703055485151708, 0, 8), (44.47908331803046, 3
3.62289768178016, 0, 8), (48.111879244679585, 36.37049078475684, 0, 8), (61.446287861093888, 46.81602723710239, 0, 8),
(74.44061209098436, 56.97613823786378, 0, 8), (80.26001942553557, 61.53870396036655, 0, 8), (112.51802034769207, 86.8
4418817516416, 0, 8)]}
{'name': 'road_3', 'values': [(119.71193552203476, 0.0, 0, 8), (54.45209708297625, 24.833505011163652, 0, 8), (42.033
63787266426, 30.29468031413853, 0, 8), (23.04117767745629, 39.59644697140902, 0, 8), (0.0, 53.45908282324672, 0, 8)]}
{'name': 'road_4', 'values': [(97.04084888310172, 0.3141777375712991, 0, 8), (95.02226764033549, 4.0362650053575635,
0, 8), (92.7032781848684, 6.56430259719491, 0, 8), (90.8356987957377, 8.248426357284188, 0, 8), (86.67683055251837, 1
1.224397515878081, 0, 8), (75.18310105637647, 18.27621994074434, 0, 8), (66.1535213496536, 23.39938178192824, 0, 8),
(51.630651876097545, 32.08315309043974, 0, 8), (28.955241052201018, 45.03200602252036, 0, 8), (16.866906660143286, 51
.922026502899826, 0, 8), (15.53627056023106, 52.09297447651625, 0, 8), (13.465760466409847, 51.502494130283594, 0, 8)
, (12.097357008839026, 50.69753995910287, 0, 8), (6.267319398699328, 46.049026941880584, 0, 8), (0.0617888115812093,
40.19259144924581, 0, 8), (0.0, 38.16177091188729, 0, 8), (1.3284906554035842, 37.228660533204675, 0, 8), (4.24231788
655743, 35.63173153810203, 0, 8), (35.622709470335394, 19.744059666991234, 0, 8), (36.136078600306064, 19.48330531083
0474, 0, 8), (54.702560709556565, 10.84434065874666, 0, 8), (75.02925084112212, 3.1146430103108287, 0, 8), (80.552381
39303401, 1.2968290681019425, 0, 8), (85.89073191676289, 0.5271606370806694, 0, 8), (90.36092718364671, 0.0, 0, 8), (
97.04084888310172, 0.3141777375712991, 0, 8)]}
{'name': 'road_5', 'values': [(0.0, 0.0, 0, 8), (1.6554038231261075, 2.13844863884151, 0, 8), (2.096167331561446, 3.7
947759833186865, 0, 8), (2.326318997424096, 5.112058524042368, 0, 8), (2.8399983244016767, 6.389560044743121, 0, 8),
(4.025622805813327, 7.866233993321657, 0, 8), (6.099448719527572, 9.060872072353959, 0, 8), (7.993820154340938, 9.684
069477953017, 0, 8), (10.158300598850474, 10.037458364851773, 0, 8), (12.884526197100058, 10.822495107538998, 0, 8),
(16.00794240576215, 12.339093423448503, 0, 8)]}

Figure 4.9: Road coordinates

4.1.7 Simulation and controlling of ego car

After extracting the road geometry, the next step is to visualize the roads in Beamng and for creating the road we need the parameters to be define which are included in road objects i.e. (x, y, z, width of roads), road material, texture length. To generate the path for ego car we need the way points (x, y, z) represents the position, speed from one way point to another which are extracted from GPS data, rotation at first way point. Figure 4.10 is the depiction of roads created in simulation using GPS coordinate on which the vehicle would navigate.

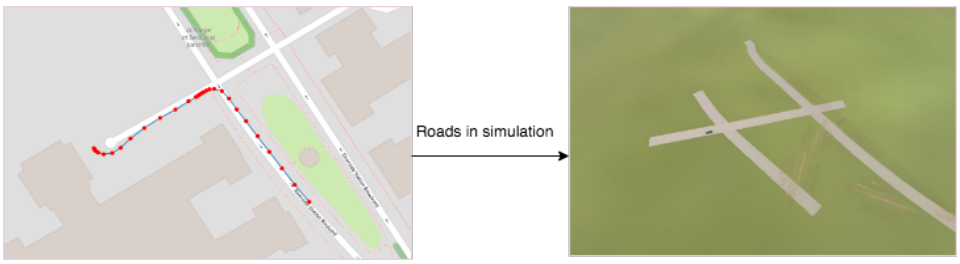

Figure 4.10: GPS points and Roads

Figure 4.11 shows the trajectory of ego car in simulation and real coordinates, it is easy to understand that the way points look similar, which shows that the translation of coordinates work perfectly.

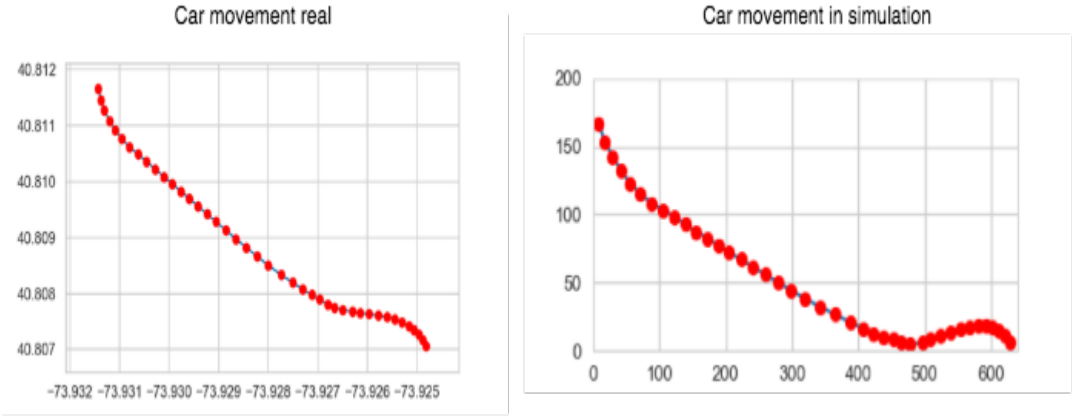

Figure 4.11: Way point real vs simulation

4.2 Step-2

Step-2 begins by detecting vehicles using the HOG object detection technique, creating a bounding box around the vehicle. Estimating speed and distance using image transformation, bounding box velocity, generating path for other vehicles with reference to

ego car coordinates and final simulation with road and the vehicles traveling in front.

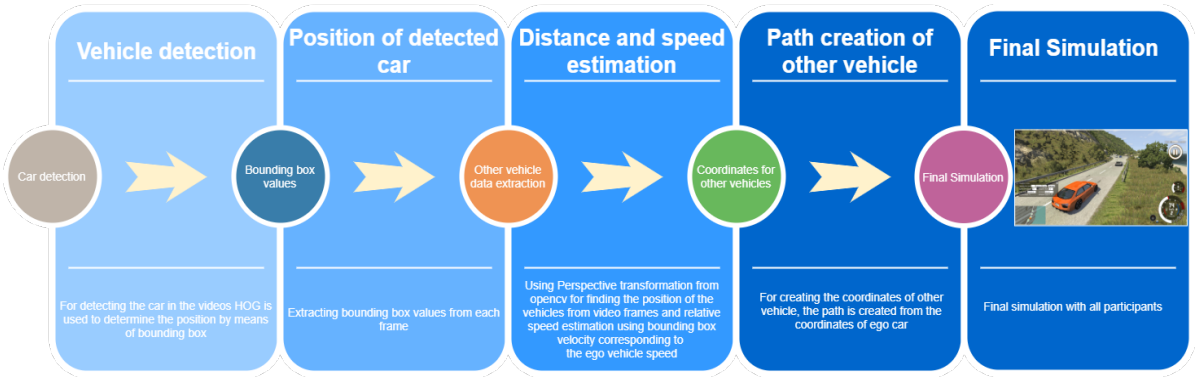

Figure 4.12: Overview of Step-2

4.2.1 Vehicle Detection

As mention in step-1 the simulation was created with ego car alone and to include other cars traveling in front of ego car, this requires as first to detect them from the frames. I utilize the object detection techniques used in [Naw15] [Taw17] and following are the steps for vehicle detection.

4.2.1.1 Object tracking using HOG

Initially all the vehicle and non-vehicle images were read using the dataset [GLU12]. Below is an example of each class of vehicles and non-vehicles:

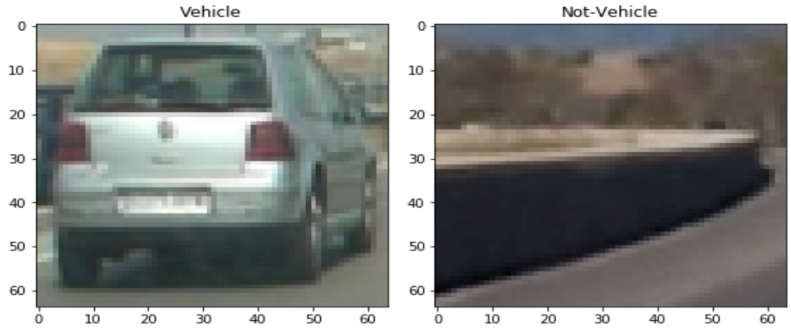

Figure 4.13: Vehicle and Not-Vehicle

The process for calculating HOG starts with image which is divided in a number of cells and for each cell the orientations are binned. Initially, we create features for car and not car. The process starts by computing the image gradient in x and y direction, finding magnitude of gradient and the direction of the gradient. The direction of gradient and magnitude is further employed to histogram of gradients with bins. Normalization is done on the histogram vector within a block.

Dividing each element of the vector give us the normalized vector. The final feature vector is calculated from entire image patch and one vector is formed by concatenating the vectors. In this stage, various colored spaces (YCrCb, RGB, HSV, LUV) and hog parameters have been explored such as (orientations, pixels in each cell and cells per block). After which images were obtained randomly from the two classes i.e. vehicle and non-vehicle distinctively and exhibited likewise to demonstrate the `skimage.hog()` [Wal+14] output resemblance. The color space and HOG parameters set of 8 orientations and (8,8) pixels per cell, (2, 2) cells per block using the YCrCb color channel. Numerous combinations of parameters were randomly tested after which color space of 8 pixels per cell with 18 orientation directives for the gradients had been chosen and a 16x16 spatial binning size and 2 cells per block setting was considered for best results.

4.2.1.2 Training the Classifier

The classifier is trained on vehicle and not vehicle features with parameters (color space, Spatial binning dimensions, histogram bins, orientation for HOG, pixel per cell, cells per block, hog channel, spatial features, histogram feature, hog feature). The training was achieved through the supervised classifier linear SVC [3] (Support Vector Classifier) [Ped+11] using a combined feature vector features of HOG, spatial features, and a histogram of color features across all three channels of YCrCb [Jia+13]. The special features were altered to resize the image to 16x16 pixels by utilizing the resultant color values for each pixel. The three featured vectors were combined and normalized for every single training image. Training images were further categorized as either each containing of a car or not. Moreover, Linear SVC was trained with a ratio of 80% of samples, the resulting 20% were used to validate the results. The accuracy in correspondence of the validation set was 98%.

Classifier	Accuracy
Linear-SVC	0.98

Table 4.1: Linear-SVC test accuracy

4.2.1.3 Selecting an image region to initiate search

This step started off by utilizing a sliding window approach, the features are used to search a vehicle for each region were calculated by creating a function with parameters (image, starting position in Y direction, stop in y direction, scaling, classifier (svc), scaling in x, orientation, pixel per cell, histogram bins, cell per block, spatial size) that can extract features using hog sub-sampling, It can make predictions and evaluated

[3] `https://scikit-learn.org/stable/modules/generated/sklearn.svm.LinearSVC.html`

against the trained model. The technique to create a window of subset of the image was achieved, after which it was moved through some standard offset, often overlapping the previous window. A trade off was being observed in between the accuracy and time, as many of the windows would be much expensive to evaluate.

Figure 4.14: Selecting a region to search

After this, another efficient approach was taken that involved the extraction of features once from a subregion of the image below the horizon. Furthermore it was subsampled with the region by overlaying windows. Each window was scaled over different factors, so that multiple box sizes could be tested efficiently.

4.2.1.4 Obtained Initial Results

Five scales using YCrCb 3-channel HOG features plus spatially binned color and histograms of color in the feature vector were searched. Below is the image for the depiction of the above process:

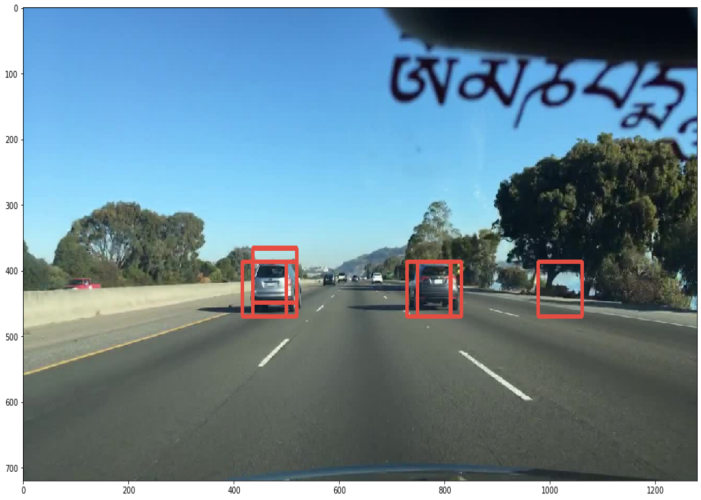

Figure 4.15: Initial Vehicle detection results

Some false positives occurred occasionally, bounding box on the boundary of the road as shown in figure 4.15.

4.2.1.5 Filtering the bounding boxes

To remove noise created by overlaped bounding boxes with different scaling factors from the list of candidate boxes, a heat map was created. In order to identify individual blobs in the heat map `scipy.ndimage.measurements.label()` [Wal+14] was used. Each bounding box had a single vote, combined with other boxes to increase the likelihood of car detection. Threshold operation was performed by setting the Zero pixels below the threshold `heat_map[heat_map <= thres] = 0`. This was resulted in a non-detection of a car when only one box was found several times as shown in the following figure 4.16.

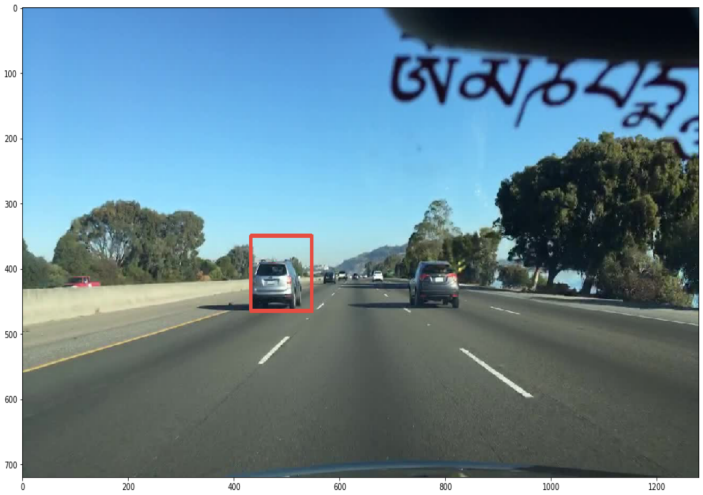

Figure 4.16: Misdetection of a car on the right

Below in figure 4.17 presenting the heatmap from a series of frames obtained through different dash cam videos, the results and the bounding boxes overlaid the last frame of video.

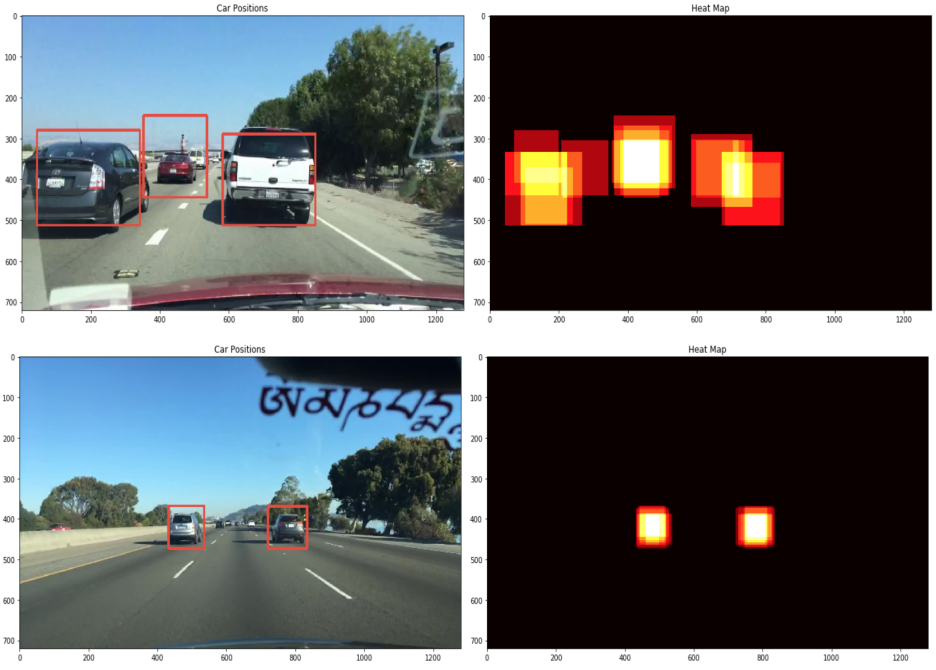

Figure 4.17: frames and their corresponding heatmaps

4.2.1.6 Region Stabilization

For coherence, a running list of bounding boxes over multiple frames was created for which each of the average color and dimension of the box were tracked. When a new candidate box on each frame was obtained it was matched with a previous box by position and dominant image color, the interpolation towards the new box was implemented. The determination of velocity in X and Y was included that was supposed to update the center of box in each frame. The combination of the above mentioned policies flattened the position and dimensions of the car bounding boxes.

After doing the experiments, the settings are applied on the videos with focus on one car travelling in front that gives the pixel values of the bounding box on each frame which are further used to find the distance and speed.

4.2.2 Finding the Position of Detected Vehicle

To find out the distance between the ego car and detected car by means of bounding box we consider to calculate the distance using geometrical image transformation methods.

For the transformation of image the concept geometrical transformation of 2d images are used, which states that the image content will not be changed but it will deform the pixel grid and further map the deformed grid into the image destination. The mapping takes place in the opposite order, from destination to source.

Transformation of perspective-projection is important in computer vision and is commonly used for gain desired computer screen presentation. It can change the sight to make it more realistic. In transformation, there are two types of projection of views that

are parallel and perspective projections. Parallel projections are the image transformation with the coordinate location into parallel lines. The transformation of the images into the view plane along lines converging to a point [J F] and a more realistic view.

I had use `cv2.getPerspectiveTransform()` fuction from [Bra00] which states that the transformation of the viewpoint will be calculated and where the pixel will shift after the transformation from the original image with the coordinates (u, v). The point with the coordinates (u_w, v_w) would be the destination of that pixel on the warped image as showed in figure 4.18. The calculation of the new position is formalised as:

$$\begin{bmatrix} u_w \\ v_w \\ 1 \end{bmatrix} = sH \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} \tag{4.3}$$

In equation 4.3 H is the homography matrix that maps the points in one image to the corresponding points in the other image.

After the calculation of perspective matrix next is to determine the center point of bounding box in x and y directions. In equation 4.4 X_c is the center point in x and in equation 4.5 Y_c represents the center point in y direction. $x1$, $x2$, $y1$, $y2$ are the bounding box values in pixels.

$$X_c = \frac{x2 - x1}{2} + x1 \tag{4.4}$$

$$Y_c = \frac{y2 - y1}{2} + y1 \tag{4.5}$$

In our case we calculate the perspective matrix by taking an image as input, giving the source, destination point `M = cv2.getPerspectiveTransform(src, dst)` and warped the image `using warpPerspective()` from opencv which will straighten effect of perspective transformation.

4.2.3 Distance estimation

After calculating the center points and Perspective matrix we can find the pixel pair in transform space x and y by using transform matrix with parameters pixel in x, y,perspective matrix, for finding the position of the vehicles from video frames I had straighten the image from bounding box center till the hood of car as showed in figure 4.18, there is a direct correlation between the pixel position and distance in meters, so the distance between the calculated position of the midpoint which is the pixel pair in y direction denoted as p_y and pixel per meter P_m, w_s represents the warped size will give us the distance between the mid point and the image destination represents the distance between the bounding box and the ego car.

$$d = w_s - \frac{p_y}{p_m} \tag{4.6}$$

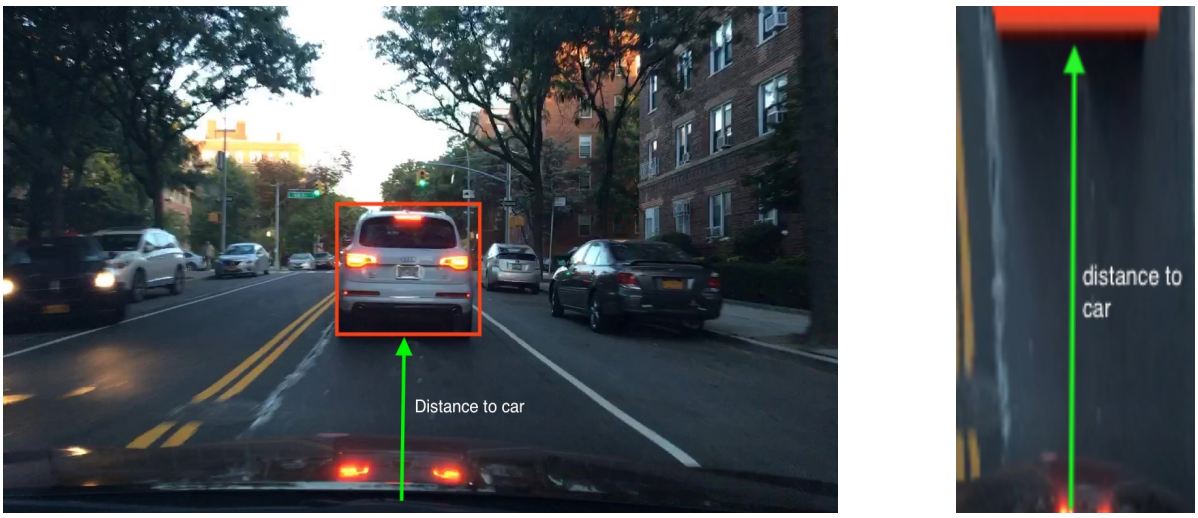

Figure 4.18: Detected car and warped image

The distance is estimated on every frame e.g. if the time of video is 20 seconds with fps 25 then in 1 second 25 times the distance will be calculated.

4.2.4 Speed Estimation

By looking how that distance changes from frame to frame, relative speed estimation was included using the bounding box velocity corresponding to the current vehicle speed. The vehicle speed in each frame is calculated using the vehicle position in each frame, the vehicle moving in consecutive frames gives us the velocity estimation in y direction vel_e and which is calculated as:

$$vel_e = p_y - Y_c \tag{4.7}$$

Where p_y is the pixel pair in y direction and Y_c represents the center point of bounding box. After calculating the vel_e, we can determine the speed by knowing how far is one

pixel and this is considered by warped size multiply the number of pixel in meters p_m. As the ego car speed e_s is known at every frame we can calculate the relative speed by the equation:

$$V = e_s + (vel_e + p_m) \tag{4.8}$$

Frames	ego_speed	other_car_speed	distance(m)
0	12.33	12.330000	4.80
1	12.33	12.330000	4.71
2	12.33	12.324922	4.70
3	12.33	12.327295	4.70
3	12.33	12.327295	4.80

Figure 4.19: Data frame of resultant values

The extracted values of speed and distance in each frames are merged by taking average over frame per second and further used for creating simulation.

4.2.5 Generating Final Simulation

To generate the simulation with leading car in front, we need to calculate the way points for the car travelling in front of the ego car. For this, we use ego car way points which are extracted from GPS data and than converted to cartesian coordinates. The coordinate y represents the front direction of ego car, z is static to 0 as we assume that the ground is flat, x represents the rotation of the vehicle and in our case we are adding the distance in y coordinate of ego car i.e. y + distance at frame 1 which was extracted in meters in the previous section.

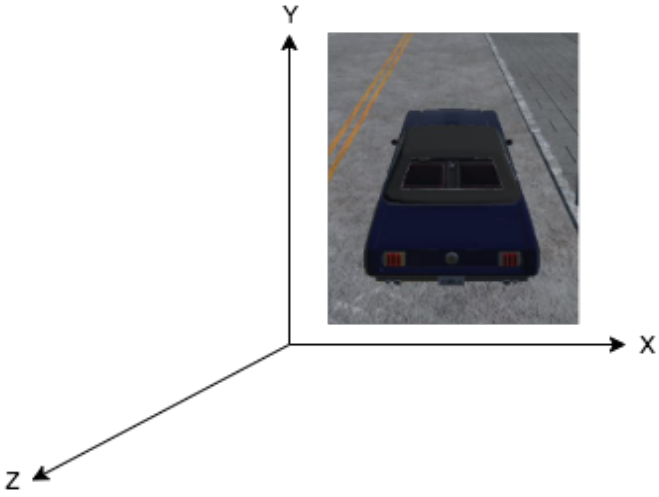

Figure 4.20: Coordinate understanding

After extracting the distance, speed and other car coordinates which are essential for creating the simulation with multiple vehicles, Ego car coordinates and roads were used from step-1, the way point and speed for the other vehicle are parsed to beamNG. Figure 4.21 represents the frame from video and simulation.

Figure 4.21: Simualtion and video frames

5 Evaluation

In this section, the Approach through which accurate results and structures are discussed, by which final results have been achieved. Firstly, I had extracted key frames every second from the video, then manually annotate key frames to create the ground truth dataset. Secondly, I had generated the simulation, run the simulation and extract the perfect pixel annotated images from the simulation every second. I draw the bounding box from the simulation frames and establish the predicted dataset. To compare bounding boxes between corresponding frames for each test video, I had used IoU to measure the accuracy between the bounding boxes.

5.1 Data set

For this research I have used the Berkeley deep drive data set [1] [Yu+18] due to the following supporting aspects:

- The dataset contains the data of almost 100,000 HD videos that cover over 1,100 hours of drives at different times of days; changing weather plus driving states.
- Videos consists Geophysical satellite locations, time splits and IMU data.
- Videos are annotated with well-defined instance levels/ frames including traffic signals,

[1] `https://bdd-data.berkeley.edu/`

traffic landmarks, people, vehicles and lane markings within the drive able area and predicable typical driving decisions from 100,000 images.
For this project I have used data columns namely: latitude, longitude, speed, timestamp.

5.2 Experimental Settings

The experiments are conducted by using gaming laptop running with windows 10 and equipped with MSI gp62, Intel Core i5-6300HQ CPU (2.30GHz), 8 GB of memory, and an NVIDIA Geforce GTX 960 GPU. In this implementation, simulation are generated by using BeamNGpy [2] [Bea18] which is an official library providing a Python interface to BeamNG.research, the research-oriented fork of the video game BeamNG.drive. It allows remote control of the simulation, including vehicles contained in it, Vehicles and the environment can be equipped with various sensors that provide simulated sensor data such as a camera feed, with options for depth values and pixel-perfect semantic annotation or a simulated Lidar sensor.

5.3 Evaluation pipeline

The focus was to evaluate the similarity and likeliness between the real video and simulation. Firstly, a metric for the evaluation was set to compare the frames at every second by using intersection over union (IOU). Originally, The berkeley deep drive dataset [Yu+18] provided the ground truth of 1 frame in each video which cause hindrance to calculate the similarity in between the remaining frames, to eliminate this problem a Scalable tool [3] [Yu+18] was used to add the frame annotations of the key frames which aid the

[2] https://github.com/BeamNG/BeamNGpy
[3] https://www.scalabel.ai/

calculation of the remaining bounding boxes. This enabled the creation of ground truth, Frames are sampled at FPS = 1 which means that 1 frame per second. The duration of the videos lies between 20 and 40 seconds, frames will be annotated and further used for evaluation. In figure 5.1 a car with bounding box containing labls x1, y1, x2, y2 can be seen.

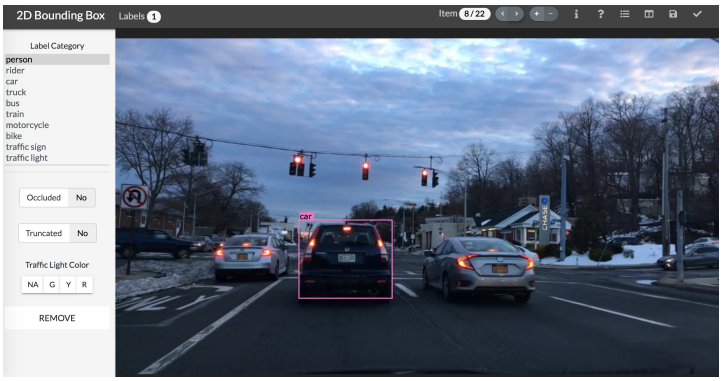

Figure 5.1: data annotation

5.3.1 Annotated key frames rendering

After the data annotation and labeling of bounding box for each key frames' ground truth. Using beamNG, by attaching the camera on the vehicle and setting the camera parameters showed in table 5.1. Annotated key frames where rendered with corresponding to frame per second. The camera direction, position was adjusted manually while the focus of view and resolution are adopted from the Berkeley deep drive dataset.

Position (x, y, z)	Direction (x, y, z)	Resolution	Focus of view
(-0.25, 0.78, 1)	(-0.25, 10, 1.1)	1280 * 720	60

Table 5.1: Camera parameter

The resultant annotated key frame extracted from beamNG is showed in figure 5.2.

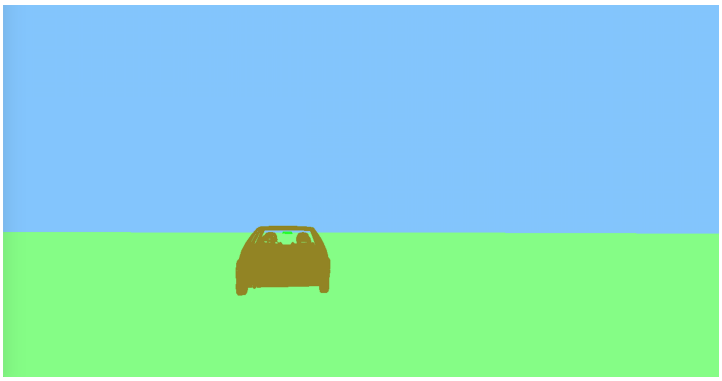

Figure 5.2: annotated frame

5.3.2 Bounding box creation on annotated frames

The contours was used to indicate the curved line that represents the boundary values and corresponding intensity, The annotated frame was converted into gray scale and threshold was applied. The function `cv2.findContours()` [Bra00] was used to draw the 2d bounding boxes surrounding the detected car with values x1, x2(width), y1, y2(height) following the csv format.

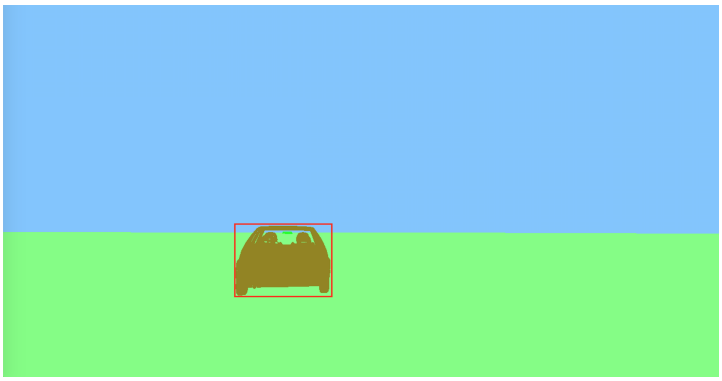

Figure 5.3: bounding box on annotated frame

Moreover, with the obtained ground truth and predicted values, for evaluation IOU metric [Rez+19] was selected which would help in the differentiation of true and false prediction and eventually help in acquiring precision, recall.

5.3.3 Intersection Over Union:

The calculation of Intersection Over Union (IOU) is based on the jaccard Index which measures the difference between two bounding boxes. It needs a bounding box of ground truth b_{gt} and a bounding box that is predicted b_p. The IOU measure the overlapping area between the b_{gt} and b_p, divided by the area of union between b_{gt} and b_p.

$$IOU = \frac{area(b_p \cap b_{gt})}{area(b_p \cup b_{gt})} \tag{5.1}$$

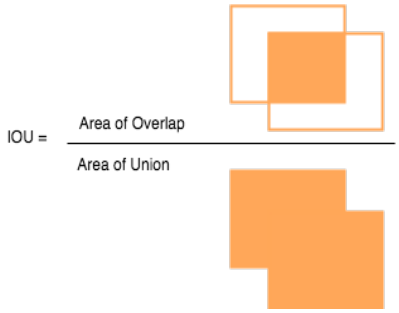

Figure 5.4: IOU Concept

The resultant value of IOU i.e. the threshold has to be defined as 0.5 which is considered a good prediction according to the metric [Rez+19] thus, result were calculated frame by frame. Figure 5.5 illustrates the results which was required to differentiate the true and false predictions for which metric IOU was implemented.

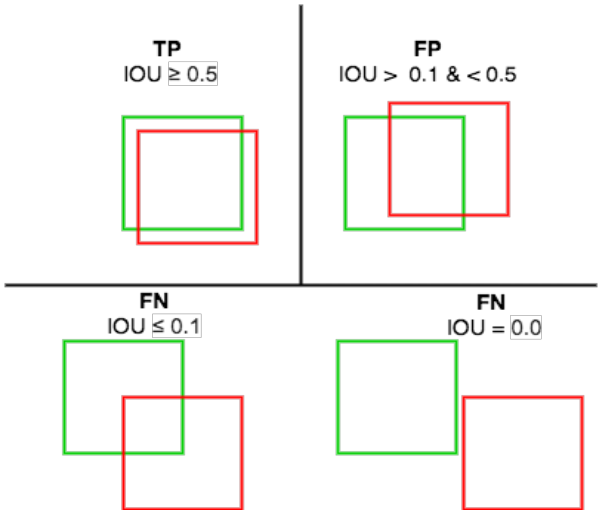

Figure 5.5: Illustration of IOU

True Positive (TP): detection with IOU $\geq$ threshold (0.5)

False Positive (FP): detection with IOU $>$ 0.1 & $<$ threshold (0.5)

False Negative (FN): detection with IOU $\leq$ 0.1

True Negative (TN): Is not applicable. It would indicate a misdetection that has been corrected. There are many possible bounding boxes in the object detection task that should not be detected in an image, so TN would be all possible bounding boxes that have not been detected correctly. That's why the metrics don't use it.

Precision:

The capacity of a model to extract the relevant objects. It is the percentage of accurate positive predictions and is given by:

$$Precision = \frac{TP}{TP + FP} \tag{5.2}$$

Recall:

Recall is a model's ability to find all specific cases (bounding boxes for all ground truth). The ratio of accurate prediction over all significant items in ground truth and is given by:

$$Recall = \frac{TP}{TP + FN} \tag{5.3}$$

F1 score:

The F1 score is a weighted average of the precision and recall and when there is no TN, the f1-score is considered as the accuracy. It shows the ratio between precision and recall. The formula for the F1 score is:

$$F1 = 2 * \frac{precision * recall}{precison + recall} \tag{5.4}$$

5.4 Evaluated Results

In figure 5.6 where x-axis denotes the number of simulations and y-axis denotes the IOU resultant values, here the average value of IOU is above the average threshold 0.5 which confirms that the predicted bounding boxes are generally closed to the ground truth. The outliers below 0.1 represent the false negative and the values greater than 0.1 and below 0.5 are the false positive predictions.

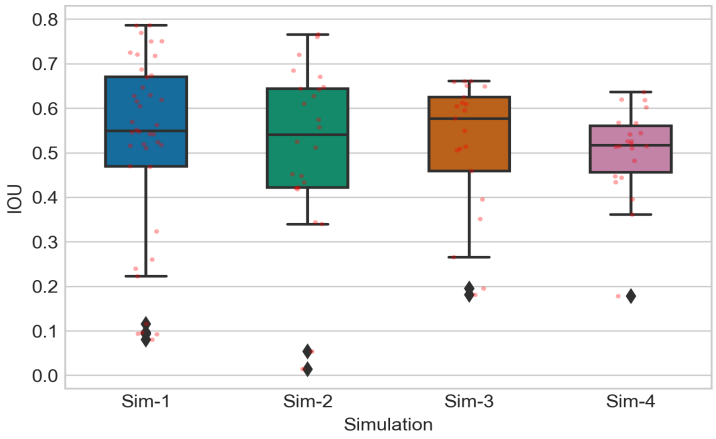

Figure 5.6: IOU Results

The table 5.2 illustrates precision, recall and f1-score for simulations where it is shown that sim-1 as the highest precision value 0.78 and sim-4 has lowest precision value 0.68 which concludes that in sim-4 the ratio of positive prediction are less as compared to sim-1. sim-2 and sim-3 has precision 0.75, 0.71 respectively. The recall of sim-3 and sim-4 is close to perfection at 1.0 which means there is no false negative in the predicted values. whereas, sim-1 and sim-2 achieved stable recall values 0.87 and 0.78 respectively. sim-3 has the highest f1-score of 0.83 whereas sim-1, sim-4, sim-2 has declined f1-score 0.82, 0.81, 0.76 respectively.

Simulations	**Precision**	**Recall**	**F1-Score**
Sim-1	0.78	0.87	0.82
Sim-2	0.75	0.78	0.76
Sim-3	0.71	1.0	0.83
Sim-4	0.68	1.0	0.81

Table 5.2: Table of Results

5.4.1 Simulation processing time

In figure 5.7 the y-axis represents time in seconds and the x-axis shows the modules involved in creating the simulation, *g_f_c* stands for gpx file creation from raw dataset, coordinate creation represents *c_c*, *r_e_o* is the method of road extraction from OSM, and *bb_s_d* stands for vehicle detection, speed and distance estimation process. *bb_s_d* took comparatively additional time than other modules. The estimation of bounding box and by analyses of boxplot in figure 5.7 the time taken by *bb_s_d* was in between 500 and 550 seconds.

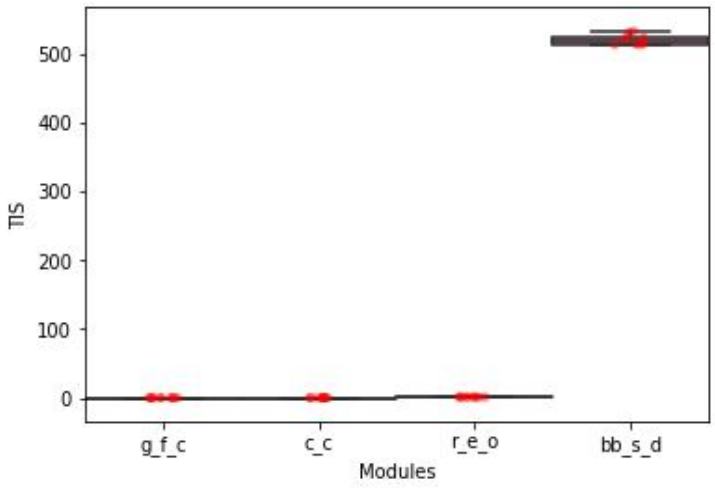

Figure 5.7: Procesing time of modules

In figure 5.8, The processing time involved in the creation of a simulation and efficiency being calculated by iterations $n = 10$. Through the obtained values, the occurrence of the outlier had been observed. During the first iteration the experimental time was much high as compared to the other iterations. Visualization of the boxplot comprehend that interquartile range (IQR) was in between 425 and 445 seconds for majority of iterations.

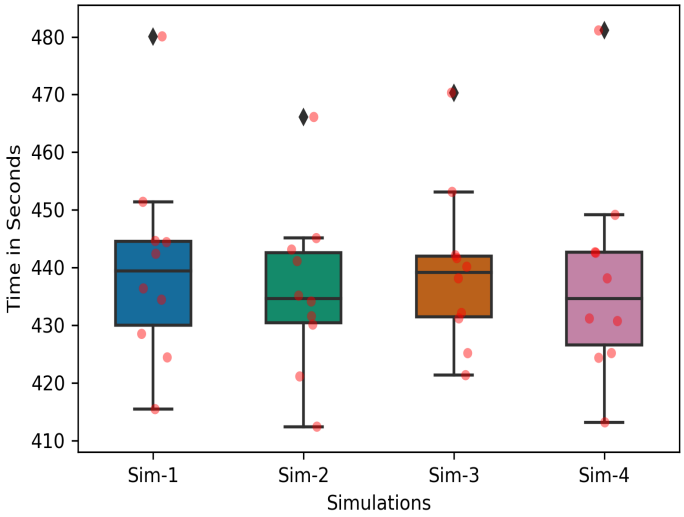

Figure 5.8: Procesing time for generating a simulation

6 Conclusion and Future Work

Simulation for self driving cars is an important aspect for achieving fully autonomous level. The evaluation of proposed research was primarily based on simulations and evaluation results shows that the proposed method is efficient and produce reasonably accurate simulations. The proposed method is viable and pave the way for future self-driving cars testing by providing a reusable feature set to develop simulations automatically. This simulations could facilitate the tester to generate test cases and examine the behavior of the autonomous car in real world scenarios which are replicated in form of simulations and would help the tester to conclude whether the self driving car functioned properly and took appropriate decisions that were expected in that certain conditions.

The object detection technique used in this research works fair enough but it can be improved by using the YOLO (You only look once) [Red+15] and the way of calculating distance from single camera images to objects by assuming that the ground is flat. Under this hypothesis, two-dimensional data from a camera image is used to model the three-dimensional universe. In the real world, this is not applicable as the infrastructure of roads is not flat there are inclined and uneven surfaces that would produce inaccurate results. The utilization of the recent research [CVI19] enabled us to conclude that the training of the deep neural network on lidar and sensor measurements provided by open source dataset for self driving cars could be executed, which would give the efficient

predictions of vehicle distance from the ego car and in the result accurate simulation will be generated.

A Code

A.1 Step-1

```
import json
from time import time
current_time_ms = lambda: int(round(time() * 1000))
start_ts = current_time_ms()

# Reading the BDD data to extract only GPS values
with open("D:/Beamng_qazi_data/Thesis_final_code/
    Final_evaluation_videos/vid-3_evaluation_best/88d118af-f4356398.
    json") as json_file:
     data = json.load(json_file)
# Creating a list
finalGps = []
for gps in data["locations"]:
    temp = []
# Getting only the values of lat, long, speed, timestamp
    for i in ['latitude','longitude','speed','timestamp']:
        temp.append(gps[i])
    finalGps.append(temp)
# Creating .txt file and multiplying the speed values d[2] * 3.6
    because the speed is measured in mps and GPX visualizer need speed
```

```
    in kph"
for d in finalGps:
    with open('D:/Beamng_qazi_data/Thesis_final_code/
    Final_evaluation_videos/vid-2_evaluation/vid-1.txt','a') as f:
        de = '{},{},{},{}'.format(d[0],d[1],d[2] * 3.6,d[3] )
        f.write(de)
        f.write('\n')
f.close()
with open('D:/Beamng_qazi_data/Thesis_final_code/
    Final_evaluation_videos/vid-2_evaluation/vid-1.txt', 'r') as f:
    a = f.read()
# Now writing into the file with the prepend line + old file data
# This is the final .txt file which will be the input in GPS visulizer
    " https://www.gpsvisualizer.com/convert_input" to create the GPX
    file
    with open('D:/Beamng_qazi_data/Thesis_final_code/
    Final_evaluation_videos/vid-2_evaluation/vid-1_1.txt', 'w') as f:
        f.write('latitude, longitude, speed, time' '\n' + a)
end_ts = current_time_ms()
ts = end_ts - start_ts
print( "processing_time:",ts )
```

Listing A.1: json to GPX

```
import overpy
import numpy as np
from tqdm import tqdm
import math
import gpxpy
import pandas as pd
from time import time

current_time_ms = lambda: int(round(time() * 1000))
```

```
10
11 start_ts = current_time_ms()
12
13 # opening the gpx file to find out max latitude,min latitude, max
       longitude, min longitude
14 with open('Final_evaluation_videos/vid-3_evaluation_best/vid-3.gpx') as
       fh:
15     gpx_file = gpxpy.parse(fh)
16
17 segment = gpx_file.tracks[0].segments[0]
18 coords = pd.DataFrame([{'lat': p.latitude,
19                         'lon': p.longitude,
20                         'zalt':p.elevation,
21                         #'speed': p.speed,
22                         'time': p.time} for p in segment.points])
23 coords.set_index('time', drop=True, inplace=True)
24
25 #Fetching Road geometry from osm using overpass API
26 api = overpy.Overpass()
27
28 # fetch all ways and nodes by giving (minimum lat, minimum long,
       maximum lat, maximum long)
29
30 result = api.query("""
31     way("""+str(coords['lat'].min())+""" ,"""+str(coords['lon'].min())+
       """ ,"""+str(coords['lat'].max())+""",
32     """+str(coords['lon'].max())+""")[highway~"."][highway!~"path|track
       |cycleway|footway"];
33     (._;>;);
34     out geom;
35     """)
36
```

```
roads = []
print('Process started...')
for way in tqdm(result.ways):
    road = []
    print("Name: %s" % way.tags.get("name", "n/a"))
    print("  Highway: %s" % way.tags.get("highway", "n/a"))
    print("  Nodes:")
    for node in way.nodes:
        print("    Lat: %f, Lon: %f" % (node.lat, node.lon))
        road.append([float(node.lat), float(node.lon)])
    roads.append(road)

print('Process finished.')
print(roads)


##  this code  will save the coords in final.txt, it is used to convert the Geodetic coordinates to geocentric

coords_1 = roads

# =======================================================
## Run this code which will save the coords in final.txt, it is used to convert the Geodetic coordinates to geocentric
# If there is no altitude remove from the function
#def coordinate_conversion(lat, lon, alt):
# If altituide is not available commit below line
def coordinate_conversion(lat, lon):
    radian_latitude = lat * (math.pi / 180.0)
    radian_longitude = lon * (math.pi / 180.0)

    """https://en.wikipedia.org/wiki/World_Geodetic_System#WGS84"""
```

```
67      b = 6378137.0  # it is a semi major axis
68      fi = 298.257223563 # flatten
69      g = 1 / fi # flatten
70      e_2 = 1 - (1 - g) * (1 - g) # calculating semi minor axis
71      t = b / math.sqrt(1 - e_2 * math.sin(radian_latitude) * math.sin(
        radian_latitude))
72
73      x = (t + 0) * math.cos(radian_latitude) * math.cos(radian_longitude
        )
74      y = (t + 0) * math.cos(radian_latitude) * math.sin(radian_longitude
        )
75
76      return x, y
77  coord_cart_x = []
78  coord_cart_y = []
79
80  def run_test():
81      calc_roads = []
82      name = "road"
83      count = 0
84      for road in coords_1:
85          values = []
86          for pt in road:
87              x, y = coordinate_conversion(pt[0], pt[1])
88              values.append([x, y])
89          pass
90          obj = {
91              'name': "{}_{}".format(name, count),
92              'values': values
93          }
94          calc_roads.append(obj)
95          # print(obj)
```

```
96          count = count + 1
97      pass
98
99      np.save('calc_roads_vid_2_2.txt', calc_roads)
100
101 run_test()
102
103 # This code will return the final road coordinates of roads in meters
104 calc_roads = np.load('calc_roads_vid_2_2.txt.npy', allow_pickle=True)
105
106
107 final_roads = []
108 for road in tqdm(calc_roads):
109     mins = np.amin(road['values'], axis=0)
110     x_min = mins[0]
111     y_min = mins[1]
112     values = []
113     for pt in road['values']:
114         x = pt[0] - x_min
115         y = pt[1] - y_min
116         values.append((x, y, 0, 18))
117     obj = {
118         'name': road['name'],
119         'values': values
120     }
121     print(obj)
122     final_roads.append(obj)
123
124 np.save('final_roads_vid_111', final_roads)
125 end_ts = current_time_ms()
126 ts = end_ts - start_ts
127 print(ts)
```

```
128 # ================================================
129
```

Listing A.2: Road extraction

```
1  # # Running the Simulation in BeamNG
2 import sys
3 from time import sleep
4 import numpy as np
5 import beamngpy
6 from scipy import interpolate
7 from beamngpy.sensors import Camera
8 from matplotlib.pyplot import imshow
9 from PIL import Image
10 from shapely.geometry import Polygon
11 from beamngpy import BeamNGpy, Scenario, Road, Vehicle, setup_logging
12 SIZE = 1024
13
14 def main():
15     setup_logging()
16     beamng = BeamNGpy('localhost', 64256, home='D:/BeamNG')  # This is
    the host & port used to communicate over
17     scenario = Scenario('xyz_Map', 'Thesis')
18
19     vehicle = Vehicle('ego_vehicle', model='etk800', licence='Qazi',
    color='Green')
20     #vehicle1 = Vehicle('ego_vehicle2', model='etkc', licence='Qazi2')
21     road_a = Road(material='AsphaltRoad_lanes', rid='main_road',
    texture_length= '5')
22     nodes = Road_a
23
24     road_a.nodes.extend(nodes)
25     scenario.add_road(road_a)
```

```
26
27 # Importing road coordinates from final_road.npy from road_final.py
28     final_roads = np.load('final_roads_vid_1.npy', allow_pickle=True)
29     count = 0
30     # nodes = []
31     for road in final_roads:
32         # name = road['name']
33         values = road['values']
34
35         road_obj = Road(material='asphaltroad_laned_nolines',
    texture_length= '5')
36         nodes = values
37
38         road_obj.nodes.extend(nodes)
39         scenario.add_road(road_obj)
40         print(nodes)
41         count = count + 1
42         print(count)
43         if count == 20:
44                 break
45
46
47     #vehicle1 = Vehicle('ego_vehicle2', model='etkc', licence='qazi')
48     orig1 = normalised_coord
49     orig = orig1[0]
50     #print(orig)
51     speed1 = speed_normalized
52     speed12 = speed1[0]
53     #speed2 = speed[0]
54     #orig4 = orig1[35]
55
56     scenario.add_vehicle(vehicle, pos = orig, rot=(0, 0,130 ))
```

```
57     #scenario.add_vehicle(vehicle1, pos = orig2, rot=(0, 0,90 ))
58     #scenario.add_vehicle(vehicle1, pos = (45.7638938, -79.08234938),
       rot=(0, 0, 180))
59     scenario.make(beamng) # The make function of a scneario is used to
       compile the scenario and produce a scenario file the simulator can
       load
60
61     path = list()
62     for i in range(len(orig1)):
63         orig = orig1[i]
64         speed12 = speed1[i]
65         #speed2 = speed[0]
66         pos = ( orig[0],
67
68                 orig[1],
69
70                 0)
71         speed = speed12
72         print(pos)
73         print(speed)
74         #print('speed')
75         #print(speed)
76         node = {
77                  'pos': pos,
78                  'speed': speed
79        }
80         path.append(node)
81
82     bng = beamng.open(launch=True)
83     try:
84         bng.set_deterministic()
85         bng.set_steps_per_second(28)
```

```
86          bng.load_scenario(scenario)
87          bng.start_scenario()
88          vehicle.ai_set_line(path)
89
90          while True:
91              bng.step(28)
92
93      finally:
94          bng.close()
95
96          x = [p[0] for p in orig1]
97          y = [p[1] for p in orig1]
98          # plt.axis('square')
99          plt.ylim(0, 200)
100         plt.plot(x, y, )
101         plt.plot(x, y, 'ro')
102         plt.show()
103
104
105
106 if __name__ == '__main__':
107     main()
```

Listing A.3: Simulation generating code

A.2 Step-2

```
1 import cv2
2 import numpy as np
3 import glob
4 import time
5 from skimage.feature import hog
```

```
import matplotlib.image as mpimg
import matplotlib.pyplot as plt
from sklearn.svm import LinearSVC
from sklearn.preprocessing import StandardScaler
from skimage.feature import hog
from lesson_functions import *
from sklearn.cross_validation import train_test_split

# Reading vehicles  and notvehicles
vehicles = glob.glob('vehicles/**/*.png', recursive=True)
not_vehicles = glob.glob('non-vehicles/**/*.png', recursive=True)

print('No. of cars: %d ' % len(cars))
print('No. of not-cars: %d ' % len(notcars))

# Defining a function that will return HOG features and visualizations
def getting_hog_feature(img, o, p_i_c , c_i_b,
                        vi=False, feature_vec=True):
    if vi == True:
        feat, hog_imag = hog(img, orientation=o,
                                  pixel_p_cell = (p_i_c, p_i_c),
                                  cell_per_block = (c_i_b, c_i_b),
                                  trans_sqt = True,
                                  visualise=vi, feature_vector=
   feature_vec)
        return feat, hog_imag
    # Rather calling by one output
    else:
        feat = hog(img, orientation=o,
                        pixel_p_cell=(p_i_c, p_i_c),
                        cell_per_block=(c_i_b, c_i_b),
                        trans_sqt=True,
```

```
                        visualise=vi, feature_vector=feature_vec)
        return feat
# Color converting function
def converting_color(img, conver='RGB2YCrCb'):
    if conver == 'RGB2YCrCb':
        return cv2.cvtColor(img, cv2.COLOR_RGB2YCrCb)
    if conver == 'BGR2YCrCb':
        return cv2.cvtColor(img, cv2.COLOR_BGR2YCrCb)
    if conver == 'RGB2LUV':
        return cv2.cvtColor(img, cv2.COLOR_RGB2LUV)

# creating of feature vector using cv2.resize().ravel() function
def bin_spat(img, size=(30, 30)):
    feat = cv2.resize(img, size).ravel()
    return feat

# Defining a function to calculate features of color histogram
def color_histogram(img, n_bins=32, bin_rang=(0, 256)):
    # Computing the color channels of histogram alonely
    chann_1_hist = np.histogram(img[:,:,0], bins=n_bins, range=bin_rang
    )
    chann_2_hist = np.histogram(img[:,:,1], bins=n_bins, range=bin_rang
    )
    chann_3_hist = np.histogram(img[:,:,2], bins=n_bins, range=bin_rang
    )
    # Concatenating  histograms into one feature vector
    hist_feat = np.concatenate((chann_1_hist[0], chann_2_hist[0],
    chann_3_hist[0]))
    return hist_feat

# Defining a function that will extrcat features from list of images
def extracting_features(imgs, c_s='RGB', spat_size=(32, 32),
```

```
                        h_bins=32, o=9,
                        p_i_c=8, c_i_b=3, hog_chann=0,
                        s_feat=True, h_feat=True, ho_feat=True):
    # Creating a list that will append feature vectors to the features
    feature = []
    # Iterating through a list
    for file in imgs:
        file_feat = []
        imag = mpimg.imread(file)
        # color conversion if the image is not "rgb"
        if c_s != 'RGB':
            if c_s == 'HSV':
                image_feat = cv2.cvtColor(image, cv2.COLOR_RGB2HSV)
            elif c_s == 'LUV':
                image_feat = cv2.cvtColor(image, cv2.COLOR_RGB2LUV)
            elif c_s == 'HLS':
                image_feat = cv2.cvtColor(image, cv2.COLOR_RGB2HLS)
            elif c_s == 'YUV':
                image_feat = cv2.cvtColor(image, cv2.COLOR_RGB2YUV)
            elif c_s == 'YCrCb':
                image_feat = cv2.cvtColor(image, cv2.COLOR_RGB2YCrCb)
        else: image_feat = np.copy(imag)

        if s_feat == True:
            s_feat = bin_spat(image_feat, size=spat_size)
            file_feat.append(s_feat)
        if h_feat == True:
            # Applying  color histogram
            h_feat = color_hist(image_feat, n_bins=hist_bins)
            file_feat.append(h_feat)
        if ho_feat == True:
            if hog_chann == 'ALL':
```

```
97                 ho_feat = []
98                 for chann in range(image_feat.shape[2]):
99                     ho_feat.append(getting_hog_feature(feature_image
    [:,:,chann],
100                                        o, p_i_c, c_i_b,
101                                        vi=False, feature_vec=True))
102                 ho_feat = np.ravel(ho_feat)
103             else:
104                 ho_feat = getting_hog_feature(image_feat[:,:,
    hog_channel], o,
105                             p_i_c, c_i_b, vi=False, feature_vec=True)
106             # Appending  new feature vector with the features list
107             file_feat.append(hog_feat)
108         feature.append(np.concatenate(file_feat))
109     # This will return the list of feature vectors
110     return feature
111
112 #  Sample size could also be reduced
113 sample_size = -1
114 vehicles = vehicles[0:sample_size]
115 not_vehicles = not_vehicles[0:sample_size]
116 # The color space could be RGB, HSV, LUV, HLS, YUV, YCrCb
117 c_s = 'YCrCb'
118 # This is Orientation for HOG
119 o = 18
120 # Pixel in each cells
121 p_i_c = 8
122 # cell in each block
123 c_i_b = 2
124 # Channels of HOG
125 hog_channel = 'ALL'
126 # The dimension of spetial bining
```

```
127 spat_size = (16, 16)
128 # The bins of histogram
129 h_bins = 16
130 # The Spatial features which are boolean
131 s_feat = True
132 # The Histogram features that could be true or false
133 h_feat = True
134 # The HOG features that could be true or false
135 ho_feat = True
136 # maximum and minimum in y direction to search in sliding window
137 y_st_sto = [400, None]
138
139
140 # Vehicle features
141 v_f = extracting_features(vehicles, c_s=c_s,
142                           spat_size=spat_size, h_bins=h_bins,
143                           o=o, p_i_c=p_i_c,
144                           c_i_b=c_i_b,
145                           hog_chann=hog_chann, s_feat=s_feat,
146                           h_feat=h_feat, ho_feat=ho_feat)
147 # Not Vehicle features
148 n_f = extracting_features(not_vehicle, c_s=c_s,
149                           spat_size=spat_size, h_bins=h_bins,
150                           o=o, p_i_c=p_i_c,
151                           c_i_b=c_i_b,
152                           hog_chann=hog_chann, s_feat=s_feat,
153                           h_feat=h_feat, ho_feat=ho_feat)
154
155 X = np.vstack((v_f, n_f)).astype(np.float64)
156 # Fitting to the column scaler
157 X_scal = StandardScaler().fit(X)
158 # Applying scaler to X
```

```
159 scale_X = X_scal.transform(X)
160
161 # Defining the labels of vector
162 y = np.hstack((np.ones(len(v_f)), np.zeros(len(n_f))))
163
164
165 # Splitting the data into training and test sets
166 ra_state = np.random.randint(0, 100)
167 x_train, x_test, y_train, y_test = train_test_split(scale_X, y,
        test_size=0.2, random_state=ra_state)
168
169 print('By using orientations:' ,o, ",pixel per cell:", p_i_c,",cell per
        block:",c_i_b)
170 print('Length of the feature vector:', len(X_train[0]))
171
172 # Using linear SVC classfier from SK learn
173 svc = LinearSVC()
174 # The time taken by SVC for training
175 t=time.time()
176 svc.fit(x_train, y_train)
177 t2 = time.time()
178 print(round(t2-t, 2), 'Time to train SVC...')
179 # Accuracy result on test set
180 print('Accuracy on test set: ', round(svc.score(x_test, y_test), 4))
181 t=time.time()
182
183
184 y_start = 400
185 y_stop = 656
186 scale = 1.5
187 image = mpimg.imread('/Users/qazi/Documents/study/Masters_Thesis/
        Thesis_final_code/Final_evaluation_videos/vid-4_evaluation/vid-
```

```
    frames/44f99ac4-e57d0b77-0000001.jpg')
188
189 img_output, boxes = search_vehicles(image, y_start, y_stop, scale, svc,
        X_scaler, o, p_i_c, c_i_b, spat_size, h_bins)
190
191 plt.figure(figsize=(20,10))
192 plt.imshow(img_output)
193 plt.show()
194
195 ystart = 380
196 ystop = 636
197 scale = 2.5
198  # reading frame from video
199 image = mpimg.imread('/Users/qazi/Documents/study/Masters_Thesis/
    Thesis_final_code/Final_evaluation_videos/vid-4_evaluation/vid-
    frames/44f99ac4-e57d0b77-0000001.jpg')
200 print(y_start, y_stop, scale, svc, X_scaler, o, p_i_c, c_i_b, spat_size
    , h_bins)
201 img_output, boxes = search_vehicles(image, y_start, y_stop, scale, svc,
        X_scaler, o, p_i_c, c_i_b, spat_size, h_bins)
202
203 print('No. of boxes found:', len(boxes))
204 plt.figure(figsize=(20,10))
205 plt.imshow(img_output)
206 plt.show()
207
208 def adding_heat(heat_map, box_lst):
209     # iterating by list of bounding boxes
210     for boxe in box_lst:
211         #  Inside each bbox 1 is added
212         # Assuming that box takes standard measuremenst ((x1, y1), (x2,
    y2))
```

```
213         heat_map[boxe[0][1]:boxe[1][1], boxe[0][0]:boxe[1][0]] += 1
214     return heat_map
215 def applying_threshold(heat_map, thres):
216     # The Zero pixels below then the threshold
217     heat_map[heat_map <= thres] = 0
218     return heat_map # This will return threshold map
219 def drawing_labeled_box(img, label):
220     # Iterating through all the detected vehicles
221     for vehicle_numbers in range(1, label[1]+1):
222         # Finding pixels correspond to each vehicle_number label value
223         non_zero = (label[0] == vehicle_numbers).nonzero()
224         #  x and y values of pixels will be identified
225         non_zero_y = np.array(non_zero[0])
226         non_zero_x = np.array(non_zero[1])
227         # Illustrate a bounding box based on minimum and maximum x and
    y
228         b_box = ((np.min(non_zero_x), np.min(non_zero_y)), (np.max(
    non_zero_x), np.max(non_zero_y)))
229         # Drawing box on image
230         cv2.rectangle(img, b_box[0], b_box[1], (0,0,255), 6)
231     return img
232
233    from moviepy.editor import VideoFileClip
234 # let us start searching bounding boxes in video which takes video as
    input and returns the video with detected vehicles
235 def video_processing(video_input, output, start=0, end=-1):
236     clip1 = VideoFileClip(video_input)
237     clip1 = clip1.subclip(start, end)
238     out_clip = clip1.fl_image(creating_cars_heat)
239     out_clip.write_videofile(output,audio=False)
```

Listing A.4: Vehicle detection

```
1
2 import math
3 import cv2
4 import numpy as np
5 from PIL import Image, ImageDraw, ImageFont
6 import pandas as pd
7 import matplotlib.image as img
8 import matplotlib.pyplot as plt
9 import json
10
11 frameNumber = 0
12 baseSpeed = 0
13 # importing the gps data to extrcat the speed
14 speedList = []
15 with open('0a4ef631-f8c1e17c.json') as json_file:
16     data = json.load(json_file)
17     for p in data['gps']:
18         speedList.append(p["speed"])
19
20 # function to get our car speed at each frame
21 def getOurCarSpeed(frame):
22     frame = frame / 2
23     print('getOurCarSpeed', round(speedList.__getitem__(int(frame)), 3)
       )
24     return round(speedList.__getitem__(int(frame)), 3)
25
26 their_speed = []
27 distance1 = []
28 myspeed = []
29
30 # To find out the mask point from an image which will give the area of
       interest
```

```
31 def making_mask(imag_size,  # The width and height
32                 hori_perc,  # The upper threshold which is  a percent of
   height
33                 bott_perc,  # The lower thresh which is a percent of
   height
34                 mask_bott_perc=1.0,  # The lower percent of width
35                 mask_topp_perc=0.5):  # The upper percent of width
36
37     img_wdth = imag_size[0] # image width
38     img_hght = imag_size[1] # image height
39
40     center_X = img_width / 2 # finding the center
41     horizon_in_y = math.floor(hori_perc * img_hght)
42     bottom_y_margin = math.floor(bott_perc * img_hght)
43     bottom = img_hght - bottom_y_margin
44     top = horizon_in_y
45
46     mask_in_bottom_left_x = math.floor(center_X - img_wdth * (
   mask_bott_perc * 0.5))
47     mask_in_bottom_right_x = math.floor(center_X + img_wdth * (
   mask_bott_perc * 0.5))
48     mask_in_top_left_x = math.floor(center_X - img_wdth * (
   mask_topp_perc * 0.5))
49     mask_in_top_right_x = math.floor(center_X + img_wdth * (
   mask_topp_perc * 0.5))
50
51     # mask points
52     mask_point = [(mask_bottom_left_x, bottom),
53                   (mask_top_left_x, top),
54                   (mask_top_right_x, top),
55                   (mask_bottom_right_x, bottom)]
56
```

```
57      return mask_point
58  # Function which will be imployed to use perspective reverse
59  '''
60  Taking Input as image and four src points in a rhombus and
61  destination points in more linear quad and warp (look the image in bird
        eye view) image to straighten effects of perspective
       transformation.
62  '''
63  def persp_rev(img, corn_src, corn_dest, imag_size):
64      # corner source is corn_src
65      src = np.float32(corn_src)
66      # distantion points where the image is needed to warp
67      dst = np.float32(corn_dest)
68
69      # Given src and dst points, calculate the perspective transform
       matrix
70      Matrix = cv2.getPerspectiveTransform(src, dst)
71     # This is to calculate the inverse matrix
72      inv_M = cv2.getPerspectiveTransform(dst, src)
73
74      # Warp the image using OpenCV warpPerspective()
75      warped = cv2.warpPerspective(img, Matrix, imag_size, flags=cv2.
       INTER_LINEAR)
76      cv2.destroyAllWindows()
77      return warped, Matrix, invM
78  '''
79   Perspective matrix setting of image_size and warped_size
80  '''
81  def make_persp_mat(img):
82      imag_size = (img.shape[1], img.shape[0])
83      warp_size = (1200, 1200)
84      # source and destination corners
```

```
85      src_corn = make_mask(img_size, 0.65, 0.05, 0.60, 0.1)
86      dest_corn = make_mask(warped_size, 0.1, 0.0, 0.4, 0.36)
87      warped, Matrix, inv_M = perspective_reverse(img, src_corn,
        dest_corn, warp_size)
88
89      return Matrix
90
91  '''
92  Below is function to perform perspective transform on a single point by
        giving x, y pixel and persp matrix (Matrix)
93  It will return the x, y pixel pair in transformed space
94  '''
95  def tr_m(p_t_xy, M):
96
97      p_t = np.array([p_t_xy])
98      p_t = np.array([p_t])
99      reserv = cv2.perspectiveTransform(p_t, Matrix)
100
101     return reserv[0][0]
102
103
104 # updating velocity on new bounding box
105 def update_vel(self):
106     #apply current vel
107     self.ground_t_1[0] += self.vel[1]
108     self.ground_t_1[2] += self.vel[1]
109     self.ground_t_1[1]+= self.vel[0]
110     self.ground_t_1[3]+= self.vel[0]
111
112 def main():
113
114     vid_frames = "D:/Beamng_qazi_data/Thesis_final_code/
```

```
    Final_evaluation_videos/vid-1_evaluation/vid_frames/0a4ef631-
    f8c1e17c-0000001.jpg"
115  vid_frames =  img.imread(vid_frames)
116  # imgplot = plt.imshow(qaziImage)
117  # plt.show()
118
119  perspectiveMatrix = make_persp_mat(vid_frames)
120  # This will return the perspective matrix of fame
121  print (perspectiveMatrix)
122  # opening the file which includes the bounding box values of each
    frame
123  with open('D:/Beamng_qazi_data/Thesis_final_code/
    Final_evaluation_videos/vid-1_evaluation/video_fps_1_results.json',
     'r') as fin:
124      data = json.load(fin)
125      ground_t = []
126      for i in data:
127          labels = i['labels']
128          ground_t_1 = []
129          for l in labels:
130              x1 = round(float(l['box2d']['x1']))
131              y1 = round(float(l['box2d']['y1']))
132              x2 = round(float(l['box2d']['x2']))
133              y2 = round(float(l['box2d']['y2']))
134              ground_t_1.append(x1)
135              ground_t_1.append(y1)
136              ground_t_1.append(x2)
137              ground_t_1.append(y2)
138              ground_t.append(ground_t_1)
139          top, left, bottom, right = ground_t_1
140          print (ground_t_1)
141  # Computing the center point of bounding box
```

```
142             centerPoint =  (left + (right - left) / 2.0, top + (bottom
    - top) / 2.0)
143             print (centerPoint)
144      # Calculating the reverse matrix
145             reserv = tr_m(centerPoint, perspectiveMatrix)
146             print (reserv)
147     # Formula used for calculating the distance
148             distance = round((((1200 - reserv[1])) / 100),1 )
149             print (distance)
150             vel = [0.0, 0.0]
151           # function for computing for all images
152         def interp(self, ground_t_1, alpha):
153             center = self.get_center_pt()
154             self.update_vel()
155             n = len(ground_t_1.ground_t_1)
156
157             for i in range(n):
158                 self.ground_t_1[i] = self.ground_t_1[i] * (1.0 - alpha)
     + ground_t_1.ground_t_1[i] * alpha
159           prev_center_y = self.tm_center[1]
160           self.tm_center = reserv
161           # velocity estimation in y direction
162           vel_est_y = res[1] - prev_center_y
163           global frameNumber
164           base = getOurCarSpeed(frameNumber)  # how fast am I going?
165            # updating velocity based on new center
166            new_center = self.get_center_pt()
167            self.vel = [new_center[0] - center[0], new_center[1] -
    center[1]]
168             self.age = 1.0
169             self.obscurred = False
170             fator = 4.0
```

```
171             global frameNumber
172             base = getOurCarSpeed(frameNumber)
173             # calculating speed
174             speed = base + ((-1 * vel[0] + -1 * vel[1]) * fator)
175             print (speed)
176 
177     their_speed = []
178     distance1 = []
179     myspeed = []
180 
181     their_speed.append(self.speed)
182     distance1.append(distance)
183     myspeed.append(getOurCarSpeed(frameNumber))
184     df_final = pd.DataFrame(data={"My Speed": myspeed, "Their speed":
    their_speed, "distance": distance1})
185     # creating a .csv file withh all the information
186     df_final.to_csv("D:/Beamng_qazi_data/Code/CarND-Vehicle-Detection/
    speed_distance_car5.csv", sep=",",
187                     encoding="utf-8")
188 
189 if __name__ == '__main__':
190     main()
```

Listing A.5: distance and speed estimation

```
1 
2 # # Running the Simulation in BeamNG
3 
4 import sys
5 from time import sleep
6 import numpy as np
7 import beamngpy
8 from scipy import interpolate
```

```
9  from beamngpy.sensors import Electrics
10 from beamngpy.sensors import Camera
11 import time
12 from beamngpy import BeamNGpy, Scenario, Road, Vehicle, setup_logging
13
14 SIZE = 1024
15
16 def main():
17     setup_logging()
18
19     beamng = BeamNGpy('localhost', 64256, home='D:/BeamNG')  # This is
    the host & port used to communicate over
20
21     scenario = Scenario('xyz_Map', 'Thesis')
22
23     vehicle = Vehicle('ego_vehicle', model='etk800', licence='Qazi',
    color='Green')
24     overhead = Camera(pos = (-0.25, 0.78, 1), direction = (-0.25, 10,
    1.1), fov = 60, resolution =(1280, 720))
25     vehicle.attach_sensor('overhead', overhead)
26     vehicle2 = Vehicle('ego_vehicle2', model='etkc', licence='Qazi2',
    color = 'Silver')
27     vehicle3 = Vehicle('ego_vehicle3', model='etk800', licence='Qazi3',
     color = 'Silver')
28     # when adding self driving car in real sceanrio
29     vehicle4 = Vehicle('ego_vehicle4', model='etk800', licence='Qazi3',
     color = 'Blue')
30     road_a = Road(material = 'AsphaltRoad_lanes', rid='main_road',
    defaultLanes = '2', texture_length= '5', defaultLaneWidth = '4',
    scale = '1 1 1', detail = '0.25')
31     nodes = Road_a
32
```

```
33      road_a.nodes.extend(nodes)
34      scenario.add_road(road_a)
35
36  # Importing road coordinates from final_road.npy from road_final.py
37
38      final_roads = np.load('final_roads_vid_1.npy', allow_pickle=True)
39      count = 1
40      # nodes = []
41      for road in final_roads:
42          # name = road['name']
43          values = road['values']
44
45          road_obj = Road(material='AsphaltRoad_lanes', texture_length= '
    5')
46          nodes = values
47
48          road_obj.nodes.extend(nodes)
49          scenario.add_road(road_obj)
50
51          count = count + 1
52          print(count)
53          if count == 20:
54              break
55
56      '''Car 1 coordinates and speed handling'''
57
58      car_1_cord = normalised_coord_car_1
59      car1 = car_1_cord[0]
60      # print(orig)
61      speed_car_1 = speed_normalized
62      speed_1 = speed_car_1[0]
63      # ================================================
```

```
64
65      '''Car 2 coordinates and speed handling'''
66
67      car_2_cord = normalised_coord_car_2
68      car2 = car_2_cord[0]
69
70      speed_car_2 = [14.6, 17.6, 14.3, 15.34, 13.30, 14.22, 14.47, 16.62,
        17.85,15.34, 16.65, 14.98, 14.3, 14.32 , 14.32]
71      speed_2 = speed_car_2[0]
72     # ====================================================
73
74      ''' Car 3 coordinates and speed handling'''
75
76      car_3_cord = normalised_coord_car_3
77      car3 = car_3_cord[0]
78
79      speed_car_3 = [5.0, 6.0 ,7.0 ,7.0 , 14.6, 17.6, 14.3, 15.34, 13.30,
        14.22, 14.47, 16.62, 15.85,13.34,12.3,12.34,15.32,20.22, 21.1,
       20.3, 23.2 ,22.0, 22.2, 21.34, 22.33, 18.11, 18.11]
80      speed_3= speed_car_3[0]
81      #====================================================
82
83      pos_car_3_start = (453.1446969569661, 5.686902701854706, 0)
84      scenario.add_vehicle(vehicle, pos = car1, rot=(0, 0,130 ))
85      scenario.add_vehicle(vehicle2, pos = car2, rot=(0, 0,130 ))
86      scenario.add_vehicle(vehicle3, pos = car3, rot=(0, 0, 90))
87
88      #adding self driving car in the scenario
89      #scenario.add_vehicle(vehicle4, pos= (510.4156331927404,
       17.21674171742052,0 ), rot=(0, 0, 90))
90      scenario.make(beamng) # The make function of a scneario is used to
       compile the scenario and produce a scenario file the simulator can
```

```
    load
91
92      '''car no 1'''
93
94      path1 = list()
95      for i in range(len(car_1_cord)):
96          car1 = car_1_cord[i]
97          speed_1 = speed_car_1[i]
98          pos = (car1[0],
99
100                 car1[1],
101
102                 0)
103         speed = speed_1
104         print(pos)
105         print(speed)
106
107         node = {
108             'pos': pos,
109             'speed': speed
110         }
111         path1.append(node)
112
113     '''car no 2'''
114     path2 = list()
115     for i in range(len(car_2_cord)):
116         car2 = car_2_cord[i]
117         speed_2 = speed_car_2[i]
118         pos = (car2[0],
119
120                car2[1],
121
```

```
122                 0)
123         speed = speed_2
124         print(pos)
125         print(speed)
126 
127         node = {
128             'pos': pos,
129             'speed': speed
130         }
131         path2.append(node)
132 
133     '''car no 3'''
134     path3 = list()
135     for i in range(len(car_3_cord)):
136         car3 = car_3_cord[i]
137         speed_3 = speed_car_3[i]
138         pos = (car3[0],
139 
140                 car3[1],
141 
142                 0)
143         speed = speed_3
144         print(pos)
145         print(speed)
146 
147         node = {
148             'pos': pos,
149             'speed': speed,
150         }
151         path3.append(node)
152 
153     bng = beamng.open(launch=True)
```

```
154
155     bng.set_deterministic() # Set simulator to be deterministic
156     bng.set_steps_per_second(8)  # With 60hz temporal resolution
157     bng.load_scenario(scenario)
158     bng.start_scenario()
159     bng.hide_hud()
160     #bng.pause()
161     bng.set_tod(tod=0.4)
162     bng.display_gui_message(msg =  "Simulation 1")
163     #bng.remove_step_limit()
164     #bng.set_steps_per_second(sps = 1)
165     vehicle.ai_set_line(path1)
166     vehicle2.ai_set_line(path2)
167     vehicle3.ai_set_line(path3)
168
169
170     for it in range(250):
171         bng.step(8)
172         #for annotation use below line
173         # view = bng.poll_sensors(vehicle)["overhead"]["annotation"]
174             #for colour frame use below
175         view = bng.poll_sensors(vehicle)["overhead"]["colour"]
176         view = view.convert('RGB')
177         filename = "Final_evaluation_videos/vid-1_evaluation/
    color_frames/frames_{}.png".format(it)
178
179         view.save(filename)
180         print("{} file saved.".format(filename))
181     pass
182
183     print("Images saved.")
184
```

```
185     y = [p[0] for p in car_1_cord]
186     x = [p[1] for p in car_1_cord]
187     plt.plot(x, y, '.')
188     plt.axis('square')
189     plt.show()
190     plt.clf()
191
192     # plotting road edges
193     road_geometry = bng.get_road_edges('main_road')
194     left_edge_x = np.array([e['left'][0] for e in road_geometry])
195     left_edge_y = np.array([e['left'][1] for e in road_geometry])
196     right_edge_x = np.array([e['right'][0] for e in road_geometry])
197     right_edge_y = np.array([e['right'][1] for e in road_geometry])
198
199     def plot_road(ax):
200       x_min = min(left_edge_x.min(),
201                   right_edge_x.min()) - 10  # We add/subtract 10
    from the min/max coordinates to pad
202       x_max = max(left_edge_x.max(), right_edge_x.max()) + 10
203     # the area of the plot a bit
204       y_min = min(left_edge_y.min(), right_edge_y.min()) - 10
205       y_max = max(left_edge_y.max(), right_edge_y.max()) + 10
206       ax.set_aspect('equal', 'datalim')
207       ax.set_xlim(left=x_max, right=x_min)
208       # pyplot & bng  coordinate systems have different origins
209       ax.set_ylim(bottom=y_max, top=y_min)  # so we flip them here
210       ax.plot(left_edge_x, left_edge_y, 'b-')
211       ax.plot(right_edge_x, right_edge_y, 'b-')
212     plt.figure(figsize=(10, 10))
213     plot_road(plt.gca())
```

```
214     plt.show()
```

Listing A.6: Final simulation creation

A.3 Evaluation

```
1
2 import csv
3 import json
4 with open('Final_evaluation_videos/vid-4_evaluation/video_12_results.
    json', 'r') as fin:
5     data = json.load(fin)
6     ground_t = []
7     for i in data:
8         labels = i['labels']
9         ground_t_1 = []
10        for l in labels:
11                x1 = round(float(l['box2d']['x1']))
12                y1 = round(float(l['box2d']['y1']))
13                x2 = round(float(l['box2d']['x2']))
14                y2 = round(float(l['box2d']['y2']))
15                ground_t_1.append(x1)
16                ground_t_1.append(y1)
17                ground_t_1.append(x2)
18                ground_t_1.append(y2)
19                ground_t.append(ground_t_1)
20    print(ground_t)
21
22
23 with open('Final_evaluation_videos/vid-4_evaluation/results/
    ground_truth_vid-4.csv', 'w', newline= '') as writeFile:
24     writer = csv.writer(writeFile)
```

```
25     writer.writerows(ground_t)
```

Listing A.7: Ground truth creation

```
1
2 import cv2
3 import csv
4 import matplotlib.pyplot as plt
5 # Load the image
6
7
8 import glob, os, re
9
10 files = os.listdir("Final_evaluation_videos/vid-4_evaluation/
     annotated_frames/")
11 files = sorted(files, key=lambda x: int(x.replace("frames_", "").
     replace(".png", "")))
12
13 cv_img = []
14 for f in files:
15     img = "Final_evaluation_videos/vid-4_evaluation/annotated_frames/"
     + f
16     print(img)
17     n = cv2.imread(img)
18     cv_img.append(n)
19 # print(cv_img)
20 print(len(cv_img))
21 print("------------------------------------------------")
22 values_bbox_final = []
23 for i in cv_img:
24     values_bbox = []
25     i = cv2.cvtColor(i, cv2.COLOR_BGR2RGB)
26     i = cv2.cvtColor(i, cv2.COLOR_BGR2RGB)
```

```
27      # Blurring for removing the noise
28      img_blur = cv2.bilateralFilter(i, d = 7,
29                                     sigmaSpace = 75, sigmaColor =75)
30      # Convert to grayscale
31      img_gray = cv2.cvtColor(img_blur, cv2.COLOR_RGB2GRAY)
32      # Apply the thresholding
33      a = img_gray.max()
34      _, thresh = cv2.threshold(img_gray, a/2+60, a,cv2.THRESH_BINARY_INV
        )
35      # plt.imshow(thresh, cmap = 'gray')
36      # Find the contour of the figure
37      contours, hierarchy = cv2.findContours(
38                                          image = thresh,
39                                          mode = cv2.RETR_TREE,
40                                          method = cv2.CHAIN_APPROX_SIMPLE
        )
41      # Sort the contours
42      contours = sorted(contours, key = cv2.contourArea, reverse = True)
43      # Draw the contour
44      img_copy = i.copy()
45      final = cv2.drawContours(img_copy, contours, contourIdx = -1,
46                               color = (0, 0, 255), thickness = 2)
47      #plt.imshow(img_copy)
48
49      # The first order of the contours
50      c_0 = contours[0]
51      # Get the 4 points of the bounding rectangle
52      x, y, w, h = cv2.boundingRect(c_0)
53      # Draw a straight rectangle with the points
54      img_copy = i.copy()
55      img_box = cv2.rectangle(img_copy, (x, y), (x+w, y+h), color = (255,
        0, 0), thickness = 4)
```

```
56      values_bbox.append(x)
57      values_bbox.append(y)
58      values_bbox.append(x+w)
59      values_bbox.append(y+h)
60      values_bbox_final.append(values_bbox)
61
62  print(values_bbox_final)
63
64  with open('Final_evaluation_videos/vid-4_evaluation/results/
        predicted_vid-4.csv', 'w', newline= '') as writeFile:
65    writer = csv.writer(writeFile)
66    writer.writerows(values_bbox_final)
67  print(x, y, x + w, y + h)
```

Listing A.8: Bounding box creation using contours

```
1
2  # importing the packages
3  import csv
4  from collections import namedtuple
5  # define the detection object image path, ground truth, prediction
6  detect = namedtuple("Detect", ["image_path", "gt", "pred"])
7
8  def b_i_o_u(box_A, box_B):
9    # x, y coordinates of the intersection bounding box
10   x_A = max(box_A[0], box_B[0])
11   y_A = max(box_A[1], box_B[1])
12   x_B = min(box_A[2], box_B[2])
13   y_B = min(box_A[3], box_B[3])
14
15   # calculating the area of intersection of rectangle
16   inter_area = max(0, x_B - x_A + 1) * max(0, y_B - y_A + 1)
17
```

```
18    # calculating the area of prediction and ground-truth
19    box_A_area = (box_A[2] - box_A[0] + 1) * (box_A[3] - box_A[1] + 1)
20    box_B_area = (box_B[2] - box_B[0] + 1) * (box_B[3] - box_B[1] + 1)
21
22    # areas - the interesection of  area
23    i_o_u = inter_area / float(box_A_area + box_B_area - inter_area)
24    return i_o_u
25
26 #  Example of detections in frames
27 examples = [
28    #Detection("xframes_0.png", [406, 457, 635, 629], [449, 453, 664,
       595])
29    Detection("xframes_9.png",  [720, 451, 984, 651], [725, 443, 1063,
       643]),
30    Detection("xframes_30.png", [433, 474, 608, 617], [410, 437, 678,
       623])]
31    #Detection("xframes_3.png", [373, 450, 577, 614], [228, 444, 608,
       664]),
32    #Detection("xframes_4.png", [379, 446, 577, 601], [370, 444, 690,
       656])]
33
34 'Read ground_truth file '
35 ground_truths = []
36 with open('Final_evaluation_videos/vid-4_evaluation/results/
       ground_truth_vid-4.csv', 'r') as csvFile:
37    reader = csv.reader(csvFile)
38    for row in reader:
39       nRow = [int(v) for v in row]
40       ground_truths.append(nRow)
41 pred_values = []
42 'Reading predicted_results file'
43 with open('Final_evaluation_videos/vid-4_evaluation/results/
```

```
    predicted_vid-4.csv', 'r') as csvFile:
  reader = csv.reader(csvFile)
  for row in reader:
    nRow = [int(v) for v in row]
    pred_values.append(nRow)

length = len(ground_truths) if len(ground_truths) < len(pred_values)
    else len(pred_values)
total_iou = 0
frame_results = []
for i in range(0, length):
  gt = ground_truths[i]
  pv = pred_values[i]
  iou = b_i_o_u(gt, pv)
  print(iou)
  frame_results.append(iou)
  total_iou += b_i_o_u(gt, pv)
avg = total_iou / len(frame_results)
print("avg:",  avg)
print(total_iou)
```

Listing A.9: Evaluation Metric creation

Bibliography

[Abc] AbcNEWS. *Police release video of Uber self-driving car accident*. URL: https://www.abc.net.au/news/2018-03-22/police-release-video-of-uber-self-driving-car/9575716 (cit. on p. 2).

[AFI14] A. Armand, D. Filliat, and J. Ibafffdfffdez-Guzman. "Ontology-based context awareness for driving assistance systems". In: *2014 IEEE Intelligent Vehicles Symposium Proceedings*. June 2014, pp. 227–233. DOI: 10.1109/IVS.2014.6856509 (cit. on p. 25).

[Bea18] BeamNG GmbH. *BeamNG.research*. Version 1.3.0.0. Oct. 11, 2018. URL: https://www.beamng.gmbh/research (cit. on pp. 3, 29, 50).

[Ber+13] Christian Berger et al. "Model-based, composable simulation for the development of autonomous miniature vehicles". In: *SpringSim*. 2013 (cit. on pp. 2, 23).

[Bra00] G. Bradski. "The OpenCV Library". In: *Dr. Dobb's Journal of Software Tools* (2000) (cit. on pp. 44, 52).

[Bre+17] Guillaume Bresson et al. "Simultaneous Localization and Mapping: A Survey of Current Trends in Autonomous Driving". In: *IEEE Transactions on Intelligent Vehicles* 2 (2017), pp. 194–220 (cit. on p. 25).

[BSV14] L. Brun, A. Saggese, and M. Vento. "Dynamic Scene Understanding for Behavior Analysis Based on String Kernels". In: *IEEE Transactions on Circuits and Systems for Video Technology* 24.10 (Oct. 2014), pp. 1669–1681. ISSN: 1051-8215. DOI: 10.1109/TCSVT.2014.2302521 (cit. on p. 24).

[Bur19] Matt Burns. *Anyone relying on lidar is doomed*. Apr. 2019. URL: https://techcrunch.com/2019/04/22/anyone-relying-on-lidar-is-doomed-elon-musk-says/ (cit. on p. 12).

[CKC16] Adam Paszke Abhishek Chaurasia, Sangpil Kim, and Eugenio Culurciello. *ENet: A Deep Neural Network Architecture for Real-Time Semantic Segmentation*. June 2016. URL: https://arxiv.org/pdf/1606.02147.pdf (cit. on p. 25).

[Com18] Blender Online Community. *Blender - a 3D modelling and rendering package*. Blender Foundation. Stichting Blender Foundation, Amsterdam, 2018. URL: http://www.blender.org (cit. on p. 15).

[Cor+16] Marius Cordts et al. "The Cityscapes Dataset for Semantic Urban Scene Understanding". In: *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016 (cit. on p. 25).

[CVI19] NEDA CVIJETIC. *To Go the Distance, We Built Systems That Could Better Perceive It*. June 2019. URL: https://blogs.nvidia.com/blog/2019/06/19/drive-labs-distance-to-object-detection/ (cit. on p. 59).

[DT05] N. Dalal and B. Triggs. "Histograms of oriented gradients for human detection". In: *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*. Vol. 1. June 2005, 886–893 vol. 1. DOI: 10.1109/CVPR.2005.177 (cit. on p. 20).

[dat] MGRS data. *Military Grid Reference System*. URL: http://mgrs-data.org/ (cit. on p. 16).

[DAG17] Igor Dosen, Marianne Aroozoo, and Michael Graham. *Automated Vehicles*. Dec. 2017. URL: https://www.parliament.vic.gov.au/publications/research-papers/send/36-research-papers/13839-automated-vehicles (cit. on p. 1).

[Erb09] Christian Erbsmehl. "Simulation of real crashes as a method for estimating the potential benefits of advanced safety technologies". In: *The 21st International TechnicalConference on the Enhanced Safety of Vehicles (ESV)*. 2009 (cit. on p. 22).

[Fin19] Geoff Fink. *Proprioceptive Sensor Dataset for Quadruped Robots*. 2019. DOI: 10.21227/4vxz-xw05. URL: http://dx.doi.org/10.21227/4vxz-xw05 (cit. on p. 13).

[Fon+18] Valentina Fontana et al. "Action Detection from a Robot-Car Perspective". In: *CoRR* abs/1807.11332 (2018) (cit. on p. 22).

[Gar] Ed Garsten. *Sharp Growth In Autonomous Car Market Value Predicted But May Be Stalled By Rise In Consumer Fear*. URL: https://www.forbes.com/sites/edgarsten/2018/08/13/%20sharp-growth-in-autonomous-car-market-value-predicted-but-may-be-stalled-by-rise-inconsumer-fear/#57849dff617c (cit. on p. 1).

[GLU12] Andreas Geiger, Philip Lenz, and Raquel Urtasun. "Are we ready for Autonomous Driving? The KITTI Vision Benchmark Suite". In: *Conference on Computer Vision and Pattern Recognition (CVPR)*. 2012 (cit. on p. 37).

[15] "Geodesy". In: *GPS Satellite Surveying*. John Wiley Sons, Ltd, 2015. Chap. 4, pp. 129–206. ISBN: 9781119018612. DOI: 10.1002/9781119018612.

ch4. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/9781119018612.ch4. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/9781119018612.ch4 (cit. on p. 31).

[GBZ16] Hana Gharbi, Sahbi Bahroun, and Ezzeddine Zagrouba. *A Novel Key Frame Extraction Approach for Video Summarization*. Dec. 2016. URL: http://www.scitepress.org/Papers/2016/57257/57257.pdf (cit. on p. 24).

[Gru+14] D. Gruyer et al. "From virtual to reality, how to prototype, test and evaluate new ADAS: Application to automatic car parking". In: *2014 IEEE Intelligent Vehicles Symposium Proceedings*. June 2014, pp. 261–267. DOI: 10.1109/IVS.2014.6856525 (cit. on p. 23).

[12] "IEEE Standard for Distributed Interactive Simulation–Application Protocols". In: *IEEE Std 1278.1-2012 (Revision of IEEE Std 1278.1-1995)* (Dec. 2012), pp. 1–747. ISSN: null. DOI: 10.1109/IEEESTD.2012.6387564 (cit. on p. 14).

[J F] Steven K. Feiner J. F. H. James D. Foley Andries van Dam. *Computer Graphics*. URL: https://ptgmedia.pearsoncmg.com/images/9780321399526/samplepages/0321399528.pdf (cit. on p. 44).

[Jan09] Volker Janssen. "Understanding coordinate systems, datums and transformations in Australia". In: (Jan. 2009), pp. 697–715 (cit. on p. 18).

[Jia+13] H. Jiang et al. "A fast method for RGB to YCrCb conversion based on FPGA". In: *Proceedings of 2013 3rd International Conference on Computer Science and Network Technology*. Oct. 2013, pp. 588–591. DOI: 10.1109/ICCSNT.2013.6967182 (cit. on p. 39).

[Lho] Christoph Lhotka. *Keplerian Orbital Elements*. URL: http://demonstrations.wolfram.com/KeplerianOrbitalElements/%20Wolfram%20Demonstrations%20Project (cit. on p. 16).

[Li+18] Li Li et al. "Artificial intelligence test: a case study of intelligent vehicles". In: *Artificial Intelligence Review* (2018), pp. 1–25 (cit. on p. 1).

[Mar+17] James Martin et al. *Certification for Autonomous Vehicles*. Dec. 2017. URL: https://www.cs.unc.edu/~anderson/teach/comp790a/certification.pdf (cit. on p. 1).

[Mob18] Mobileye. *Sensing the Future*. Apr. 2018. URL: https://www.mobileye.com/ (cit. on p. 13).

[Mot16] California Department of Motor Vehicle. *Google autonomous car crash report*. Aug. 2016. URL: https://www.dmv.ca.gov/portal/wcm/connect/6448a850-67fa-4136-938c-28c6cd9a4afd/Google_081616.pdf?MOD=AJPERES&CVID= (cit. on p. 2).

[Naw15] Shah Nawaz. "HOG-SVM Car Detection on an Embedded GPU". PhD thesis. Nov. 2015 (cit. on p. 37).

[NHT18] NHTSA. *Automated Vehicles for Safety*. Nov. 2018. URL: https://www.nhtsa.gov/technology-innovation/automated-vehicles-safety (cit. on p. 1).

[Olb] Roland Olbricht. *Overpass API*. URL: http://overpass-api.de/ (cit. on p. 32).

[Ope17] OpenStreetMap contributors. *Planet dump retrieved from https://planet.osm.org*. 2017. URL: https://www.openstreetmap.org (cit. on pp. 4, 31).

[PAP13] MORGAN STANLEY BLUE PAPER. *Self-Driving the New Auto Industry Paradigm*. Nov. 2013. URL: https://orfe.princeton.edu/~alaink/SmartDrivingCars/PDFs/Nov2013MORGAN-STANLEY-BLUE-PAPER-AUTONOMOUS-CARSEFBC9A-SELF-DRIVING-THE-NEW-AUTO-INDUSTRY-PARADIGM.pdf (cit. on p. 1).

[Ped+11] F. Pedregosa et al. "Scikit-learn: Machine Learning in Python". In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830 (cit. on pp. 20, 39).

[PE12] M. Platho and J. Eggert. "Deciding what to inspect first: Incremental situation assessment based on information gain". In: *2012 15th International IEEE Conference on Intelligent Transportation Systems*. Sept. 2012, pp. 888–893. DOI: 10.1109/ITSC.2012.6338670 (cit. on p. 24).

[PGE12] M. Platho, H. Grofffdfffd, and J. Eggert. "Traffic situation assessment by recognizing interrelated road users". In: *2012 15th International IEEE Conference on Intelligent Transportation Systems*. 2012, pp. 1339–1344 (cit. on pp. 2, 24).

[Pop14] Gabriel Popescu. "Pixel geolocation algorithm for satellite scanner data". In: June 2014 (cit. on p. 17).

[Ram+18] Vasili Ramanishka et al. "Toward Driving Scene Understanding: A Dataset for Learning Driver Behavior and Causal Reasoning". In: 2018 (cit. on p. 23).

[Red+15] Joseph Redmon et al. *You Only Look Once: Unified, Real-Time Object Detection*. 2015. arXiv: 1506.02640 [cs.CV] (cit. on p. 59).

[Rez+19] Seyed Hamid Rezatofighi et al. "Generalized Intersection over Union: A Metric and A Loss for Bounding Box Regression". In: *CVPR*. 2019 (cit. on p. 53).

[SAE] SAE. *Automated driving*. URL: https://www.sae.org/misc/pdfs/automated_driving.pd (cit. on p. 6).

[Sch] Adam Schneider. *GPS visualizer*. URL: https://www.gpsvisualizer.com/ (cit. on p. 28).

[Spe+13] J. Spehr et al. "Efficient scene understanding for intelligent vehicles using a part-based road representation". In: *16th International IEEE Conference on Intelligent Transportation Systems (ITSC 2013)*. Oct. 2013, pp. 65–70. DOI: 10.1109/ITSC.2013.6728212 (cit. on p. 23).

[Taw17] Tawnkramer. *CarND-Vehicle-Detection*. Aug. 2017. URL: https://github.com/tawnkramer/CarND-Vehicle-Detection (cit. on p. 37).

[Thea] TheEconomist. *Why Uber self-driving car killed a pedestrian*. URL: https://www.economist.com/the-economist-explains/2018/05/29/why-ubers-self-driving-car-killed-a-pedestrian (cit. on pp. 2, 3).

[Theb] TheGuardian. *Uber dashcam footage shows lead up to fatal self-driving crash video*. URL: https://www.theguardian.com/technology/video/2018/mar/22/in-car-footage-shows-fatal-self-driving-crash-video (cit. on p. 2).

[usa] usatoday. *Uber self-driving car crash*. URL: https://www.usatoday.com/story/money/cars/2018/05/24/uber-self-driving-car-crash-ntsb-investigation/640123002/ (cit. on pp. 2, 3).

[Wal+14] Stéfan van der Walt et al. "scikit-image: image processing in Python". In: *PeerJ* 2 (June 2014), e453. ISSN: 2167-8359. DOI: 10.7717/peerj.453. URL: https://doi.org/10.7717/peerj.453 (cit. on pp. 38, 41).

[Was] Jacob Wasserman. *mplleaflet library*. URL: https://github.com/jwass/mplleaflet (cit. on p. 31).

[Yu+18] Fisher Yu et al. "BDD100K: A Diverse Driving Video Database with Scalable Annotation Tooling". In: *ArXiv* abs/1805.04687 (2018) (cit. on pp. 49, 50).

[ZF17] ZF. *Forward-Looking Radar*. July 2017. URL: https://www.zf.com/products/en/lcv/products_51104.html (cit. on p. 12).

[Zha+15] L. Zhao et al. "Ontology-based decision making on uncontrolled intersections and narrow roads". In: *2015 IEEE Intelligent Vehicles Symposium (IV)*. June 2015, pp. 83–88. DOI: 10.1109/IVS.2015.7225667 (cit. on p. 25).

[Zof+16] M. R. Zofka et al. "Testing and validating high level components for automated driving: simulation framework for traffic scenarios". In: *2016 IEEE Intelligent Vehicles Symposium (IV)*. June 2016, pp. 144–150. DOI: 10.1109/IVS.2016.7535378 (cit. on pp. 2, 23).

Eidesstattliche Erklärung

Hiermit versichere ich, dass ich diese Masterarbeit selbstständig und ohne Benutzung anderer als der angegebenen Quellen und Hilfsmittel angefertigt habe und alle Ausführungen, die wörtlich oder sinngemäß bernommen wurden, als solche gekennzeichnet sind, sowie, dass ich die Masterarbeit in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegt habe.

Passau, May 22, 2020

qazi mujahid

Qazi Mujahid