FELADATKIÍRÁS

Jelenetértelmezés megvalósítása autonóm járművekhez mély neurális hálók segítségével
A gépi tanulás új módszerei az intelligens érzékelés számos területét forradalmasították az elmúlt évtizedben. Ezen módszerek közül külön figyelmet érdemel a mély tanulás (Deep Learning), amely a gépi tanulás legtöbb területén state of the art megoldásnak számít. A mély tanulás egyik legfontosabb alkalmazása a jelenetértelmezésben (scene understanding) történik, ahol az környezetben található objektumok minél több jellemzőjét és ezek kapcsolatait kívánjuk feltárni.

Az utóbbi néhány évben a mély tanulás kutatásának egyik fontos fókusza a különböző mobilis robotokban történő alkalmazása, amelyek a robot aktuális környezetének átfogó ismeretét igénylik. Az alkalmazások közül kiemelhető az autonóm járművek területe, ahol az objektumokról meghatározott információ teljessége és megbízhatósága kiemelten fontos.

A diplomatervezés során a hallgató feladata egy olyan algoritmus késztése, amely képes bemeneti kép vagy videofolyam alapján az abban található objektumok felismerésére és a jármű számára releváns információk (sebesség, forma, relációk stb.) kinyerésére.

A hallgató feladatának a következőkre kell kiterjednie:

- Tanulmányozza át a téma releváns szakirodalmát. Vizsgálja meg, hogy más műhelyek milyen megoldásokat alkalmaznak.
- Készítsen rendszertervet egy megoldásra, amely képes a jelenetértelmezés elvégzésére.
- Végezze el az algoritmus fejlesztését és tanítását.
- Tesztelje a megoldás pontosságát és hatékonyságát, valamint végezze el a tanuló algoritmus validációját.
- Vizsgálja a megoldást valósidejűség szempontjából.

Tanszéki konzulens: Szemenyei Márton

Budapest University of Technology and Economics
Faculty of Electrical Engineering and Informatics
Department of Control Engineering and Information Technology

Driving Scene Understanding in Simulation with Stereo RGB imaging and CNN synergy

Master's Thesis

Author
Najib Ghadri

Advisor
Márton Szemenyei

May 30, 2020

Contents

HALLGATÓI NYILATKOZAT

Alulírott *Ghadri Najib*, szigorló hallgató kijelentem, hogy ezt a szakdolgozatot meg nem engedett segítség nélkül, saját magam készítettem, csak a megadott forrásokat (szakirodalom, eszközök stb.) használtam fel. Minden olyan részt, melyet szó szerint, vagy azonos értelemben, de átfogalmazva más forrásból átvettem, egyértelműen, a forrás megadásával megjelöltem.

Hozzájárulok, hogy a jelen munkám alapadatait (szerző(k), cím, angol és magyar nyelvű tartalmi kivonat, készítés éve, konzulens(ek) neve) a BME VIK nyilvánosan hozzáférhető elektronikus formában, a munka teljes szövegét pedig az egyetem belső hálózatán keresztül (vagy autentikált felhasználók számára) közzétegye. Kijelentem, hogy a benyújtott munka és annak elektronikus verziója megegyezik. Dékáni engedéllyel titkosított diplomatervek esetén a dolgozat szövege csak 3 év eltelte után válik hozzáférhetővé.

Budapest, 2020. május 30.

Ghadri Najib
hallgató

Abstract

Autonomous driving is undoubtedly the future of transportation. The comfort that it brings us is what drives us to work on making it real. We already have autonomous systems in public transportation in abundance, but it is different when we talk about the car roads. Driving a car requires near-human intelligence due to the nature of the environment, in fact it is impossible to define the environment. A train's or subways's environment can be defined mathematically and hence controlled easly, but for a machine to drive a car, it has to understand what we understand, and what we understand is even hard to define ourselves.

Computer science has come a long way, and we have already seen the rise of artificial intelligence algorithms and their effectiveness. Out of these methods Deep Learning and Convolutional Neural Networks are key tools in achieving our goal. With these algorithms computers learn general concepts of the world, and this is essential to make a safe autonomous driving (AD) system. We will see in this work briefly what they are and how they work.

Some notable companies have already achieved a high level of AD, most notably Tesla, and another AD supplier MobilEye. These companies use algorithms that are developed globally and publicly and I am going to use the same algorithms to partly achieve what they have achieved.

In this work I am going to create a Scene Understanding system specialized for driving situations. I choose to evaluate the system on a virtual car driving simulation called CARLA Sim, that is going to benefit us to measure our rate of success.

I researched how existing autonomous driving systems have been built, and inspired by them I designed a system that is capable of recognizing important information for a car on the road. I used stereo imaging of multiple RGB cameras mounted on top of our virtual car for depth estimation and used trained Convolutional Neural Networks to then perform further infomration extraction from the images and perform detection for each frame of the simulation. I made a 3D webvisualizer that is able to show us the difference between ground truth information extracted programatically from the simulator and the detection infomration while simultaneously play a montage video of the simulation. Finally I evaluated the system and measured it's validity for real situations and provided further improvement notes on my work. This thesis is also published on https://najibghadri.com/msc-thesis/ where you can try the 3D webvisualizer.

Chapter 1

Introduction

I am very passionate about artificial intelligence and as much inspired by the work of tech companies such as Tesla. Tesla has managed create cutting edge technology, creating compelling and practical electric cars combined with their Tesla Autopilot system. It has become iconic to sit in a Tesla and watch it drive itself. Tesla has already driven 3 billion drives on autopilot, their access to data is most likely number one in the world. There are other important companies who develop autopilot systems, one of them is MobilEye an Israeli subsidiary of Intel corporation that was actually a supplier of Tesla until they set apart in part due to disagreements on how the technology should be built, which is an important topic that I will talk about.

There are a couple of topics we should establish first. The first being levels of autopilot systems as defined by SAE (Society of Automotive Engineers) (figure 1.1).

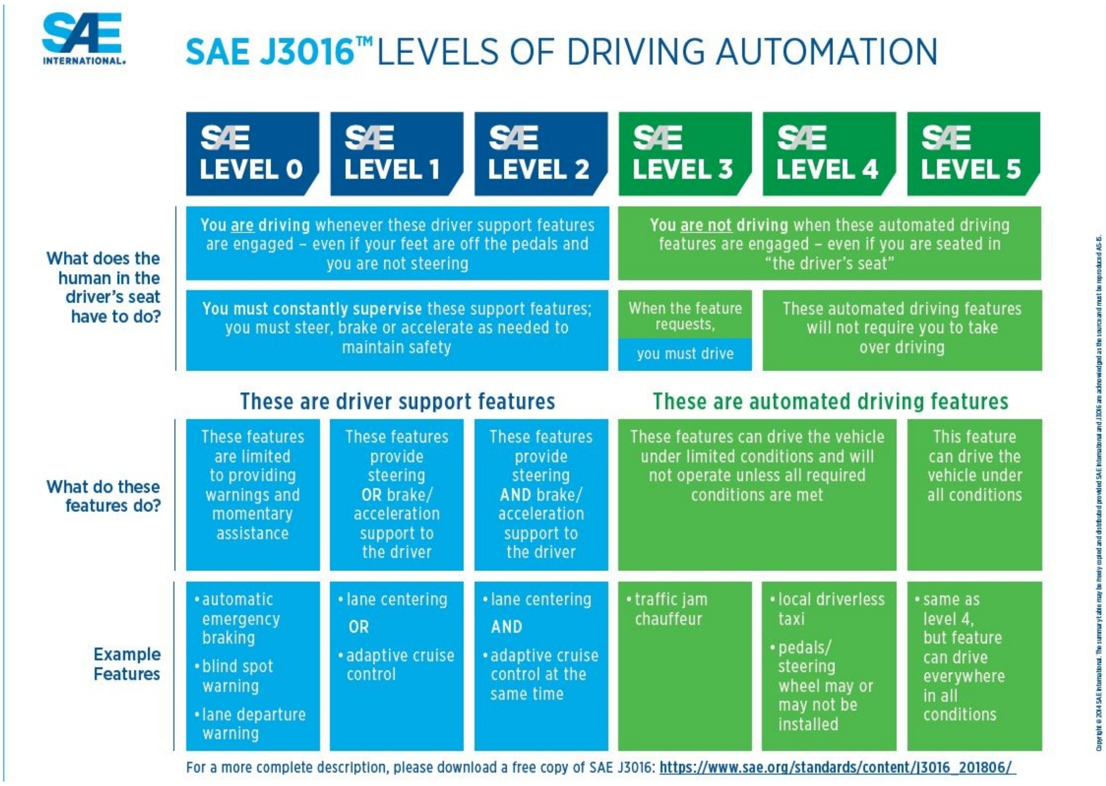

Figure 1.1: Levels of driving automation defined in SAE J3016 [1]

From level 0 to 2 are automations where the human is still required to fully monitor the driving environment. Tesla's autopilot is Level 2 which is partial automation that includes control of steering and both acceleration and deceleration. From Level 3 the human is not required to monitor the environment. Full automation, where the driver is not expected to intervene and the vehicle is able to handle all situations is on Level 5. In order to achieve that level the autopilot must fully understand the environment.

This is however very difficult. The algorithms that we know today are not enough to achieve understanding of the environment yet. Even Convolutional Neural Networks (CNNs) are not cabale of understanding deep concepts of the world. CNNs are mainly used in computer vision and are useful when we want to recognize patterns that appear anywhere in 2D images. Today we are able to calssify images, detect and localize objects, segment images to very high accuracy, however this doesn't mean the computer *understands* the scenes. Furthermore these algorithms are trained very specifically: To build a detection neural network (NN) first a meticulous dataset must be created that tells the algorithm what must be detected - we call this the ground truth, or training data set. Then the NN must be trained and optimized until it yields a low error on the test dataset. We call this Deep Learning due to the fact that the networks contain millions of parameters that are trained through hundreds of thousands of iterations. This is not close to what might be general AI.

In this sense we can argue about the meaning of "scene understanding". There is research going on in the direction of general AI most notably in my opinion by Yann LeCun the chief at Facebook AI and professsor at NYU, who works a concept called energy-based models. The Energy-based model that is a form of generative model allow a mathematical "bounding" or "learning" of a data distribution in any dimension. Upon prediciton the model tries to generate a possible future for the current model in time, where the generated future model acts as the prediciton itself. Generative adversarial networks are a type of these models. This is in contrast to the other main machine learning approach that is the discriminative model which is what we use mostly. Perceptrons such as NNs and CNNs, support vector machines fall into this category, however the distinction is not clear.

For the purpouse of this thesis hence it is important to define what a system capabale of understanding scences in driving situations means. The essentials are the following:

- Lane and path detection
- Driveable area detection
- Object detection: cars, pedestrians, etc.
- Object localization in 3D real world space
- Object tracking and identification
- Foreign object detection: anything that shouldn't be where it is
- Traffic light and sign understanding
- Handling occlusion of objects
- Pedestrian crossing detection
- Knowledge of surroundings and road for example with the help of high definition maps

In an ideal world, where all cars are autonomous these perceptions would be enough, however the future of self-driving cars is going to be a transition, where both humans and machines will drive on the roads. We humans already account for each other (we try as we can), but self-driving cars will have to account for us too. We might not be smart but driving on the road sometimes requires improvization to save a situation and we might need a more general AI.

For the vehicle to understand it's surroundigs first of all it needs sensors. Each company goes differently about the sensor suite, and it is quite interesting to examine each solution. I will talk about this in the next chapter, Related works.

1.1 Proposed solution

As we said to develop our system first of all we need data. There are a lot of datasets available on the internet for car driving. They include object detections, segmentations, map data, lidar data. Some of the most notable ones are the nuScenes dataset[1], Waymo dataset[2] from Google's self-driving car company or the Cityscapes dataset[3] and more. Each of these datasets are very good, however they are not really helpful for our case.

In order to localize objects in 3D space I use stereo imaging. Each AD system today employs stereo camera setting because it is a very simple and cheap but accurate way of estimating depth for each pixel in an image. In order to have the *freedom* to create a custom camera setting I cannot rely on these datasets. Furthermore, I want to measure the success rate of my detector however there is no dataset that contains all the necessary information, because in fact it is not possible to collect everything from the real world.

This is why I choose to use a *simulation* instead to test the system. Using a simulation gives a huge ammount of freedom. My research work started in looking for simulators that let me extract data from the simulation in each frame and let's me create custom world scenario and sensor settings.

After an extensive research of self-driving car simulators of I found CARLA Simulator[4] [2] (a screenshot is seen on figure 1.2) to be the most advanced one that is also opensource. CARLA is a quite mature simulator with an API that fulfills our requirements.

I set up the virtual vehcile with 10 RGB cameras mounted on the roof creating 4 stereo sides as shown on figure 1.3. As the title of the thesis says, I only used RGB cameras and no other sensors. This is a similar approach to what Tesla is taking, except for the radar sensors, contrary to almost all other players in the field who also employ a Lidar sensor for depth data including MobilEye and Waymo. Lidar data is good for correction, but it is better if the AI can equally perform using only RGB cameras, since it is a more general solution that is closer to how we humans percieve the environment.

The detector uses state-of-the-art detection, localization and segmentation model Detectron2 [3] a MASK R-CNN conv net model based on Residual neural networks and Feature Pyramid Networks trained on the COCO general dataset[5].

Finally I develop a 3D webvisualizer that lets us replay the ground truth and detection log simultaneously and compare the error between the two.

[1]nuScenes dataset https://www.nuscenes.org/
[2]Waymo dataset https://waymo.com/open/
[3]Cityscapes dataset https://www.cityscapes-dataset.com/
[4]CARLA Sim http://carla.org/
[5]COCO dataset http://cocodataset.org/

Figure 1.2: A screenshot from CARLA

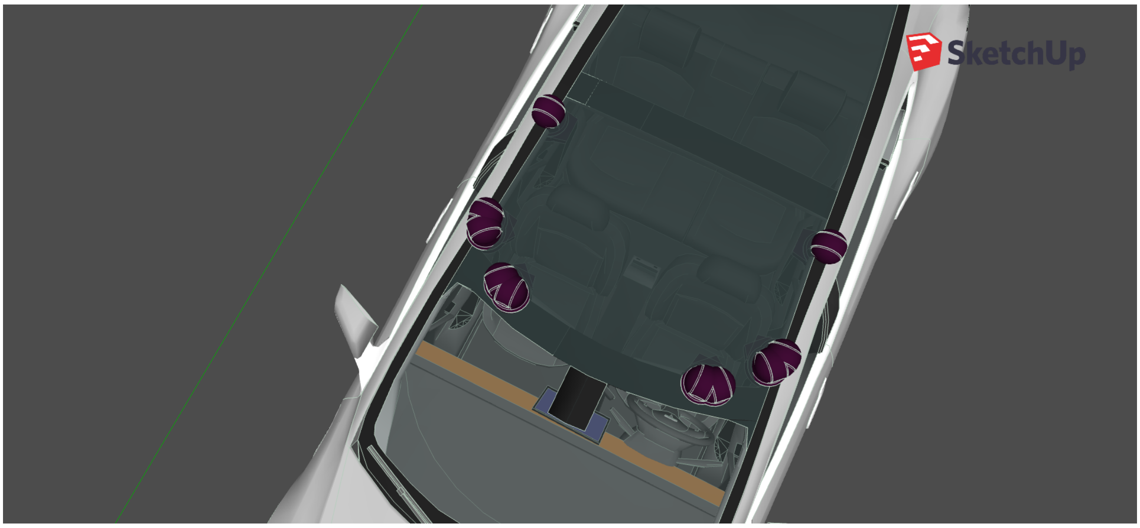

Figure 1.3: How the cameras are set up on the roof

figure 1.4 depicts this taskflow.

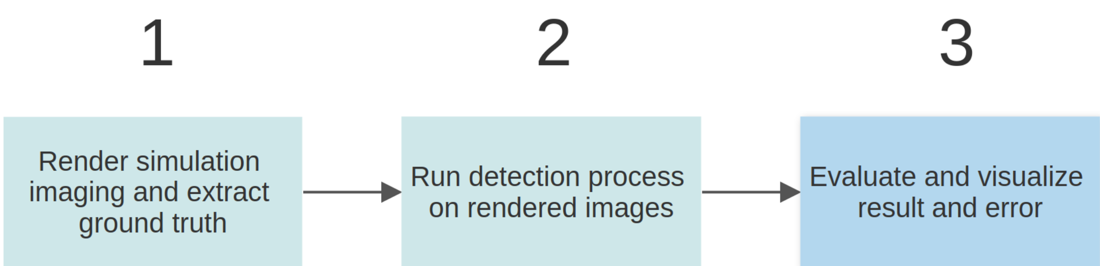

Figure 1.4: Task flow

1.2 Summary of results

The result is a detector that is capable of localizing vehicles, and pedestrians on the road up to 100 meters with an accuracy of 50cm in an angle of 270°centered to the front. The algorithm is written in Python and uses PyTorch, with that on an NVIDIA Titan X GPU the detector can perform in 2.7FPS for one side, ie. for two cameras. In an embedded optimized system using C or C++ code this can easily be improved to even 60FPS creating a real-time system. The code cannot perform lane detection yet, but that would have been the easier part. The webvisualizer let's us relplay the simulation frame by frame and see the detection error for each actor in the scene. It also shows a montage the original, detection and depthmap. Below, figure 1.5 shows a screenshot of the webvisualizer in action.

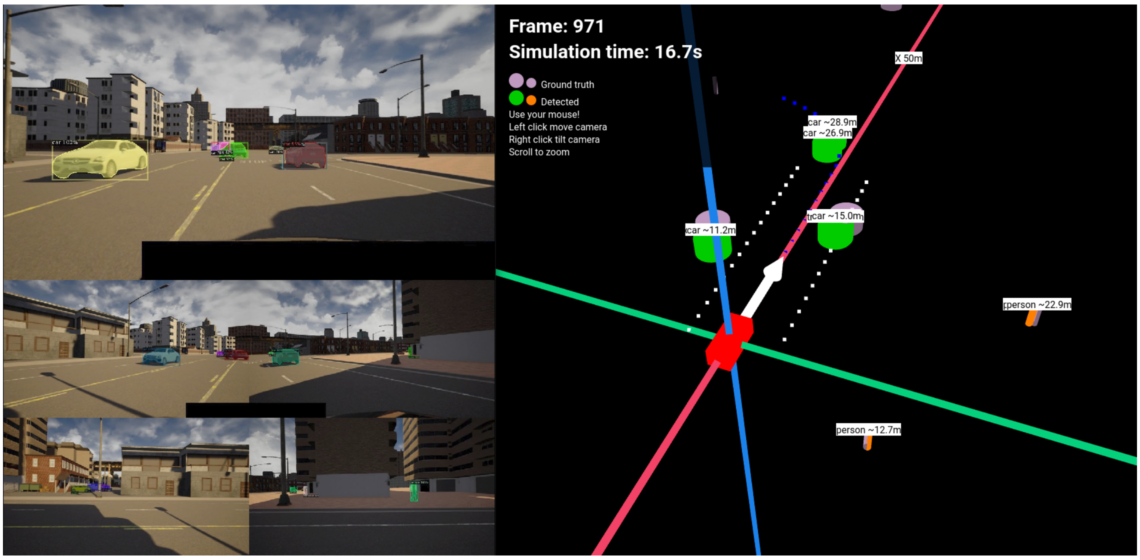

Figure 1.5: 3D wevisualizer

All of the code for the thesis, detector, simulator configuration and webvisualizer is available on https://github.com/najibghadri/msc-thesis and you can access the webvisualizer and interactively replay and test simulations on https://najibghadri.com/msc-thesis/.

1.3 Thesis structure

In Chapter 2 I give an overview of the widely used sensors for peception in the automotive industry: RGB cameras, radar, Lidar and ultrasonic sensors. In Chapter 3 I talk about different kinds of perceptions, state-of-the-art Convolutional Networks and computer vision algorithms that are useful for our use-case.

In Chapter 4, I analyze and compare different self-driving car solutions: Tesla and Waymo self-driving cars and MobilEye autopilot. In Chapter 5, I introduce CARLA Simulator and some notable features of it.

In Chapter 6 I define the technical assumptions that I made in order to simplify the task and the resulting limitations.

Chapter 7 introduces the Carla simulator details the design and implementation of the simulator configuration, the detector algorithm and the webvisualizer.

Then in Chapter 8 I present different measurements and results, and in Chapter 9 I present experimentations that ended up not being part of the detection. Finally I discuss ways to improve the system in Chapter 10 and close with a conclusion.

Chapter 2

Sensors

Selecting the right sensors to understand the environment is half the task. Combining multiple sensors to collect data for further information extraction is called sensor fusion. In this chapter we are going to detail the most widely used sensors for scene understanding for autonomous vehicles and compare them.

Radar, utrasonic and LiDar sensors basically all work the same: emit a wave, wait until it returns and estimate the distance based on the time difference, and estimate the speed calculating the frequency shift - this is the Doppler effect: an increase in frequency corresponds to an object approaching and vice versa. A visualization is seen on figure 2.1.

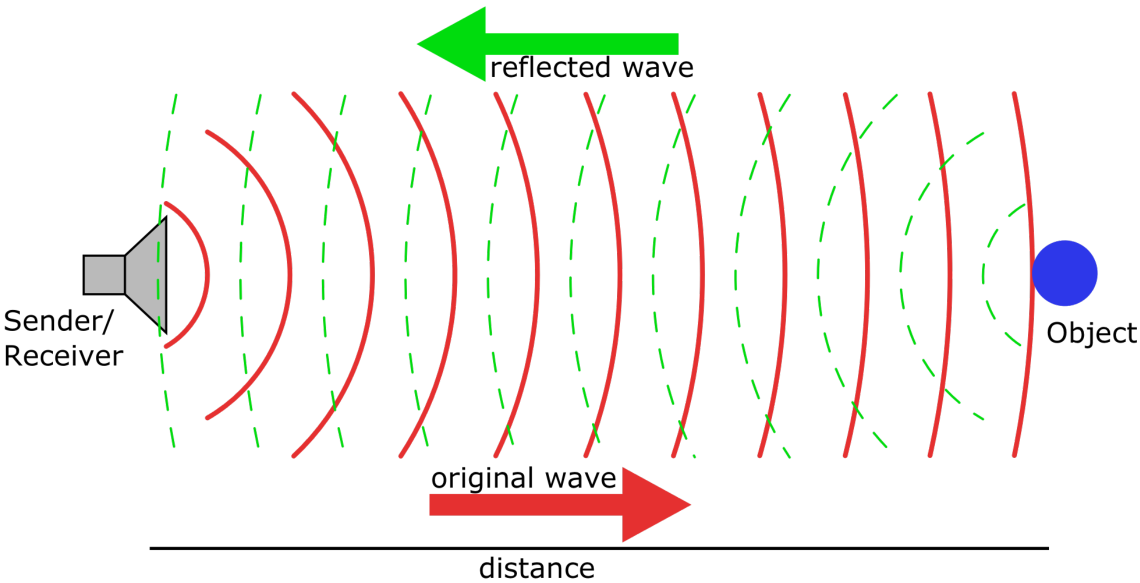

Figure 2.1: Sensing object with wave emission and reflection

Thus calculating the distance is a simple equation:

$$Distance = \frac{Speed\ of\ wavefrom * Time\ of\ Flight}{2} \tag{2.1}$$

However they use different waves: Radar works with electromagnetic waves, ultrasonic sensors work with sound waves and LiDar works with laser light.

2.1 Radar

Radar sensors at the front, rear and sides have become an essential component in modern production vehicles. Though most frequently used as part of features like parking assistance and blind-spot detection, they have the capability to detect objects at much greater range – several hundred meters in fact.

Radar sensors are excellent at detecting objects, but they're also excellent for backing up other sensors. For instance, a front-facing camera can't see through heavy weather. On the other hand, radar sensors can easily penetrate fog and snow, and can alert a driver about conditions obscured by poor conditions. Radar is robust in harsh environments (bad light, bad weather, extreme temperatures).

Automotive radar sensors can be divided into two categories: short-range radar (SRR), and long-range radar (LRR). The combination of these types of radar provides valuable data for advanced driver assistance systems.

Short-range radar (SRR) Short-range radar (SRR): Short-range radars (SRR) use the 24 GHz frequency and are used for short range applications like blind-spot detection, parking aid or obstacle detection and collision avoidance. These radars need a steerable antenna with a large scanning angle, creating a wide field of view.

Long-range radar (LRR) Long-range radar (LRR): Long-range radars (LRR) using the 77 GHz band (from 76-81GHz) provide better accuracy and better resolution in a smaller package. They are used for measuring the distance to, speed of other vehicles and detecting objects within a wider field of view e.g. for cross traffic alert systems. Long range applications need directive antennas that provide a higher resolution within a more limited scanning range. Long-range radar (LRR) systems provide ranges of 80 m to 200 m or greater.

2.2 Ultrasonic

Ultrasonic (or sonar) sensors like radar, can detect objects in the space around the car. Ultrasonic sensors are much more inexpensive than radar sensors, but have a limited effective range of detection. Because they're effective at short range, sonar sensors are frequently used for parking assistance features and anti-collision safety systems. Ultrasonic sensors are also used in robotic obstacle detection systems, as well as manufacturing technology. In comparison to infrared sensors in proximity sensing applications, ultrasonic sensors are not as susceptible to interference of smoke, gas, and other airborne particles (though the physical components are still affected by variables such as heat), and they are independent of light conditions. They also work based on reflected emission.

Ultrasound signals refer to those above the human hearing range, roughly from 30 to 480 kHz. For ultrasonic sensing, the most widely used range is 40 to 70 kHz. At 58 kHz, a commonly used frequency, the measurement resolution is one centimeter, and range is up to 11 meters. At 300 kHz, the resolution can be as low as one millimeter; however, range suffers at this frequency with a maximum of about 30 cm.

You can see the sensor suite of Tesla figure 2.2 from Tesla Autopilot website [1].

[1] Tesla autopilot https://www.tesla.com/autopilot

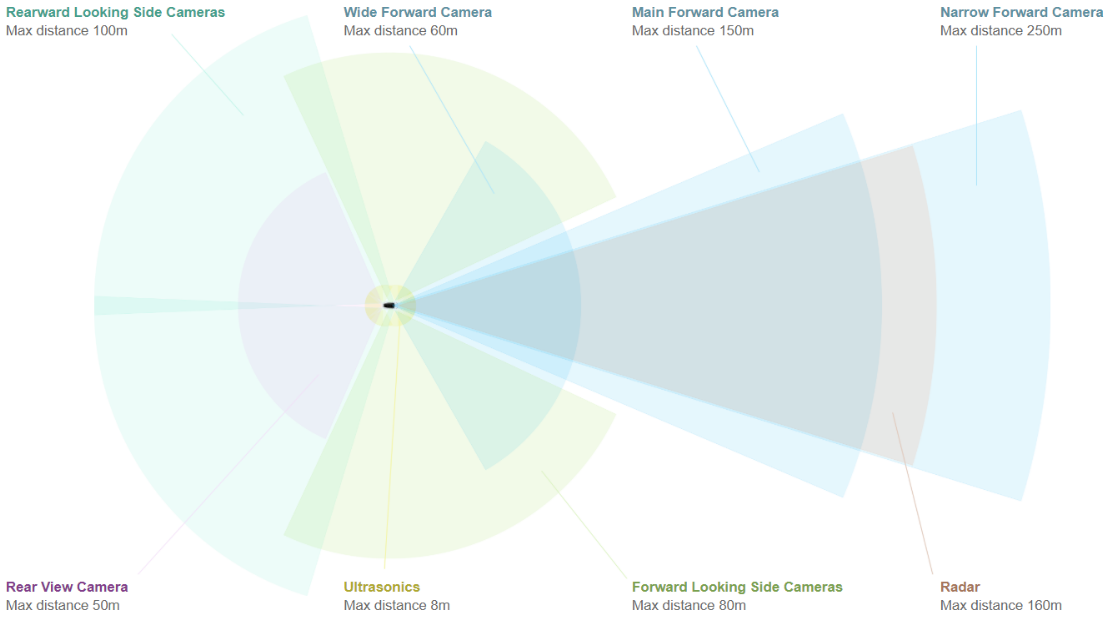

Figure 2.2: Tesla sensor suite infographic

2.3 LiDAR

As Radar is to radio waves, and sonar is to sound, LiDAR (Light Detection and Ranging) uses lasers to determine distance to objects. Lidar sometimes is called 3D laser scanning. It does this by spinning a laser across its field of view and measuring the individual distances to each point that the laser detects. This creates an extremely accurate (within 2 centimeters) 3D scan of the world around the car.

The principle behind LiDAR is really quite simple. Shine a small light at a surface and measure the time difference it takes to return to its source. The equipment required to measure this needs to operate extremely fast. The LiDAR instrument fires rapid pulses of laser light at a surface, some at up to 150,000 pulses per second. A sensor on the instrument measures the amount of time it takes for each pulse to bounce back. Light moves at a constant and known speed so the LiDAR instrument can calculate the distance between itself and the target with high accuracy. By repeating this in quick succession the insturment builds up a complex 'map' of the surface it is measuring.

The three most common currently used or explored wavelengths for automotive lidar are 905 nm, 940 nm and 1550 nm, each with its own advantages and drawbacks.

Lidar sensors are able to paint a detailed 3D point cloud of their environment from the signals that bounce back instantaneously. It provides shape and depth to surrounding cars and pedestrians as well as the road geography. And, like radar, it works just as well in low-light conditions.

You can see how a lidar sensor from Luminar[2] reconstructs the environment in figure 2.3.

Currently, LiDAR units are very big, and fairly expensive - as much as 10 times the cost of camera and radar — and have a more limited range. You will most often see them mounted on Mapping Vehicles, but as the technology becomes cheaper, we might see them on trucks and high-end cars in the near future.

[2]Luminar https://www.luminartech.com/

Figure 2.3: Luminar LiDAR in action

2.4 RGB Cameras

2.5 GPS & WPS

The Global Positioning System is the perfect example of how sensor technology grows smaller and more ubiquitous over time. Originally introduced for military applications in 1974, GPS probes today can be found in cameras, watches, key fobs, and of course, the smartphone in your pocket.

The lesser-known WPS stands for Wi-Fi Positioning System, which operates similarly. When a probe detects satellites (GPS) or Wi-Fi networks (WPS), it can determine the distance between itself and each of those items to render a latitude and longitude. The more devices a GPS/WPS probe can detect, the more accurate the results. On average, GPS is only accurate to around 20 meters.

For WPS the most common and widespread localization technique is based on measuring the intensity of the received signal, and the method of "fingerprinting". Typical parameters useful to geolocate the wireless access point include its SSID and MAC address. The accuracy depends on the number of nearby access points whose positions have been entered into the database. The Wi-Fi hotspot database gets filled by correlating mobile device GPS location data with Wi-Fi hotspot MAC addresses.

Chapter 3

Computer vision

After collecting data from the sensors we choose we need to implement the right algorithms to extract information from the sensor data. In this chapter I start with explaining basics of computer vision and then move on to advanced convolutional neural netowrks that will help our goal.

Computer Vision, often abbreviated as CV, is defined as a field of study that seeks to develop techniques to help computers “see” and understand the content of digital images such as photographs and videos.

The problem of computer vision appears simple because it is trivially solved by people, even babies. Nevertheless, it largely remains an unsolved problem based both on the limited understanding of biological vision and because of the complexity of vision perception in a dynamic and nearly infinitely varying physical world.

CV algorithms can be categorized into image processing and computer vision algorithms. The goal of image processing is to produce an image which is more advantageous for our purposes. Image processing is often used to prepare images for further analysis . On the other hand, the goal of computer vision is to transform the input image to a higher abstraction level, thus to extract and extract information from the image in a more compact manner. To further differentiate, the term machine vision is used if these methods are applied in embedded systems (cars, robots, production machines, smartphones, etc.).

Due to recent results in machine learning-aided computer vision applications, these algorithms form a separate domain called learning vision. The most dynamically advancing branch of learning vision utilizes deep learning. Other computer vision techniques and algorithms are called tradition vision.

Traditional vision applications consist of four main steps, the first of which is the recording of images sequences. Usually there is a second step which is image correction, where we might reduce noise or apply effects to the images. We will see later that in our case wil will need to occlude parts of the image where the ego car appears in order that it won’t detect itself as another car. In the third step task-relevant features are extracted, properties which can help solve the given problem. In the last step, called decision-making, the applied algorithm uses the available data to determine the final output.

3.1 Challenges in Computer Vision

Object detection is considered to be the most basic application of computer vision. Rest of the other developments in computer vision are achieved by making small enhancements on top of this. In real life, every time we, humans open our eyes, we unconsciously detect objects.

Since it is intuitive for us, we fail to appreciate the key challenges involved when we try to design systems similar to our eye. Some challenges for computers are:

- Variations in viewpoint
- Difference in illumination
- Hidden parts of images, occulsion
- Background Clutter

3.2 Traditional approaches

Various techniques, other than deep learning are available enhancing computer vision. Though, they work well for simpler problems, but as the data become huge and the task becomes complex, they are no substitute for deep CNNs. Let's briefly discuss two simple approaches.

3.2.1 KNN (K-Nearest Neighbours)

In the KNN algorithm each image is matched with all images in training data. The top K with minimum distances are selected. The majority class of those top K is predicted as output class of the image. Various distance metrics can be used like L1 distance (sum of absolute distance), L2 distance (sum of squares), etc. However KNN performs poorly - qute expectedly - they have a high error rate on complex images, because all they do is compare pixel values among other images, without any use of image patterns.

3.2.2 Linear Classifiers

They use a parametric approach where each pixel value is considered as a parameter. It's like a weighted sum of the pixel values with the dimension of the weights matrix depending on the number of outcomes. Intuitively, we can understand this in terms of a template. The weighted sum of pixels forms a template image which is matched with every image. This will also face difficulty in overcoming the challenges discussed in earlier as it is difficult to design a single template for all the different cases.

3.3 Neural networks

3.4 Deep Learning

3.5 Convolutional Neural Networks

3.6 Detection and Segmentation

3.6.1 Object Classification Detection, Localization

(AlexNet, LeNet, VGG) R-CNN, Fast, Faster Segmentation Networks Mask R-CNN - Detectron2 Detectron YOLO

3.6.2 Segmentation

3.7 Bounding box detection and orientation

3.8 Key point detection

3.9 Voxelization

PointNet VoxelNet

3.10 Tracking

others SORT Deep Sort

3.11 Lane and road detection

Road detection Driveable Road Lane detection: sliding window, curve fit

3.12 3D vision

Camera model and Calibration 3D reconstruction Stereo vision Depth estimation

3.13 Datasets

Datasets - KITTI, MARS, COCO, Waymo, nuScenes

Basics: depth from radar/stereo cameras: I choose only cameras

Chapter 4

Related work

It is important for a self-driving company to openly detail their technical solution because it let's people trust the autopilot.

4.1 Tesla

- Miles done - Risk - Tesla eight cameras, 12 ultrasonic sensors, and one forward-facing radar. - Their view on simulations expensive.

4.2 Waymo

4.3 MobilEye

- do some pros/cons - Other Simulations - The simulation idea for dataset and ground truth instead of dataset - Drawbacks, limitations - Pros cons

Chapter 5

CARLA Simulator

CARLA's mission is to create a simulator that can simulate sufficient-enough real-world traffic scenarios so that it is more accessible for researchers like myself to research, develop and test computer vision algorithms for self-driving car.

CARLA[2] is an open-source simulator for autonomous driving research. It is written in C++ and provides a very accessible Python API to control a lot of the simulaton execution. It has been developed from the ground up to support development, training, and validation of autonomous driving systems. In addition to open-source code and protocols, CARLA provides open digital assets (urban layouts, buildings, vehicles) that were created for this purpose and can be used freely. The simulation platform supports flexible specification of sensor suites, environmental conditions, full control of all static and dynamic actors, maps generation and much more. It is developed by the Barcelonian university UAB's computer vision CVC Lab and supported by companies such as Intel, Toyota, GM and others. The repository for the project is at https://github.com/carla-simulator

It provides scalability via a server multi-client architecture: multiple clients in the same or in different nodes can control different actors. Carla exposes a powerful API that allows users to control all aspects related to the simulation, including traffic generation, pedestrian behaviors, weathers, sensors, and much more. Users can configure diverse sensor suites including LIDARs, multiple cameras, depth sensors and GPS among others. Users can easily create their own maps following the OpenDrive standard via tools like RoadRunner. Furthermore it provides integration with ROS[1] via their ROS-bridge

I used CARLA 9.8.0 in the project that was the latest at the time (2020 March 09). Carla has a primary support for Linux so I could run it easily on Ubuntu. It requires a decent GPU otherwise the simulation is going to be slow.

It's important to mind the coordinate system used in Carla, because later when we will extract data the axes must be mapped to the correct data points. Since Carla is built with Unreal Engine [2] it uses the coordinate system as in figure 5.1: X coordinate is to the front of the ego actor, Y is to the right of ego and Z is to the top.

[1]Robot Operating System (ROS) https://www.ros.org/
[2]Unreal Engine https://www.unrealengine.com/

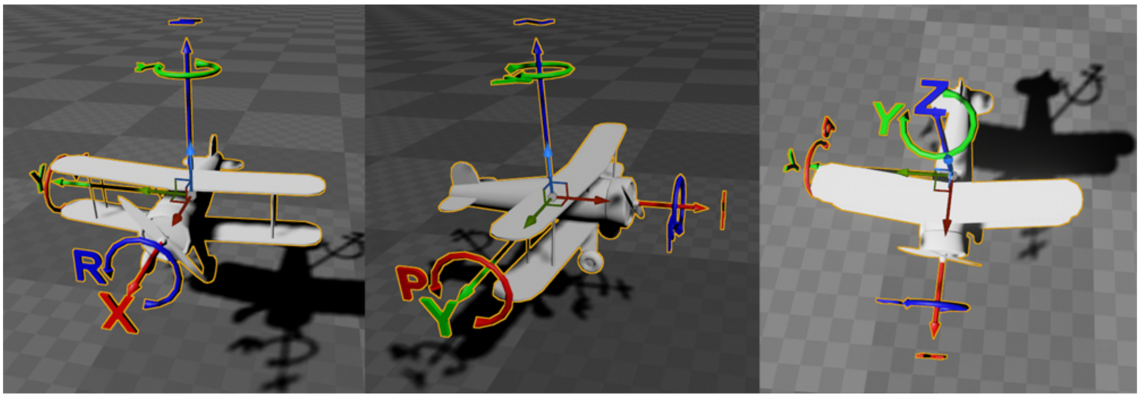

Figure 5.1: Carla coordinate system

5.1 Is a simulation enough?

I believe the future of self-driving car research and development is in part with simulations and in part with real-world training as well. To develop a self-driving AI from ground up it is certainly advisable to first develop and test the algorithms in a simulation.

In order to create simulations that are rich and different Carla provides a large variety of actors and maps. The traffic manager can also be parametrized to control how pedestrians and vehicles move: their speed, minimum distance, and even "aggressivity" towards each other, which means how willing are they to collide instead of waiting until the actor in front moves away. This is actually useful as it helps unlock possible traffic deadlocks. The latest CARLA provides 8 maps but in newer versions they will be adding new maps. You can see a screenshot of each rendering in the 6 maps I used in figure 5.2.

Figure 5.2: Variety of maps in Carla

A simulation obviously can't return the variety and exact nature of scenarios that happen in *nature*. However I believe they are sufficient for testing an entry-level self-driving system and that with the use of simulations a company can lower the costs of development. The rise of simulators itself shows there is a need for the market.

5.2 CARLA Simulation sensors

The Carla simulator's API support a wide range of sensors: RGB Cameras, LiDAR, Radar, GPS, gyroscope, accelerometer, compass and more. These are very easy to use, If you are interested I recommend reading the sensors reference in their documentation [3]

Carla also provides miscellaneous sensors that help collecting ground-truth data for deep learning applications. This includes semantic segmentation camera, depthmap camera and other simple ones such as collision detector. figure 5.3

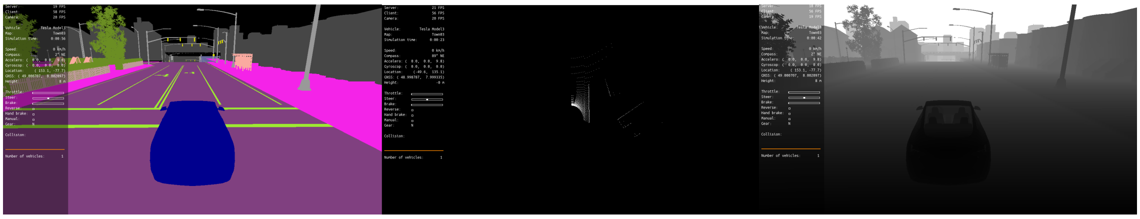

Figure 5.3: Different sensors and cameras in Carla (semantic segmentation, lidar, depthmap)

5.2.1 Other simulators

There are a couple of other dedicated projects for simulators. There is Deepdrive from Voyage auto[4], an American AD supplier, NVIDIA has a project going on called Drive Constellation[5] which is said to be very advanced but is not opensource, and another project called RFPro[6]. However these are either not opensource or not mature enough. CARLA Simulator[7] was by far the best one.

[3]CARLA sensors reference https://carla.readthedocs.io/en/latest/ref_sensors/
[4]Deepdrive Voyage https://deepdrive.voyage.auto/
[5]NVIDIA Drive Constellation https://developer.nvidia.com/drive/drive-constellation
[6]RFPro http://www.rfpro.com/
[7]CARLA Sim http://carla.org/

Chapter 6

Assumptions made and limitations

In order to simply the task of scene understanding we need to define boundaries to measure the success of the detector.

6.1 Daylight situation

First of all we are going to specialize to day-light situations only. This detection with RGB cameras at night is difficult, in order to achieve that we need other sensors such as Radar, Sonar or LiDAR. As we are only using RGB cameras we arge going to assume that all driving situations occur in daylight.

6.2 Flat plane assumption

Another important assumption is that the driving field and landscape area is flat. It isn't difficult to detect object that are a bit higher on the picture but it is difficult to recognize the curvature of the plane on the image. In case the detector can interpret curvature and the ego car is on an angled road the angle data from the gyroscope sensors has to be take into account and subtracted from the percieved angles. It is generally true that inorder to recognize true information about the world the relative position and orientation has to be taken into account.

In order to reduce this complexity, we are going to only take into account the objects' position on planar coordinates.

6.3 Path, lane and road detection

As described before there are many ways of detecting lane and the easiest is to use the Hough transform and detect the lanes directly in front of the car. However this is not a robust solution: this only gives good results in good illumination and weather situations. It is true that most situations are like this but there are still many unpainted roads, dirt roads or simply due to lightning and weather the lane edges won't be clear.

One robust solution would be to take into account the vehicles in front and behind us and interpret their path as the right path and regress the lane to their path.

Another solution is to take into account previously driven paths. This is the approach Tesla takes however it is not clear how exactly.

6.4 Orientation

It would be important to

6.5 Tracking

- No tracking - No consistency through time

6.6 Only detection

No control - just scene understanding: simple steer control explain

Chapter 7

Design and implementation

Let's recap the task flow of the task I described in the Introduction: After configuring the simulator with the designed camera setting I render multiple traffic scenarios in different maps provided by CARLA while extracting all necessary information into a log file to later compare the detection log with

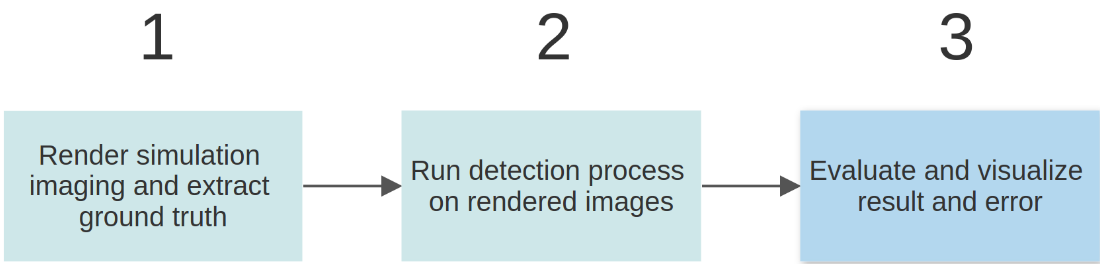

Figure 7.1: Task flow

7.1 Tools used

Very soon it became obvious that a Linux operating system is the right way to develop everything. I have been using Ubuntu before this project as well so I was already familiar with everything. The main IDE I used throughout the project is Visual Studio Code, which thanks to it's openness and community has a lot of extensions that helped me develop in fact every part of the thesis: Python, Nodejs and Javascript for the webvisualizer and finally LaTeX and ofcourse git support.

I also used Conda which is I think an essential tool when you want to develop ML and AI projects with Python. Conda makes it easy to create and use separate Python environments. This is very important because different implementations of algorithms require different versions of the same packages thus it keeps a clean separation. The drawback is that consecuently it requires a lot of drive memory.

Upon developing the algorithm and experimenting with it I used Jupyter Notebook which is a Python runtime on top of the bare one and a web-based IDE at the same time. With Jupyter Notebook it is very easy to change and re run the code thanks to it's "kernel" system, which keeps the value of variable and imported packages between executions.

For the GPU-intensive tasks such as simulation and convnet calculations in the detector I was provided with a remote Titan X GPU[1] by my university.

7.2 Choosing the sensor suite

Mounting cameras around the vehicle to have an all around vision is an essential design strategy, as we have seen in the work of other companies in Chapter 4. However we will need to determine depth as well. I decided to use only cameras in a stereoscopic structure to create 4 stereo sides around the vehicle. The following image shows the design setting with field of views visualized in figure 7.2.

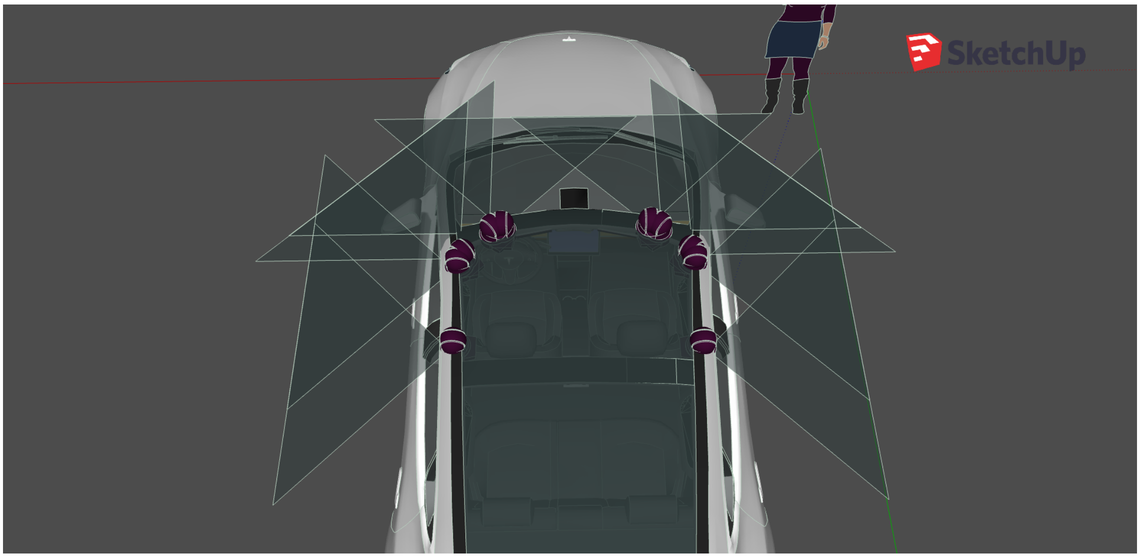

Figure 7.2: The stereo camera setting I used on top of the virtual Tesla Model 3

In details:

- Front stereo: two cameras looking straight to the front 0.8 meters apart
- Right corner and left corner stereo cameras: the cameras are on the diagonal corners of a 20 cm wide 20cm tall triangle creating two 45°angled stero vision.
- Right and left side stereos are turned 90°to the sides and they are apart 0.5 meter.

The cameras are 1.5 meters above the ground.

The advantage of puting stereo cameras apart to a relatively large distance is that it increases the accuracy of the stereo block matching algorithm to a further distances. The drawback however is that a smaller portion of the right and left side images are going to intersect hence creating a smaller field of view. However due to the corner stereo cameras this is not a problem for us.

[1] Titan X GPU https://www.nvidia.com/en-us/geforce/products/10series/titan-x-pascal/

7.3 Configuring the simulation

Carla simulator can be ran in two time-step settings: variable and synchronous. In real-world perception it is a complex task by itself to synchronize multiple cameras with each other so that when the algorithm calculates information based on data from multiple sensors they all correspond to the same moment in time with an error boundary. In a simulation however we can have the freedom to synchronize the simulation timesteps themselves and collect all imaging data between each timestep. Setting Carla to synchronous timestep ensures that all images in a certain frame are collected and respond to the same moment.

I used 30FPS timestep setting so that physics calculations are still realistic but the performance is not too bad. We also have to account for the size of the generated images: it was good to half the size of the image datasets from a 60FPS setting. Increasing the traffic participants also degrades the performance. I usually used 200 vehicles and 100 pedestrians for each map, that resulted in realistic traffic scenarios.

I recorded different scenarios of approximately 1 minute, which means 1800 frames on 30FPS. On the Titan X machine it it took 15 minutes to render 1 simulation minute, i.e. it ran the simulation with 2FPS. Note, this is different from the simulation time-step which we fixed to 30FPS. Since I collect 10 images in each frame it results in a dataset of 18000 images.

The camera setting I used is an undistorted camera that takes 1280x720 resolution images, i.e HD 720p images, compressed with JPEG to yield a reasonable size. This way one image is on average 215 kilobytes instead of 1MB which is a very good compression rate and this was the limit where I did not see any difference in detection accuracy.

In a real-world systems images go straight to the GPU and CPU unit and they get downscaled to the choosen size before feeding into the algorithm. I had to resort to compression because of the research nature of the project: I reran and tested the accuracy of the detector many times on the same dataset.

Using an undistorted camera matrix only means that we need to use one less back transformation matrix in the detection calculations. In real-world the intrinsic camera matrix is calculated and corrected for cameras that are mounted on cars and it is part of the calculation.

Besides imaging we have the ground truth log data. During the simulation, besides rendering images I coded a logger that logs the necessary information of the state of the simulator for each frame. This information is built up in a json-like dictionary, and at the end of the simulation it is saved to one file, that I call the framelist.

7.4 Extracted data

Naming the images in an organized way is important to make it easy to read the images in a structured way upon detection. Each image starts with the number of the frame it was taken in. Starting the simulator server Carla increases a frame counter starting with 1. To know which image corresponds to which camera, the framenumbers are postfixed with a label. figure 7.3 shows the postfixes for each image.

In each frame I log information about the current state of the simulation. For the purpouses of the final detector the following information gets logged in each frame:

Figure 7.3: L2/1, R1/2: Right side/Left side first and second cameras, LC(2/1), RC(1/2): Right corner, left corner cameras, FL FR: Front left, front right cameras

- Frame's number: the value of the frame counter at each frame
- For all walker and vehicle actors in a 100 meter radius from the ego car:
  - Id: corresponds to the actor's unique id among other actors.
  - Relative position: X, Y, Z coordinate of the actor in the CARLA coordinate system (see figure 5.1)
  - Distance: Euclidean distance from the ego car
- Waypoints: these are center and left-right points of the lane the egocar is currently in up to 30 points forward. These were meant to be the ground-truth data for lane-detection

This information is then exported into a JSON file with the following format:

```
frameList: [
    {
        frame: Number,
        actors: [
            {
                type: car|pedestrian,
                id: Number,
                relative_position: {
                    x: Number,
                    y: Number,
                    z: Number,
                }
            },
        ],
    },
]
```

For a one-minute simulation the ground-truth json file is approximately 20 megabytes. It isn't optimal to save information like this for longer simulations. In those cases it is recommended to use a binary format. Carla provides a way to save binary information of the recording but unfortunately there were issues with recording that way, so I ended up with this custom log format. However it ended up being very beneficial, because the webvisualizer simply loads the json files (detection and ground truth) into two JavaScript objects.

7.5 Detector

The plan: three images of detection montage!!!

7.5.1 OpenCV

Intro to it

7.5.2 Detectron2

- Occulsion - Detection images! - Pretrained models - Comparison - Object detection and localization - Instance segmentation

7.5.3 Depth estimation

7.5.3.1 Triangulation

- Triangulation

7.5.3.2 Stereo Block Matching Algorithm

- Stereo Block Matching Algorithm (newer)

7.5.3.3 Result

-

7.5.4 Back projection

- Camera model and coordinates

- Camera setting - Inverse transformation explain, Translation: same matrix as camera why - Yaw pitch roll, euler matrix - Affine matrix

7.5.5 Final pseudo-code

Final pseudo code

```
pseudo code
```

7.6 Web visualizer

In order to measure the accuracy Web visualizer - Framework - Usage and results

7.7 Additonal scripts

-Start script -Montage script

Chapter 8

Results

8.1 Accuracy

- Explaining errors - Fine tuning: - Results I am proud of - Precision, recall acc, danger

8.1.1 Fine tuning

- Depth mean vs mode Table here - All sides: FPS avg: 0.53 FPS TITAN X - If saving pictures: FPS avg: 0.29 - One side: - Three sides: - FPS of one side my computer vs Titan X - Different models and their accuracy and FPS one side - Mask R CNN

8.2 Free Z coordinate

- Car tilt problem - Carla position problem - Z coordinate hack explain why its ok, CARLA issue Show the difference!

8.3 Night results

8.4 Hardware requirements

- Dangerousness - Hardware requirements

For more results visit najibghadri.com/msc-thesis

Chapter 9

Experimental results

9.1 Tracking

9.2 YOLO

9.3 Lane detection

9.4 3D Bounding box detection

9.5 Keypoint detection

9.6 Night results

Chapter 10

Improvement notes

10.1 Optimal sensor suite

Less sensors: rectified cameras - exo stereo

10.1.1 Data correction

Car position, tilt, velocity detection and correction, odometric correction

10.2 Tracking and correlation

10.3 Depth correction

Size based depth correction Parallax motion based depth correction

10.3.1 Size based

10.3.2 Monodepth

10.3.3 Parallax motion

10.4 Lane, path and road detection

Road segmentation, path based on other actors Drivable area reconstruction from other actors - more robust

10.5 unsupervised learning methods

Energy based method - Yann LeCun - Latent space for possible outcomes - Traffic situation understanding - Surrounding understanding

10.6 Keypoint based detection and orientation

Orientation, keypoint detection, wheel, etc detection

10.7 3D reconstruction

Voxel reconstruction of actors

10.8 Traffic light understanding

10.9 Foreign object detection

White list based - difficult problem! (https://link.springer.com/article/10.1186/s13640-018-0261-2)

Chapter 11

Conclusion

Working on this thesis has been a very unique experience because the whole filed was new to me before getting into it. Usually thesis projects require that the student works on the same project for 4 semesters, however I took a different road unfortunately or not. I did my previous research work in Web APplications and Applied blockchain technology. Then I took an optional a deep learning class and it sparked my interest for AI even more. Taking this project was a risk and I had to learn about basic computer vision processing methods, algorithms, 3D vision, the camera model, convolutional neural networks and deep learning and even a little bit of game engines because of the simulator. But in the end I learned a lot of things and I hope I can use this knowledge soon in a nice AI company perhaps one that works on autopilots.

The final scene understanding algorithm is not a system that can be applied by itself in a real scenaro, however it builds on the same basic ideas for scene understanding for cars. The work of companies like Tesla and Waymo constitues many top researchers in the field. In Hungary this market is yet in very early stages but companies like BOSCH or a smaller company like AIMotive are already present and working on the field with a good pace.

Acknowledgements

Ez nem kötelező, akár törölhető is. Ha a szerző szükségét érzi, itt lehet köszönetet nyilvánítani azoknak, akik hozzájárultak munkájukkal ahhoz, hogy a hallgató a szakdolgozatban vagy diplomamunkában leírt feladatokat sikeresen elvégezze. A konzulensnek való köszönetnyilvánítás sem kötelező, a konzulensnek hivatalosan is dolga, hogy a hallgatót konzultálja.

Bibliography

[1] J3016B: Taxonomy and Definitions for Terms Related to Driving Automation Systems for On-Road Motor Vehicles - SAE International. URL https://www.sae.org/standards/content/j3016_201806/.

[2] Alexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio Lopez, and Vladlen Koltun. CARLA: An open urban driving simulator. In *Proceedings of the 1st Annual Conference on Robot Learning*, pages 1–16, 2017.

[3] Yuxin Wu, Alexander Kirillov, Francisco Massa, Wan-Yen Lo, and Ross Girshick. Detectron2. https://github.com/facebookresearch/detectron2, 2019.

Appendix

http://deeplearning.iit.bme.hu/notesFull.pdf