DEPARTMENT OF INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Master's Thesis in Informatics

Development of an autonomous parking application for L4 Fiasco.OC / Genode using Microsoft AirSim and Matlab Simulink

Kimiya Beheshti shirazi

DEPARTMENT OF INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Mastesr's Thesis in Informatics

Development of an autonomous parking application for L4 Fiasco.OC / Genode using Microsoft AirSim and Matlab Simulink

Entwicklung einer Autonomen Einpark-Anwendung für L4 Fiasco.OC/Genode unter Verwendung von Microsoft AirSim und Matlab Simulink

Author:	Kimiya Beheshti Shirazi
Supervisor:	Prof. Dr. Uwe Baumgarten
Advisor:	Sebastian Eckl, M.Sc
Submission Date:	16. August 2019

I confirm that this master's thesis is my own work and I have documented all sources and material used.

Munich, 16.08.2019 Kimia Beheshti Shirazi

Acknowledgments

I would like to thank Prof. Uwe Baumgarten for giving me the opportunity to write this thesis at his chair and also for all of his supports and advise during this work.

Many thanks to my advisor, Sebastian Eckl for his invaluably wise advice, his patience and in particular for the spontaneous and fruitful discussions.

Finally, I want to thank David Werner to generously share his experience in autonomous parking with me and all the other persons in the Chair of Operating Systems for the welcoming and supportive atmosphere.

Abstract

Autonomous Parking is an interesting field of research in autonomous driving because parking, and especially parallel parking, has always been a challenge for drivers, even experts. Although study on autonomous parking has been ongoing for more than a decade, it is still open for new ideas and researchers because most of the proposed methods are not completely practical or suitable to apply on real vehicles on public streets. When we speak about autonomous parking, we always confront two research aspects. First is parking detection and research on algorithms to separate vacant and occupied parking spots. The second is path planning and motion generation algorithms to perform parking maneuvers. This thesis presents an implementation of a parallel parking algorithm. For this, a complete parking scenario which includes all parallel parking steps from finding a parking place to controlling the vehicle to move it into the detected parking place. For the parking vacancy detection step, the ACFObjectDetector from the Matlab Computer Vision Toolbox was, used and for the motion generation step a parallel parking algorithm for a non-holonomic vehicle has been implemented. The motion planning algorithm is based on distance measurements with respect to the vehicle's environment and surrounding objects. The approach has been implemented on Carla simulator by using python as API.

Contents

1. Introduction

Autonomous driving has attracted a great deal of attention either in research organizations or in industry. In recent years, important car manufactures all over the world have invested lots of money to produce semi-automatic or even full-automatic vehicles. Among many of the autonomous driving areas that researchers has been working in these days, Autonomous Parking (AP) is one of the oldest subject in autonomous driving. One of the first idea for car parking was proposed in 1934 [1]. However, this research did not lead to a produced vehicle. One of the successful AP project presented by the researchers from (Institute for Research in Computer Science and Automation (INRIA)) in mid of 1990. They applied this method on an electric car called Ligier [2]. Many researchers impressed by their algorithm and new algorithms has been published based on this method i.e, [3]. However, most of the parking algorithms has been proposed theoretically and not all of them practically used for parking. Among companies who applied Autonomous Parking in their produced cars, we can point to Toyota, Ford, Lincoln, Jeep,Volkswagen, BMW, Mercedes-Benz and AUDI. However, their systems still depends on human and accelerating/braking inputs are not fully automated. Bosch company has recently planed to produce a fully automated parking system which allows the driver to leave the car and activate parking from his smartphone [4]. Apart from the commercial issues and the fact that most of these companies should adapt to new technologies to survive beside their competitors, what are the benefits of ADAS and AP for drivers or in general for passengers? One of the arguments of making Autonomous Vehicle (AV) is to save the fuels because most of the AVs are powered by electricity instead of gasoline [5]. Other studies shows that most of the accidents are because of human's error like driver's inattention, distraction, stress or tiredness. In particular, AP could help drivers to find parking spots easily because drivers do not have a wide view of the parking or maybe there are parking spots which are not obvious because either they are far from the driver or they are hidden by other cars or objects [6]. AP offers more efficient parking by reducing waiting time of passengers to search for parking spot. After they arrive to the destination they could just drop by and vehicle will search for a parking place by itself.
Autonomous parking consists of two main problems: detection and maneuver. Many researches has been done in both areas as will be described in details in the next chapter 3. The methods have been defined for detection phase, are either based on sensors or visual features. Hence, detection phase itself can be categorized into 2 groups: free-space-based and parking-slot-marking-based methods [7]. Free-spaced-base methods use sensors to find a free space between adjacent vehicle and most of them use UltraSonic sensors. Parking-slot-marking-based methods tries to use visual features like line markers from the parking lot which is also called marking points. In maneuver step the problem is to find a perfect algorithm to control maneuvering of the vehicle to detect parking lot. Many researches have

been also devoted to this area. Most of the parking maneuver algorithms are based on these two methods: tracking and posture stabilization [8]. The idea of first method is to design a control path in which an actor (Robot or vehicle) should follow this path as a reference trajectory. Second one tries to stabilize the position of the actor to a desired final posture. This thesis presents a research on whole parking process from detection phase to maneuver phase and specifically considers parallel parking implementation.

1.1. Motivation and Problem Statement

Although many approaches have been presented on AP but most of the methods are designed for a robot or what is called Car Like Mobile Robot (CLMR) like [8] or [3]. However, there are less methods which specifically designed for a real vehicle so this field is still open for new research either to complete last methods or to make them practical on a real car. In addition, most of the recent methods like [9] and [10] are designed for garage-based parking which is mostly based on the vertical or diagonal movement of vehicle(2.2) so fewer methods considered parallel parking. However, it seems that automation of parallel parking is more necessary because parking on streets is more difficult than a parking-lot as it is more difficult to find a parking place on the street where there is more traffic and also more barriers to prevent parking there and the probability of human's errors on street parking are more than parking in garage. Besides, for parking on street(or parallel parking) many thing should be considered to prevent accidents. In garage parking, parking places are clear and there is less traffic(from people or cars) there and even sometimes there are helpers in garage to guide the drivers.
This thesis implements a simulation of parallel parking using a new simulator called Carla. Implementation includes a complete parking process from the the detection of parking place to control the vehicle movement to the parking lot. Python will be used as API which is directly connected to Carla and for the detection part, Matlab tools has been used. This implementation is a Combination of visual-based and sensor-based methods. In detection part, camera sensor is used to provide visual data for vehicle detection and to make a perfect maneuver algorithm which protected vehicle from crashes, sensors are applied to detect obstacles and make range measurements.

1.2. overview

According to the above descriptions, diagram 1.1 illustrates an overview of this thesis. Chapter 2 presents a background and fundamental knowledge which is required to follow this work. Chapter 3, describes related works and methods that presented in visual-based as well as sensor-based parking detection and also various parking-path methods. Detail explanation of each step of thesis which has been seen in flowchart 1.1 can be found from Chapter 4 to Chapter 6. Chapter 7 illustrates the implementation of each step and all of the challenges during implementation phase. Chapter 8 provides an Evaluation of the implemented steps and considers how well are the simulation results. Chapter 9 says about limitations and

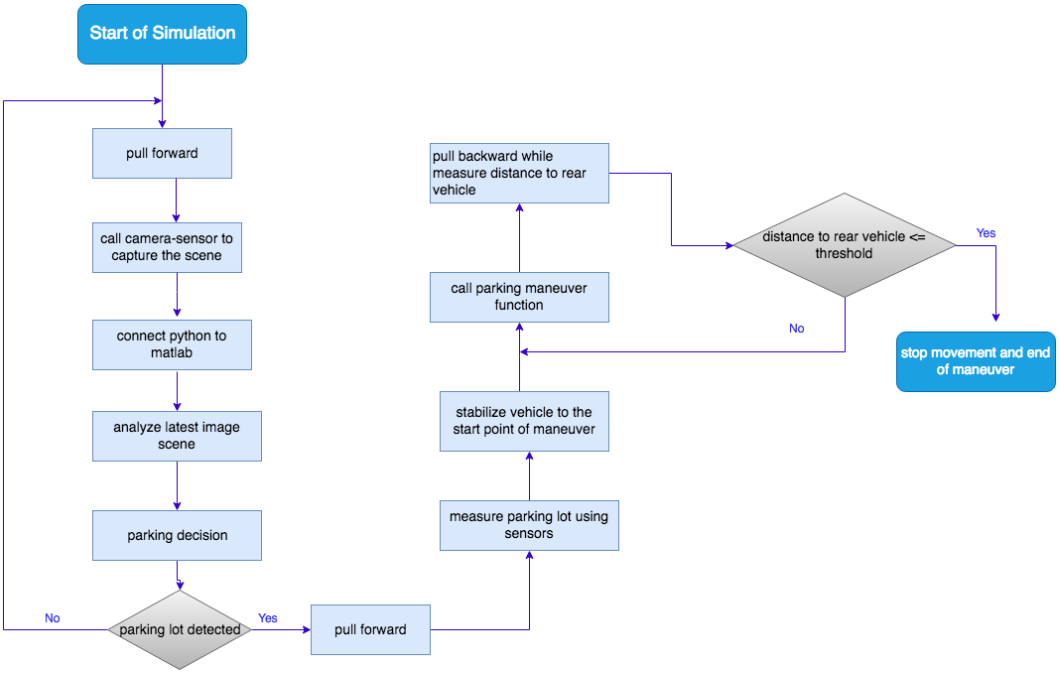


Figure 1.1.: System Review

problems during this work. Chapter 10, gives an outlook on possible future developments and finally Chapter 11 summarizes this thesis.

2. Fundamental Concepts

This chapter presents an explanation of the tools and softwares which have been used during development and also the environment where the system has been developed.

2.1. Simulation Environment

As it has also been mentioned, Carla has been used as a simulator for this parking algorithm. David Werner and his colleagues [11] implemented parking maneuver algorithm using Speed Dreams 2 simulator in a project at this chair. The goal is to evaluate the results of both simulators(Carla and SpeedDreams) to see how the maneuvering algorithm works in different scenarios.

2.1.1. Carla Simulator

Carla is a an Open source simulator for autonomous driving research which presented by A. Dosovitskiy et al. [12] as an Urban Driving Simulator at 2017. They presented this simulator from the ground up to support training, prototyping and validation of autonomous driving models. The goal of designing Carla was to study three approaches about AV: 1. classic modular pipeline in: rule-based planner and maneuver controller. 2. deep network trained end-to-end via imitation learning. 3. reinforcement learning approach [12]. Environment of Carla includes Urban layouts, buildings, pedestrians, street signs and multiple vehicle models. At first release, Carla just included two towns as described here [12] but now in its last release-Carla0.9.6, there are seven different Towns(maps). What makes Carla different from other simulators is multiple weather situation which could be set during simulation(fig 2.1). Carla developers believe that To make the simulation more similar to the real driving environment, different weather conditions i,e. rainy weather when the street is slippery for driving, or snowy, dark and fuggy weather should be also considered. In most of the presented simulation, weather situation has been neglected because developers mostly care about driving algorithms and maneuvering AV regardless of the environment where maneuver should be performed. Most of the images for detection parts are taken with the best camera quality but what if an AV confronts a situation like rainy weather where the street is more slippery than the environment it was trained or a dark weather when there is not enough light to take perfect photos for vehicle detection. So this was a smart idea from Carla developers to implement simulation in different weather environment.
Carla is implemented on UnrealEngine4(UE4) and its environments is a 3D environment which composed of static objects as building, street signs vegetation as well as dynamic objects like vehicles and pedestrians. The interesting point is that all of the accompanying assets

Figure 2.1.: Carla-Different Weather Conditions

which have been created by Unreal-Editor are released with Carla as open-source. In addition, Carla provides some sensors, pseudo-sensor readings and also a range of measurements associated with agent like the location and rotation of vehicle with respect to the world coordinate system.

2.1.2. Client-Server Sides

Carla simulator is a client-server system. Server side runs and renders Carla world which includes simulation and scenes created by Unreal-Engine. Client side provides interface for users to interact with server like spawning new vehicles or controlling properties of simulation. Client API is implemented in Python which is responsible for interaction between autonomous agent and server via sockets. Carla PythonAPI is a module that could be imported into python scripts which controls simulators and retrieve data like controlling vehicles or attach sensors into them and then reading data generated by sensors. Diagram 2.2 illustrates the client-server system in Carla.

2.1.3. Sensors

As explained above, Carla provides some sensors but this is still under-development. Here is a description of the Carla sensors which have been used during this work.

Camera-Sensor

This sensor was the first sensor presented by Carla developers which provides RGB cameras and pseudo-sensors. Pseudo-sensors presents ground-truth depth and semantic segmentation [12]. Sensor's parameters are 3D location, 3D orientation respecting to the vehicle coordinate, field of view and depth of field so it represents 3 modalities: normal vision, ground-truth depth and ground-truth semantic segmentation. Depth-camera or ground-truth depth provides a view which codifying the distance of each pixel to the camera. Semantic segmentation, classify objects in the scene as they will be presented in different colors. Segmentation is based

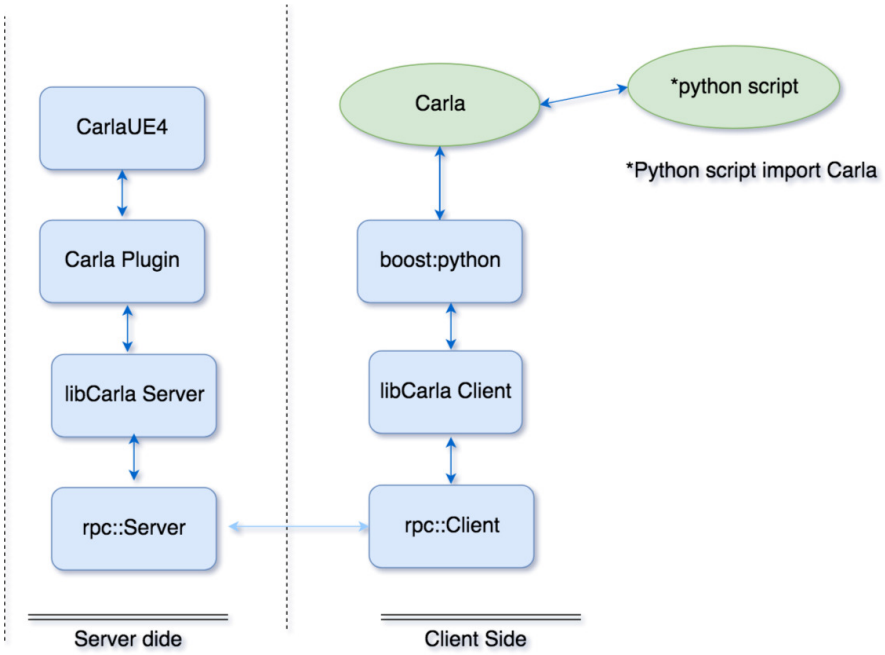


Figure 2.2.: Client-Server system in Carla

on road, vehicles and building. Fig 2.3 shows samples of different modalities in camera-sensor. In this work, we use camera sensor as a normal vision and it captures different photos during simulation. Results will be used for vehicle detection in our autonomous parking scenario.

Gnss-Sensor

Global Navigation Satellite System (Gnss) sensor is attached to an actor to show its current position. This sensor is not considered as a detection sensor and does not provide any information of other vehicles or surrounding obstacles. However, it is sometimes useful to get the location of obstacles which are already detected. This location is Geo reference which defined by open-drive map definition.For example it shows longitude, latitude or altitude position of the actor. In this thesis, Gnss was used to help us finding the distance from the vehicle to the detected vehicle by attaching it to other vehicles.

Obstacle-Sensor

This sensor detects other obstacles during simulation. Detection depends on the view and range of the sensor. This sensor is also supposed to provide distance to detected obstacle. However, this distance measurements have some problem yet.

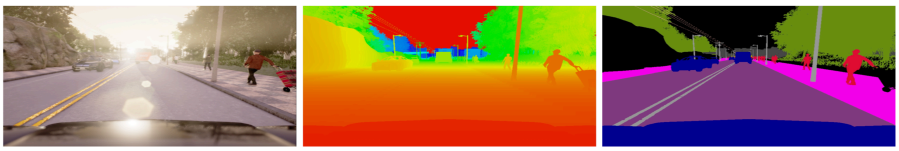


Figure 2.3.: Carla-Camera Sensor Results

2.2. Common Concepts in Autonomous Parking

This section describes some of the expressions and concepts which has repeatedly been used in the next chapters.

2.2.1. Ego Vehicle

Ego-vehicle refers to the self-driving vehicle to which all of the driving algorithms(here parking maneuver) would be applied. Hence, ego-vehicle is our actor/agent that is connected to sensors and we want to monitor its behavior. In other words, when we talk about Autonomous Vehicle, we have to look beyond the first person's singular as developers call it ego-vehicle [13].

2.2.2. Non-Holonomic Vehicle

The idea of our parking maneuver algorithm, is the vehicle is non-holonomic which means that there is no intolerable velocity constraint on vehicle [2]. The idea of the parking maneuver method which have been used in this thesis, is to control a car based on its steering angle and velocity. In other word, it tries to turn vehicle's steering to reach to the expected point so there should not be any constraints on magnitudes of velocity and steering angle.

2.2.3. Parking Types

There are three types of parking lots as it can be seen in 2.4 [9]

- **Vertical Parking:** Vehicles should be parked in the boxes of a row which is vertical to the side of the parking place. This type is also called Perpendicular parking lot.
- **Diagonal Parking:** Diagonal parking is very similar to vertical parking but the separation lines of parking places(boxes) are diagonal.

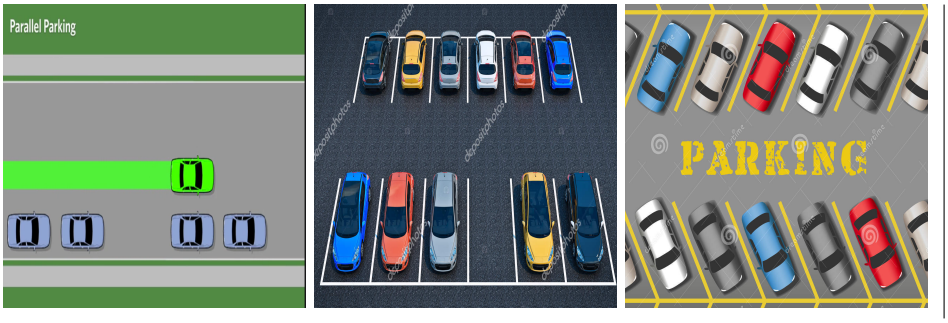


Figure 2.4.: Parallel Parking-Vertical Parking-Diagonal Parking

- **Parallel Parking:** This parking lot can be seen on both sides of a street that that vehicles are in a row which are positioned parallel to each other. This thesis also works on this type of parking.

2.3. Deep learning and machine learning

Deep Learning (DL) and Machine Learning (ML) are fields of Artificial Intelligence (AI) while DL is a sub-field of ML. In both fields we study about design of an algorithm which could learn based on data we provided which are called training data. Hence, trained algorithm could present some results in the future without programming again based on what it has learned. There are three types of learning:[14]

- **Supervised-Learning:** Training process is based on some labeled data as the labels says the algorithm what should be predicted.
- **Unsupervised-Learning:** There is no labeling or control during training process and the algorithm should find patterns in data. This learning type can be used in visualization and analysis process.
- **Reinforcement-Learning:** This method which contains an agent(robot) which is learned to behave in an environment by taking actions and quantifying results. Robot should takes next action based on the current state and gets reward for its correct decision so it will learn based on these rewards. This method has been used in play games like chess, traffics lights and other control systems.

Above learning methods are common in both DL and ML. However, DL is one way of executing ML. DL tries to make a deeper execution of Neural Network (NN). As NN is defined with three layers(fig 2.5). Deep learning adds multiple weights or hidden layers between input and output layers of shallow NN to make it deeper [14]. One of the other difference between ML and DL is that ML always needs a guidance to learn but DL is better in classification learning so DL could learn on its own because it structures the algorithm in NN layers in a way that could be learned and made decision. However, DL needs lots of

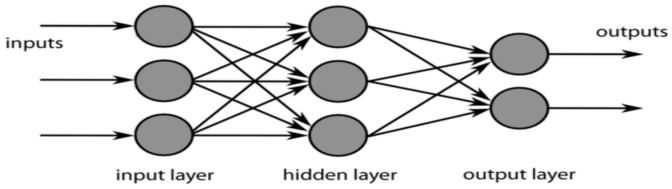


Figure 2.5.: Neural Network Layers

data, memory and time for learning as in some cases when there are many data, it would take months to finish learning process in Deep Learning.

2.3.1. Object Detection Methods

After reviewing the concepts of learning(ML and DL), here are some of the methods and algorithms for learning which are used in object detection. During this thesis two of these methods(Faster-RCNN and ACFObjectDetector) have been used for vehicle detection as it will be explained in chapter 4.

- **Histogram of Oriented Gradients (HOG)**- HOG is an old and popular method. Since it is not using a specific deep learning algorithm, it is a fast method. This simple method could not be handled to detect objects in different orientation or object rotation.
- **CNN:** Convolutional Neural Network (CNN)s are similar to the human neural networks but they added some synapses(weights) so empowered NN and complex dataset could be learned through the network [15].
- **R-CNN-FastRCNN-FasterRCNN:** Regions with CNN features (R-CNN) is region based CNN and is a DL method which adds some manageable number of regions into CNN so it combines rectangular region proposals with CNN features. The process in R-CNN algorithm is as follows: [16]
  1. Find regions in image that might contains objects which are called region proposals.
  2. Extract CNN features from the following regions.
  3. Classify objects by using extracted features.

  In order to speed up R-CNN, Fast-RCNN was introduced. Fast-RCNN uses an external edge boxes algorithm for generating regions and it is faster than R-CNN because its detector process the entire image instead of classifying each region and as FastRCNN

R-CNN Detector	Description
trainRCNNObjectDetector	• Less time to train an object detector, but detection time is slow. • Allows custom region proposal
trainFastRCNNObjectDetector	• Allows custom region proposal
trainFasterRCNNObjectDetector	• Optimal run-time performance • Does not support a custom region proposal

Figure 2.6.: Comparison of CNN Methods

contains computations for overlapping regions, that is more efficient than RCNN. And finally FasterRCNN is the fastest detection method because instead of using an external edge boxes algorithm(as in FastRCNN), it adds Region Proposal Network (RPN) boxes directly to the network for generating regions. Fig 2.6 compares these methods [16].

- **MaskRCNN:** Recently a new method on FasterRCNN has been defined which is called Mask-RCNN [17]. This method extends FasterRCNN by adding a branch for segmentation masks on each region of interests(ROI). It provides all of the outputs generated by FasterRCNN: class labels and bounding-boxes. Besides, Mask-RCNN adds an additional mask output which include pixel to pixel alignment of detected objects. Fig 2.7 illustrates some results of MaskRCNN.

- **You Only Look Once (YOLO):** YOLO is another object detector in Deep Learning Which is presented to provide a faster approach than other object detectors and even FasterRCNN. This method runs deep learning CNN on an input image to produce network predictions. Then in the next step object detector decodes all predictions and generates bounding boxes.

- **ACFObjectDetector:** Aggregate channel features (ACF) is a simple method in object detection and its detection is based on object features. Aggregate channel features (ACF) recognizes specific objects based on training images and object ground truth locations [18]. This method has been also selected for this work because after testing several methods, ACF presented better results in our data.

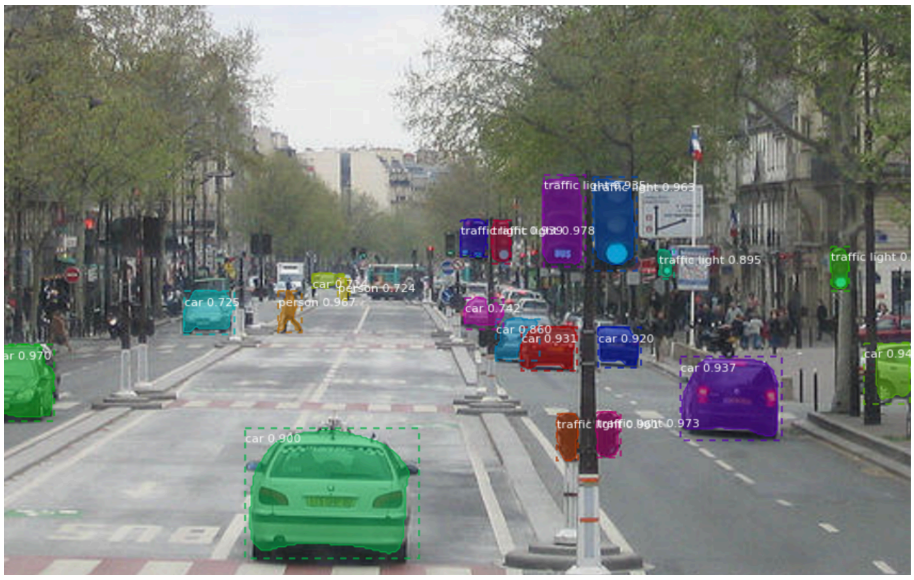


Figure 2.7.: Mask-RCNN

3. Related Work

This chapter provides an overview of some of the most relevant works on Autonomous Parking (AP). As it has been also mentioned in chapter 1, there are two main steps in AP: 1- parking vacancy detection and 2- path/motion generation for parking maneuver. Related works in each field have been explained here.

3.1. Parking Detection

Detection of parking slot could be based on sensors or visual aspects. As it has been explained before, parking detection methods are in 2 categories: 1- free space-based approaches and 2- parking slot marking-based approaches. There is always this criticism about sensor-based methods that they depend on adjacent vehicles to find a parking vacancy and parking place which detected in this way is always surrounded by exactly two vehicles so when there is no car in front of one vehicle and still there is enough place for parking, this parking bay could not be detected [10]. However, the second method(visual method based on line marker) is not also perfect because it could be only applicable during day time or when there are enough lights to extract the features, photos from the parking environment should have a great quality to make it possible for classifier to find the features [19]. Here refers to some of the recent works in both approaches.

3.1.1. Free Space-Based or Sensor-Based

In sensor based parking vacancy detection we could point out this recent work [20] based on Radar sensor as they used radar sensor for detection in all kinds of parking (parallel and perpendicular parking lots). They presented some sorts of filtering calculation and classification of parking spaces in order to optimize Radar results as they claim on 95 percent of accuracy in detected parking lots with an inexpensive algorithm. [21] used UltraSonic sensors in parking occupancy detection and they claim that their method provides accurate measurements from sides of the parking with simple calculation. In [22] UltraSonic sensors have been used both in parking detection and parking maneuver. In [23] from Hanover university of Germany, a novel idea of using Lidar sensor as a new sensor for parking occupancy detection has been presented. They believe that modern sensors provide better facility for detection and data collection and also using Lidar would help to make a more detailed and precise measurements. [19] presents a method to find free parking space between vehicles by scanning laser-radar. They believe that UltraSonic sensors have been successful to find parking vacancy in parallel parking but could not be used in perpendicular(or vertical) parking because of the fact that UltraSonics could work in a situation that incident

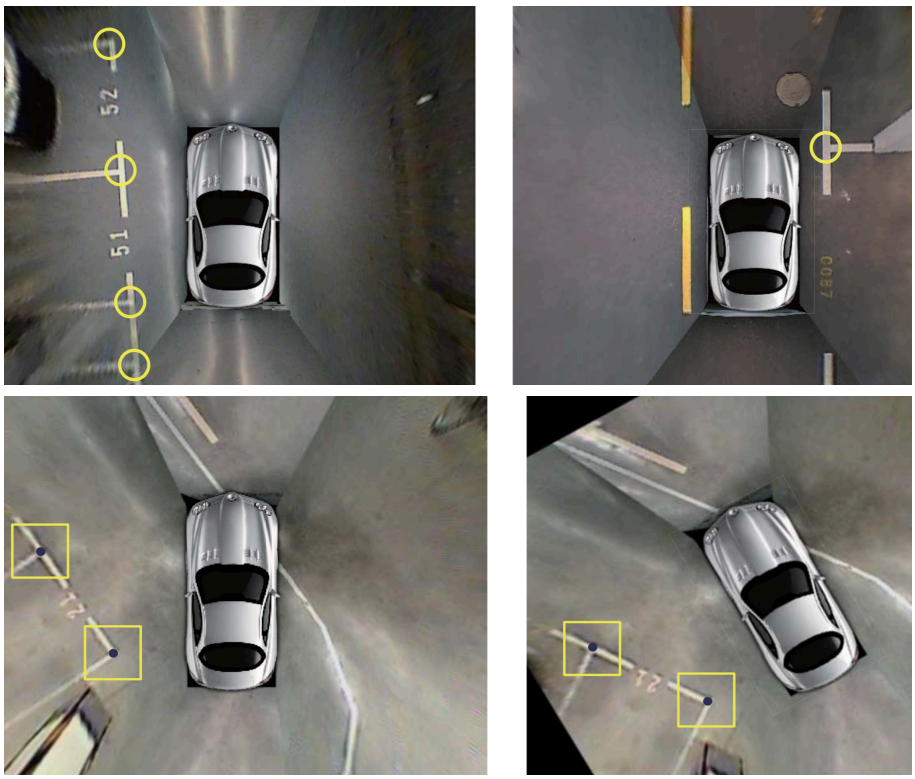


Figure 3.1.: Marking points to define parking lots [25]

angle between UltraSonic sensor and sides of adjacent vehicle is small and as this angle in perpendicular parking situation is so large, range recognition usually fails. The method that [19] used for parking space detection is based on corner detection. Target parking position is found based on the corner detection of parking spaces and corners are divided to round and rectangular. Finally,[24] compares different sensor approaches and tests detection results in fixed and mobile sensor systems. Fixed approach, installs sensors in parking bays. Mobile approach is when the sensor mounted on the vehicle and gathers data as the car is moving. It has been said that mobile approach provides less accuracy but in [24] the focus is to present a novel idea to increase accuracy. They used Sonar-sensors for mobile detection and it was working in small sensor ranges. They have also mentioned that Mobile sensors would be more expensive than fixed sensors.

3.1.2. Marking-Based or Vision-Based

As one of the best vision-based approach, we could refer to [7] where they proposed a detection method based on marking points in parking lot. Marking points are edges of the parking places which are the conjunction of line markers that separates parking place for each vehicle as it can be seen in fig 3.1. Here surrounded view images of different car direction are used to detect marking points where a binary classifier applied to all local images in

data base for detection. Then parking places are detected from these marking points. The approach is to take every two cross points of two parking segments and according to their lengths by applying a pattern called Gaussian line temple, parking lots will be detected. They completed their algorithm in [25] by using NN and DCNN based Networks and also a bigger data base of surrounding view images. For training process YOLO2 method base on DCNN has been used and during training process, position of all marking points in dataset(surrounding images) has to be learned. There are number of rotations for each image in database. After detection of marking point, in the next step, a pattern classifier for marking points will be used to define which points are in the same class and which two cross points make valid entrance line for parking. Finally in the last step, depth of the parking slot will be measured and find which of the points sets in the last step, make a rectangular which is a parking lot. Although This method is really perfect and it precisely defines parking places, it does not clarify whether the parking place is empty or occupied. For parking occupancy detection we could refer to Bayes classifier methods and Region Growing Algorithm. [10] used Bayes classifier for feature extraction and classification of parking slots and available parking lots were detected based on this features. In order to figure out whether a parking lot contains a car, region growing algorithm has been used. The approach is to examine pixels of parking lot images as it calculate each pixel of the image and in each step examines the neighboring pixels of initial points and determines whether the neighbors are similar to the initial ones. When the neighboring pixels are different from the last ones, it means that they contains vehicles. [26] proposed a method to distinguish empty parking places from occupied ones by segmentation of parking area into blocks using calibration. The approach is to covert image into gray-scale blocks and in each block white color refers to car and if the block is black, it means that it does not contain a car and it's a free parking lot. Threshold value is calculated in every block to determine if the block contains white color(car) or that is empty. [27] is another method to determine occupied parking places based on 3-layer Bayesian hierarchical detection framework or BHDF. Based on the geometric structure of parking slots divides them to car occlusions and environment occlusions. Prior knowledge to the network is required for this segmentation and training step is based on the labeling of the vehicles so when the method could not find vehicle in the image, it would be classified as environment occlusion. [28] also represents a way of parking space detection based on edge orientation histogram (EOH) density. The novelty of this approach is that it could handle object detection in dynamic light effects and even in dark images and night scenes which was not possible in last methods. They claim that this method could work with low quality of camera and in different light setting in parking area. To make this algorithm work in different light setting, they used dynamic mixing of multiple features as these multiple features make algorithm strong enough to detect objects in variant lights.

3.2. Path-Planning and Motion-Generation

After determination of empty parking lots, it's time to maneuver. Various algorithms for generation of path motion and parking maneuver has been proposed during last decade. As

it has been explained in 1 there are two types of path-planning. Here are some examples from both methods:

3.2.1. Fuzzy Logic Control

In this method, human driving experience are gathered to create a database which is in the form of IF-THEN rules. These rules shows the steps to a robot or a selfless car [9]. This database needs to be trained in advance and the goal is to control the vehicle to the desired position by using these rules on database. Fuzzy system should select suitable movement at each time from the knowledge database according to its internal states. [8] Uses fuzzy parking control method to control the steering angle of a Car Like Mobile Robot (CLMR). They proposed different methods for forward and backward maneuver and also applied their algorithm for both garage and parallel parking so in total 4 movement methods have been defined. Backward and forward Fuzzy Garage Parking Control(FGPC) and 2 methods for Backward and forward Fuzzy Parallel Parking Control(FPPC). Real-time image processing was used to define the next motion. Image information were used to define next movement and end position. Color detection method was used to calculate the end position. A host computer which connected to the robot were capturing the working environment via the vision system and calculation of the posture of CLMR by analyzing image information and sends commands which contains the next movements to the robot/CLMR via wireless radio modem. We can refer to [29] as another example. Here the idea was based on the decomposition of a parking maneuver into sub-maneuvers that could be realized in an open loop control so the main challenge was the transition between these sub-maneuvers which could be done by machine learning classifiers. RF-kernel was used as a learning classifier and General Radial Basis Function (GRBF) classifier was used to find the similarities based on the template. Classifiers were trained to find the detection points where transition from one sub maneuver to to the next one is required. So the idea was to find the best position of the vehicle based on machine learning. Here machine algorithms helps to find features through images and detect vehicle position then determine whether the posture of vehicle is good respects to the map of the parking area or decide if the maneuver should be done.

3.2.2. Reference Trajectory

This method is based on a reference path as the path depends on the environment model as well as the vehicle model like its speed, steering and etc. Igor Paromtchik and his colleague from INRIA institute [2] developed a smooth motion algorithm which was applied to LIGIER electric vehicle of French company. They presented an iterative algorithm which used a combination of feedback and planning approach. Maneuver consisted of N-iterative motions aimed at lateral displacement of the vehicle. The idea was to control the vehicle maneuver based on its velocity and steering angle. They used 14 ultrasonic sensors for the range measurements and getting information from the environment. K. Jiang et al. in [3], believed that algorithm represented in [2] was not considering possible collision and longitudinal boundary during maneuver. So they proposed a new algorithm by defining some constraints

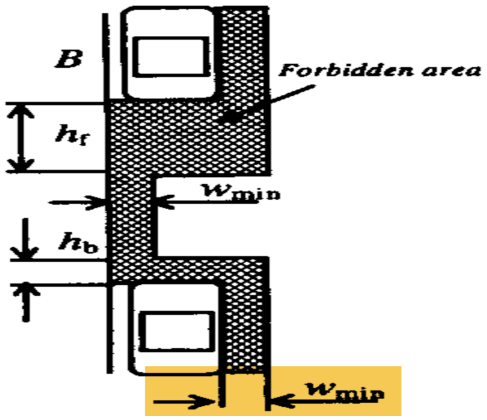


Figure 3.2.: Forbidden area in parallel parking maneuver [3]

for parking maneuver i,e. they defined forbidden area for movement in parking lot as we could see in fig 3.2. Motion path was formed by two circles(or 2 arcs). One of the circle movement is clockwise and the other one is in opposite direction for controlling forward and backward movements as they are in opposite direction. Fig 3.3 illustrates the path. This algorithm planned a path for robot or CLMR outside of the forbidden area. They claims that their algorithm could also work in a small parking spaces. However, the last method was applied on a real car and that is still used for LIGIER autonomous vehicles and this one was just implemented on a CLMR. [9] presents another example of reference path method which is only for vertical parking lots.

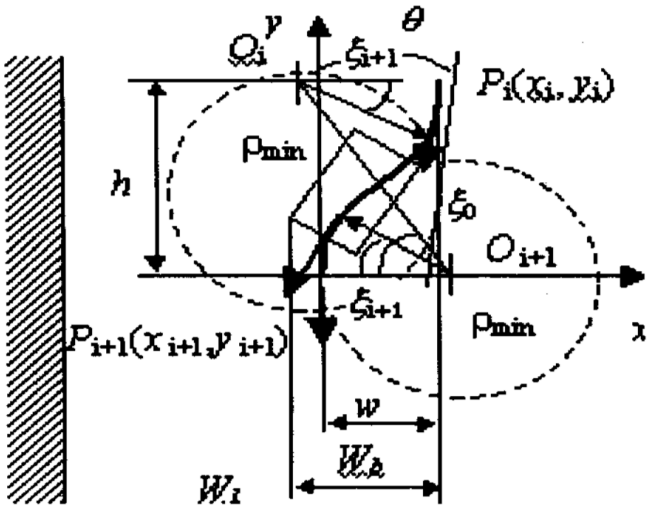


Figure 3.3.: Motion-path formed by two circular arcs[3]

4. Parking Detection

First step in Autonomous Parking (AP) is to find a parking vacancy. This step is really important and could be the base step in parking scenario because it parking maneuver could be affected by that. For example if there is not enough space, parking maneuver would not be successful. Hence, in this research many times and efforts have been taken to find a best method of parking space detection. Various approaches has been presented for parking vacancy detection in recent years. We selected vision method and Machine Learning methods for parking detection. Some ML and NN methods have been also presented previously, among which we could point ti [28]. In[28] orientation histogram and density computation were used to identify empty parking lot. As it's mentioned in 3, many of the vision based methods uses parking space line markers in street and garages to define different parking areas and marking points as in[25]marking points are defined as edges of each parking area and after finding this point it would be easier to find a free area for parking. One of the problem which is addressed about vision ways is that they always need to have visual based data of parking-lot to make decision of parking occupancy otherwise it is more reliable method than sensor because although sensors would provide precise data of distances and measurements, they could not process other information of the scene as position of vehicle, object detection and they just consider cars as data and other features would be ignored. In this work, a simple vision base method has been used. At first, vehicles in front of the ego vehicle should be detected and then based on vehicles' positions and sizes and comparing them to the main(ego) vehicle, free parking place would be detected.

4.1. Vehicle Detection

The idea is to detect cars in front of the ego vehicle and based on the distance between first two vehicles ahead, parking space would be detected. Detection is based on the images taken by camera-sensor. Camera sensor is placed in front of the ego vehicle and during simulation while vehicle passes the street, images from side of the street are captured and sent to the detection algorithm. The goal is to detect vehicles in images so it is a textbox object detection problem. As it mentioned in 2, lots of machine learning algorithms could be used to detect object in an image. The most common ones are as follows:

- **Histogram of Oriented Gradients (HOG)**
- **Convolutional Neural Network**
- **R-CNN, FastRCNN and Faster-RCNN**

- **YOLO**
- **ACF**

In this thesis, three detectors were trained(two detectors from FasterRCNN by setting different training parameters and one detector from ACF method) and finally ACFObjectDetector was selected as our vehicle detection algorithm because it provided more accurate detection on our specific training data. Despite our expectation that RCNN would provide better result because of its popularity, detector from ACF worked better in our training data. Another reason could be referred to the training dataset as FasterRCNN requires a specific training dataset. Since the results of ACFObjectDetector were more satisfied and this method was working better in our data, ACF detector has been selected and working on better algorithm for training FasterRCNN postponed to future works. Next section presents an explanation of the methods which have been used. Evaluation results from the detector of each method after training step, has been also provided.

4.1.1. FasterRCNN

At first test, 656 labeled images(results of groundTruth labeler) fed into the network in learning process and Epochs set to be 40 as it supposed to provide a more accurate data by training with more epochs and learning iterations. However, if these epochs number exceeded certain values, network would be over-trained. 40 epochs resulted in 26240 iteration for each step of training (as RCNN has 4 training steps) and accuracy rate for each step was closed to 98-100 percent. Fig 4.1 illustrates a moment of training process in the last training step. Fig 4.2 illustrates results of RCNN detector and ACF detector respectively(from left to right). First left image in the first was the testing image to examin results of different detectors. Second image(first row) shows detection result of FasterRCNN(trained with 40 Epochs) and it detected two of 3 vehicles. Finally, the last image in first row is the result of ACF detection where all vehicles covered with bounding boxes which means that ACF detected all of the 3 vehicles. In second row another image that includes 2 cars, has been tested and again the results of RCNN and ACF could be observed from left to right. As it can be seen, FasterRCNN detected none of the vehicles as it shows empty bboxes and empty scores(0*1 empty). Second image is the result of ACF which shows 3 detection(for 2 cars in image?) because sometimes we could get more than one detection for one vehicle and here the first 2 bboxes are for first vehicle and the second bboxes is for the other vehicle. However, in this situation where there are more than one detection for one vehicle with different scores, strongest score would be selected. Here after selecting the strongest scores by eliminating same detection(bboxes for one vehicle which have overlaps), the result of ACF is the last photo in row 2 which shows that it detected both vehicles but RCNN could not detect any of them. There was a probability that RCNN was over-trained because of 40 epochs or many repetitive training samples.

So next time we decided to reduce number of repetitive data as far as possible. This time 188 images were chosen in the training step and Epochs reduced to 10 numbers so we had 1880 learning iteration in each step and accuracy rate was more than 82 as it can be seen in

39	25200	16:43:11	0.6486	100.00%	0.84	0.0010
39	25400	16:51:17	0.7217	100.00%	0.89	0.0010
40	25600	16:59:25	0.8702	100.00%	0.98	0.0010
40	25800	17:07:32	0.6082	100.00%	0.84	0.0010
40	26000	17:15:37	0.5804	100.00%	0.76	0.0010
40	26200	17:23:45	1.0040	100.00%	1.03	0.0010
40	26240	17:25:22	0.5748	100.00%	0.76	0.0010

Step 4 of 4: Re-training Fast R-CNN using updated RPN.
--> Extracting region proposals from 656 training images...done.

Training on single CPU.

Epoch	Iteration	Time Elapsed (hh:mm:ss)	Mini-batch Loss	Mini-batch Accuracy	Mini-batch RMSE	Base Learning Rate
1	1	00:00:09	0.2187	96.09%	0.51	0.0010
1	200	00:01:15	0.3902	92.86%	0.63	0.0010
1	400	00:02:19	0.1343	96.09%	0.70	0.0010
1	600	00:03:23	0.1666	97.22%	0.72	0.0010
2	800	00:04:27	0.0901	97.66%	0.44	0.0010
2	1000	00:05:31	0.1157	96.88%	0.42	0.0010
2	1200	00:06:35	0.2736	92.19%	0.57	0.0010
3	1400	00:07:39	0.0984	96.88%	0.64	0.0010
3	1600	00:08:42	0.2032	96.09%	0.65	0.0010
3	1800	00:09:46	0.1595	97.66%	0.66	0.0010
4	2000	00:10:50	0.0676	98.21%	0.56	0.0010
4	2200	00:11:54	0.0570	99.22%	0.47	0.0010
4	2400	00:12:58	0.1233	96.09%	0.46	0.0010
5	2600	00:14:02	0.0156	100.00%	0.36	0.0010
5	2800	00:15:05	0.0616	100.00%	0.70	0.0010
5	3000	00:16:09	0.0830	97.66%	0.45	0.0010
5	3200	00:17:13	0.0824	98.44%	0.41	0.0010
6	3400	00:18:17	0.0466	100.00%	0.38	0.0010
6	3600	00:19:21	0.0627	99.22%	0.47	0.0010
6	3800	00:20:25	0.0254	100.00%	0.39	0.0010

Figure 4.1.: FasterRCNN Training Process - Set to 40 Epochs

Figure 4.2.: Detection results of ACFObjectDetector and FasterRCNN

Training on single CPU.

Epoch	Iteration	Time Elapsed (hh:mm:ss)	Mini-batch Loss	Mini-batch Accuracy	Mini-batch RMSE	Base Learning Rate
1	1	00:00:02	1.3047	35.16%	0.82	0.0010
1	50	00:01:47	1.1571	100.00%	0.77	0.0010
1	100	00:03:34	1.0158	100.00%	0.75	0.0010
1	150	00:05:21	0.7642	100.00%	0.79	0.0010
2	200	00:07:07	0.9493	100.00%	0.97	0.0010
2	250	00:08:55	1.0523	96.09%	0.98	0.0010
2	300	00:10:42	1.1591	100.00%	1.17	0.0010
2	350	00:12:28	0.6930	100.00%	0.86	0.0010
3	400	00:14:12	0.9760	100.00%	1.02	0.0010
3	450	00:15:56	0.8676	100.00%	0.97	0.0010
3	500	00:17:41	1.1579	100.00%	1.12	0.0010
3	550	00:19:26	0.6494	100.00%	0.86	0.0010
4	600	00:21:11	0.5311	100.00%	0.71	0.0010
4	650	00:22:56	1.1418	100.00%	1.07	0.0010
4	700	00:24:41	0.7021	98.44%	0.82	0.0010
4	750	00:26:27	1.9615	100.00%	1.65	0.0010
5	800	00:28:13	0.5132	100.00%	0.74	0.0010
5	850	00:29:59	1.3405	97.66%	1.23	0.0010

Figure 4.3.: Training process of FasterRCNN with 10 epochs

4.3. Although the accurate rate of training was reduced compared to the last time, detection results were exactly the same as last detector(trained by 40 epochs) for same testing samples.

4.1.2. ACFObjectDetector

ACFObjectDetector with 5 training steps have been generated and same dataset from training last detector(FasterRCNN with 10 epochs) has been used to see the difference. 4.4 shows a moment of ACF training. Evaluation results can be seen in 4.5. Same testing samples were used to measure the efficiency of three different detectors. Two R-CNN detectors provided the exact same precision rates of 20 percent. ACF has the better detection rate (70 percent)

4.2. Detection of Parking Vacancy

After finding detected vehicles, next step is to find parking vacancy or an empty space between two adjacent vehicles which is perfect for parking. As already mentioned above, most of the defined methods like [7] depends on marking lines provided by parking-lots. However, as our environment was on a street which did not offer these line marking or any sign near parking places. Besides, the scenario is to find a parking vacancy and implement maneuver in the street not a garage with ordered lanes and signs to guide drivers to parking-places. In parallel parking in a random place on the street, there is always these problems that line markers in most places are too small and difficult to observe so it would be difficult to detect them by computer and most of the streets does not even have these markers for parking. In other words, city streets are full of line marks as traffic line, crossroads or lane dividers

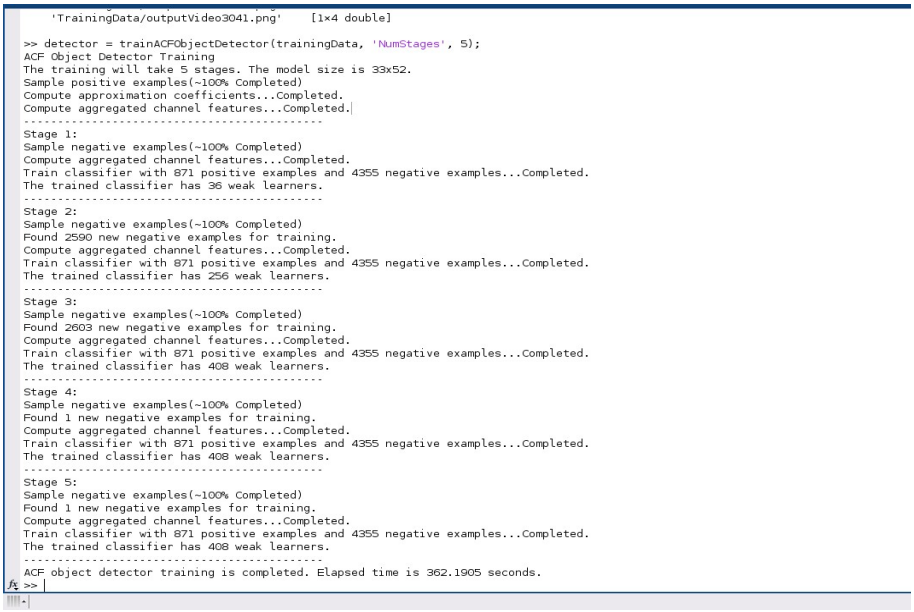


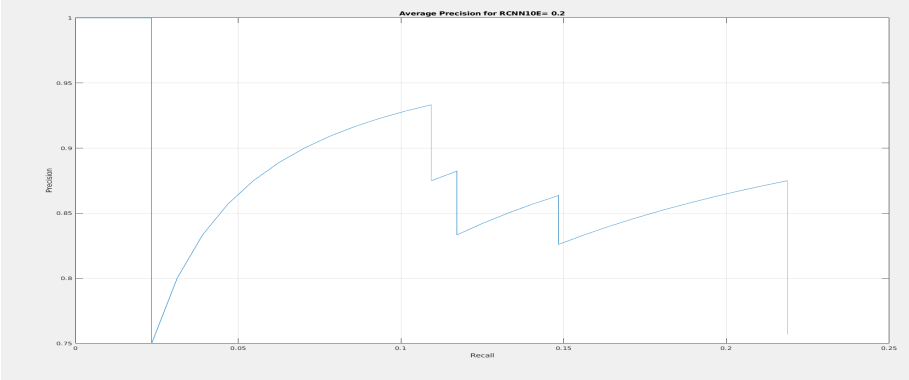
Figure 4.4.: Training ACFObjectDetector

which would be difficult to separate them from parking lanes so most of the previous works in vision-based did not help to find a place of parking. The only thing we had in this project was bounding-boxes which provided by vehicle detection algorithm(ACF). Bounding boxes are the boxes highlights detected vehicles as it could be also seen in detected photos in fig 4.2 and they provides us pixel coordinates of each detected car and each of them has a form of [x, y, width, height]. As it can be seen again in fig 4.2 bboxes in each frame of detection, have partially overlapped and these overlaps could be used to find empty space between vehicles. If two vehicles has a high overlap ratio, it can be concluded that they are so closed together and the distance between them is very small so it would not be possible to park another car between them and vise versa. Hence, the idea is to measure how much two vicinity objects are overlapped. In this work IOU(Intersection Over Union)has been used to find overlap ratio of the boxes. IOU is a common measurement in computer vision and most of the detection algorithms have libraries to implement it. IOU can be defined by equation 4.1 and it can be measured by the amount of pixels where two objects overlap dividing to the amount of pixels covered by both objects.

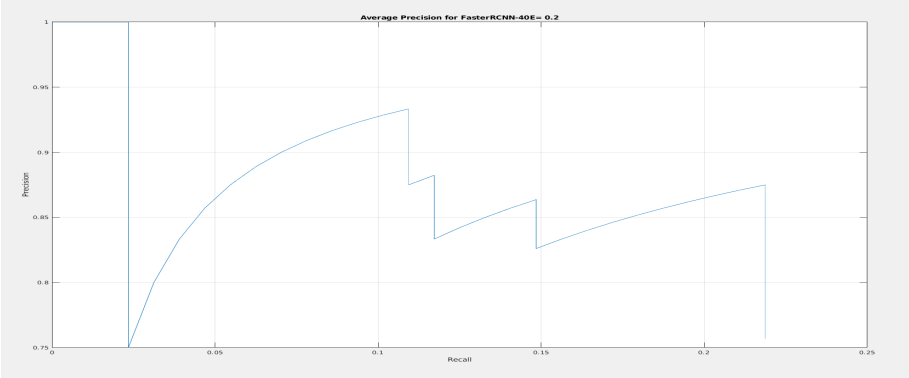
$$IntersectionOverUnion = \frac{Intersection of Boxes}{Union of Boxes} \tag{4.1}$$

Overlap ratio will give us how much a car's bounding box overlaps with other car's bounding boxes i,e if this ratio is lower than 0.15, this means that their overlap is so small and there is a parking space between them but if this overlap is high (like more than 0.2) it means that the space between two vehicle is not big enough to place another car. However, we did not make parking decision just based on ratio data and after estimation of parking space and the probability of empty place, sensors are used to measure the exact distances between two

FasterRCNN detector - training epochs set to 10



FasterRCNN detector - training epochs set to 40



ACF detector - trained in 5 steps

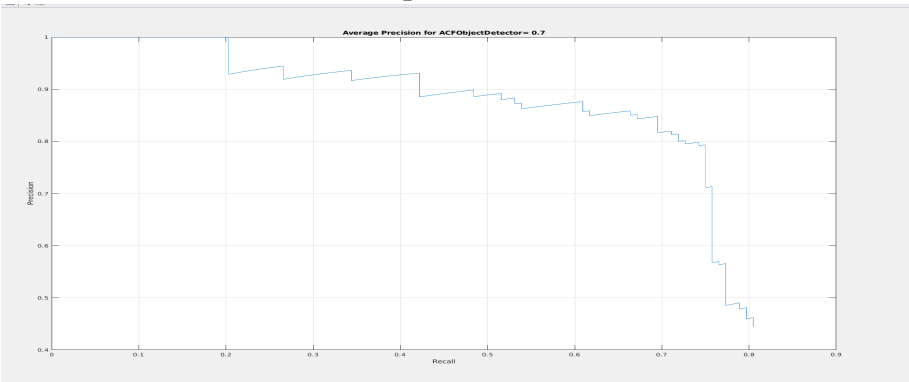


Figure 4.5.: Evaluation results of 3 detectors

vehicles(by low overlap rates). so this ratio gives us the likelihood of parking vacancy and is not the final decision. a detailed explanation about the sensors and how ego vehicles sets to the starting point of maneuver, has been provided in chapter 5.

5. Vehicle Stabilization

Before start of the maneuvering step, vehicle should be set to the stabilized situation or that should be moved to a start position for maneuver. This step is also called positioning phase which is really important to make a collision free parking movement. Vehicle should have a correct starting position that is parallel to the front parked vehicle before entering to the parking place [30]. this step is considered as an interface between parking-detection(last chapter) and maneuvering step(chapter 6) 6. After the estimation of parking vacancy, vehicle moves slowly to the parking place and during this movement, sensors in the right side of the car are started. Sensors provide information of non-player vehicles' location cover the empty parking space(parking vacancy). Based on the exact distance between two non-player cars, L(length of the parking place) and W(width of the parking) could be measured. values of L and W are used in the next step to make parking maneuver algorithm. When the vehicle reach to the second non-player vehicle(after passing empty space), it should be stopped and prepared for parking. In this work there are three sensors in the right side of the vehicle. Fig 5.1 illustrates place of the sensors. Each of the sensors give the distance to the detected-vehicle. When ego reaches to the location of second car(after passing parking space), distance between ego-vehicle and this car is measured by three sensors. If all sensors show the same distance value, it means that ego-vehicle is parallel to the non-player car and should be stopped at this point. Fig 5.2) shows the starting position of the vehicle before moving to the parking lot.

Figure 5.1.: Sensor places in ego-vehicle

Figure 5.2.: Start of Parking Maneuver

6. Parking Maneuver

Various path planning algorithms have been presented for parking generation and steering motion. However, most of them are designed for a general motion of a robot or what is called Car Like Mobile Robot (CLMR) [31]. These methods might not be usable for a real vehicle to produce a real time parking on streets. The assumption of most of these recent methods is that the actor(robot or car) is Non-holonomic which means there is no intolerable velocity constraint on vehicle [2] i,e. Focus of these algorithms is to control the steering Angle and velocity of the vehicle or how to turn the steering to reach the expected point in a controlled speed. The method presented in [2] has been chosen for our parking maneuver simulation. This method is based on the idea of sinusoidal control function [32]. This approach is a feedback and planning method which means that motion control procedure is an iterative process which repeatedly localize the vehicle until a specified location of vehicle relative to its environment is reached. In the first iteration, vehicle enters to the parking place and gets a satisfied location compares to its environment but for optimizing the motion to set a perfect place in parking-bay where vehicle has the precise distance with adjacent cars and side of the parking-bay, this maneuver should be repeated in more iteration. iteration depends on the area of the parking-bay(its lateral and longitudinal measurements) and also size of the vehicle. This algorithm considers steering angle and velocity magnitudes as main maneuver parameters. Calculation of steering-angle and velocity depends on the calculation of the whole maneuvering time, maximum speed and maximum steering wheel of the vehicle which is applicable to the parking maneuver. It is obvious that this values depend on the model of the ego-vehicle, environment situation and also different simulators offer different values. Before calculating these main parameters of maneuver(speed and steering angle) and even before the calculation of maximum time, it is important to know the height and width of the parking bay and also size of the ego vehicle. According to above description, this parking maneuver algorithm could be classified as the following steps:

- Measurements of parking place(find longitudinal and lateral displacement)
- Calculation of maximum values of maneuvering time(T_{max}) and steering angle(Φ_{max}) by evaluating vehicle model which is explained in 6.1.
- Steering vehicle based on the maximum values of the last step while processing the distances to other obstacles(like side of the parking and other vehicles) to prevent from collision
- Check vehicle location relative to the environment and expected parked place and decide when to stop steering

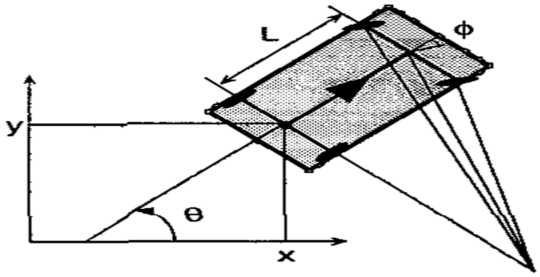


Figure 6.1.: Model of a non-holonomic vehicle [2]

In this work, as the detection of the parking place and setting the position of the vehicle has been done in the separate steps (in the last steps using machine learning algorithms), we made a little changes in the process of the algorithm. In our method, maximum time and steering angle is calculated just one time before the start of maneuver and after 5 but in [2] this calculation is done before each iteration start. In the current project, we just made the maneuver in one iteration and as the vehicle reaches to the goal position in the first iteration we refused to make this movement iterative. However, to get an optimal position where vehicle has precise distance to the adjacent cars (say in the center of two vehicles) and also a satisfied distance to the sides of the parking-bay, this algorithm should be applied attractively with also some forward movement. In this project, the goal was to place the car inside of the parking without collision but for making a precise optimum location and as the goal was reached in the limited time of research, the other process of testing an iterative algorithm or making a nice position of the vehicle between two other cars, postponed to future works. calculation of maneuver parameters is based on the accurate kinematic and dynamic model of the vehicle. So next section provides an explanation of model and motion of the vehicle. Then base on this model, in 6.2 section, a detail explanation of how T-max and Ø-max are calculated before start of parking motion has been provided and finally in 6.3 section, motion algorithm and how the vehicle is steered and moved to the parking-bay based on T-max and Ø-max values has been explained.

6.1. vehicle model

A four-wheeled vehicle model with its coordinates and steering can be seen in fig 6.1. As it can be seen vehicle's location described by three coordinates (x,y,θ) where x and y are coordinates of the midpoint of vehicle's rear wheel and θ is the orientation angle of the vehicle. Motion of the vehicle can be described by following equations:

$$\begin{aligned} \dot{x} &= v \cdot cos(\phi) \cdot cos(\theta) \\ \dot{y} &= v \cdot cos(\phi) \cdot sin(\theta) \\ \dot{\theta} &= \frac{v}{L} \cdot sin(\phi) \end{aligned} \tag{6.1}$$

Where v refers to the longitudinal velocity, ϕ is the steering angle and L is a the wheel base and constant value referred to wheel base and can be defined in range 1.5 < L < 2 [2]. This equation assumed that the vehicle is non-holonomic because using the derivatives of car's coordinate declared that these values are non-intolerable. Besides, these equations are valid for a vehicle which moves on a flat ground so they ignored the friction and slippage between the ground and vehicle's wheels. Future calculation of parking maneuver are based on this equations(of vehicle model) and as we know parallel parking movement affects on the lateral displacements of the vehicle into the parking-bay.

6.2. Calculation

As already mentioned above, some calculation should be done before the start of the maneuver because to calculate the value of steering angle and velocity at each time of maneuver, the values of maximum steering and maximum time of maneuver should be known in advanced which also depends on the size/model of the car and environment situation like parking-bay's area so before going to the next section(motion phase), here an explanation of how this maximum values are calculated before start of the maneuver.

6.2.1. Maneuver Time Calculation

Before starting the calculation of steering angle and velocity for parking maneuver (6.3), we should know the duration of whole parking maneuver. Computation of T is an iterative process based on the evaluation of vehicle's motion (6.1). The idea is to change the values of x and y (location of the vehicle) in the path of the parking-bay to estimate the duration of passing parking area. To this end, values of x and y should be changed in an iterative process. Following constraint should be satisfied during these iterations.

$$|(x_T - x_0) \cdot \cos\theta_0 + (y_T - y_0) \cdot \sin\theta_0| < D_l \tag{6.2}$$

D_l is the longitudinal displacement for the car in the parking-bay or length of the parking place and in each iteration, this constraint should be checked besides the value of ΔT is added to the T value in each time (starts from 0). ΔT is the sampling period of the movement which depends on the maneuver's speed and in simulation depends on the simulator and its sampling-rate. The calculation process also depends on the changes of steering angle and velocity at each time as their equations have been defined in the next section6.3. Based on the 6.1, the changes in x and y for each iteration is defined as follows:
When $\phi(t_n) = 0$:

$$\begin{aligned} \theta(t_n) &= \theta(t_n - 1) \\ x(t_n) &= x(t_n - 1) + v(t_n)\Delta T \cos\theta(t_n) \\ y(t_n) &= y(t_n - 1) + v(t_n)\Delta T \sin\theta(t_n) \end{aligned}$$

and when $\phi(t_n)$ is not 0:

$$\theta(t_n) = \theta(t_n - 1) + \frac{v(t_n)\Delta T}{L} sin\phi(t_n)$$
$$x(t_n) = x(t_n - 1) + \frac{L}{tan\phi(t_n)}[\sin\theta(t_n) - \sin\theta(t_n - 1)]$$
$$y(t_n) = y(t_n - 1) + \frac{L}{tan\phi(t_n)}[\cos\theta(t_n) - \cos\theta(t_n - 1)] \tag{6.3}$$

So in each iteration, x and y is calculated according to the following equations and their values compared to the constraint(6.2.1) plus magnitude of T is increased by ΔT. when the constraint is violated, iteration will be stopped and the value of T is returned as the maximum duration of parking maneuver (T_{max}) to the next step.

6.2.2. Steering Angle Calculation

Calculation of Φ_{max} is pretty similar to T_{max}. This is also an iterative process where values of Φ is started from an empirical value of steering angle which is the maximum magnitude of steering angle that vehicle's wheels could get and that depends on the model of the vehicle and again simulator. Φ value is reduced during the iteration till the following constraint is satisfied. Here we consider lateral displacement of parking and Φ value is reduced by ΔΦ in each iteration till the lateral constraint is satisfied and in that time the last value of Φ is given as the maximum steering angle for the maneuver.

$$|(x_0 - x_T) \cdot \sin\theta_0 + (y_T - y_0) \cdot \cos\theta_0| < D_w \tag{6.4}$$

D_w is the lateral displacement or width of the detected parking-bay.
Calculation of x and y c as the same for T calculation process.(6.3)

6.3. Motion phase

After preparing all of the values which are required for maneuver and getting knowledge about parking bay and distances, vehicle should be moved to the parking-bay. As already mentioned above the parking movement in our applied method is based on steering the wheels of the vehicle in an appropriate velocity so here there are two calculations(speed and steering angle). These calculations are done during the maneuver. In other words, vehicle is steered till it reached a satisfied place inside the parking-bay. for setting the values of steering and velocity at each time of maneuver, these equations have been used [2].

$$\phi(t) = \phi_{max} \cdot k_\phi \cdot A(t), \qquad 0 \leq t < T \tag{6.5}$$
$$v(t) = v_{max} \cdot k_v \cdot B(t), \qquad 0 \leq t < T \tag{6.6}$$

As it can be seen from these equations maneuver started from time of 0 (t=0) and maximum period of maneuvering is T which is calculated in advance 6.2.1. In each time of maneuver

new values for ϕ and v are set to the vehicle. Φ_{max} is what explained in 6.2.2. v_{max} is the maximum speed that a vehicle can get and again depends on the vehicle's model as well as environment situation which is also an empirical value during calculation. coefficient k_ϕ in steering angle calculation regards to the movement direction which is +1 when vehicle should be moved to the right side and -1 when it moves to the left.(depends on the parking situation) and k_v in velocity calculation refers to the forward or backward movement(+1 for forward and -1 for backward). $A(t)$ and $B(t)$ functions are calculated as follows:

$$A(t) = \begin{cases} 1, & 0 \leq t < t' \\ cos(\frac{\pi(t-t')}{T^*}), & t' \leq t \leq T - t' \\ -1, & T - t' < t \leq T \end{cases} \tag{6.7}$$

$$B(t) = 0.5 \cdot (1 - cos(\frac{4\pi t}{T})), \qquad 0 \leq t \leq T \tag{6.8}$$

As we see their calculation depends on T^* which is maneuvering time for a one complete turn or the time that would take for vehicle to turn from $-\Phi_{max}$ to $+\Phi_{max}$. t' is defined as: $t' = \frac{T-T^*}{2}, T^* < T$

6.3.1. End of Maneuver

In this approach time of the whole maneuver T_max is estimated in advance but this is the estimation and it is also considered as a constraint for the maximum time of maneuver so to the stop time of maneuver should have also other constraints. In the presented work, distance of the vehicle to the backward obstacle (or other parked) vehicle and also lateral distances to the side of the parking are also checked during the maneuver and when vehicle reached to the satisfied distance from the rear vehicle, it should be stopped and situation considered as parked situation. In addition, to make sure that is a safe maneuver(collision-free), distances from all sides of the vehicle to other obstacles should be checked during maneuver and when the vehicle is about to contact with other vehicles or obstacle, it should be stopped and maneuver start again by positioning the vehicle to the first location. This is also explained in detail in the next chapter.

7. Implementation

As it mentioned in 2, Carla simulator has been selected for our implementation. In this work one of the built in maps of Carla has been used. In order to access Carla's assets like vehicles and maps, it provides a library which can be imported to python script and all we need as map(which is called World in Carla) and actors(or vehicles) and their properties are called through this library. First step of simulation was to define a favorite map which was suitable to our parallel parking project and spawn all vehicles in the map. As it has been already mentioned in the last chapters, this work is divided in 3 main steps:

- **Detection phase**
- **Positioning phase**
- **Maneuvering phase**

The implementation process of each of them has been presented in the following sections.

7.1. Detection Phase

For the detection phase we used Matlab's tools and Machine Learning libraries so it was also required to have access to Matlab and Matlab should be connected to Carla simulator. First the detection process in Matlab and then the method which have been used for Matlab-Python connection will be explained.

7.1.1. Train ACF Detector

As already mentioned in 4.1, after comparing evaluation results of different detection methods, ACF(Aggregate Channel Features) algorithm has been chosen for the detection part. Before starting any detection, a detector is required so first step is to train ACF detector. we need to provide a dataset for the training step. Hence, different scans of vehicles in different locations and also in different orientations were provided. In this work about 150-200 images from all Carla vehicles were taken but before sending these data to training phase, Ground Truth objects should be defined in each image. Ground Truth object contains information about data source and list of all label definitions [33]. In other words all of images should be labeled in order to define ROI(Region Of Interest) to emphasize on object(here vehicle) which should be detected. For labeling Ground-Truth-Labeler application from Matlab [34] has been used as it provides automatic labeling which makes a fast process and also it supports different image formats in various sizes and it is also possible to import unordered list of data to it.

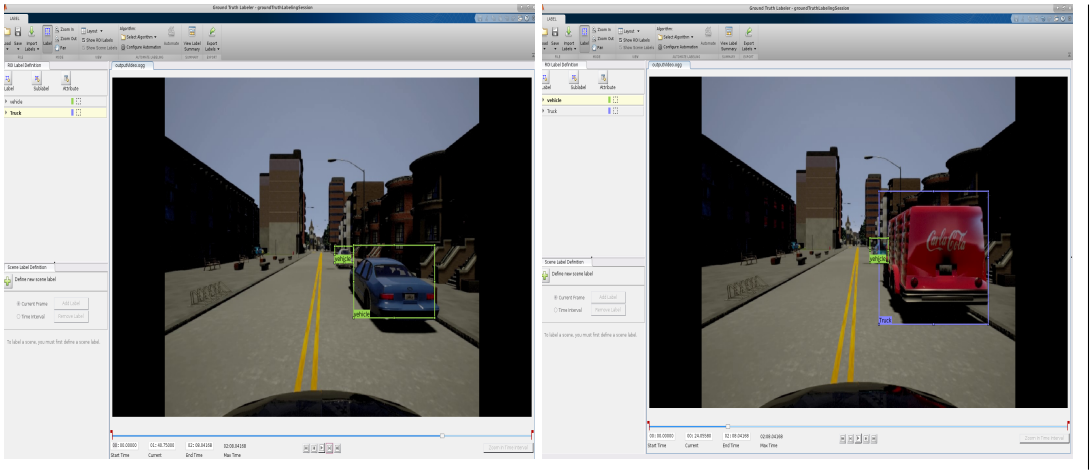


Figure 7.1.: GrounTruthLabbeler Application

Figure 7.1 shows the environment of GroundTruthLabbeler and labeled vehicles using this application.

Training function of ACF library(called trainACFObjectDetector) has been used for learning phase and training property adjusted to 5 steps of training (5 is the empirical value that gave us a good result). 7.2shows the last step of training ACF detector.

After testing step, the detector was saved as it should be called for each time of running parking project. so second step was to test and evaluate the detector to guarantee that it will work in our parking project.

7.1.2. Testing Detector

For testing the trained detector, a set of testing images which were not from training set, were selected. This images were labeled by GrounfTruthLabbeler to define the expected results. Raw images(testing set before labeling) were given to the detector in a for loop and detection results from all images were saved to a table. Finally, result table was compared to the expected results(labeled images) using evaluateDetectionPrecision function from Matlab. This function shows the rate of detection. As we have also seen the graph results of different detector in chapter 4, ACFObjectDetector had the best rate in out data.(70 percent accuracy)

7.1.3. vehicle Detection

The tested detector 7.1.2 is called whenever we run the program and before start of parking maneuver to find a parking vacancy by using detect() function of ACF library. This function gets an image as input argument and compares it with the detector and its outputs are scores and bboxes' coordinates.

Fig 7.3 illustrates an example of applying ACFObjectDetector on a test image. As it can be seen the results are shown as Scores and BBox. Scores refer to the detection score or rate of detection in percentage. BBox shows the place of detected vehicle and also height and width

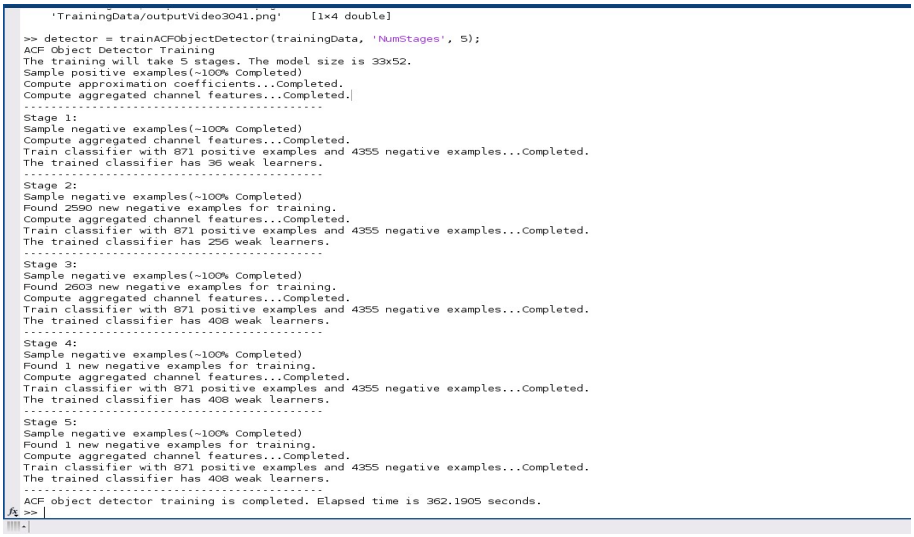


Figure 7.2.: Training ACFObjectDetector

Figure 7.3.: Results of ACF Detector-Bounding Box (BBox) and Scores

Figure 7.4.: Detected Vehicles

of the box surrounding vehicle. BBox can be defined as [x, y, width, height]. we used these values to estimate size of the vehicle. In order to see bboxes around the vehicles, Matlab's annotation function of Image processing toolbox could be used. After applying annotation function to image, bboxes will be added to image so detected vehicles would be covered by boxes around them fig 7.4

7.1.4. parking vacancy detection

As it has been explained in 4 to estimate the place of parking vacancy, bounding boxes are used to find empty parking place. However, before calculating the overlap ratio, we should select bboxes with the strongest detection rate(strong scores). Another problem is that in some cases there are multiple bboxes for one detected vehicle with different detection scores. So if there would be multiple bboxes for one vehicle, BBox with the highest scores should be selected. After selection of the strongest boxes, each of the vehicles in front of the ego vehicle would have one bbox covering them, as we could see in fig 7.4 from detection results, this bboxes have overlaps. In this work, we use these overlaps as a criteria to discover parking vacancy. If the rate of overlap is very small, it means that two adjacent vehicle are far from each other so there is enough place for parking between the. However, when this ratio is high it means that two vehicles are closed and there is no other space for a new car between them. Here overlap rates were calculated by using Matlab bboxOverlapRatio function. When this ratio is smaller than the threshold, it could be ignored so there could be a parking spot between 2 vehicles. Here, searching for parking place would be stopped and the area of detected parking place would be calculated by sensor and compared with vehicle's size to make sure that it is a suitable place for parking.

7.1.5. Connect Matlab to Python-API

In order to access our trained detector during simulation and find parking place, Matlab should be connected to Carla because the default API in Carla is python so we should have made a bridge between python-API(which is directly connected to Carla) and our Matlab

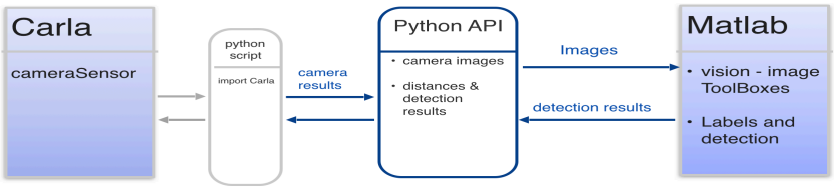


Figure 7.5.: Python-Matlab connection

codes(to access detector and detection libraries in Matlab). Figure 7.5 illustrate interface and the steps for connecting Matlab to Python.

Various methods and libraries has been defined in recent years either by Matlab and Python developers. In this work at first we decided to use MatlabEngine library which is offered by Matlab [35]. Matlab-engine provides different methods to read data from Matlab to python and vise versa. At first there were some settings in python and Matlab before using it as it should be installed in both sides (python and Matlab) and import to their libraries. However, this method was not successful for us because what we wanted to import from Matlab to python was our trained detector and the format of detector was not supported by Matlab engine. Detector type is specific type of double struct in Matlab and as it can be seen from type mapping table provided by Matlab [36], struct type always translated to dict type in python and when detector was translated to dict type, everything were disturbed in that as we could not get the bounding boxes and scores and also was not possible to access detector's parameters and functions for detection in that format. So other ways of connection should have been tested. There are also different tools offered by python as Numpy/Scipy libraries offered different tools for this goal. After testing different methods, finally what is offered by pymatlab [37] has been selected which was the fast way of connection and it also support different data types from Matlab. The good point is that we could access to the whole .m file at once without need of going through Matlab codes and read each variable separately. That is really straight forward where pymatbridge installed once and could be imported to our python files to access all of the .m file. Fig 7.6 shows the interface function for calling parking.m file from Matlab in PythonAPI. File parking.m includes detection function in Matlab which calls ACFDetector and calculate the detection results.

According to 7.1.3 section, to detect vehicles and find parking place we need to get an image of the vehicles and send it to detect() function in Matlab to compare it with detector. As we need image data for detection, we used a camera sensor which is attached to the ego vehicle and provides image files to send to Matlab.

```
from pymatbridge import Matlab


def interface(data):
    image=data.save_to_disk('/home/user1/Downloads/Carla9.5/PythonAPI/my projects/camera_result/%06d.jpg' % data.frame_number);
    print('this is the path to send to matlab: ',image);
    mlab = Matlab()
    mlab.start()
    overlap = mlab.run_func('parking.m', {'img': image})

    return overlap
```

Figure 7.6.: Call Matlab file from Python

```
global image;
camera_sensor.listen(lambda image: parking_decision(image));

def parking_decision(data):
 camera_sensor.stop();
 cameraControl = carla.VehicleControl(throttle=0);
 vehicle.apply_control(cameraControl);
 output=interface.decision(data);
 overlapRatio = output['result'];
 print('the rate of Overlap:', overlapRatio);
 if overlapRatio < 0.1:
   print('the parking place is between the next 2 cars');
   parking_preparation();
 else:
   print('no parking place has been detected right now');
   cameraControl = carla.VehicleControl(throttle=0.5);
   vehicle.apply_control(cameraControl);
   print('car is moving');
   camera_sensor.listen(lambda image: parking_decision(image));
```

Figure 7.7.: Camera-Sensor

Camera sensor

Camera sensor is placed in front of the ego vehicle and in each capture, its output which is an image file will be sent to .m file (Matlab) to find the parking vacancy. The idea is that in each tick or capture of the sensor, vehicle is stopped from moving and also camera sensor is stopped and wait for the result of Matlab file to see if the parking place is detected or not. If the parking place is there, camera sensor will be stayed deactivated and we will go to maneuvering step. Otherwise, it camera-sensor turned on again and vehicle set to continue to its forward moving and again by another capture, last process repeated again till a parking place would be detected. The codes for detection part and analyzing camera data are written in parking-decision function and can be seen in Fig 7.7. This code will be executed each time camera sensor provides a new data (image file) and is an iterative process till the detection of parking place.

The idea of deactivating camera sensor after each capture is to wait for the .m file or Matlab's result after each time of capturing. so it will makes a synchronization between Matlab and python. python is always faster than Matlab and that is also because of the process is done by Matlab as reading all detection libraries and comparing to the whole trained data in detector takes more time of process. so in order to sync the process, vehicle and sensor will be deactivated. Otherwise, if there would be no synchronization, vehicle continue its movement and camera sensor generates other photos and send them to Matlab while the process of last image in Matlab has been not finished yet. so many of output images from camera would be dropped and will not processed in Matlab even if they may contain important information. Also if vehicle continue to its movement while Matlab is in the process it may finished the path and passed through all vehicles and parking place while no photos has been sent during its movement and .m file is still processing last result. As there might be many repetitive photos in camera-sensor's result, it has been set to get photo in every 5 second instead of every second because in that way there would be many redundancy where many same JPEG files would be sent to .m file that makes the program really slow. As it has been seen in 7.7, overlap ratio is returned from Matlab as a result of detection in each sent image. Here this rate is compared with the threshold of 0.1(empirical value). If the overlap rate is less than threshold, the space between two vehicles ahead will be considered as a parking-bay.

7.2. Positioning phase

After detecting parking place, vehicle should be set to the start position for parking. we know that parking place is between the next two cars so the ego vehicle should pass these cars and stop beside the second car(parallel to that) to start a reverse movement to the parking vacancy. As already mentioned in 5 chapter, longitudinal and lateral displacement of the parking-bay should be also measured and this values should be compared to the ego vehicle's size to ensure that this is a right parking place where vehicle would have a safe movement without collision. Besides, these data will be also needed to implement maneuver algorithm in the next step. For the measurement of parking area, also distance to the obstacles and detection of other cars Obstacle-sensor of Carla has been used. As we have seen in 5.1, three sensors have been placed in the right side of the vehicle because parking place in our environment is in the right direction and one sensor the rear of vehicle car is for controlling distance to adjacent vehicles and ensure collision-free maneuver. Sensor in the center of right side of the vehicle is for detection of other vehicles. During stabilization phase, vehicle moves slowly in the left side of the parked vehicles and results from the sensor are read. we used a counter to record number of detected vehicles by the sensor when the counter comes to 3, it means that vehicles should be stopped and this sensor will be stopped. Why this is 3? because at start point where we get the result of Matlab detection, vehicle is located in the side of a non-player where stabilization phase is started. This non-player was not in the last image of camera sensor where parking vacancy detected but when vehicle starts to move this non-player will be also detected by obstacle sensor so counter will be also increase by that detection at start

```
#********************** Set vehicle's start position for parking maneuver **************

def parking_preparation():
 vehicle.apply_control(carla.VehicleControl(throttle=0.4));
 sensor_obstacle_rightSide.listen(lambda data: detect_non_players(data));

def detect_non_players(data):
   current_vhcl = data.other_actor.id;
   if config.nonplayer != current_vhcl and current_vhcl != 0:
     config.count += 1;
     config.nonplayer = current_vhcl;

   if config.count == 3:
     print('time of parking');
     frontDistance=data.distance;
     sensor_obstacle_rearRight.listen(lambda data: stablePosition(frontDistance, data));

   def stablePosition(frontDistance, event):
     if event.distance == frontDistance:
        vehicle.applyControl(carla.VehicleControl(throttle=0.0,steer=0.0));
        parking_maneuver();
```

Figure 7.8.: Parking-preparation

of the movement. Counter should become to 3 means that vehicle has passed 2 vehicles in front besides the first one at the start of moving. To find longitudinal displacement of parking bay, sensor locations after passing the first vehicle(counter=2) and in start of detection of second vehicle(right after counter becomes 3) will be recorded. Distance of these 2 locations is longitudinal displacement of parking-bay. when second vehicle is detected, vehicle should be stopped moving. In this step distances of 3 right sensors to the parked vehicle should have the same value to ensure that vehicle is in parallel to the parked car. The codes for this step has been written in a function called parking-preparation as it can be seen in fig 7.8.

7.3. Maneuvering Phase

Implementation of this phase realizes the concepts presented in chapter 6. All of the implementation codes has been written in python using python mathematical tools and numpy libraries. Calculation of steering angle, time and velocity has been written in maneuver.py file which has been imported to the main file (parking.py) and will be called during simulation and after positioning phase. When vehicle stops near to the front parked vehicle, parking-maneuver() function will be called. At first of maneuver T_{max} and Φ_{max} calculation function is called from the maneuver.py and then by having the whole maneuver time (T_{max}), in an iterative process parking function from maneuver.py will be called. In each iteration time value is sent to the parking function which returns steering angle and velocity at each time stamp. So parking function will be called in each time of simulation and its maximum value is maximum of maneuvering time (T_{max}). Fig 7.9 shows the parking maneuver function.

As it can be seen in 7.9, to control collision, obstacle sensors are used and each time the sensor generates data, another function called control-maneuver() will be called so the distance of vehicle to detected obstacles will be checked in each time stamp and if the distance is considered to be safe, parking function from maneuver.py is executed. Parking() function gets time and also vehicle data as input arguments to set the vehicle's arguments as speed

```
#**********************parking maneuver step*****************************

def parking_maneuver():
 time.sleep(3);
 #calculate T-max and Ø-max
 maneuver.calculate_maneuverTime(vehicle);
 maneuver.calculate_max_steeringAng(vehicle);
 time.sleep(3);
 sensor_obstacle_rearCenter.listen(lambda data: control_maneuver(data));

 def control_maneuver(data):
   for t in numpy.arange(0,config.T,config.sampling_period):
     if data.distance <= 1.2:  #check distance to rear car
       vehicle.apply_control(carla.VehicleControl(steer=0,throttle=0));
       break;
     maneuver.parking(t,vehicle);
     time.sleep(0.05);
```

Figure 7.9.: Parking-maneuver function

and steering. So maneuver.parking(t, vehicle) access to the steeringAngle(t) and velocity(t) in maneuver.py and each time of simulation, it gets new steering and velocity values and apply this new values to the vehicle. As we can also see from the above codes, end of maneuver depends on the sensor output as this output, when vehicle's distance to rear side is small, ego-vehicle should be stopped and the situation is considered as parked. So the loop will be broken and steering and velocity values will be set to 0 and vehicle will be stopped. It is also important to mention that, because of the problem we had in Carla to set velocity values, this value is not set directly and is set through throttle parameter. Because by setting throttle value, velocity of vehicle will be also affected as their values are not independent and also if we just set the velocity by using set-velocity() function in Carla without setting throttle, engines brakes will decrease vehicle velocity [38]. Fig 7.10 shows how the velocity has been set in this implementation. As it can be seen, velocity of vehicle is compared with the expected velocity which we get from velocity(t) function and if it would be higher, an increase is applied to the brake otherwise, throttle would be increased so we set the velocity by controlling the throttle like in real car. Time maneuver and phi are defined as global value so they could be accessed by all functions and could be changed during calculation we also needed to define a a maximum value for velocity which should be accessed by velocity(t) function. All of these global and constant values have been defined in config.py file and are accessible with all the other python scripts. Sampling period of the maneuver simulation should be also defined and all of the calculations are based on this sampling-period. Sampling-period in Carla is adjusted by FPS which is fps rate or sampling rate of simulation. That is also possible to set the fix sampling rate of simulation in Carla and in this project, it is set to -fps 20 in each time of simulation. Every time we perform Carla we should set the parameters $-benchmark - fps20$ where benchmark means that simulation will be in a fix sampling rate and fps 20 means that simulation is in every 0.05s (seconds). Hence, sampling-period of maneuver has been defined to 0.05 in config.py and this value will be used to calculate T_{max} and Φ_{max}. During calculations of time, steering, loops are incremented by this value and in a different sampling

```
velo = vehicle.get_velocity().y;
control=vehicle.get_control();
throttle = control.throttle;
brake = control.brake;
if velo < abs(v):
  print('throttle:',throttle,'control.throttle:',control.throttle);
  throttle = throttle + 0.5;
else:
  brake = brake + 0.5;
  print('brake value:',control.brake);
```

Figure 7.10.: Set Velocity For Parking Motion

rate, maneuver would not be worked or sampling-period in config.py should be changed by changing fps.

8. Evaluation

This section considers functionality of the presented approach and how it works in comparison with real parking with a real driver(not automatic ones). To this end, at first the detection process (detection of vehicle and parking-bay) will be evaluated and then maneuver process and the problems during maneuvering process will be discussed.

8.1. Finding parking place

As already mentioned above, ACF method were finally selected for the detection phase as its rate on our training data was the best compared to other methods. The trained detector of this method was successful to detect our vehicles. Detection itself is done in Matlab and our main API is python(PythonAPI to import Carla) so we had to make an interface vetween Python and Matlab but this connection process, made our system a little slow. using camera-sensor from Carla and waits for the Matlab detection result was the only problem we had. As it was explained in the last chapter, after each tick(or capture) of camera-sensor, vehicle should be stopped till the detection results come from Matlab file. These stops makes our vehicle a bit different from what we see in normal car parking. As we explained already, by changing the camera sensor attributes and increase the time between captures, this process can become faster but we should be careful that if this time between ticks increased a lot like more than 0.4 seconds, then some parts of the simulation that would contain a parking place would be ignored and would not be sent to the detection function. So this way would not also helpful to make a faster detection process. The other problem is that each time that camera-sensor generate a new data(JPEG file), we should connect to Matlab again which is the method used by pymatbridge. Although it is the best method for connecting that lets to access the whole .m file as once, each time we send a new data we should make a new connection to Matlab. This would be a problem for our system and also it would cause hardware problem because when there are many calls to Matlab and many connection and disconnection, it causes OS problem and system stops so for the scenario that parking vacancy is so far from the ego vehicle and many photos should be sent to Matlab before the vehicle reaches to the detected parking place, this method could not work well. So maybe it would be a good idea to change Matlab to python itself by translating detection parts from Matlab to other environment so there would not require to make these connections between Matlab and python. This idea can be considered as a future works.

8.2. maneuver evaluation

Here maneuvering phase has been evaluated and also the detailed problems and limitations to make a perfect parking motion are considered.

8.2.1. Lateral Displacement of Parking-bay

As it has been already mentioned in chapter 7, the measurements to get the width of parking-bay or lateral displacement is based on the location of parked vehicles and distance between the detected vehicle and the corner side of the ego vehicle. So because of the problem at first we had with the obstacle sensor as it was just detecting Carla actors as vehicles, we could not detect the sideways or side of the parking to directly measure the distance of ego vehicle to the sideways and get lateral displacement of the parking. So what we get as a lateral distance is not really precise value which may also affect on parking motion and the calculation of formulas to get the steering and velocity magnitudes. This is assumed that by having providing more precise values, parking motion would be also improved.

8.2.2. Measuring sensors' distances during maneuver

As it has already mentioned, in order to prevent collision during movement, distances from other vehicles should be checked from obstacle sensors results. This measurement is also used to find the stop point as the sensor in the back of the vehicle compares the distance to the backward car and if it reaches the certain distance, steering angle as well as speed parameters are set to zero and the motion algorithm would be stopped. Here is a problem that the ego vehicle has not completed the parking motion yet and it is not in a parallel and perfect position compared to other adjacent cars but as the distance to the rear obstacle reached to the goal distance, vehicle should be also stopped without completing the motion so here some other forward and backward motions are required to make the perfect parking so further step of this research project would be to improve this method and make the parking algorithm as iterative [2]. This means that in the next iterative movement direction is changed to opposite(i,e. in our case the second movement would be forward movement) and then these processes will be repeated till the car get the expected position in the parking-bay.

8.2.3. Slippage and Frictions During Maneuver

In this algorithm, frictions and slippage between vehicle's wheels and the ground are not considered and formula for steering and velocity measurement are only based on the vehicle's own parameters as steering and velocity. The problem is that in different executions even with the same scenario and no changes in the code or values, some little changes in vehicle movement can be seen like the final position in the parking place may have been a bit different from the last executions or how deep the vehicle enters to the parking in the first turn might be changed in different executions. In order to make our program more reliable in the real world, the friction values and environment situation should be also considered. This can be done by making some changes in formulas or by choosing a new maneuver algorithm.

9. Limitations

This chapter points out the most important limitations and problems during simulation so the solutions to these problems are considered as future works in order to improve the implementation.

9.1. sensor measurement and access to a suitable sensor

As already mentioned above, Carla is a very new simulator which is still under active development. Many of its features are still under test. Among them, we can point to the sensors that are currently offered by Carla. Popular sensor for autonomous parking during recent years was UltraSonic sensors [21] but this sensor is not yet presented by Carla. The other choice was Radar sensor which has also provided great results in autonomous driving project during last years [39]. However, Carla offers Lidar sensor and use of Lidars for autonomous parking project would be a really novel idea as most of the previous methods were using UltraSonics so by testing this algorithm with Lidar, we could compare the results to Ultrasonic methods. The problem was that the output data we were receiving from Lidar sensor were not totally reliable and perfect, i.e, they did not contain all information of surrounding obstacles and also the 3D visual results(extracted as .ply formats) were not clear enough to make range measurements. Hence, in the limited time of thesis research we were not able to figure out Lidar problems or fix it for our project. Last solution was to substitute Lidar with other sensors offered by Carla. Obstacle sensors and their out put results were the most suitable choice for this work. The idea of this sensor is to detect surrounding obstacles and get the distance to them which was what we were searching for. However, in the earlier version of Carla that we were using, obstacle sensor was not working correctly as the detection of obstacle was a problem in earlier version and finally in Carla version 0.9.5, it has been improved so in the time of publishing this version by Carla developers, we finally could use sensors in our project but still measurement(like distance to detected object) was not working. Fig 9.1 shows some results of distance information provided by obstacle sensor. As it has been shown it always shows the same distance. The problem was because of a mistake in the sensor code that it gets Radius measurements instead of distance which we tried to fix this problem and finally get the distance from it. However, for the parking maneuver to get the parking area measurements and information like lateral displacement of parking bay, distance to other obstacles like side of the parking-bay should be measured so sensor should detect objects other than vehicles like street curbs or sides of the parking place. Instead of getting the distance to the side of the street, distances to the side of the parked vehicles were measured and in this way we just made an estimation of the lateral displacement of the

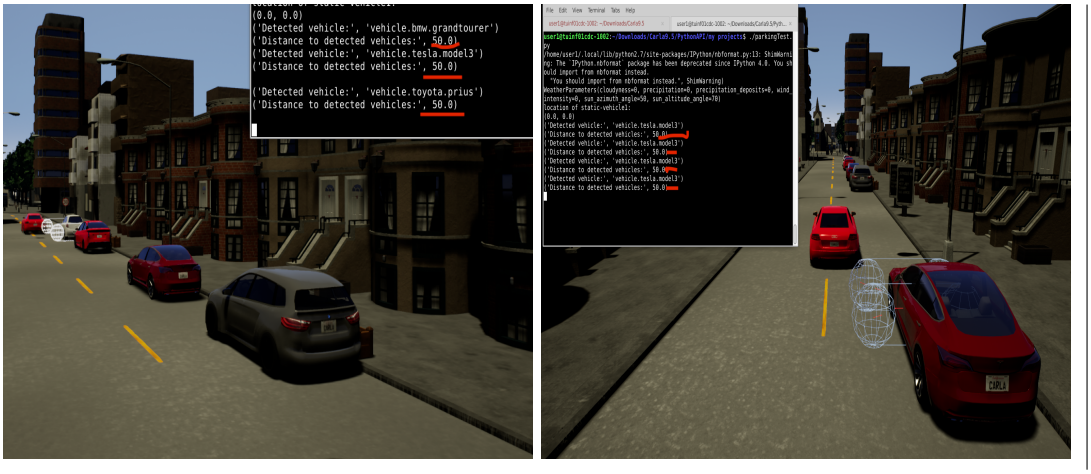


Figure 9.1.: Obstacle Sensor Results

parking-bay which is not an exact measurement. So finding of a perfect sensor which would help us to meet our expectation and measurement goals could be considered as a future work that would also results on better implementation.

9.2. API problems

The main language supported by Carla is Python as its API is PythonAPI but it is also possible to add C++ through libCarla(library provided by Carla). One of the problem as it has already been mentioned in the previous chapters, was using Matlab and its tools for machine learning. Although we managed to make an interface between Matlab and python and finally enjoyed the machine learning tools and libraries of Matlab, this interface made a detection process really slow and we had a synchronization problem between Matlab and python as we decided to make interruption by stopping vehicle and deactivating sensors in each process but this several stop/move vehicle made the program look a bit different from real world where a real driver searching for a parking place and also several connection and disconnection to Matlab engine causes OS problem and when these connect/disconnect are a lot, they stopped simulator from running so the program will be crashed and finally it the whole system and OS will be stopped. These are some of the API limitation during simulation.

10. Future Work

This chapter presents all of the possible improvements of this simulation in the future. The goal is to use this method as a practical and reliable way of parking on a real steer without help of driver.

10.1. Design a new sensor to improve range measurements

Sensors are the main problem as it has been mention in the last chapter 9 which are not giving perfect measurement result. In order to make this parking maneuver algorithm practical before the deadline of master thesis, best solution was to choose between current sensors offered by Carla simulator. This was working fine but for making better result and more accurate range measurement, a better sensor should be used. One idea would be to design a new sensor which can also be interface between different sensors in Carla for instance a sensor could be defined here which would be just attached to the ego vehicle and find a detected vehicle and from the id of detected vehicle find its location and the coordinate of each edge of the vehicle to get a precise distance. we could also use Lidar sensor here and fix the problem of it to show 3D visual results. Other solution is to get the sensors from other applications and find a way to connect it to Carla or to the Lidar channels that are currently presented by Carla. So a new sensor should be designed in future development.

10.2. Improve maneuver algorithm

As already pointed out in the previous chapters, the idea of this maneuver algorithm is to implement iteratively to get the better results. It can be also seen in real parallel parking in street(not garage parking) that when driver wants to park in a small parking place, there should be some iterative forward/backwards movement of the car to get the perfect position in the parking place. By the first maneuver(as we implemented here), vehicle enters to the parking place but its position in the parking is not always perfect. That may be parked crooked or may not be positioned as the same line as other adjacent vehicles. Hence, to make a better position, in the next maneuver it should move forward and get the opposite direction for movement(velocity signs changed from -v to +v for moving forward) and also opposite direction of steering angle(from $-\phi$ to $+\phi$ and vise versa). This movements will be repeated till the vehicle gets a perfect place in the parking-bay as a driver always does in parallel-parking. So another ongoing improvement is to make the maneuver iterative to contain forward movements besides backward movement. To this end, a perfect final constraint for the end of this iteration should be defined. This constraint would define the

end position of the vehicle in the parking place. Sensors are also required here to measure precise distances to rear and front cars to decide for final location of vehicle in parking-bay. Number of iteration depends on the ego vehicle's size as well as parking's area.

10.3. test in other simulator

Another idea is to test this method in other simulators to see how algorithm would be worked in different scenarios and use various tools that other simulators provide i.e, for measurements, sensors, map designing or other vehicles provided with them. That would be a great idea to see if the current detection method find parking vacancy in other environment and also if this implementation of parking maneuver, our codes and calculations are not limited to Carla simulator which would ensure that this algorithm is reliable and could be prepared to use on a real street.

10.4. consider traffic during parking

In this simulation parking scenario has been implemented in a quiet street where there is no traffic , no other vehicles moving on street and no human. The only obstacles were parked vehicles. In order to make this work similar to real parking challenge on streets, it would be a good point to consider traffics of car and human during simulation.

11. Conclusion

This study explains an implementation of parallel parking algorithm and the results of this method. As car manufactures have considered autonomous driving over the past decade, much research has been devoted to this area. Most results, however, are still hypothetical. In this thesis we specifically worked on the implementation of parallel parking and presented an implementation of parallel parking on the side of a street without line markings or defined parking spaces. Parallel parking consists of different steps: vehicle detection, parking vacancy detection, setting the vehicle starting position and finally parking maneuver to bring the vehicle into the appropriate parked position, relative to other vehicles on the street without crashes into other vehicles or surrounding obstacles. In this implementation a combination of vision- and sensor-based methods have been used. A vision based approach and machine learning algorithms were used for the detection of surrounding objects and to perform distance measurements. Although, the presented implementation was successful, it is not yet ready to apply to real vehicles on the street. Sensors should be improved to provide more precise measurement and to this goal, a study on different sensors or may be a design of a sensor which is using a variation of sensors[5] should be done to get a safe and robust maneuver. In addition, parking maneuver algorithm should be improved a little as described in 10.

A. Appendix

A.1. Parking vacancy detection in Matlab

```
function ratio=parking(path)
  ratio = 100;
  I = imread(path.img);
  detector = load('~/matlabR2018b/bin/newDetectors-8thJun/acfDetector.mat');
  [bboxes, scores]=detector.acfDetector.detect(I);
  if isempty(scores)
    sprintf('nothing detected!')
  else
    [selectedBboxes,selectedScores] = selectStrongestBbox(bboxes,scores,'RatioType',
    'Union','OverlapThreshold',0.1)
     for i=1: length(selectedScores)-1
      if i+1 <= length(selectedScores)
        if selectedScores(i) > 20 && selectedScores(i+1) > 20 && selectedBboxes(i,4) > 40
          B1 = selectedBboxes(i,:);
          B2 = selectedBboxes(i+1,:);
          distance=abs(B1-B2)
          if distance(1,1) <= 130
            ratio=bboxOverlapRatio(selectedBboxes(i,:), selectedBboxes(i+1, : ))
          else
            sprintf('vehicles do not have enough space from each others')
          end
        else
          sprintf('not enough scores to decide!')
        end
    else
      sprintf('all detection been processed')
      break;
   end
 end
end
```

A.2. Calculation of maneuver time

```
def calculate_maneuverTime(vehicle):
  x=vehicle.get_location().x;
  y=vehicle.get_location().y;
  orientAngl=vehicle.get_transform().rotation.yaw;
  ts=0;
  cond = True;
  while cond:
    for ts in numpy.arange(0,config.T,config.sampling_period):
        s_angle = steeringAngle(ts);
        velo = velocity(ts);
        if(s_angle == 0):
          orientAngl_lastStep = orientAngl;
          orientAngl = orientAngl;
          x = x + (velo * config.sampling_period * math.cos(orientAngl));
          y = y + (velo * config.sampling_period * math.sin(orientAngl));
        else:
          orientAngl_lastStep = orientAngl;
          orientAngl = orientAngl + (((velo * config.sampling_period)/
             config.vehicle_length)*math.sin(s_angle));
          x = x + ((config.vehicle_length / math.tan(s_angle)) * (math.sin(orientAngl)
            - math.sin(orientAngl_lastStep)));
          y = y - ((config.vehicle_length / math.tan(s_angle)) * (math.cos(orientAngl)
           - math.cos(orientAngl_lastStep)));
        cond=longitudinal_condition(vehicle.get_location().x,x,vehicle.get_location().
           y,y,vehicle.get_transform().rotation.yaw);
        print('longitudinal cond:', cond);
        config.T += config.sampling_period;
        print('T calc values',config.T);
    config.T -= config.sampling_period;
```

A.3. Calculation of steering-angle

```
def calculate_max_steeringAng(vehicle):
  x=vehicle.get_location().x;
  y=vehicle.get_location().y;
  orientAngl=vehicle.get_transform().rotation.yaw;
  ts=0;
  cond = False;
  while not cond:
    config.phi_max -= 0.0872665
    for ts in numpy.arange(0,config.T,config.sampling_period):
    s_angle = steeringAngle(ts);
    velo = velocity(ts);
    if(s_angle == 0):
      orientAngl_lastStep = orientAngl;
      orientAngl = orientAngl;
      x = x + (velo * config.sampling_period * math.cos(orientAngl));
      y = y + (velo * config.sampling_period * math.sin(orientAngl));
    else:
       orientAngl_lastStep = orientAngl;
       orientAngl = orientAngl + (((velo * config.sampling_period)/
          config.vehicle_length)*math.sin(s_angle));
       x = x + ((config.vehicle_length / math.tan(s_angle)) * (math.sin(orientAngl)
          - math.sin(orientAngl_lastStep)));
       y = y - ((config.vehicle_length / math.tan(s_angle)) * (math.cos(orientAngl)
          - math.cos(orientAngl_lastStep)));
    cond=lateral_condition(vehicle.get_location().x,x,vehicle.get_location().
       y,y,vehicle.get_transform().rotation.yaw);
```

List of Figures

List Of Abbreviations

ACF Aggregate channel features. 10, 19

ADAS Advanced Driver Assistance Systems. 1

AI Artificial Intelligence. 8

AP Autonomous Parking. iv, 1, 2, 12, 18

API Application Programming Interface. iv, 2, 5, 42

AV Autonomous Vehicle. 1, 4, 7

BBox Bounding Box. 33–35, 52

CLMR Car Like Mobile Robot. 2, 15, 16, 27

CNN Convolutional Neural Network. 9, 10, 18, 52

DL Deep Learning. 8–10

Gnss Global Navigation Satellite System. 6

HOG Histogram of Oriented Gradients. 9, 18

INRIA Institute for Research in Computer Science and Automation. 1

ML Machine Learning. 8, 9, 18, 32

NN Neural Network. 8, 9, 14, 18

R-CNN Regions with CNN features. 9, 18, 21

RPN Region Proposal Network. 10

YOLO You Only Look Once. 10, 14, 19

Bibliography

[1] *Automatic parking*. URL: https://en.wikipedia.org/wiki/Automatic_parking#Commercial_systems.

[2] I. E. Paromtchik and C. Laugier. "Autonomous parallel parking of a nonholonomic vehicle". In: *Proceedings of Conference on Intelligent Vehicles*. Sept. 1996, pp. 13–18. DOI: 10.1109/IVS.1996.566343.

[3] K. Jiang and L. D. Seneviratne. "A sensor guided autonomous parking system for nonholonomic mobile robots". In: *Proceedings 1999 IEEE International Conference on Robotics and Automation (Cat. No.99CH36288C)*. Vol. 1. IEEE, May 1999, 311–316 vol.1. DOI: 10.1109/ROBOT.1999.769997.

[4] D. J. C.-M. C. Dr. Gene Yeau-Jian Liao Dr. Chih-Ping Yeh and M. H. Liu. "Automatic Parking Vehicle System". In: American Society for Engineering Education, 2016, 2016.

[5] F. Hucko. *Master ThesisThe development of autonomous vehicles*. Aalborg University Copenhagen, 2017.

[6] J.Nyambal and R.Klein. "Automated parking space detection using convolutional neural networks". In: (Nov. 2017), pp. 1–6. DOI: 10.1109/RoboMech.2017.8261114.

[7] L. Li, L. Zhang, X. Li, X. Liu, Y. Shen, and L. Xiong. "Vision-based parking-slot detection: A benchmark and a learning-based approach". In: (July 2017), pp. 649–654. ISSN: 1945-788X. DOI: 10.1109/ICME.2017.8019419.

[8] T.-H. S.Li and S.-J. Chang. "Autonomous fuzzy parking control of a car-like mobile robot". In: *IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans* 33.4 (July 2003), pp. 451–465. ISSN: 1083-4427. DOI: 10.1109/TSMCA.2003.811766.

[9] Y.Zhang, S.Tang, and Z.Zhang. "A novel parking control algorithm for a car-like mobile robot". In: (Apr. 2012), pp. 377–381. DOI: 10.1109/ICNSC.2012.6204948.

[10] J.Chen and C.Hsu. "A visual method for the detection of available parking slots". In: (Oct. 2017), pp. 2980–2985. DOI: 10.1109/SMC.2017.8123081.

[11] A. Reisner, A. Weidinger, and D. Werner. "IDP-Project: Autonomous Parking Scenario". In: Informatik F13 - Chair of Operating Systems - Technical University of Munich, 2017.

[12] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun. "CARLA: An Open Urban Driving Simulator". In: *Proceedings of the 1st Annual Conference on Robot Learning*. 2017, pp. 1–16.

[13] *Ego-Vehicle*. URL: https://www.gpsworld.com/intelligent-transportation-systems-require-the-ego-vehicle/.

[14] *ML vs. DL*. URL: https://www.datacamp.com/community/tutorials/machine-deep-learning.

[15] J.Nyambal and R.Klein. "Automated parking space detection using convolutional neural networks". In: (Nov. 2017), pp. 1–6. DOI: 10.1109/RoboMech.2017.8261114.

[16] *RCNN methods*. URL: https://ch.mathworks.com/help/vision/ug/faster-r-cnn-basics.html.

[17] *Mask RCNN*. 2018. URL: https://arxiv.org/abs/1703.06870.

[18] *ACFObjectDetector*. URL: https://ch.mathworks.com/help/vision/ref/acfobjectdetector.html?s_tid=doc_ta.

[19] H.G.Jung, Y.H.Cho, P.J.Yoon, and J.Kim. "Scanning Laser Radar-Based Target Position Designation for Parking Aid System". In: *IEEE Transactions on Intelligent Transportation Systems* 9.3 (Sept. 2008), pp. 406–424. ISSN: 1524-9050. DOI: 10.1109/TITS.2008.922980.

[20] R.Prophet, M.Hoffmann, M.Vossiek, G. Li, and C.Sturm. "Parking space detection from a radar based target list". In: (Mar. 2017), pp. 91–94. DOI: 10.1109/ICMIM.2017.7918864.

[21] W.-J. Park, B.-S. Kim, D.-E. Seo, D.-S. Kim, and K.-H. Lee. "Parking space detection using ultrasonic sensor in parking assistance system". In: (June 2008), pp. 1039–1044. ISSN: 1931-0587. DOI: 10.1109/IVS.2008.4621296.

[22] T.Wu, P.Tsai, N.Hu, and J.Chen. "Research and implementation of auto parking system based on ultrasonic sensors". In: (Nov. 2016), pp. 643–645. DOI: 10.1109/ICAMSE.2016.7840267.

[23] F.Bock, D.Eggert, and M.Sester. "On-street Parking Statistics Using LiDAR Mobile Mapping". In: (Sept. 2015), pp. 2812–2818. ISSN: 2153-0017.

[24] P.Ball, R.Liao, C.Roman, S.Ou, and E.Pow. "Analysis of fixed and mobile sensor systems for parking space detection". In: (July 2016), pp. 1–6. DOI: 10.1109/CSNDSP.2016.7574009.

[25] L.Zhang, J. Huang, X.Li, and L. Xiong. "Vision-Based Parking-Slot Detection: A DCNN-Based Approach and a Large-Scale Benchmark Dataset". In: *IEEE Transactions on Image Processing* 27.11 (Nov. 2018), pp. 5350–5364. ISSN: 1057-7149. DOI: 10.1109/TIP.2018.2857407.

[26] N.Bibi, M.N.Majid, H.Dawood, and P.Guo. "Automatic Parking Space Detection System". In: (Mar. 2017), pp. 11–15. DOI: 10.1109/ICMIP.2017.4.

[27] C.-C. Huang, S.-J. Wang, Y.-J. Change, and T. Chen. "A Bayesian hierarchical detection framework for parking space detection". In: (Mar. 2008), pp. 2097–2100. ISSN: 1520-6149. DOI: 10.1109/ICASSP.2008.4518055.

[28] K. Choeychuen. "Available car parking space detection from webcam by using adaptive mixing features". In: (May 2012), pp. 12–16. DOI: 10.1109/JCSSE.2012.6261917.

[29] G. Notomista and M. Botsch. "A Machine Learning Approach for the Segmentation of Driving Maneuvers and its Application in Autonomous Parking". In: *Journal of Artificial Intelligence and Soft Computing Research* 7.4 (2017), pp. 243–255. URL: https://content.sciendo.com/view/journals/jaiscr/7/4/article-p243.xml.

[30] K. Jiang and L. D. Seneviratne. "A sensor guided autonomous parking system for nonholonomic mobile robots". In: 1 (May 1999), 311–316 vol.1. ISSN: 1050-4729. DOI: 10.1109/ROBOT.1999.769997.

[31] T. .-.-. S. Li and S.-J. Chang. "Autonomous fuzzy parking control of a car-like mobile robot". In: *IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans* 33.4 (July 2003), pp. 451–465. ISSN: 1083-4427. DOI: 10.1109/TSMCA.2003.811766.

[32] R. M. Murray and S. S.Sastry. "Steering nonholonomic systems using sinusoids". In: *29th IEEE Conference on Decision and Control*. Dec. 1990, 2097–2101 vol.4. DOI: 10.1109/CDC.1990.203994.

[33] *groundTruth*. 2019. URL: https://ch.mathworks.com/help/vision/ref/groundtruth.html.

[34] *GroundTruthLabeler*. URL: https://ch.mathworks.com/help/driving/ref/groundtruthlabeler-app.html.

[35] *matlabEngine*. 2019.

[36] *matlabEngine-data types*. 2019. URL: https://ch.mathworks.com/help/matlab/matlab_external/handle-data-returned-from-matlab-to-python.html?searchHighlight=Handle-2520Data-2520Returned-2520from-2520MATLAB-2520to-2520Python.

[37] *PyPI*. 2019. URL: https://pypi.org/project/pymatlab/.

[38] *velocity and throttle*. 2019. URL: https://github.com/carla-simulator/carla/issues/1652issuecomment-513626791.

[39] R. Prophet, M. Hoffmann, M. Vossiek, G. Li, and C. Sturm. "Parking space detection from a radar based target list". In: *2017 IEEE MTT-S International Conference on Microwaves for Intelligent Mobility (ICMIM)*. Mar. 2017, pp. 91–94. DOI: 10.1109/ICMIM.2017.7918864.