Budapest University of Technology and Economics
Faculty of Electrical Engineering and Informatics
Department of Automation and Applied Informatics

Autonomous valet parking system in dynamically changing environment

Scientific Students' Association Report

Authors
András Kondákor
Gábor G. Varga
Márton Antal

Supervisors
Dr. Ákos Nagy
Domokos Kiss
Gábor Csorvási

October 28, 2020

Contents

List of Figures

List of Tables

Kivonat

A teljesen autonóm, illetve önvezető autó megvalósítása korunk egyik legjelentősebb mérnöki kihívása. Ami valaha tudományos-fantasztikumnak számított, az a technológia fejlődésével mára már valósággá válhat. Noha az önvezető autók még korai stádiumban járnak, de elterjedésük forradalmasíthatja a forgalomirányítási és közlekedési rendszereket. Ennek elérése érdekében több összetett kihívást kell leküzdeni először.

Az önvezető autókhoz kapcsolódó számos megoldandó probléma egyike a parkolóhelyek automatikus detekciója. A jövőben a lakosság számának növekedésével a járművek száma is folyamatosan növekedni fog, így egyre nagyobb lesz az igény a parkolóházak iránt, elegendő parkolóhely biztosításának céljából. Azonban a parkolóhely keresés könnyen fáradságos munkává válhat, melyet nem lehet figyelmen kívül hagyni. Időt, energiát és üzemanyagot fogyaszt, valamint a sofőrnek akár többször is köröznie kellhet a parkolóban, mire végül talál egy üres helyet. A dolgozat az automatizált parkolóhely detektálás kérdésének megoldása érdekében bemutat egy ultrahangos szenzorokat és LIDAR-okat használó módszert.

A feladat teljesítésének egyik szükséges feltétele a jármű pályájának megtervezése. Dinamikus környezet révén kizárólag alacsony számításigényű módszerek működhetnek megbízhatóan. Ennek a problémának az áthidalására mintavételen alapuló fa-építő algoritmust és a Reed's-Shepp lokális tervező folytonos görbületű változatát használó módszert mutat be a dolgozat. Ezeket a tervezőket felhasználva a megoldás olyan utat eredményez, melyet a robot anélkül tud követni, hogy meg kellene állnia a kerekek kormányzásához.

A közeljövőben a hagyományos és az autonóm járművek párhuzamosan lesznek jelen a mindennapokban, így az utóbbiaknak le kell tudnia küzdenie az ebből adódó nehézségeket. A mozgó objektumokkal (mint például a többi autóval és a sétáló emberekkel) rendelkező környezetben működő autonóm járműnek képesnek kell lennie kezelni a velük való találkozást. Ennek megoldására egy RGB-D kamerát alkalmazó algoritmust mutat be a dolgozat, ahol az objektumokat gépi tanulás segítségével detektálják és mozgásukat becsülik a mélységi információ felhasználásával.

A dolgozat a fent említett kihívásokkal foglalkozik, és arra törekszik, hogy lehetséges megoldásokat találjon rájuk. Az első három fejezetben a megvalósított algoritmusok kerülnek bemutatásra, melyek a fő kérdésekre összpontosítanak, mint a parkolóhelyek felismerése, az útvonaltervezés és a dinamikus objektumok detektálása. Ezt követően bemutatásra kerül, hogy ezek a megoldások hogyan alkotnak egy rendszert, amely képes az autonóm parkolási feladat végrehajtására. Az utolsó részben az elvégzett tesztek kerülnek ismertetésre és ezek alapján jövőbeni fejlesztési javaslatok kerülnek felvetésre.

Abstract

Building a fully autonomous or self-driving car is one of the most demanding engineering challenges of our century. What was once considered a science-fiction can become a reality as the technology matures. However, self-driving cars are fairly in their infancy today, but once implemented they can revolutionize traffic management and transport systems. In order to achieve that, numerous complex challenges must be overcome first.

One of the many scenarios associated with self-driving cars is related to automatic parking space detection. In the future, as the population grows the number of vehicles will continue to rise and it is likely that there will be an increasing demand for parking garages to provide sufficient parking space. Nevertheless, finding a parking spot can easily become an issue which cannot be neglected. It consumes time, energy, fuel and the driver may have to circle in the parking lot until finally finding a vacant spot. To solve the issue of automated parking space detection, this paper presents a method using ultrasonic sensors and LIDARs.

A necessary condition for the problem is planning a collision-free path for the car. Being in a dynamically changing environment, the considered algorithms must provide trajectories with low computational demand that can be followed precisely for car-like robots. To cope with this problem, a sampling-based tree-growing planning method in conjunction with a continuous-curvature version of the Reed's-Shepp local planner is used. The coupling of these planners results in such paths which the robot can follow without ever having to stop for reorienting its wheels.

In the near future, conventional and autonomous vehicles will operate simultaneously, so the latter ones need to deal with the resulting difficulties. For a robot functioning in an environment with dynamic objects, e.g. other cars and people walking in the garage, it is crucial to manage situations where they cross each other's paths. To resolve this, an algorithm employing an RGB-D camera is implemented, where objects are detected with the help of machine learning and their movement is predicted utilizing depth information.

This paper deals with the aforementioned challenges and aims at finding potential solutions to them. In the first three chapters, the implemented algorithms are presented. These parts are focusing on the main issues which are vacant parking space recognition, path planning and dynamic object detection. After that, it is explained how these solutions create a system, which is able to perform autonomous valet parking. Finally, the system is tested in a simulated environment, where the operation is examined and suggestions are proposed about possible future improvements.

Chapter 1

Introduction

In this chapter, we first present the problem which we are undertaking to solve. After that, a brief overview of the implemented system is introduced. Finally, a short description of the employed framework is given.

1.1 Valet Parking Problem

The car is one of the most commonly used means of transport, without which today's life would be inconceivable. As the population grows, the number of privately-used vehicles increases as well, but the number of parking spaces remains for the time being [1]. It is assumed that numerous parking garages and parking lots will be built in the future to overcome this issue. As technology improves, drivers do not need to bother themselves by searching for an appropriate parking space and parking their cars. Instead, they just simply get off of their cars at the entrance of these parking garages and go about their businesses. In the past and present, this service was already available in the form of Valet Parking offered by many restaurant, hotels, stores and other establishments, where the aforementioned task was performed by a so-called valet. In our research, we aimed at achieving this by leaving the human resources out and lean completely on autonomous, intelligent cars to carry the task out.

First and foremost, finding a proper parking space is a burden for many drivers which cannot be neglected. Owing to the increased demand for autonomous driving and advanced driver assistant systems, automatic parking system is a topic which have been widely researched. Automatic parking systems start by recognizing vacant parking spots, but since parking lots and garages are uncontrolled environments, various obstacles and illuminations are present, therefore it is a challenge for such systems to properly detect available locations. Furthermore, these systems prefer to utilize sensors already installed on mass produced vehicles for ease of commercialization. Among a variety of range-finding sensors, ultrasonic sensor based approaches are the most popular [2]. However, even if finding a proper parking spot but parking a vehicle is always a troublesome problem to drivers because it is hard to know the exact turning time [3] and it is also generally considered a high-stress maneuver [4]. New drivers, disabled people, unskilled and especially aged drivers can take benefit from our proposed system and also those who wish to eliminate the time spent on the tedious process related to parking. It is hopeful that the concept of our system will revolutionize parking scenario in the future.

1.2 System Overview

As Section 1.1 gave an introduction about Valet Parking and the task to be solved, this section gives an overview about what we call an Autonomous Valet Parking System and explains the methods and decisions chosen to accomplish it.

In our simulated environment, the initial pose of the car is at the entrance of the parking garage. As soon as the car enters the parking garage it starts to circle in the parking lot while continuously detects the parking spaces and looks for free spaces. Once a proper free space is found, which will be called the preferred space, a path is planned to that spot, so that the car can autonomously carry the parking maneuver out. While in motion, it also scans its dynamically changing environment for potential objects, such as moving people. Therefore, our proposed system consists of three main components. First, Parking Space Detection is introduced in Chapter 2. Second, Path Planning is represented in Chapter 3. Third, Dynamic Object Detection is described in Chapter 4. Valet Parking Manager is the integrator component of our system which will be presented in detail in Chapter 5. At the end ouf our paper, future improvements will be put forward.

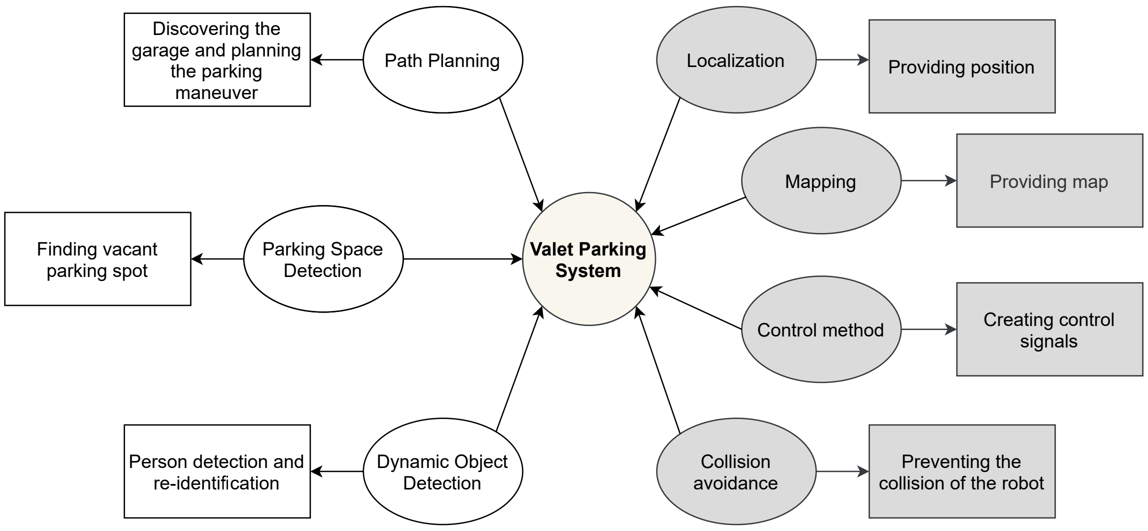


Figure 1.1: *Components required for Autonomous Valet Parking. White color indicates the parts developed by us, while gray color marks the already existing elements.*

In order to solve the proposed task, a parking garage was modeled according to the real parking lot found on the -2nd floor in Building Q of Budapest University of Technology and Economics. Underground and indoor parking lots are one of the most challenging environments for automatic parking systems due to their dim lighting, reflections on road surfaces, low contrast markings, and the presence of pillars [2]. Since almost all parking slots in underground and indoor parking lots are of a rectangular type, this paper focuses on this type of parking slot. Other types (diamonds, slanted, and parallel) are rarely located in these situations due to their disadvantages in terms of space efficiency. Underground and indoor parking lots also have much smaller spaces compared to outdoor parking lots.

As mentioned in Section 1.1, the ease of commercialization is crucial in deciding what kind of sensors to deploy. Therefore, ultrasonic sensors are most widely used as they are easy to attach to vehicles at low cost. The ultrasonic sensor based detection method has been widely used and is currently favored by the majority of car manufacturers. These kind of sensors correctly measure obstacle distances when the target surface is nearly perpendicular to the transmitter [2]. Fortunately, this is the exact case in the most underground and indoor parking lots as the parked vehicles are perpendicularly located with respect to the transmitter due to the parking slot.

However, unlike outdoor parking lots, underground and indoor parking lots include many pillars. Since a pillar has a narrow width compared to parked vehicles, the ultrasonic sensor based detection method has difficulty estimating its position due to inexact range data. In addition, it is hard to separate a pillar and a parked vehicle if they are located closely next to each other, thus deteriorating the obstacle position estimation accuracies of the ultrasonic sensor based method [2].

To overcome this issue, we employed LIDARs as well. Laser scanners have achieved remarkably accurate performance for recognizing free spaces since they produce highly accurate range data. At the same time, it must also be admitted that for the time being, they are used less frequently in real cars due to their high price. However, it is likely that the price will fall over time.

Furthermore, Section 2.1 provides ample examples that typical parking solutions use camera technology to detect parking spaces either on their own or combined with data fusion mostly with ultrasonic sensors. We decided that we leave the camera based solutions out for parking space detection so it will entirely rely on distance measurement sensors due to the fact, that the poses of parking spaces are known in advance.

The second core component of the application is the path planning. Car manufacturers have been working on designing cars more safe and autonomous. As a result of this, the majority of the latest models are equipped with drive-by-wire technology. This means, that the computers can fully control the car. They can apply break and throttle signals, and turn the steering wheel without the intervention of the driver.

A parking lot can be considered as a partially structured environment. The car should follow the lanes and when a free space is detected, it should safely maneuver itself to the place. During parking space detection, the velocity of the car should be fast enough not to block the others, also the car must not deviate too much from the ideal path. In the parking phase precision is more important, and staying inside the lane is not a priority, human drivers also cross the lane to properly execute the parking maneuver.

Designing feasible path is computationally expensive, usually a frequency of at least 10Hz must be met. Feasibility must consider the constraints of the car. Furthermore, since the car would be driving in a hybrid environment (among autonomous and human-driven cars), the path should be "human-like" so it will not surprise the participants of the traffic.

Chapter 3 describes the investigated and implemented algorithms in the project, that were considered based on the aforementioned properties.

In Chapter 4 the problem of object detection is investigated, as the third part of the application is visual-based dynamic object detection. In autonomous vehicles, it is crucial to gather as many information about the environment as possible. There are several sensors that can be used for this task, but cameras have the best potential without a doubt. In the application, an RGB-D camera is chosen for object detection which extends the wealth of possibilities even more. In the first part of the chapter, the different approaches of visual-based object detection are summarized and after that, the chosen solution is explained. Although the actual solution is used for person collision detection, the implemented structure is adaptable, with plenty of possible improvements.

As for the implementation, we employed the Robot Operating System framework which will be introduced in Section 1.3.

1.3 Robot Operating System

Robot Operating System, ROS for short, is a language-independent framework over Linux, to develop software for robots. It is a set of applications, libraries and conventions to facilitate the development of complex and robust robotic systems. The entire software is open source, freely accessible, and supported by an entire community, thus ensuring that each other's work can be used to develop further improvements. The main features of the framework are that it is peer-to-peer, device based and enables the usage of multiple programming languages. It contains example navigation algorithms, machine vision modules, unified data structures for different sensor types, robot geometric descriptions, display tool, and of course much more as well [5].

1.3.1 Package

A ROS program consists of *packages*. A package performs a task or function. When there is a need for a new task, it can be done by creating a new package. Multiple packages exist in a workspace side by side and they are compiled at the same time. In a package there are two main files, namely the `package.xml` and the `CMakeLists.txt`. The official translation system for ROS is catkin. Packages can be created in a so-called catkin workspace. This catkin workspace stores every package, each with its associated `package.xml` and `CMakeLists.txt` files. The `package.xml` file contains the description of the package, the name and contact details of the author, the license (e.g. BSD), and the package dependencies. `CMakeLists.txt` contains information for CMake translator, such as the required compiler version, what message and service files need to be generated, what nodes to create, what is the path to link directories, etc [6].

1.3.2 Node

Within a package, the base unit of the system is the *node*. The node is a separate, executable program that can be written in one of the supported languages (C++, Python, Lisp). A node can collect sensor data, run some calculations, or can be responsible for controlling a motor, for example. Nodes can communicate with each other using the ROS client library (roscpp, rospy). Client libraries also allow nodes written in different languages to communicate with each other. Nodes can publish and/or subscribe to a topic. In addition, they can provide services to other nodes. On startup, a node always announces itself to the master [7].

1.3.3 Master

The master is responsible for registering nodes and services, allowing the nodes to find each other and exchange data. The master should always start first when the system is ready to run. ROS is a distributed system therefore nodes can run on different devices within the network. In this case, we need to specify the addresses of each devices and they must register at which address the ROS master is running, as only one master can be in a given ROS system. From this time, communication between nodes takes place transparently through the network. In addition, the master also includes the parameter server, which is a centralized database where parameters related to the system can be stored [8].

1.3.4 Topic

The *topic* is a communication channel based on the principle of publish/subscribe. A node publishes data to a topic, and any other node can subscribe to that topic. Multiple nodes can subscribe to a topic, even from different devices. The subscription is implemented by specifying a so-called CallBack function in the program, which is called when a new message arrives (see Figure 1.2). There is always only one, predefined type of message which can be published to a specific topic. Topics and nodes present in the system can be displayed with the *rqt_graph* tool, which displays the relations between nodes and topics as a graph [9].

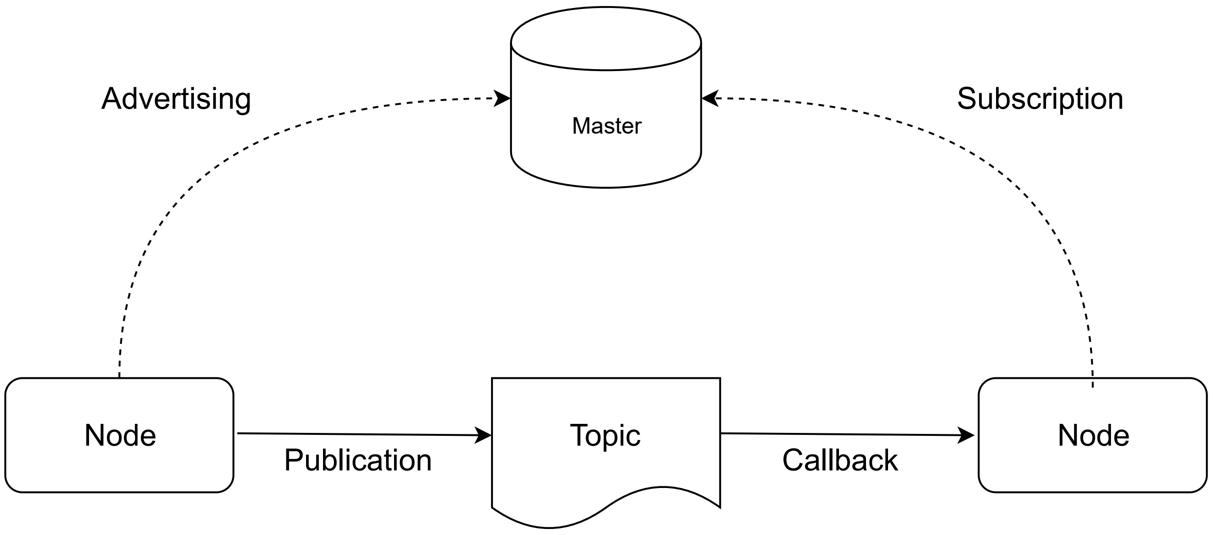


Figure 1.2: *Communication in ROS based on the principle of publish-subscribe*

1.3.5 Service

Service is another type of communication between nodes. A service enables a node to send a request, to which the service responds. In other words, a node can create services that can be used by other nodes. This can also be thought of as a remote function call, the request means the input parameters and the response corresponds to the return value. Service's input parameters and its return values must be defined in a .srv file [10].

1.3.6 Message

Communication through topics is done by sending *messages*. Publisher and subscriber nodes can communicate with each other, provided they use the same type of message. This means that a topic is clearly determined by the type of message posted on it. A message is a mixed data structure. It may contain primitive data types (bool, float, integer, string) and other message types already defined. The fields of the message must be defined in a .msg file [10].

1.3.7 RViz

RViz is a visualization program suitable for displaying information related to sensor data and robot status. By subscribing, it displays the content of various topics in 3D, provided that this is possible. Its primary purpose is displaying and debugging that are extremely useful during development process. Maps, images and ultrasonic and laserscan data are displayed for example by using RViz [11].

1.3.8 Gazebo

Gazebo is a physics engine especially designed for robot and environment simulation. It is free, open-source and widely employed among ROS users. It uses URDF (Unified Robot Description Format), which is a structured XML to describe the robot: links, joints, motors, sensors and so on. Gazebo supports a wide variety of sensors e.g., LIDARs, ultrasonic sensors, camera, odometry and IMU. It is possible to import 3D files, so one can place the robot into a proper model of the real world. A simulator helps a lot, especially during program development. Many scenarios, algorithms and ideas can be tested quickly in a safe environment, also it gives freedom to the developer to try out the program without the need of a hardware. Despite all this, one should always keep in mind that the simulator simplifies the real world [12].

Chapter 2

Parking Space Detection

Nowadays, finding a proper parking space becomes an increasingly difficult task. Most of the times, even if vacant spots are available, the drivers may not have any information about them. It is either because the free spot is too far from them or it is hidden by some other cars or objects large enough to hide the spot. In the past, and maybe at some places even now, parking spaces are managed by personal staff working in the parking lot who might not have a total view of the available parking spaces. Sometimes the driver has to check for a vacant space by circling in the parking lot, and the process for searching a free space is not only time but also energy-consuming as well as it is a wastage of fuel. Serious traffic congestion may also occur due to unavailable parking space. Automated parking space detection is a major part of an autonomous valet parking system which aims at solving the aforementioned issues.

2.1 Related work

Various methods and techniques have been proposed to overcome the problem of parking space detection in congested areas. Nyambal and Klein [1] proposed an approach for a real-time parking space classification based on Convolutional Neural Networks (CNN) using Caffe and Nvidia DiGITS framework. The training process has been done using DiGITS and the output is a caffemodel used for predictions to detect vacant and occupied parking spots. The system checks a defined area whether a parking spot (bounding boxes defined at initialization of the system) is containing a car or not (occupied or vacant). Those bounding boxes coordinates are saved from a frame of a video of the parking lot in a JSON format, to be later used by the system for sequential prediction on each parking spot. They achieved significant results regarding that their system did not get triggered when a human passed through the classification area as well as a street sign did not trigger the classifier if it partially covered a vacant parking spot.

Suhr and Jung [2] put forward a method that reliably detects and tracks vacant parking spaces in underground and indoor environments by fusing only those sensors which are already installed on mass-produced vehicles: an AVM (Around View Monitor) system, ultrasonic sensors, and in-vehicle motion sensors. The proposed method detects vacant parking spaces by combining two complementary approaches: Free space-based and parking slot marking-based. The free space-based approach finds vacant parking spaces by recognizing adjacent vehicles. This is one of the most popular approaches as it can be implemented using various range-finding sensors. However, this approach has a fundamental drawback in that it cannot find free spaces when there is no adjacent vehicle and its accuracy depends on the positions of adjacent vehicles. The parking slot marking-based approach finds parking spaces by recognizing markings on road surfaces. Unlike the free space-based approach, the performance of this approach does not depend on the existence and positions of adjacent vehicles. However, it cannot be used in cases where parking slot markings are not present or are severely damaged.

Guo *et al.* [13] advanced a vision based smart parking framework to assist the drivers in efficiently finding suitable parking slot and reserve it. A web camera is deployed to get images of the parking area and image processing techniques are used to detect the presence or absence of cars to count and locate the available parking spaces. The status of the parking lot is updated whenever a car enters or leaves the parking lot. Initially, they segmented the parking area into equal size blocks using calibration. Then, they classified each block to identify the car and intimate the driver about the status of parking either reserved or free.

Kakinami *et al.* [14] presented a system to detect the empty spaces available for parking between vehicles. To detect the parking space, this system combines information coming from an ultrasonic sensor and a 3D vision sensor. The profiles of parked vehicles are modelled by a couple of vertical planes: a longitudinal plane and a lateral plane. The empty space between profiles is the detected parking space. Longitudinal planes are computed with ultrasonic data while lateral planes are obtained from 3D vision data. As a result of their research, it was shown that 3D vision and ultrasonic sensing technologies could mutually complement each other in order to detect and localize parking spaces.

Lin *et al.* [15] designed a vehicle transverse automatic parking auxiliary system which can assist the driver to the roadside parking. Their topic was mainly concerned with the development of a vehicle transverse automatic parking assist system. The entire system included front-wheel drive, turning mechanism, transverse drive mechanism and ultrasonic distance detecting control. Ultrasonic sensors were placed at the front and rear right side of the car so that the measurement mechanism primarily determined whether an obstacle is parallel to the car and thus whether the detected space is of appropriate size. Both Kamarudin *et al.* [16] and Spitzner *et al.* [17] implemented automatic parking space detection in a very similar way to [15], primarily using ultrasonic sensors. Moreover, it is also identical in [15], [16] and [17] that the proposed algorithms were implemented in a small-scaled, car-like mobile robot.

Lee *et al.* [4] dealt with parking space detection by employing ultrasonic sensor. Using the multiple echo function, the accuracy of edge detection was increased. The single echo function means that one ultrasound pulse set is sent and just one echo is received. Then the distance is calculated by Time of Flight (ToF). The ultrasonic sensor detects the nearest object placed in its effective beam width by the single echo function and it estimates the direction of the object or assumes that the direction is an orthogonal angle. Contrarily, the multiple echo function means that after an ultrasound pulse set was sent and the first echo was received, it also waits for more echos at each „Multiple Echo Resolution Time". Then the distance is calculated by ToF of the first echo and the other echos. The significance of this method is that a surface shape can efficiently be classified whether it is an edge or a plane.

Last but not least, Anwar *et al.* [18] presented an ultrasonic sensor based autonomous car parking system. The system has the ability to self-park the vehicle with coordination between the sensors and to likewise park the car through mobile phone application remote control. To achieve the purpose of autonomous parking the system searches for an appropriate parking space, performs obstacle detection and generates PWM signals from the controller to the servo motors to achieve parking. The car parking system proposed is a compact module that can be integrated into any vehicle. In this implementation, the vehicle searches for an empty parallel parking space such that the space is at least 1.8 times the actual length of the vehicle. If the free space is less than 1.8 times the actual length of the vehicle, then it tends to find other free space.

2.2 Methodology

In the sequel, both the used and developed parts and components will be introduced which were employed in implementing Parking Space Detection. Table 2.1 gives an overview.

Component	**Function**
Map Server	Providing the map
AMCL	Localization
Minkowski Difference, GJK algorithm	Deciding whether two shapes are overlapping or not
Parking Space Detection Node	Determining the state of parking spaces

Table 2.1: *Components of the system related to Parking Space Detection*

2.2.1 Mapping

Although mapping is not the major part of our paper, we cannot leave it without a word as a map is needed to solve the task. If a mobile robot does not have any information about its surroundings, but it has to build a map and localize itself within it, then the task is often referred to as SLAM - Simultaneous Localization and Mapping. This topic is also well researched and now many off-the-shelf algorithms are available. In our project we experimented two commonly used SLAM algorithms, namely GMapping and Cartographer, both of them are capable of generating an adequate map. GMapping provides a map with lower resolution and has less sensitivity to map changes therefore it requires less computational demand. On the other hand, Cartographer is a more precise and complex SLAM algorithm, it provides various other services than just basic SLAM and can adapt to the changes in the dynamic environment faster. Its drawback is, that running the Cartographer requires almost the full computational power of a modern mid-range PC.

2.2.1.1 Map Server

Map Server is a ROS Package providing the Map Server ROS Node [19]. Its primary task is transmitting the map data as a ROS Topic. Maps managed by this tool are stored in a pair of files. The YAML file describes the map meta-data and identifies the corresponding image file. The image file encodes the occupancy data. It describes the occupancy state of each cell of the world in the color of the corresponding pixel. In the standard configuration, light gray pixels are free, black pixels are occupied, and mid-gray pixels are unknown (see Figure 2.1). Color images can be used too, but they will be converted to grayscale images.

The YAML file contains the following required fields:

- **image:** Path to the image file. It can be relative or absolute.
- **resolution:** Resolution of the map in meters/pixel.
- **origin:** The 2D pose of the bottom-left pixel in the map in the format of (x, y, yaw). Yaw is interpreted as a counterclockwise rotation.
- **occupied_thresh:** The pixels which occupancy probability is greater than this threshold value will be considered completely occupied.

- **free_thresh:** Pixels with occupancy probability less than this threshold value are considered completely free.
- **negate:** Whether the white/black, free/occupied semantics should be reversed. The interpretation of the threshold values remain unaffected.

Besides from providing an already existing map, Map Server also enables to save a currently generated map for later use. This will produce the above-mentioned image and YAML files. A more detailed description (for example about the value interpretation of Map Server) can be found in [19].

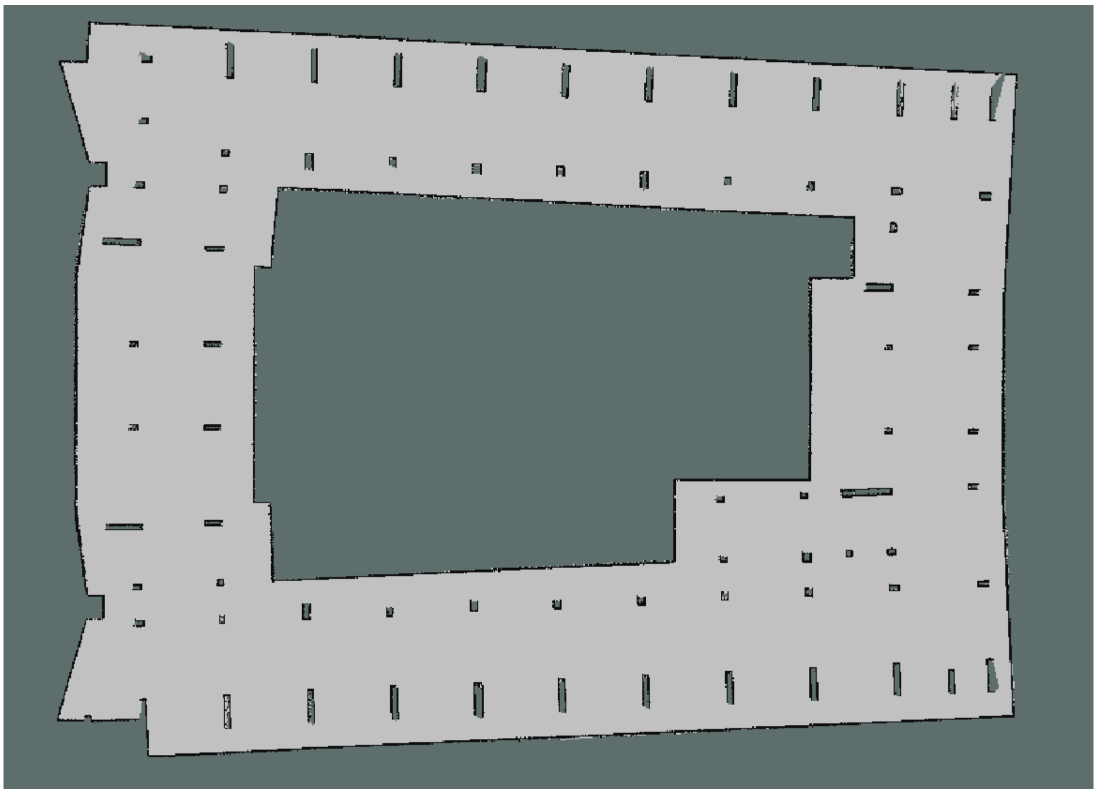
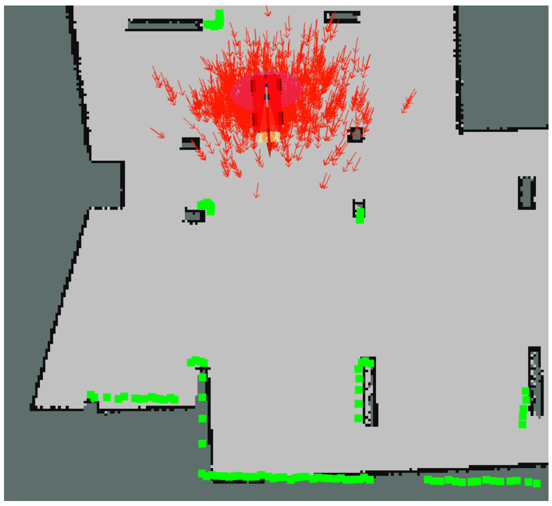


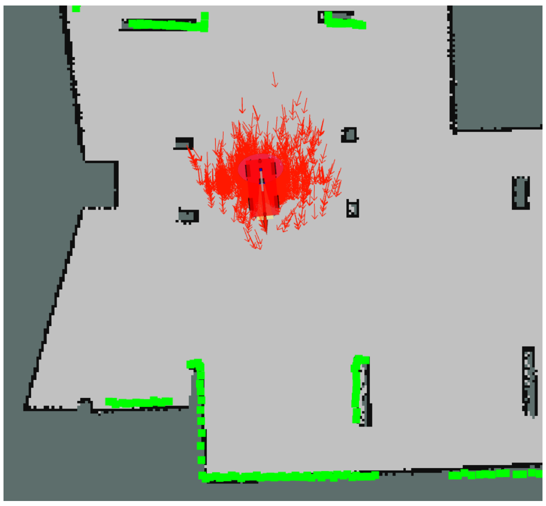
Figure 2.1: *The map provided by Map Server*

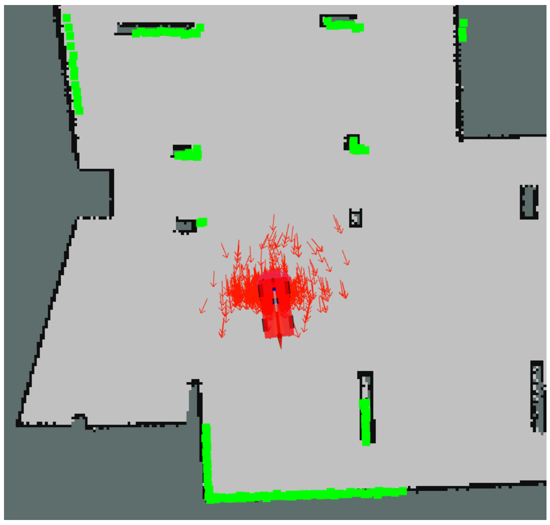
2.2.2 Localization

In order to find the pose of the car on the map, AMCL was employed. AMCL is a probabilistic localization system for a robot moving in 2D [20] and it is one of the most common methods among today's robotic localization solutions [21]. It implements the Adaptive Monte Carlo Localization approach, hence its name. As a prerequisite, it needs a known map in which the robot can be placed. This is fulfilled by the Map Server presented in Section 2.2.1.1.

During its operation, AMCL generates hypotheses about the pose of the car and puts arrows indicating these on the map. While the car is in motion, these hypotheses are filtered out by a so-called particle filter, thus the arrows are gradually grouped around points that the algorithm considers potential positions of the car. Finally, the arrows are arranged in a group which is estimated as the real position of the car and so the actual localization is realized.


(a) Initial state


(b) Intermediate state (1)


(c) Intermediate state (2)

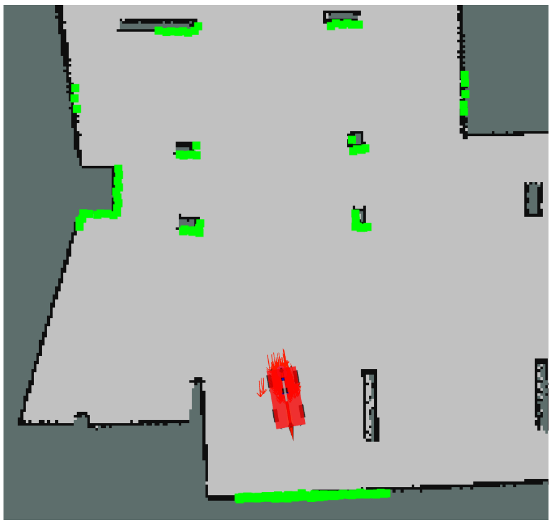

(d) Final state

Figure 2.2: *AMCL in operation. As the car moves, the hypotheses (tiny red arrows) will be grouped around the real pose of the car. The green dots represent the merged front and rear LIDAR scans. It can also be noticed that as AMCL becomes more confident with its pose estimation, the green dots become congruent with the map.*

ROS provides the AMCL package which works with laser scan data, takes in a laser-based, known map, transforms messages and outputs pose estimates. On startup, AMCL initializes its particle filter according to the parameters provided. Amongst the AMCL parameters, it can be given an estimation of the car's initial pose. The initial covariances can also be set, i.e., how much the initial hypotheses depend on each other around the starting point. This affects in how much space and in what direction the algorithm will place the arrows. This can help in some cases, but if the car starts from another position, it is not worth relying on this, as this may cause the algorithm to find a wrong position that counts for the best result locally but not globally.

If the car is positioned incorrectly on the map, we have an option to use a service from AMCL called *global_localization*. This service initiates global localization, wherein all particles are dispersed randomly through the free space in the map and it gradually rearranges them around the actual position of the car.

However, if we do not know that the car is in the wrong position, this solution is not appropriate. AMCL also provides another option, which is turned off by default. The solution is if the algorithm constantly adds some random hypotheses to the existing ones, assuming that the car is in the wrong place, thus giving a chance to find the real position.

Last but not least, Figure 2.3 shows the differences between localization using odometry and AMCL. Odometry is the determination of the amount of displacement along the path of the robot. Knowing the angular rotation of the wheels and the orientation of the steering, the pose of the robot at the next time can be calculated. A major advantage of odometry is that given the aforementioned data, it is always able to estimate the new pose of the robot. The disadvantage is that due to its integral nature, the pose calculation error can increase beyond all limits and keeping the error within a given limit requires pose verification using a periodically independent reference.

In ROS, base frame is attached to the robot, in odom frame the relative displacement of the robot is calculated using odometry and global frame usually means the map frame. The transition between these coordinate frames takes place with the help of coordinate transformations. During operation, AMCL estimates the transformation of the base frame (*/base_frame*) in respect to the global frame (*/map_frame*) but it only publishes the transform between the global frame (*/map_frame*) and the odometry frame (*/odom_frame*). Essentially, modifying this transform accounts for eliminating the drift that occurs using Dead Reckoning [20].

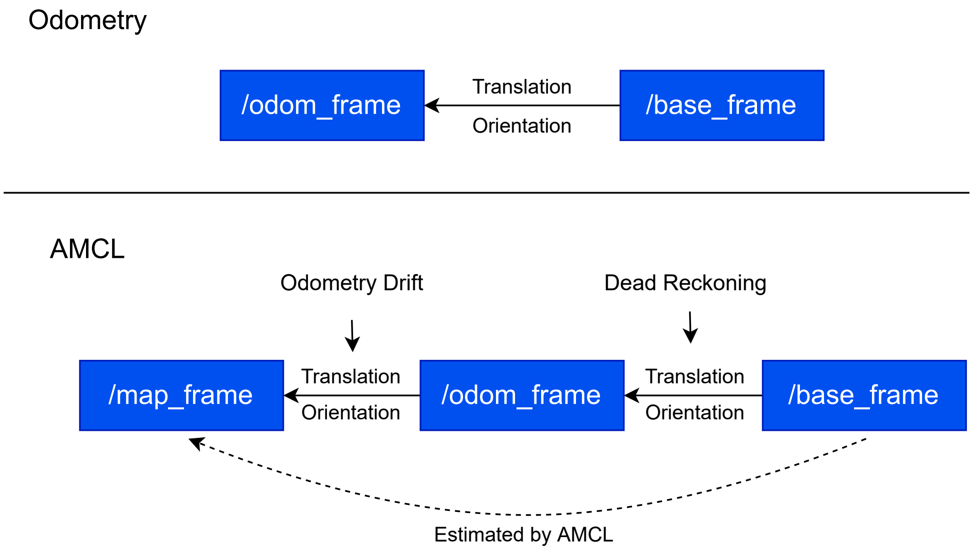


Figure 2.3: *Differences between odometry and AMCL* [20]

More information about the topics, services and parameters of the AMCL package provided by ROS is available in [20] while about the theory of localization and the algorithm of AMCL is found in [21].

2.2.3 Minkowski Difference

The implemented parking space detection algorithm presented in Section 2.2.5 incorporates frequent polygon overlapping checks based on the GJK algorithm, which relies on the concept of Minkowski Difference. The computation of Minkowski Sum and Minkowski Difference is crucial for many applications, such as robot motion planning, morphological image analysis and computer-aided design and manufacturing [22].

Let us assume that there are two shapes. The Minkowski Sum of those shapes is all the points in $ShapeA$ added to all the points in $ShapeB$:

$$A \oplus B = \{a + b | a \in A, b \in B\} \tag{2.1}$$

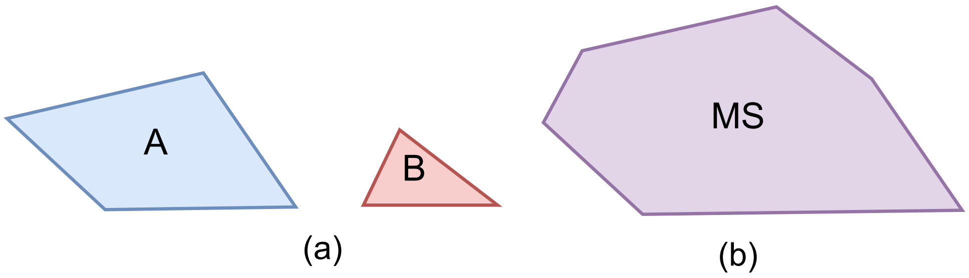


Figure 2.4: *(a) Two polygons A and B (b) The Minkowski Sum of Polygon A and B* [22]

The Minkowski Difference of those shapes is all the points in $ShapeB$ substracted from all the points in $ShapeA$:

$$A \ominus B = \{a - b | a \in A, b \in B\} \tag{2.2}$$

The Minkowski Difference is most commonly employed in obstacle detection algorithms of mobile robots due to its following, defining factor: if two convex shapes are overlapping (intersecting) the Minkowski Difference will contain the origin [23].

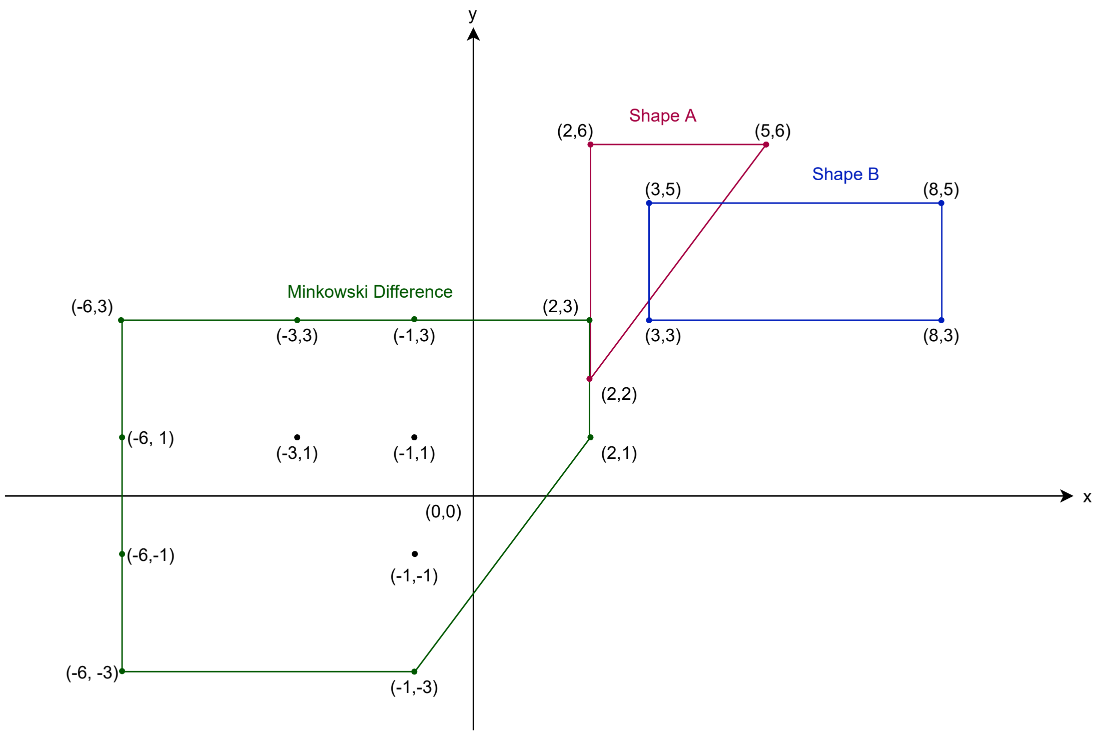


Figure 2.5: *A concrete example of Minkowski Difference*

Figure 2.5 demonstrates an example where the Minkowski Difference is performed on $ShapeA$ and $ShapeB$. These two convex shapes are intersecting each other, therefore the resulting shape (the Minkowski Difference) contains the origin.

2.2.4 GJK Algorithm

The example in Figure 2.5 highlights a major disadvantage of the Minkowski Difference. Performing the operation requires the number of vertices of $ShapeA$ multiplied by the number of vertices of $ShapeB$ multiplied by 2[1] substractions. Both shapes are convex and defined by the outermost vertices, therefore the Minkowski Difference must be only performed on the vertices of the shapes. However, regarding arbitrary polygons, its computational cost can still be significant due to the possible large number of vertices. This is where the GJK algorithm facilitates the operation.

The Gilbert-Johnson-Keerthi distance algorithm or simply GJK algorithm is an iterative method which converges quickly and can run in near constant time. A major prerequisite, that GJK can only operate on convex shapes since it is based on the Minkowski Difference. GJK's original intent was to determine the distance between two convex shapes. GJK can also be used to return collision information and can be supplemented by other algorithms as well [24].

A substantial benefit of the GJK algorithm is that there is no need to calculate the Minkowski Difference. Instead, a polygon can be built inside the Minkowski Difference that attempts to enclose the origin. This polygon is called the Simplex. If the Simplex encompasses the origin then the Minkowski Difference also contains the origin [24]. The Simplex is built by using a so-called Support Function. The task of the support function is to return a point inside the Minkowski Difference given two shapes. The easiest method would be to take a point from $ShapeA$ and from $ShapeB$ and substract them to obtain a point in the Minkowski Difference. However, it is not adequate if the support function provides just a random point. Instead, it is expedient to make the support function dependent on a direction and it is worth choosing the farthest point in a given direction. Choosing the farthest point has significance because it creates a simplex which contains a maximum area therefore increases the chance that the algorithm exits quickly. In addition, the fact can be used that all the points returned this way are on the edge of the Minkowski Difference and therefore if a point cannot be added past the origin along some direction then it is certain that the Minkowski Difference does not contain the origin. This increases the chance of the algorithm to exit quickly in non-intersection cases.

Figure 2.6 demonstrates an example of building a simplex, using $ShapeA$ and $ShapeB$ from Figure 2.5 and performing the support function three times. First with the direction of $(-1,-1)$ which results in point $(-6,-3)$, second with the direction of $(1,-1)$ wich results in point $(-1,-3)$ and last with the direction of $(-1,1)$ which results in point $(-6,3)$. Incidentally, it is also noticeable that if the direction of $(-1,0)$ were used then all the points of $(-6,3),(-6,1),(-6,-1)$ and $(-6,-3)$ would be sufficient. Looking at Figure 2.6 it can be clearly seen that the Simplex does not encompass the origin but it is known from Figure 2.5 that the two shapes are overlapped. The problem here is that even if the farthest point is chosen in a given direction but randomly choosing a direction did not yield a Simplex that contained the origin. If instead the direction of $(1,1)$ were to be chosen for the third Minkowski Difference point that yields a Simplex shown in Figure 2.7 and now it contains the origin so it can be determined that there is overlapping.

As it is clearly seen, the choice of direction can affect the outcome. It can also be noticed that if we obtain a Simplex that does not contain the origin another point can be calculated and used it instead. This is where the iterative part of the algorithm comes in. The GJK algorithm cannot guarantee that the first 3 points chosen are going to contain the origin nor can it guarantee

[1]Although in terms of "vector substractions" this 2 multiplier here is superfluous, but computer calculations are performed on scalars, and 2D points have x and y coordinates which must be substracted from each other.

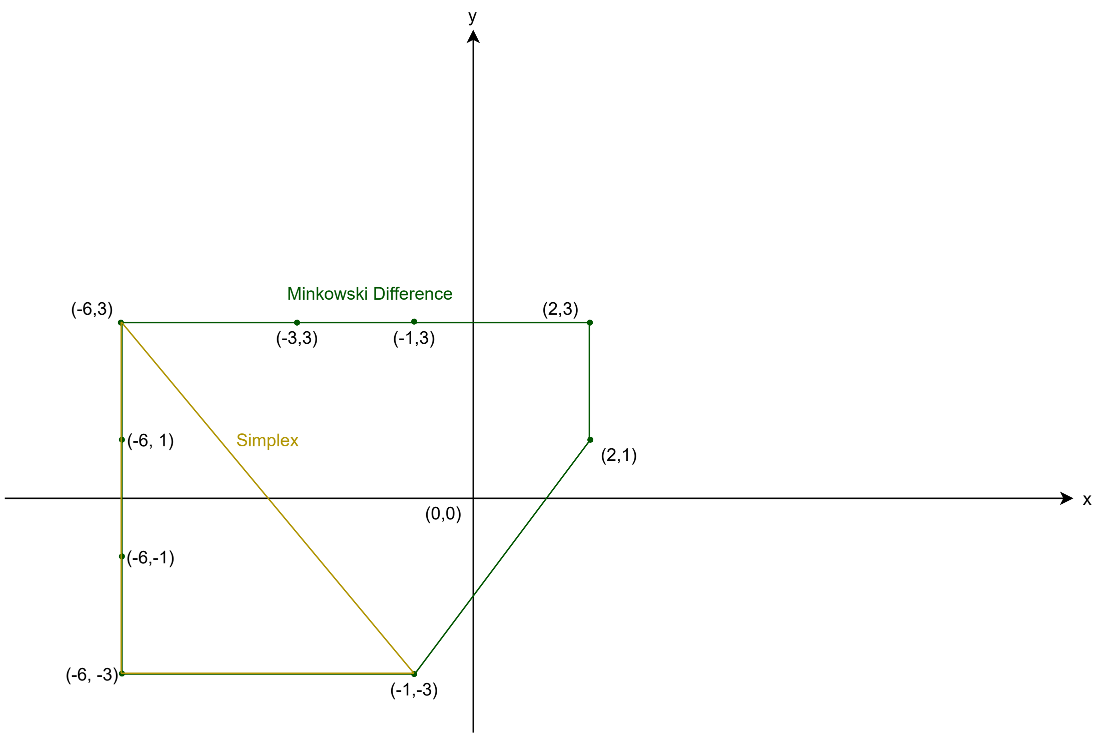


Figure 2.6: *Creating a Simplex*

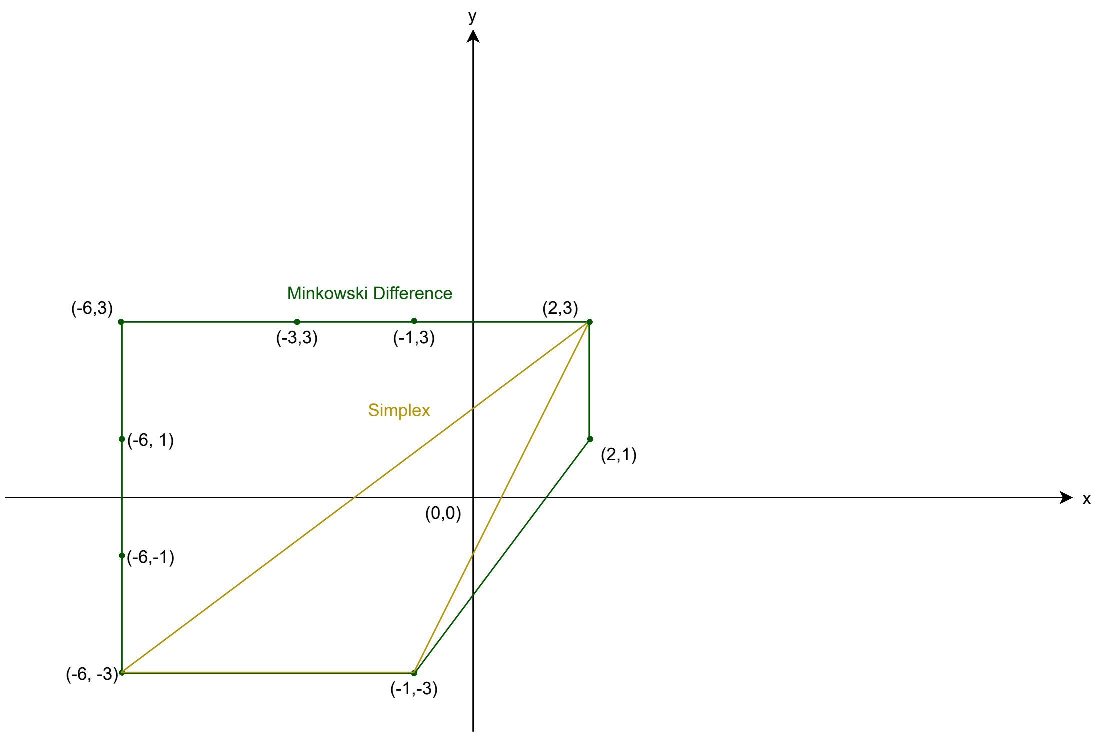


Figure 2.7: *Creating another Simplex*

that the Minkowski Difference contains the origin. However, it can modify how the points are chosen by only choosing points in the direction of the origin.

Basically, only the first direction for the first Minkowski Difference point needs to be chosen, because it is expedient to choose the second direction as the opposite of the first so that the initial Simplex has a maximum area. After that the direction should always point towards the origin. There are numerous options for choosing the first direction, for example an arbitrary direction, the direction created by the difference of the centers of the shapes, etc. Any will work, in our implementation it is the first point of $ShapeA$ pointing towards the first point of $ShapeB$ in a given data structure. After that the algorithm iteratively performs and checks the following:

1. Does the current Simplex contain the origin? If yes, there is overlapping and the GJK algorithm terminates.

2. Is the Simplex able to get closer to the origin so that it encompasses it? If no, there is no overlapping and the algorithm terminates.

More information about the implementation part and the theory of GJK algorithm can be found in [25, 24], respectively.

2.2.5 Parking Space Detection Algorithm

By now we have a map provided by the Map Server presented in Section 2.2.1.1. The car is localized on this map by employing AMCL described in detail in Section 2.2.2. This Section focuses on the implementation and demonstrates how the Parking Space Detection algorithm is realized.

The modeled parking garage was simulated according to the real parking lot found on the -2nd floor in Building Q of Budapest University of Technology and Economics. The generated map of this parking lot can be seen in Figure 2.1. The parking spaces are stored in a YAML file based on the authentic blueprint of the real parking garage. This is significant, because in our implementation one YAML file contains one layout of parking spaces. If the map changes for some reason, the parking spaces stored in the YAML file must also be redefined or modified. The following values are stored for every parking space:

- **id:** An identification number that uniquely determines the parking space.

- **center:**

  - **position:** 2D coordinates (x, y) of the center of the parking space.
  - **orientation:** Orientation of the center of the parking space in quaternions (x, y, z, w). We decided to use quaternion because it is a built-in, default type in ROS which facilitated the implementation process.

- **width:** Width of the parking space.

- **length:** Length of the parking space.

A ROS node was programmed in C++ language called Parking Space Detection Node. On startup, it reads the YAML file and interprets its content. If one of the data is corrupted (for example there is an invalid quaternion value), it warns the user and aborts the program. In this way it is ensured that the parking space detection only occurs when the parking spaces are properly defined. Furthermore, the node also supplements its internal data structure with a field called *free*. Initially, all the values of *free* are set to *false* which means that every parking space is assumed as being occupied. This is a security precaution since it conveys less danger if a vacant parking space is recorded as occupied as its opposite. Moreover, that is the reason why *free* is not stored in the YAML file, e.g. to avoid unnecessary redundancy. As a matter of course, when the car moves in the parking garage it will explore its surrounding environment and update the states of the parking spaces with the help of ultrasonic sensors and LIDARs. The following steps are performed at each examination:

Step 0: Identifying nearby parking spaces in order to reduce the computational demand of the further steps: a simple distance test between the center of the parking spaces and the center of the rear axis of the car is executed. Further steps only apply to nearby parking spaces.

Step 1: Dividing each parking spaces into $2 \cdot N$ equal sized cells and extending them by one cell in the direction of the road, so that all in all there are $2 \cdot (N + 1)$ sections. The value of N is calculated as follows:

$$N = \text{Rounding}\left(\frac{length}{width/2}\right) \tag{2.3}$$

The extension is virtual, only needed for the detector algorithm, displaying the parking spaces uses the normal size. Then, it is examined whether the 2D projection of the possible maximum ultrasonic cone (which is a triangle) overlaps with all cells. This is important, because it ensures that at least one ultrasonic sensor can have a maximum coverage of the parking space. Determining the intersection of the 2D projection of the possible maximum ultrasonic cone as a triangle and each equal sized cells is fulfilled by the GJK algorithm presented in Section 2.2.4. If there is an intersection, Step 2 follows. If there is no intersection, then the state of the parking space is unchanged. It is to be noted that the enlargement by one cell towards the road is advantageous so that if a car sticks out of a parking space, the detector algorithm still has the chance to identify it correctly.

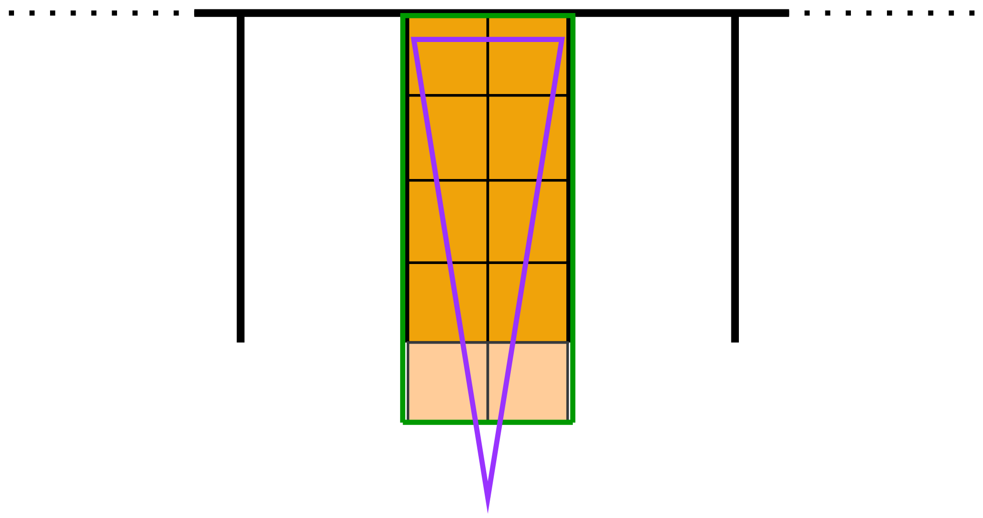


Figure 2.8: *Dividing a parking space into equal sized cells (orange). The added cells are drawn in a lighter color. The green rectangle represents the enlarged parking space while the purple triangle is the 2D projection of the ultrasonic cone.*

Step 2: It is investigated whether the enlarged parking space as a rectangle intersects with the 2D projection of the currently measured ultrasonic cone as a triangle. To decide this the GJK algorithm is called. If there is no intersection, the state of the parking space remains unaffected. This is relevant if for some reason the sensor cannot see the parking space, even though it would have the opportunity. This may be because another car is parked in front of the parking space or a pillar hides it from the sensor's field of view. Due to the presence of pillars, underground and indoor parking garages are one of the most challenging environments for automatic parking space detection systems [2]. But if there is intersection, Step 3 proceeds.

Step 3: The Parking Space Detection node receives a pointcloud which contains the merged points of the ultrasonic sensors and the LIDARs. If the enlarged parking space contains any of these points then its state is set to occupied otherwise it is set to vacant. A simple distance measurement test diminishes the computational demand. Only those points are checked which are closer to the center of the parking space than half the diagonal of the parking space. So this *critical distance* in this case is the radius of the circle enclosing the rectangle.

Figure 2.9 demonstrates an example where all the parking spaces are vacant. Parking spaces are displayed as rectangles where color green indicates vacant while red occupied parking spaces. Moreover, the green dots are the merged front and rear LIDAR scans, white dots are the merged ultrasonic and LIDARs pointcloud and the purple cones are the ultrasonic sensor measurements. When ultrasonic sensors measure above 99% of their maximum range, the measured range values are not added to the merged pointcloud, because it cannot be determined whether there is an object or the sensor has simply reached its maximum measurement limit.

The car starts from the bottom left corner and as mentioned earlier, initially all the parking spaces are regarded as occupied. As the car comes in motion, it runs the parking space detection algorithm and updates the state of the parking spaces if necessary. Figure 2.9 illustrates two more things as well. First, parking spaces 5 and 82 are not yet detected as vacant because at this current moment, there is no ultrasonic cone which overlaps with every equal sized cell, therefore their previous states remain (Step 1 from earlier). Second, in order to avoid false recognitions resulting from the uncertainty of AMCL, where the algorithm detects the points of the wall as occupied parking spaces (otherwise this could happen, for example in case of parking spaces 4 and 81) therefore the length and width values of each parking spaces were reduced by 10%.

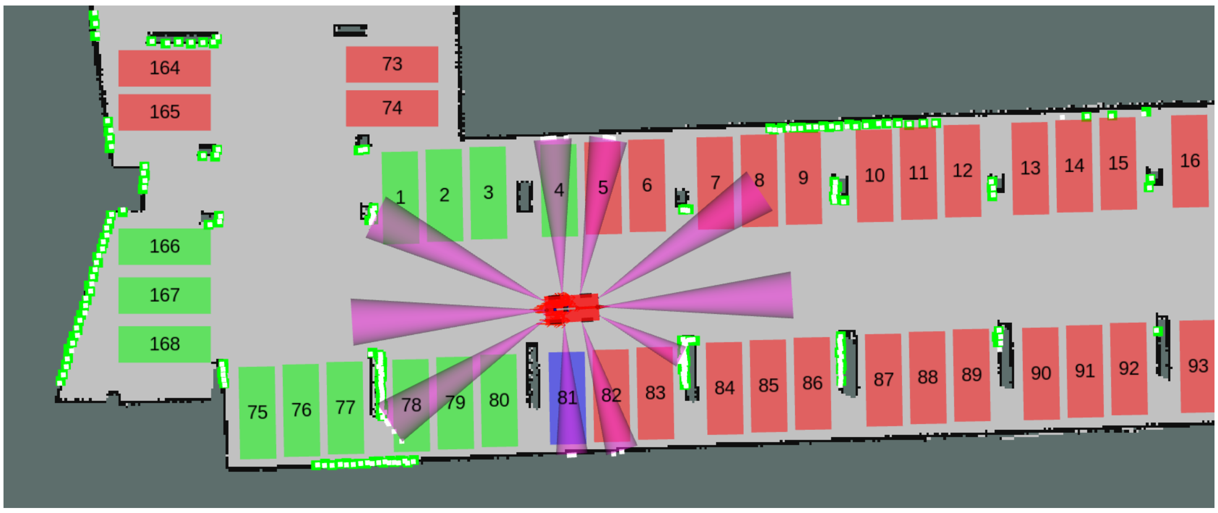


Figure 2.9: *Detection of parking spaces*

The typical runtime of the parking space detection algorithm is approximately 4-6 milliseconds which can be considered remarkably robust, especially knowing that there are 170 parking spaces and 10 ultrasonic sensors which are to be examined in each iteration.

Last but not least, the Parking Space Detection node has a ROS Service through which the setting of a preferred parking space can be switched on and off. If it is turned on, the node determines the close parking spaces (similar to Step 0 from earlier) from which it selects a free parking space which is closest to the center of the car's rear axis. In Figure 2.9 the preferred parking space is blue. If amongst the close parking spaces there is no vacant spot, then there is no preferred parking space and the car keeps in motion waiting for a close and free space which will be the preferred.

Chapter 3

Path planning

Path planning is a mandatory process for autonomous mobile robots, that finds a path between two configurations. The robot is often surrounded by obstacles, that can be standing or moving, and it must be aware of. Finding a collision-free, safe and optimal path in such a scenario is a difficult problem. This makes the topic actively researched in recent decades.

Optimal paths could vary depending on the application. Algorithms often optimize for length or travel time, but there are more special approaches also, for example minimizing the total amount of turning. Sometimes, not only the environment of the robot causes constraints but the robot itself.

A system is holonomic when the number of controllable degrees of freedom is equal to the total degrees of freedom. So a holonomic robot can drive straight to a goal that is not in-line with its orientation. These robots do not have intrinsic kinematic constraints, so finding a feasible path is much easier. Robots built on castor wheels or omnidirectional wheels is a good example of holonomic drive.

A non-holonomic robot has less controllable degrees of freedom than the total so it can not follow any geometric path. It can not drive straight to a goal that is not in-line with its orientation, meaning the robot must either rotate to the desired orientation before moving or rotate as it moves. Creating a path for such a robot should be done carefully so that the robot's constraints are not violated and it can precisely follow the desired trajectory. Car-like vehicles are example of non-holonomic robots, since the controllable degrees of freedom is two (acceleration/breaking and turning angle of steering wheel), but the total number of its freedom is three (position and orientation).

Path planning is a broad and complex field, the exact needs vary on applications, therefore no best solution exists. A common practice is separating the process into two steps, a global planning which uses a priori information of the environment, and a local planning phase where the constraints of the robot and the dynamic obstacles are taken into consideration. There are numerous algorithms for both used in recent applications and some of them are also addressed in this document.

3.1 Global planners

Global path planning is a major component in the navigation process. It contains of finding a global path between two robot configurations in a cluttered environment. It is also a well-studied research area, since it is explicitly used in many fields outside robot motion planning, like: network optimization [26], video games [27] or biology [28]. The core ideas are often connected to graph-search algorithms which is also a well and actively researched field. Most of the commonly used global planning algorithms for a single mobile robot can be categorized into one of these groups:

- Graph search algorithms: Graph is created from the map, as the following: nodes are free spaces in the map and edges represent the cost of getting there (usually the length of that path). There are some widely used algorithms that can find the shortest path in this graph. These algorithms are usually complete, meaning that it will terminate with a solution if it exists.
  - Dijkstra's algorithm [29]: Starting from the initial node and marks all the neighboring nodes with the cost to get there. If a node has no more edges to check, it chooses the least costly node, and calculates the cost of its adjacent nodes. If multiple path exists to the same vertex, the cheaper one will point to it. Once the goal is reached, the algorithm terminates and the shortest path is presented with the pointing edges. It always finds the shortest path.
  - A* algorithm [30]: This method is very similar to Dijkstra's one, but it is addressing its weaknesses. A general problem with the former one is that it discovers the graph in all directions. A* introduces a heuristic function which helps to priorities selecting nodes in the direction of the goal. This algorithm is being described in more depth later on.
  - D* algorithm [31]: D* is also an improvement of the A* method. Generally the former algorithm is quite fast and finds optimal solution, but when an obstacle appears in the path of the robot, re-planning is very expensive. To overcome this issue, D* starts planning from the goal and has the ability to change the cost of the path. This enables keeping the previously calculated path outside the obstacle, so can find a new path quicker.
- Sampling-based algorithms: Instead of considering the whole state space, it is reduced by sampling. Usually a graph or tree is created and formerly mentioned or commonly used graph algorithms are used to find solution. In practice, most of these algorithms are only resolution complete, meaning that the outcome depends on the used resolution. It can happen that some solution is missed if the state-space is not discretized with enough precision or the maximum time elapsed.
  - Rapidly-exploring random trees: an algorithm designed to randomly select points from the search space. The selected point is connected to the existing graph. One (or more) trees are grown from the samples and when there is a path between the start and goal configuration a basic graph search is used. Many types of RRT algorithms have been developed, which usually vary in point selection, tree connection, node number or uses heuristic functions.

  - Rotate-Translate-Rotate (RTR): similarly to RRT a tree is grown but the sampling, node selection and extension steps are different. This provides good and fast solution in congested spaces. Because of its favorable computational time, this solution was chosen and later will be described in more depth.

- Intelligent, heuristic based methods: In the last few decades, there has been a rapidly increasing interest towards more heuristic methods. Some of them are inspired by biological systems e.g.: swarm optimization [32] and genetic algorithms [33], [34]. While other algorithms were introduced with the advancement of the neural networks [35]. In general, these algorithms usually provide optimal solution, but their computational demand is much higher than the previous methods and is rising with a great magnitude as the map resolution is increasing

3.1.1 RTR planner

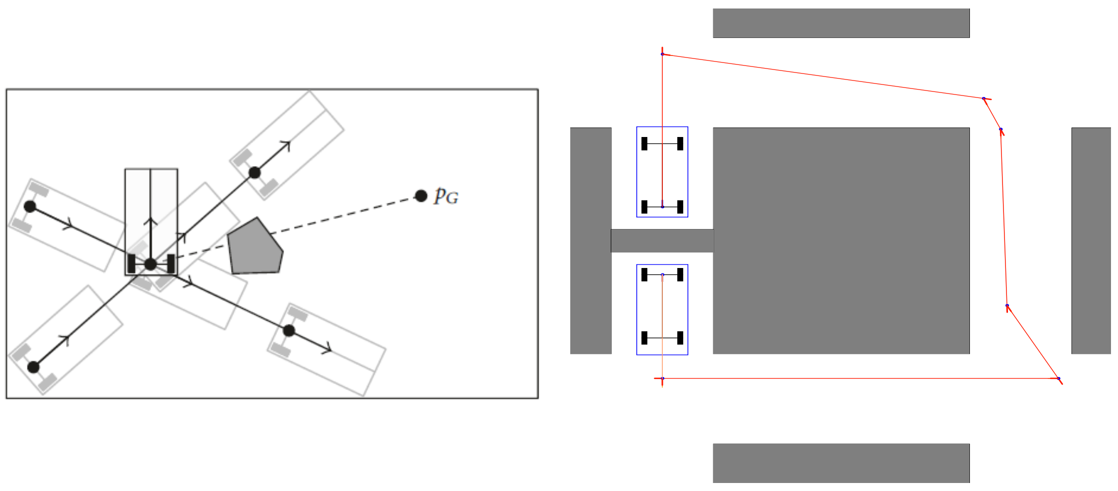
As mentioned earlier briefly, RTR, is short for Rotate-Translate-Rotate, is a planner algorithm inspired by the Rapidly Exploring Random Tree. The detailed walk-through of the algorithm can be found in this paper [36], here a brief overview is presented.

The algorithm builds a topological tree in the free space configuration. It consist of 3 steps: sampling, node selection and tree extension. It is mainly designed for differential drive robots, and is proved to provide quick solution with high success rate in narrow environment.

The first step is sampling, and is different from the one commonly used in RRT algorithms. A random position named guiding position, p_G is selected on the free plane. The algorithm only considers the position, without orientation.

Next step is vertex selection, where the nearest configuration $q_{nearest}$ in the tree is determined. This is realized with the basic metric function, Euclidean distance, hence no special metric is needed.

The last step is the tree extension and this is the main difference to the RRT method. From the selected $q_{nearest}$ configuration, rotation is applied to achieve the desired orientation that points toward p_G. If it successfully performed without collision, then translation is applied to both forward and backward until the first collision. In narrow space the robot often collides even in the rotational step, but RTR-planner performs the transnational extension just at the colliding orientation. Furthermore, the rotation is tried again in the other turning direction as well, and tree extension is applied on collision (or on successful alignment with p_G). Figure 3.1a shows this step in case of the rotational step is blocked. This extension strategy results in a more aggressive space exploration than the basic RRT.



(a) Collision occurs in the rotational step [36] (b) Path planned with RTR

Figure 3.1: *RTR planner examples*

Similarly to bi-directional RRT, the RTR builds two trees, one from the initial the other from the goal configurations. Connecting the trees is attempted in every iteration. The newly added transnational element is checked for intersection with the other tree. If one is found and the necessary rotation at the intersection is collision-free, then the path exists in the merged tree.

3.1.2 A* planner

According to the related works [37], an improved version of the A* can be used in structured environment, similar where the robot will operate. Therefore, for first the basic A* planner is introduced in this section. This particular planner aims to resolve the main issue of Dijkstra's algorithm. Although the path found by the latter one is optimal in length, it's drawback is the required resources. Dijkstra starts exploring the nodes based on their distance from the starting configuration in ascending order, meaning many nodes are discovered even though they are not in the direction of the goal. It can be seen easily, that it gets computationally expensive as the world is expanding.

A* introduces a heuristic function $h(n)$, that underestimates the cost of reaching the goal from each node. Such function must meet the following criteria:

- $h(goal) = 0$
- For any two adjacent nodes x and y:
  - $h(x) \leq h(y) + d(x, y)$, where $d(x, y)$ denotes the length of the edge

First one states, that if the goal is achieved, the heuristic function must provide 0. The second statement means, the function must return less than or equal the real distance between the goal and node x. Two of the most commonly used of expressions in two dimensional problems are Euclidean Distance and Manhattan Distance.

If the sets of criteria is met, the heuristic function is called admissible and is guaranteed to return a least-cost path. Lets denote any node as n and the goal with g, the form of the distances:

1. Euclidean distance:

$$h(x_n, y_n) = \sqrt{(x_n - x_g)^2 + (y_n - y_g)^2} \tag{3.1}$$

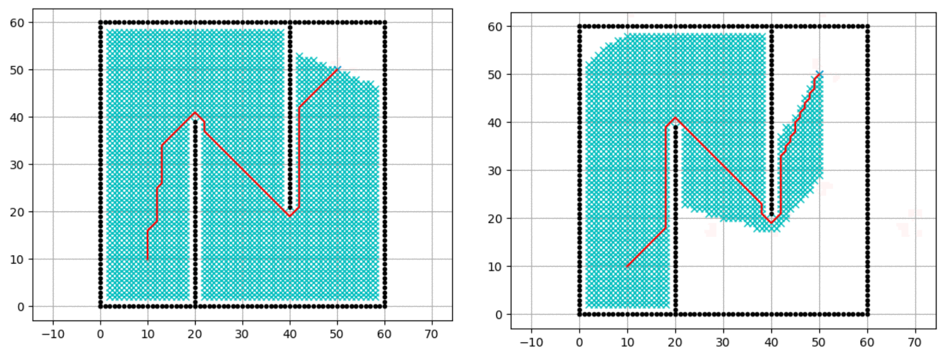
2. Manhattan distance [38]:

$$h(x_n, y_n) = |x_n - x_g| + |x_n - y_g| \tag{3.2}$$

The overall cost of a node $f(n)$ is the sum of the path length from start point to the node $g(n)$ and the heuristic value of the function:

$$f(n) = g(n) + h(n) \tag{3.3}$$

It terminates when the goal is selected (meaning there is a path in a graph) or there are no more nodes to select. Selection is usually done from a priority queue, where the adjacent nodes of the already visited ones are listed in a descending order of function f. The first one, with the least cost is chosen and the value f and g of its neighbors are updated and added to the priority queue. Each node keeps track of its predecessor, so when the algorithm is stopped at the goal, the shortest path can be determined.



(a) *Dijkstra's algorithm* (b) *A star algorithm*

Figure 3.2: *Comparison of optimal planners* [39]

Figure 3.2 shows a comparison between Dijkstra's algorithm and A*. Although the shape of the found path is different, they are equal in length. The blue crosses marks the discovered area and shows the effectiveness of the A* planner over the Dijkstra algorithm.

3.1.3 Hybrid A* planner

The largest disadvantages of the previous approaches is that the paths are sampled in discrete space (grid map) thus the resulting paths are also discrete. This implies errors in the motion execution part (when following the line), usually it is smoothed with approximation or in the local planning phase, but that requires more computation and admissibility might gets violated.

In Hybrid A* search, nodes can be in any continuous point on the grid, considering the non-holonomic constraints. The state space usually has three dimensions, containing the x, y

position and θ orientation. The next, reachable states are determined using precomputed primitives that are known to be followable for the car. At each state, these motion primitives are placed, and its end positions are saved as adjacent, reachable nodes to the open set (priority queue).

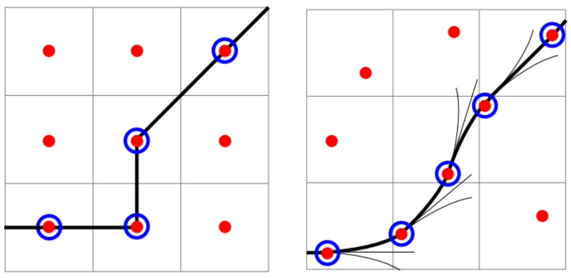

(a) Regular A* (b) Hybrid A*

Figure 3.3: *Comparison of A* and Hybrid A* planners* [40]

Furthermore, a cost function is still used. One that computes how expensive is it to reach the adjacent position (which is still known from the primitives) and a heuristic function to estimate the cost of reaching the goal. The sum of these function is the total cost, which determines its position in the priority queue. Every node keeps track of its parent which is later used for the final path reconstruction.

The algorithm got more recognition after it was successfully used in the DARPA Urban Challenge in 2007 [40]. Many papers were published afterwards about its operation and results [37, 41, 42].

3.1.3.1 Hybrid A* planner in structured environment

In many scenarios, it is useful be able to influence the final trajectory of the path by assigning waypoints. Suppose the robot should follow lanes, turn or go across junctions, it is highly preferred to stay in the lane as much as possible, or perform the turn without high deviation from the ideal curve. This is a crucial step for real-world applications. In our application, the parking garage also has lanes, that the car should be able to follow, so this algorithm was chosen as one of the planners.

A typical representation of such waypoints is a directed graph, that captures the topological structure of the environment. This either requires prior knowledge and created manually, or it might be generated automatically. In this project the former solution was used. In offline mode, we manually selected the lane points that are feasible and preferred. This graph is then loaded together with the map, and is taken into consideration during global planning.

For the structured planner, the heuristic function is extended and new distance metric is introduced. Given this lane network, represented as a graph: $G = \langle V, E \rangle$, where V is a set of vertexes and E is a set of edges, with α_E denoting the angle of an edge. Euclidean distance can be computed from a vehicle configuration $q = (x, y, \theta)$ to the graph:

$$D(G, q) = \arg\min_i \langle E_i : |\alpha_E - \theta| < \alpha_{min} \rangle D(E_m, q) \tag{3.4}$$

where $D(E_m, q)$ is the Euclidean distance between the edge that is determined by the $\arg\min$ expression and the point of the robot configuration. The $\min$ expression means, we first look for the edge that has almost the same heading as the robot. This is especially helpful to avoid following the oncoming lane, and staying on the preferred path in turns. Also, for a human user point of view, to further increase the traveling experience it is favorable to reduce direction change and reverse traveling as much as it is possible. This soft-constraint is also added to the cost, as a penalty, so the algorithm tends to chose paths that are more "human-friendly".

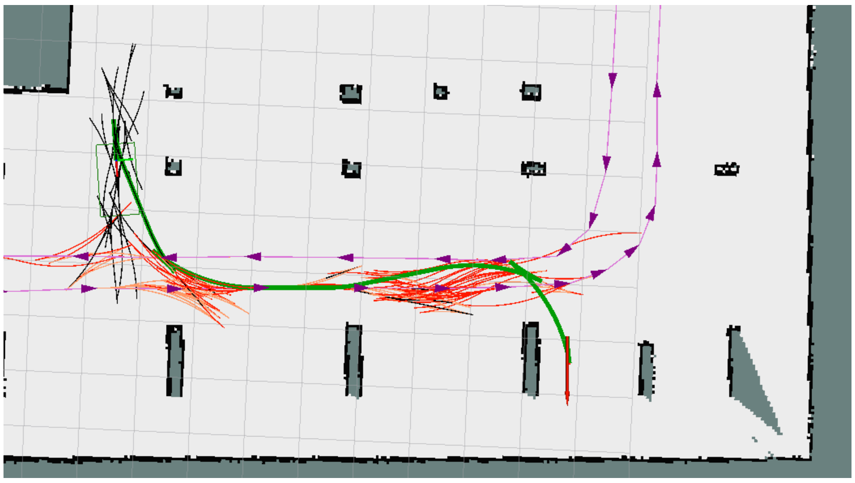


Figure 3.4: *Path primitive of the Hybrid A* in structured environment*

Figure 3.4 shows the progress of the Hybrid A* planner in structured environment. On the left side of the image, the green rectangle shows the initial pose of the car. On the right bottom side the red vector indicates the goal. The green curve is the final trajectory. Also, black and red motion primitives can be seen, which shows the progress of the algorithms.

3.2 Local planners

A local planner is responsible for generating feasible path for the robot with respect to its intrinsic constraints in the absence of obstacles. A global planner e.g. from the previous section, provides a collision-free path which usually consists of straight segments (in case of graph based planners). In case of non-holonomic robots the local planner has the task of connecting these configurations, provided by the global planner, with smooth and followable trajectory. Properties of the path are dependent on the type of the used local planner.

After a feasible path is generated, it is checked against collision, and if it fails then new iteration is started.

Local planners are strongly coupled with steering methods (sometimes they refer the same) and used in decomposition-based path planning. The exploration of collision-free configuration space involves a huge amount of steering operations and computation. In order to achieve real-time planning, the efficiency and the selected methods are crucial.

3.2.1 Car model

Designing local path for non-holonomic robots has been in scope of many research teams [43, 44, 45]. These research works use a simplified model for the car-like robot and compute paths made up of line segments connected with tangential circular arcs of minimum radius determined by the kinematics of the car. Such model is described with the following equations:

$$\dot{x} = cos(\theta)v, \tag{3.5}$$
$$\dot{y} = sin(\theta)v, \tag{3.6}$$
$$\dot{\theta} = \frac{1}{L}\tan(\phi)v, \tag{3.7}$$

where the notations match with Figure 3.5. The (x, y, θ) is the car configuration, v is the longitudinal speed, L is the distance between the front and rear wheel axles and ϕ refers to the steering angle.

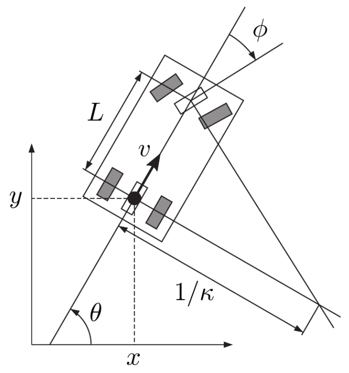


Figure 3.5: *Ackermann model for car-like robots* [36]

The turning radius can be expressed as: $\rho = \frac{L}{\tan\phi}$. The shortest path between two configurations for the simplified car was first established by Dubins [44] for a car moving forward only, and later by Reeds and Shepp [45] for the car moving in both directions. Although the resulting path is optimal in terms of length, the curvature of this type of path is discontinuous. Discontinuities occur between segments and arcs, also between arcs with opposite direction of rotation. In order to follow such path precisely the car would have to stop at each curvature discontinuity. This behavior is not preferred for human beings so curvature continuity is a desirable property. Continuity can be described with one more parameter: steering velocity. This is a finite, upper bounded variable of the robot. The implied equation that need to be met:

$$|\sigma| \leq |\sigma_{max}| = \frac{\Omega_{max}}{L\cos^2\phi} \tag{3.8}$$

where σ denotes the curvature change rate, Ω is the actual steering angle rate of change (steering velocity).

One of the most wide-spread algorithms is presented in the next section, the RS method is using the simpler model, and an improved version is described later that overcomes the problem of discontinuity is using the extended model of the car.

3.2.2 Reeds-Shepp's paths

Reeds and Shepp (RS in short) were the first who proposed a solution to find the shortest path for the simplified car model. This allowed the car to go forward and backward at a constant velocity or turn with minimum turning radius. This limitation enabled them to show that the shortest path between two configurations on a 2D plane belongs to a family of 48 paths. These paths are a result of at most 5 primitives' concatenation. A primitive in this context is either a line segment or a circular arc. Sussman and Tang [46] further restricted the possible path pool to 46 based on the minimum principle of Pontryagin.

Let S denote a straight line segment and C a circular arc with a radius ϱ. Left and right turns are represented with L and R. Furthermore subscripts denote the length of the straight line or the angle of the arc. The direction is represented with (+) for forward and (-) for backward, and cusps, where the direction change happens is with "|". The following 9 groups and 46 motion primitives can describe the shortest path:

Base word (groups)	**Sequences of motion primitives**
$C\|C\|C$	$(L^+R^-L^+)(L^-R^+L^-)$
$CC\|C$	$(L^+R^+L^-)(L^-R^-L^+)(R^+L^+R^-)(R^-L^-R^+)$
$C\|CC$	$(L^+R^-L^-)(L^-R^+L^+)(R^+L^-R^-)(R^-L^+R^+)$
CSC	$(L^+S^+L^+)(L^-S^-L^-)(R^+S^+R^+)(R^-S^-R^-)$ $(L^+S^+R^+)(L^-S^-R^-)(R^+S^+L^+)(R^-S^-L^-)$
$C\|C_\beta\|C_\beta C$	$(L^+R^+_\beta L^-_\beta R^-)(L^-R^+_\beta L^+_\beta R^+)(R^+L^+_\beta R^-_\beta L^-)(R^-L^-_\beta R^+_\beta L^+)$
$C\|C_\beta C_\beta\|C$	$(L^+R^-_\beta L^-_\beta R^+)(L^-R^+_\beta L^+_\beta R^-)(R^+L^+_\beta R^-_\beta L^+)(R^-L^+_\beta R^+_\beta L^-)$
$C\|C_{\pi/2}SC$	$(L^+R^-_{\pi/2}S^-R^-)(L^-R^+_{\pi/2}S^+R^+)(R^+L^+_{\pi/2}S^-L^-)(R^-L^+_{\pi/2}S^+L^+)$ $(L^+R^-_{\pi/2}S^-L^-)(L^-R^+_{\pi/2}S^+L^+)(R^+L^+_{\pi/2}S^-R^-)(R^-L^+_{\pi/2}S^+R^+)$
$CSC_{\pi/2}\|C$	$(L^+S^+L^+_{\pi/2}R^-)(L^-S^-L^-_{\pi/2}R^+)(R^+S^+R^+_{\pi/2}L^-)(R^-S^+R^-_{\pi/2}L^+)$ $(R^+S^+L^+_{\pi/2}R^-)(R^-S^-L^-_{\pi/2}R^+)(L^+S^+R^+_{\pi/2}L^-)(L^-S^+R^-_{\pi/2}L^+)$
$CC_{\pi/2}SC_{\pi/2}\|C$	$(L^+R^-_{\pi/2}S^-L^-_{\pi/2}R^+)(L^-R^+_{\pi/2}S^+L^+_{\pi/2}R^-)$ $(R^+L^-_{\pi/2}S^+R^+_{\pi/2}L^-)(R^-L^+_{\pi/2}S^+R^-_{\pi/2}L^+)$

Table 3.1: *Path primitives and groups for optimal path from Reeds and Shepp*

Between two configurations, the optimal path can be found in this pool of groups. RS is usually used to connect the points of the global path that is almost fully feasible for a non-holonomic robot. The geometric path has nonzero clearance from the obstacles and the steering method verifies the *topological property*, so the approximation succeeds in finite time [47].

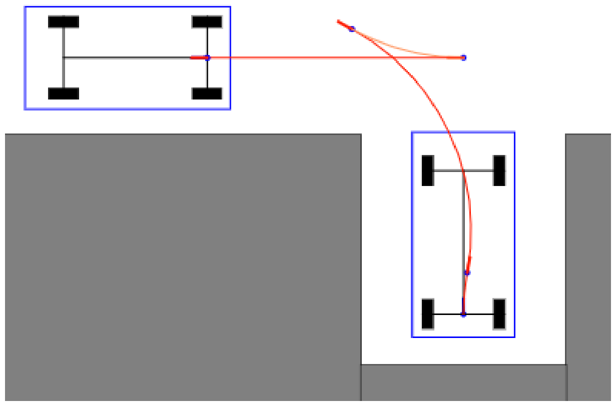


Figure 3.6: *Path planned with RS algorithm*

Figure 3.6 shows a parking maneuver using the RS algorithm.

3.2.3 Continuous-Curvature generalization of Reeds-Shepp's paths

As mentioned earlier, the drawback of the path generated with an RS algorithm for a car-like robot is that it has curvature discontinuities. This either has to be addressed in a way, otherwise the resulting trajectory is not feasible for a human observer or will result as a jumping error for the steering controller. Therefore path smoothing algorithms have been published over the last three decades. The used curves that results in a continuous curvature path can be categorized into two groups: curves that have closed-form expression e.g. B-splines [48], polynomials [49] or polar splines and parametric curves whose configuration point is a function of their arc length e.g. clothoids [50], cubic spirals or intrinsic splines [51].

CCRS is the abbreviation of Continuous-Curvature Reeds-Shepp algorithm. This steering method computes paths with the following properties:

- continuous-curvature
- upper-bounded curvature
- upper-bounded curvature derivative

The algorithm is similar to the RS in terms of path generation, but in order to ensure continuity it introduces CC-Turns and uses them instead of circular arcs. The final path is a sequence consisting of CC-Turns and straight line segments. It was shown, that the CCRS path is not optimal in terms of length, however they converge to the optimal RS path as the sharpness of the curvature goes to infinity. To achieve the desired continuity between the segments and arcs, a new motion primitive has to be introduced.

3.2.3.1 Clothoid

Clothoid is a curve which curvature changes linearly with its curve length. The curvature change is described one parameter: sharpness, denoted by α, which is constant all along the shape. It does not have a closed form of expression. The curvature, denoted by κ, as a function of the length of the curve:

$$\kappa(s) = \kappa(0) + \int_0^s \gamma\alpha d\xi = \kappa(0) + \gamma\alpha s \tag{3.9}$$

To derive the equations for the configuration along the clothoids, first the Fresnel-integrals have to be introduced:

$$C_F(r) = \int_0^r \cos\left(\frac{\pi}{2}\xi^2\right) d\xi, \tag{3.10}$$

$$S_F(r) = \int_0^r \sin\left(\frac{\pi}{2}\xi^2\right) d\xi, \tag{3.11}$$

where ξ is the variable of the integration. The (x, y, θ) configuration can be expressed as a function of length:

$$x(s) = \gamma\sqrt{\frac{\pi}{|\alpha|}} C_F\left(\sqrt{\frac{|\alpha|}{\pi}} s\right), \tag{3.12}$$

$$y(s) = \gamma sgn(\alpha)\sqrt{\frac{\pi}{|\alpha|}} S_F\left(\sqrt{\frac{|\alpha|}{\pi}} s\right), \tag{3.13}$$

$$\theta(s) = \gamma\frac{\kappa s}{2}, \tag{3.14}$$

where γ denotes direction of the motion.

When clothoids are used for path planning, eight different types can be distinguished and are shown in 3.7. The sign of sharpness $sgn(\alpha)$ is either *P*ositive or *N*egative. The robot's direction γ can be *F*orward or *B*ackward. Lastly, the sign of the absolute curvature change $sgn(\frac{d|\kappa|}{ds})$ determines whether a clothoid is *in-type* or *out-type*.

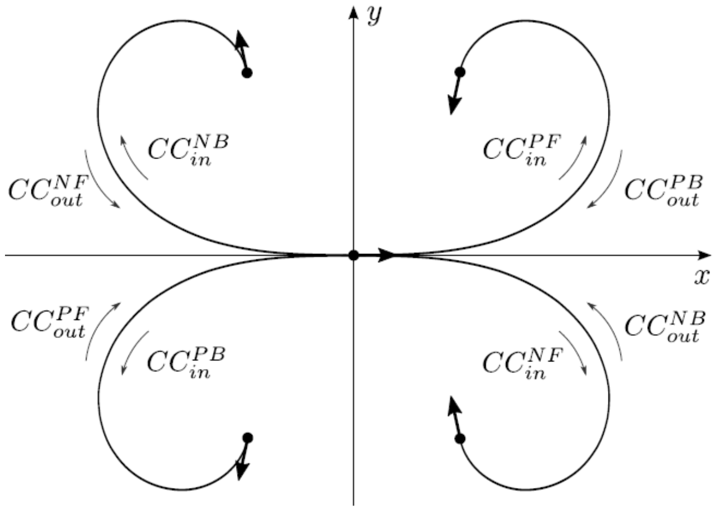


Figure 3.7: *Different types of clothoid curves* [36]

Defining clothoids can be done in various different ways. It was shown by Wilde [52] that a clothoid can be described with a (κ, δ) pair, the curvature and the deflection of the curve at a certain point. The paper only covers the case of forward motion, and the more general case was presented in [36]. Furthermore, a new representation is shown in the paper, that helps us to reduce the number of free parameters from two to only one. This makes the explicit computation of these curves simpler. The former variable pair can be written as:

$$\alpha = \frac{\kappa^2}{2\delta} = \frac{\kappa_{CC}^2}{2\delta_{CC}} \tag{3.15}$$

where κ_{CC} and δ_{CC} can be any fixed coherent pair. For simplicity, it is preferred to point them to the endpoint, where they have maximum value along the curve. With this representation the necessary equations that fully describe the *in-type* clothoid:

$$x(\delta) = sgn(\delta_{CC})\frac{\sqrt{2\pi|\delta_{CC}|}}{\kappa_{CC}} C_F\left(\sqrt{\frac{2|\delta|}{\pi}}\right), \tag{3.16}$$

$$y(\delta) = \frac{\sqrt{2\pi|\delta_{CC}|}}{\kappa_{CC}} S_F\left(\sqrt{\frac{2|\delta|}{\pi}}\right), \tag{3.17}$$

$$\theta(\delta) = \delta, \tag{3.18}$$

$$\kappa(\delta) = \kappa_{CC}\sqrt{\frac{\delta}{\delta_{CC}}}. \tag{3.19}$$

The expressions for the out-type clothoids are very similar. The changes occur in the sign of the maximum deflection: $\delta_{CC}^{out} = \delta_{CC}^{in}$ and in the calculation of the total deflection: $\delta_{out} = \delta_{in} - \delta_{CC}^{in}$. The C_F and S_F represents the corresponding Fresnel-integrals.

3.2.3.2 CC-Turns

A CC-Turn is a geometric element made up of three parts:

- in-type clothoid
- circular arc
- out-type clothoid

It has continuous-curvature along the whole turn and is used in CCRS instead of a simple circular arc.

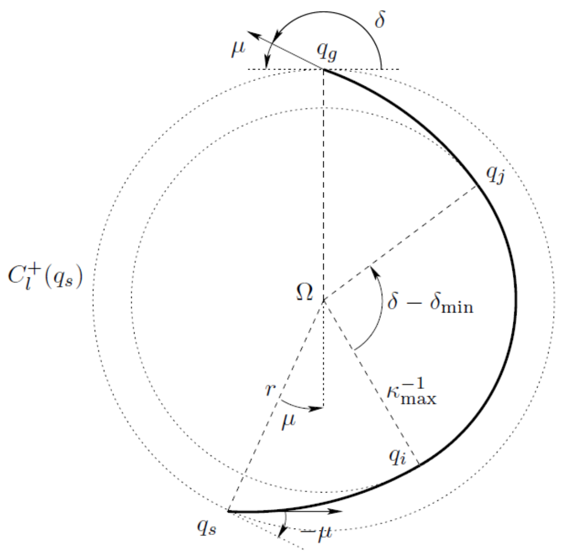


Figure 3.8: *General case of CCTurn* [53]

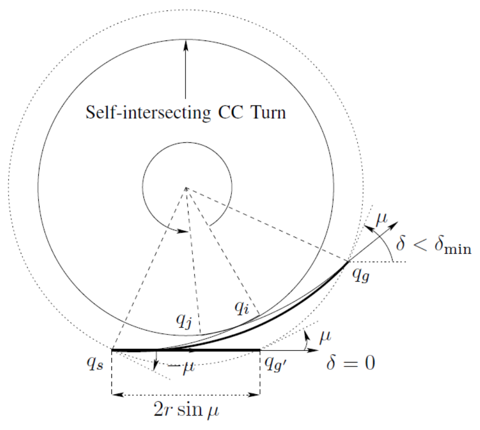
In this general case, the first part of the CC-Turn is a clothoid arc of sharpness $\alpha = \alpha_{max}$ whose curvature goes from 0 to κ_{max} followed by a circular arc with radius $R = 1/\kappa_{max}$ and an *out-type* clothoid with sharpness $-\alpha$ whose curvature goes back to 0. The sign of α and κ determines the shape if the CC-Turn. From implementation point of view, it can be assumed without loss of generality, that the starting configuration is $(0, 0, 0, 0)$, where the last element is the curvature. After this primitive is constructed, it can be transformed to any pose in the plane. The q_s and q_g are the start and goal configuration, and the q_i, q_j are the beginning and the end of the circular arc.

There are cases when the CC-Turn is degenerate. Let δ_{min} denote the minimum deflection required for the in-type clothoid to reach the maximum curvature κ_{CC} from 0 followed by an out-type clothoid that goes from κ_{CC} to 0. The first scenario is when the total amount of deflection is less then the minimum δ_{min}, so there is no circular arc in the CC-Turn. This is usually referred as *Elementary path* and can be seen on Figure 3.9a. Calculation of δ_{min} :

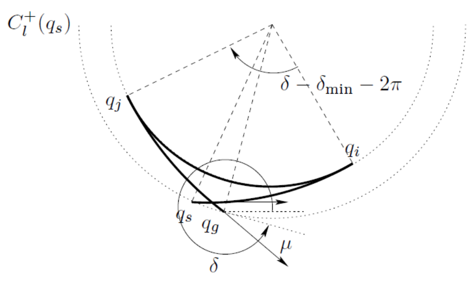
$$\delta_{min} = \frac{\kappa_{max}^2}{\sigma_{max}}, \tag{3.20}$$

where σ_{max} is the upper bound of the curvature change rate. Lets denote v the constant velocity of the robot, than σ can be computed as the following: $\alpha_{max}|v| = \sigma_{max}$

The other case is when the total amount of deflection has a value from the following range: $\delta \in [\delta_{min} + \pi; 2\pi)$. In this case, it is possible to further reduce the length. The RS-car (and therefore the CC-car also) can make backward motions (unlike the Dubins car), so it is possible to refine the CC Turn to start moving backwards when it reached the end of the first, in-type clothoid. Moving back with a deflection of $\delta - \delta_{min} - 2\pi$ will result in the exact same position as it would be if it had traveled all along the circular arc. This path is not only shorter but is more likely to be collision free, since the *bounding box* of the total trajectory is way smaller. This case can be seen in Figure 3.9b.



(a) Elementary path

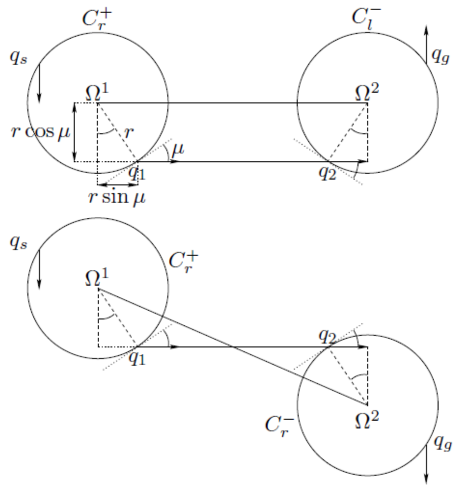


(b) Backward arc path

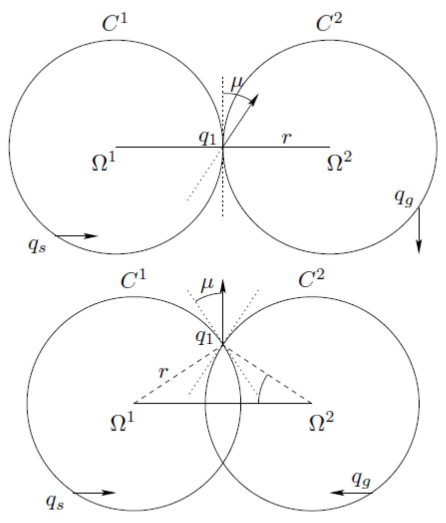
Figure 3.9: *Degenerate versions of the CC-Turn* [53]

3.2.3.3 CC Path

To reach any arbitrary configuration on the free space, connecting CC-Turns and segments are necessary, their result is called CC Path. Constructing them is similar to the RS method. The main difference here is that now two circles are connected with a μ-tangent line segment. To help understanding the scenario, Figure 3.10 shows a typical CC path consisting of a C-S-C and C-C sequence where C denotes a CC-Turn.



(a) CSC sequence



(b) CC sequence

Figure 3.10: *μ-tangent line connection of the common cases of CC-path* [53]

The necessary groups for path planning in case of CCRS are similar to the ones for the RS path. One possible grouping was described more detailed in [54].

Papp D. had a CCRS implementation for a previous version of our system. Our implementation reused some of his ideas regarding the grouping of the path type. Also, we had to integrate the algorithm to the currently used robotic framework. Furthermore, we focused on improving the computational speed as well. According to the book, written by Nikolaus Correll [55], collision checking could take up to 90% of the execution time in path-planning problems so a successful and quick local method is mandatory. Usually during collision checking, the local path planner is invoked until a free path is not found.

Some best practices we used during implementations:

- if it is possible only compute the points once and transform them later to the required configuration
- use look-up tables wherever it is possible

We followed the first one for every geometric element — clothoid, arcs, CC-Turns. The latter one was used for the Fresnel-integrals to fasten the calculation of clothoids [56]. The values were stored for elements, having maximum deflection of $\pi/2$ and was multiplied by a gain calculated from the varying parameters of the primitive. Along with these improvements, our implementation has become around 4 times quicker than the earlier, making it a better option to choose as a local planner in highly congested areas.

3.2.3.4 Result comparision of CCRS and RS path

Figure 3.11 shows a perpendicular parking maneuver situation. This kind of parking happens the most in case of a parking garage.

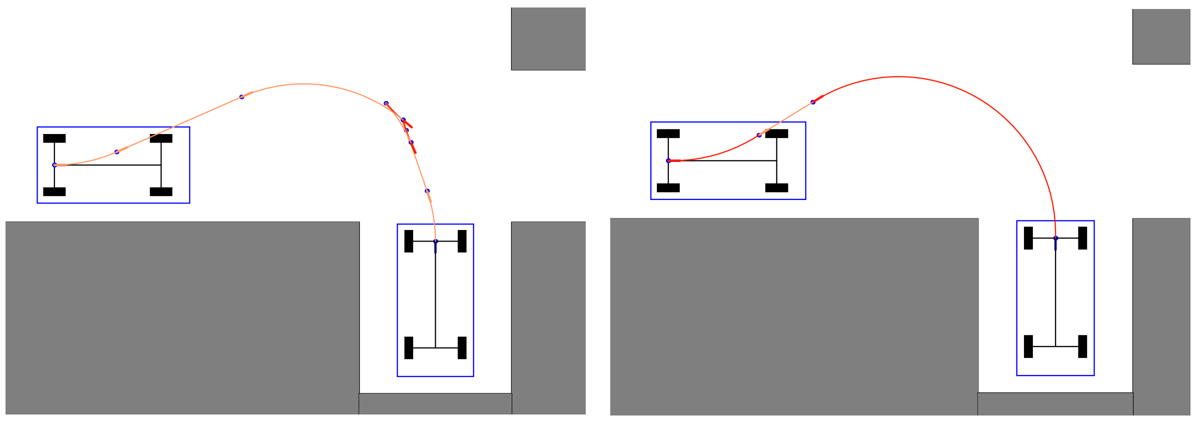

(a) Planned with RTR+RS (b) Planned with RTR+CCRS

Figure 3.11: *Perpendicular parking maneuver with different planners*

The path planned with RS is more crude and would not be comfortable for a human passanger. The path is not only contains a backward motion part, but also a couple of discontinuities where the car must stop, and reorient its wheels. On the other hand, the path designed by the RTR+CCRS planner combination provides a smooth and continuous curvature path, which the car can execute without stopping. Also, this path is more human-like, therefore is preferred in a hybrid environment, where autonomous and human-driven cars are present.

3.2.4 Approximator

In this section, the aforementioned approximator, that was used together with the global and local method, is described. The main task is to divide the global paths into smaller parts that can be followed by the robot. When the path is created by the global planner, $(start, goal)$ configurations should be given to the local planner. At first, the global path's start and goal configuration is given, but the local planner is most likely to fail to return a collision-free path. In the next iteration the local planner should be invoked for planning with a $(start, goal)$ point pair, where the new goal point shall be closer to the start then the previously selected goal configuration. The next goal point in case of this approximator is chosen as the following:

- the end configuration of the primitive before the last one
- if there are only one primitive left, then the last one is halved

The shape and length of the primitives depends on the used global planner. In our case for the RTR planner, the primitives are straight line segments and rotational movements.

If the local planner successfully provides a path for the invoked start and end configuration, then in the next iteration, the reached goal will be chosen as start and the global goal will be the goal. To prevent an endless division of the last segment, a minimum distance is specified, and the approximation report failure.

3.3 Methodology

The path planning methods were chosen in the aspect of the project's goal. As it was described earlier in Chapter 1 our goal was to create and automated valet parking system. That being said, the used algorithms have to perform well in parking lots and in narrow, dynamically changing area. Based on the previously presented literature and algorithm review, we ended up using two, different methods. The first one is the Hybrid A* planner in structured environment and the second one is a combination of the RTR and CCRS with the approximator described.

In the presented scenario, we have captured the map in advance with Cartographer and is using AMCL for localization in runtime. Since the map was available, we manually added a track graph for the lanes. They helped the Hybrid A* planner to keep the car in the correct side of the road during it is scanning for an empty parking lot. The properties of this planner makes it a suitable and reasonable candidate. For the parking task, we wanted to test another method, that can provide successful solution fast enough in narrow spaces. It can happen from time to time, that we have to invoke the planner as the environment is changing, so we chose the mentioned combination: RTR+CCRS. Furthermore the parking maneuver is performed in very narrow area, so it is crucial for the robot the follow the path very precisely. The result of the combined planners always provide such path with the properly measured and tuned parameters.

Chapter 4

Dynamic object detection

Autonomous navigation of robots and cars requires adequate models of their static and dynamic environment. In many situations, for instance in the given parking space detection example, the map of the static environment can be constructed offline and used as a reliable starting point. In a more general situation, where the static environment is unknown, the standard solution is based on perceptions of the distance from objects, then processing the measurements to build a local representation of the surrounding scene and finally integrating the local representation into a global one [57].

In contrast, dealing with the dynamic environment is always challenging since the focus is more on the interpretation of an unknown situation, so the robot is able to react properly, which is a difficult and a computationally demanding task. Furthermore, detecting moving objects from an autonomous vehicle is a major precursor to many activity recognition, object recognition and tracking algorithms.

4.1 Object detection

Detection can be thought of as general or object-specific detection. In the case of general detection, the goal is typically to find the area in which the vehicle can safely navigate. For this task, it is expedient to use distance-based detection, to which we can apply active and passive sensors. The former involves sensors based on RADAR, LASER or LIDAR and ultrasonic, while the latter includes optical (camera) sensors. In the case of object-specific detection, there are two main approaches, one is called traditional object detection whilst the other one uses learning-based algorithms to solve this task. In traditional object detection, the recognition algorithms typically focus on the outline, shape, structure, or symmetry of the object based on an image taken by a camera and look for the features that define it [58] . As for deep learning, it rejects the traditional programming paradigm where problem analysis is replaced by a training framework where the system is fed a large number of training patterns (sets of inputs for which the desired outputs are known) which it learns and uses to compute new patterns [59].

For clarification, in this paper, the problem definition of object detection means the task to determine where objects are located in a given image (object localization) and which category each object belongs to (object classification).

The first part of this chapter gives a brief overview about the typical sensors used to solve the issue of object detection and declares the choosen sensor. The second part is about the different methods used in object detection.

4.1.1 Sensors used in object detection

The most typical sensors used for object detection in autonomous vehicle systems are sonars, LIDARs, RADARs and cameras. Including more sensors into sensor fusion system benefits with better performance and the robustness of the solution since each sensor has its own advantages and drawbacks [58, 60].

Ultrasonic sensors are inexpensive, but have short-range, which can be further reduced by dusty environments, and reflections impair their accuracy due to their poor angular resolution. They are typically used in automatic parking systems and low-speed adaptive cruise control because they are not suitable for accurate object localization for the reasons mentioned [58] .

Laser-based distance sensors promise better resolution but are much more complex and expensive technology. They are accurate, capable of high-resolution 3D measurement and are insensitive to environmental influences, however, dark, light-absorbing surfaces can cause problems [58, 60] .

RADARs have relatively long distance but low angular resolution, so they are not really effective in scenarios with multiple objects. These sensors are typically used for adaptive cruise control, collision avoidance and emergency braking systems as well as for classification in object detection [60] .

The use of cameras as sensors is becoming more and more popular, in addition to the fact that image processing technologies have evolved a lot during the last few decades. At the same time, these devices have also become cheaper, more compact and their quality got better. Nor can we ignore the fact that the computing power available to computers has also increased dramatically. These advances in technology have contributed greatly to be able to use computer image processing as a real-time method in practice [61, 62] .

Although RADAR and LIDAR systems used in the automotive industry can detect obstacles effectively, their ability is limited when it comes to distinguishing objects. The camera, on the other hand, can be used for this and many other purposes. However, for a moving vehicle in a dynamic environment in most cases, it is crucial to have information about the distance from the recognized object so that collision can be avoided. There are methods to estimate the distance based on a single image input, but the accuracy of these methods is questionable [63].

To resolve this contradiction, the sensor that is chosen in this paper for object detection is an RGB-D camera. These type of cameras capture RGB colour images augmented with depth data each pixel. A variety of techniques can be used for producing the depth estimates, such as time-of-flight imaging, structured light stereo, dense passive stereo, laser range scanning, etc. While many of these technologies have been available to researchers for years, the recent application of structured light RGB-D cameras to home entertainment and gaming has resulted in the wide availability of low-cost RGB-D sensors, that are well-suited for robotics applications [64, 65, 66] .

4.1.2 Object detection based on traditional vision

As it was stated earlier, to gain complete image understanding, one should not only concentrate on classifying different images but also try to precisely estimate the locations of the objects contained in each image. These two tasks, classification and localization are the two key steps to a complete object detection algorithm.

In traditional techniques, a computer vision solution that performs classification usually consists of the execution of several algorithms in a sequence called an algorithmic pipeline [67]. The first step is a pre-processing phase with image correction like noise filtering or intensity

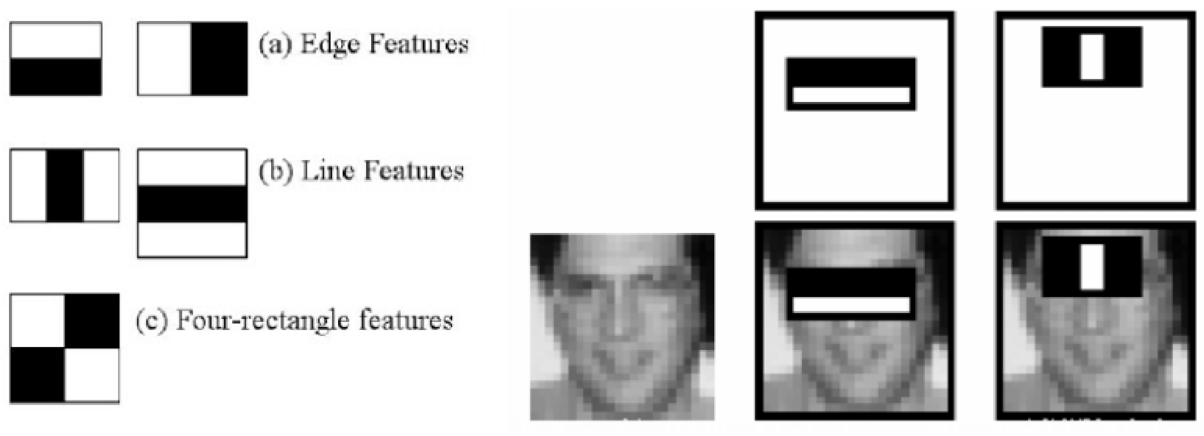


Figure 4.1: *Haar-like features (left) and their use for facial detection (right)* [67]

transformation. This is followed by a feature extracting phase, the purpose of which is to transform the information found in the pixels of the image to a a higher level of abstraction, which are image characteristics . These image features (SIFT, SURF and BRIEF are the most common for object detection [59]) are designed to easily separate task-relevant information from interfering effects in the space they define. The last step is the decision phase in which the algorithm assigns a tag to the given image based on the image characteristics [67].

In the sequel, some traditional classification and segmentation techniques are explained, to get a general view of this topic.

4.1.2.1 Classification techniques

Viola-Jones: This method is also known as Haar Cascade detector because it uses a special feature type, called Haar-like features. Even though it is good for general object recognition, it is typically used for face detection (4.1 [67]). The Haar features examine a detail of the image using a binary window by subtracting the pixels below the black parts from the sum of the intensities of the pixels under the white parts. This gives a signed number that describes the similarity between the window and the image detail. A large positive result means similarity, a large negative result means contrast, while a result around zero means a complete lack of similarity [67].

One of the main drawbacks of using Haar-like features is that even for a relatively small part of the image, there are a huge number of different features which would require a tremendous amount of computation. This number is reduced with the help of several optimization methods but this takes a lot of additional effort. Another disadvantage is that the characteristics used are not invariant to image transformations, so the algorithm performed with their help is only able to compensate for this problem to a certain extent [67, 68].

Bag of (Visual) Words: The original 'Bag of Words' idea is that a text can be classified by topic based on the relative frequency of words that appear in the text.

The same concept can be easily adapt to the case of image classification where the 'words' are local image features. These are usually corner-like points coded into a transformation-invariant descriptor. These features simultaneously encode the look and structure of the image in a small local slice, and from their relative location, the global shape can also be inferred. Their main disadvantage is that their calculation is quite expensive.

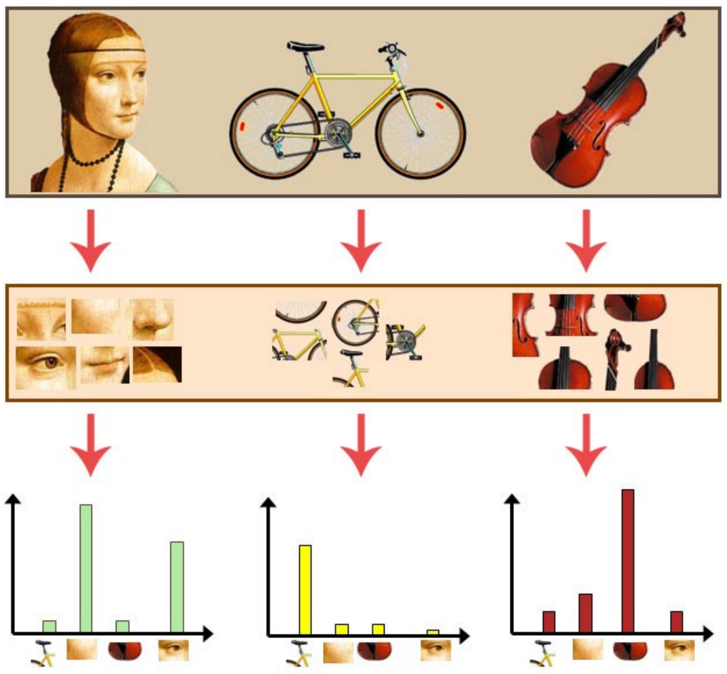


Figure 4.2: *Histogram of visual words* [67]

A so-called visual dictionary is also needed so that the classification can be done based on it. The objects are represented as a cluster of words, and the choice of the number of clusters is basically the task of the designer, who has to find a compromise value between reducing the compactness of the clusters and over-increasing the number of groups. If the visual dictionary is successfully constructed with the help of clustering, the new local image features can be easily assigned to its words by minimizing the square error from the word centers [67] .

The histogram of the visual words in a given image expresses which relative frequency of the words in the visual dictionary occurs in the image in Figure 4.2. A big shortcoming of this method is not utilizing any information about the absolute or relative position of each visual word to make the classification, so it lacks the ability to localize the position of the object.

Deformable part-based model (DPM): This method overcomes the mentioned drawback of the previous one by describing each class as a graph of words, where the edges of the graph represent the geometric relationships between each word. These connections are not completely rigid, but can vary within certain limits (deformation). As it can be seen in Figure 4.3 the features used by deformable submodels are basically convolutional filters, which have two types: central (root) and partial (part) filters. The former has the task of giving a large response at the centre of the object, while the latter must give a large response at the position of the previously mentioned parts of the object [67, 69].

4.1.2.2 Segmentation techniques

In the field of computer vision, there are several scenarios when it is important to classify each pixel into separate objects. This task is called segmentation, which output is usually a multi-state image, which contains the labelled pixels [67].

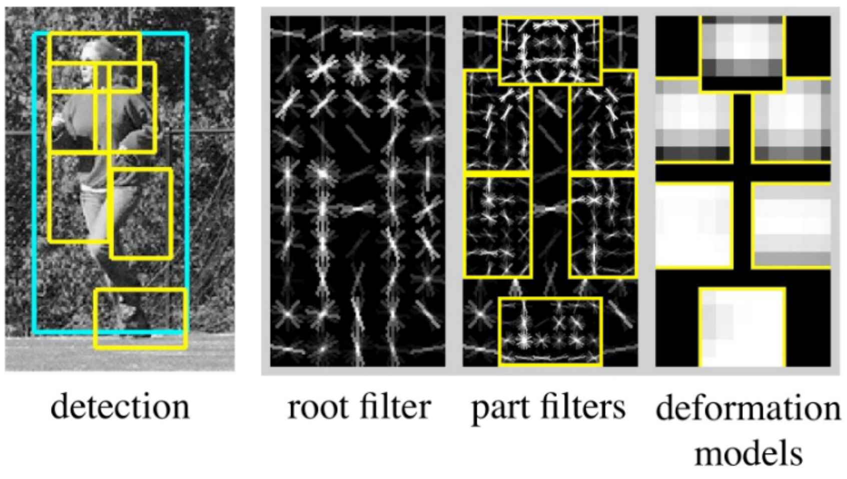


Figure 4.3: *Visualisation of the two type of filters on an example image* [67]

The simpler techniques are based on colour and intensity thresholding or clustering. Other methods try to find related regions based on different inclusion criteria, these are called region-based methods. There are also edge, motion, and graph-based methods [67].

Clustering The essence of clustering is to divide the points of a set of points defined in any space into some subset, that is, a cluster, so that the clusters thus obtained satisfy some compactness criterion as much as possible. Although several algorithms have been proposed for clustering, one of the most common solutions is the k-means procedure [67].

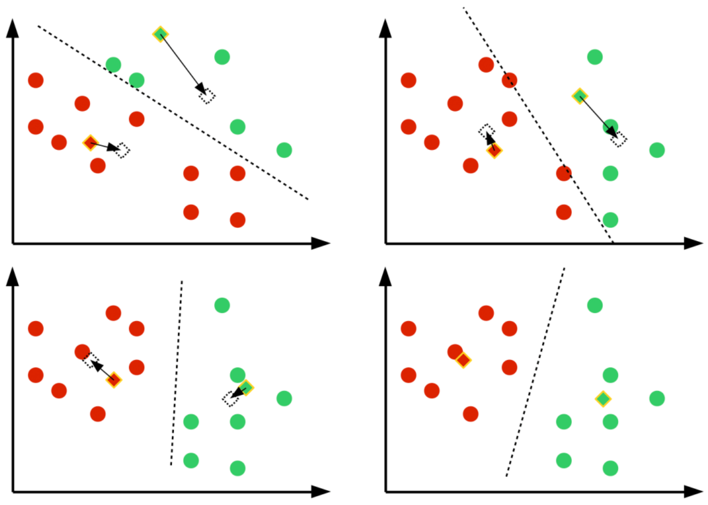


Figure 4.4: *The process of k-means clustering* [67]

K-means The k-means method intends to classify the set of points into k clusters so that the sum of the squared distances of the elements of these clusters from the centre of the cluster is minimal [67]. It uses an iterative algorithm whose initialization step is to randomly place k centres in the space stretched by the data points. It then repeats the next two steps until the algorithm converges. Figure 4.4 shows the steps of k-means. As a first step, you assign each point to the cluster centre closest to it, thus changing the cluster assignment of each point. It then assigns a new value to each cluster centre, which will be the arithmetic mean of the points in that cluster. This step changes the position of the centres, so that which point belongs to which cluster, so that the iteration can continue to run.

4.1.3 Object detection using machine learning

Traditional vision techniques use descriptive analysis to solve the problem, and as a result, the algorithms they use are exact and can easily be formulated as some series of instructions and logical conditions. Deep learning, on the other hand, takes a completely different approach. It rejects the traditional programming paradigm by replacing problem analysis with a training framework, which tries to discover the rules that underlie a phenomenon. This method is called predictive analysis [59] .

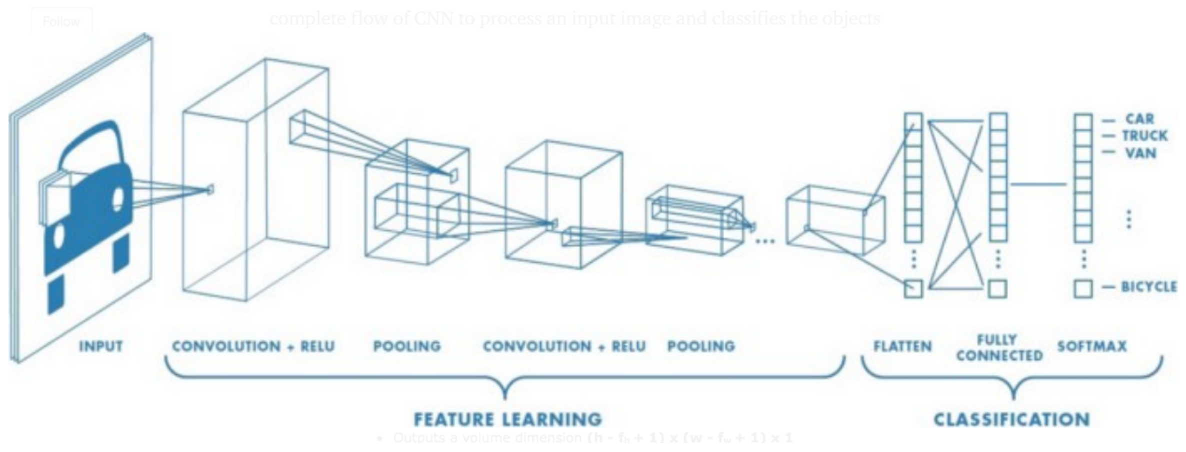


Figure 4.5: *Complete flow of a CNN to process input image classification* [70]

Although there are many types of neural networks in deep learning, most of them are not considered effective in the field of computer vision, because an average image has at least hundreds of pixels and usually three channels, so traditional fully-connected architectures would generate millions of parameters, which easily leads to overfitting and requires lots of computation [67].

Perhaps the most commonly used neural networks in object detection are CNNs or Convolutional Neural Networks [71]. In a CNN architecture each input image passses through a series of *convolutional layers*, *pooling*, *fully connected layers* (FCs) and finally a *activation function* classifies the object. The flow is shown in figure Figure 4.5, the different parts of CNN has different functions which are described in the sequel [70].

Convolution Layer Convolution is a mathematical operation that extracts features from the input with the help of a kernel (or filter) that preserves relationship between pixels. Based on the chosen kernel the operation can perform edge detection, blurring, sharpening etc.

Convolution layers have a parameter called *stride*, which determines the number of pixels to be shifted with kernel during convolution.

Sometimes the filter with a given stride does not perfectly fit the input image. In this case there are two options, either the image is padded with zeros or the part where the filter did not fit is dropped [70].

ReLU ReLU stands for Rectified Linear Unit for a non-linear operation. It can be described as follows:

$$f(x) = max(0, x) \tag{4.1}$$

ReLU's purpose is to introduce non-linearity into the neural network. Other non linear functions can substitute it, but in data science ReLU is tend to be the best solution due to its performance [70].

Pooling Layer This layer helps to reduce the number of parameters, in case the image is too large. This step reduces the dimensionality of each map but keeps the important information. This step usually called as spatial pooling or downsampling. The typical types are max, avarage and sum pooling [70].

Fully Connected Layer The Fully Connected layer is a traditional Multi Layer Perceptron [72] that uses a softmax activation function in the output layer. The Softmax function takes a vector of arbitrary real-valued scores and squashes it to a vector of values between zero and one that sum to one. The output from the convolutional and pooling layers represent high-level features of the input image. The purpose of the Fully Connected layer is to use these features for classifying the input image into various classes based on the training dataset [70].

Activation function This is the final step of the CNN, which classifies the output.

The frameworks of generic object detection methods can mainly be categorized into two groups [71]. One starts by generating region proposals and then classifying each of them into different object categories similarly to the traditional object detection pipeline. The other tries to solve the two problems (categorization and localization) in one step. These are usally labelled as regression-, classification-based or single shot frameworks.

4.1.3.1 Region Proposal-Based Frameworks

There are several region proposal-based methods like R-CNN [73], Fast R-CNN [74], Faster R-CNN [75], spatial pyramid pooling (spp-net) [76] etc. To have a general view about this approach the next pharagraph briefly explains evolution of R-CNN into Faster R-CNN.

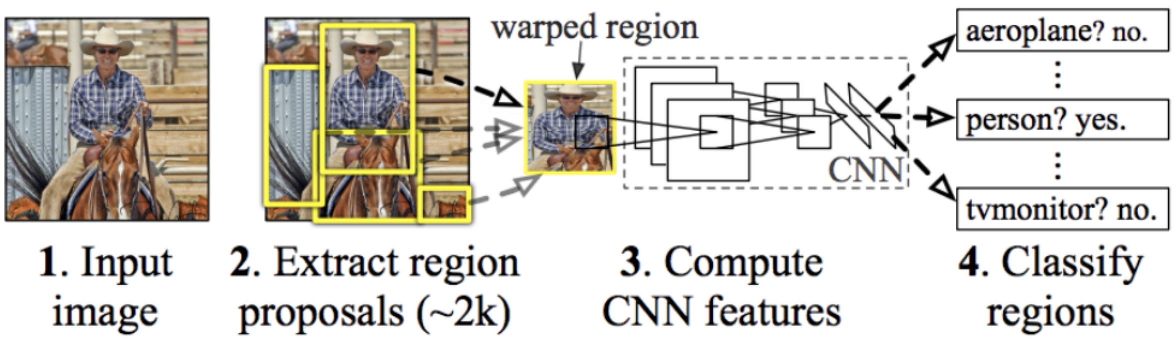


Figure 4.6: *Architecture of R-CNN* [77]

R-CNN The R-CNN adopts a selective search to generate about 2000 region proposals for each image. These 2000 candidate region proposals are warped into a square and fed into a convolutional neural network that produces a 4096-dimensional feature vector as output. The CNN extracts the features and feds them into an SVM (Support Vector Machine [78]), to classify the presence of the object within the region. In addition, the algorithm also predicts four values which are offset values to increase the precision of the bounding box [71]. Figure 4.6 shows the architecture of the method.

The drawbacks of this method that to train the network one should classify 2000 region proposals per image and the processing time is far from real time, as it takes around 47 seconds to test each image.

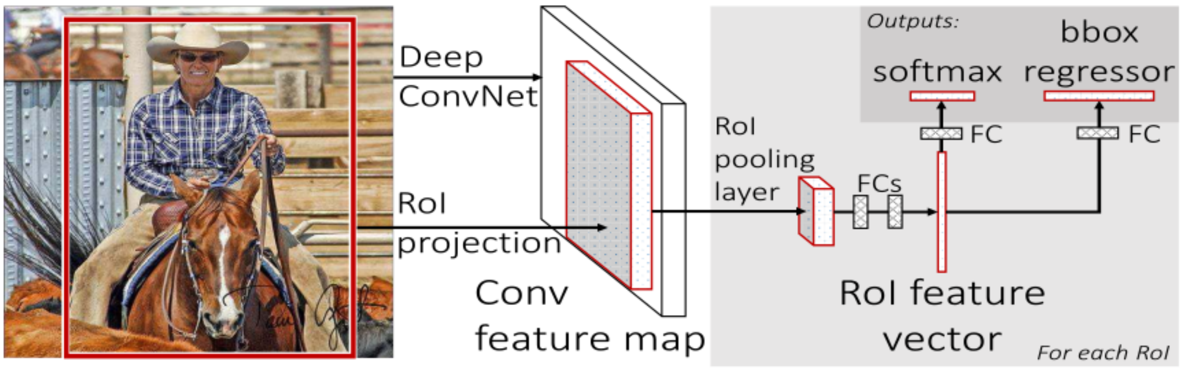


Figure 4.7: *Architecture of Fast R-CNN* [77]

Fast R-CNN Figure 4.7 shows the architecture of Fast R-CNN. Instead of feeding the region proposals to the CNN, the input image is directly fed to the neural network to generate a convolutional feature map. From this, it identifies the region of proposals and with the help of a RoI pooling layer it reshapes the proposals into a fixed size so they can be fed into a fully convolutional layer [71, 77].

With these modifications Fast R-CNN is able to produce output in every 2 seconds. While this is a significant improvement compared to the previous algorithm, this method can not be used in real time applications [79].

Faster R-CNN Both of the previously mentioned algorithms use selective search to gather region proposals. This turned out to be a slow and time consuming process which affects the performance of the network. Faster R-CNN enchances its performance by leaving the network to learn these region proposals [77].

Instead of using a selective search algorithm on the convolutional feature map (like Fast R-CNN does), it uses a separate network to predict the region proposals. The rest of the steps are similar to Fast R-CNN [77].

With these improvements, Faster R-CNN is finally able to detect obects under 0.2 seconds, which means it can operate at 5 frames per second [79].

4.1.3.2 Single Shot Detection Frameworks

Frameworks that are able to map image pixels straightly to bounding box coordinates and class probabilites seem to outperform region proposal-based methods in terms of execution speed[71]. To enhance performance, these frameworks tend to use lower resolution which affects their accuracy. The two state of the art frameworks that are described here are YOLO [80] and SSD [81].

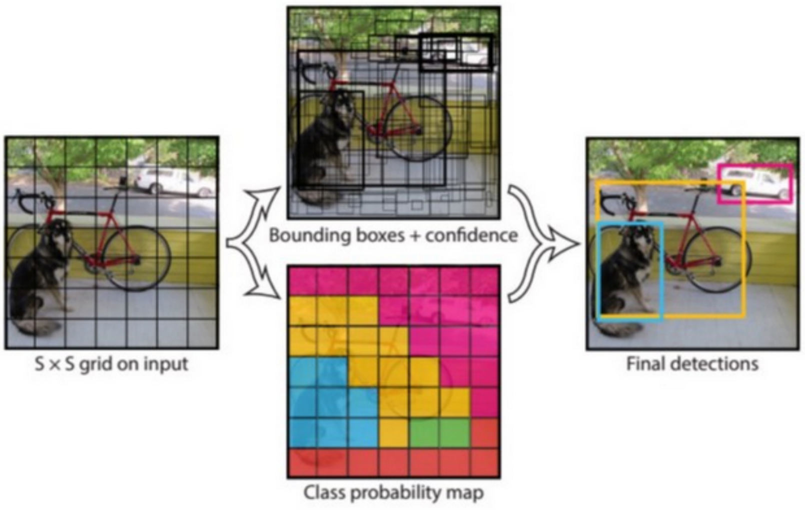


Figure 4.8: *The model of YOLO* [80]

You Only Look Once (YOLO) Yolo divides the image using an SxS grid and each grid cell is responsible for predicting the object centered in that grid cell. Each of them predicts B bounding boxes (BBs) and confidence scores for those boxes. Each bounding box consists of 5 predictions: x, y, w, h and $confidence$. x, and y are the coordinates of the center of the box, while w and h are the width and height relative to the image size. There are also C conditional class probalities predicted in each cell. As a result, all cells has $Bx(C + 5)$ outputs, which are produced by a $1x1$ convolutional filter. Figure 4.8 shows the general process of the algorithm.

YOLO has a difficulty in dealing with small objects in groups, which is caused by strong spatial constraints imposed on bounding box predictions [71].

There are newer version of YOLO, trying to improve the performance of the original one. In YOLO2 the width and height of the enclosing rectangle are estimated in relation to a reference rectangle (so-called anchor box), of which there are a total of B pieces (one for each estimate). The width and height values of each anchor box are determined using B-element clustering on the rectangles in the training database. It is also important to mention that during detection, YOLO may find an object more than once, in which case we keep the highest confidence value of the predictions of too similar a shape, while discarding the rest. This step is called non-maximum suppression [67] .

Single Shot MultiBox Detector (SSD) Instead of fixed grids SSD takes the advantage of a set of default anchor boxes with different aspect ratios and scales to discretize the output space of bounding boxes. During prediction the network generates scores for each object category present in each default box. The network combines predictions from multiple feature maps with different resolutions, which helps it to conveniently handle objects with various sizes [81].

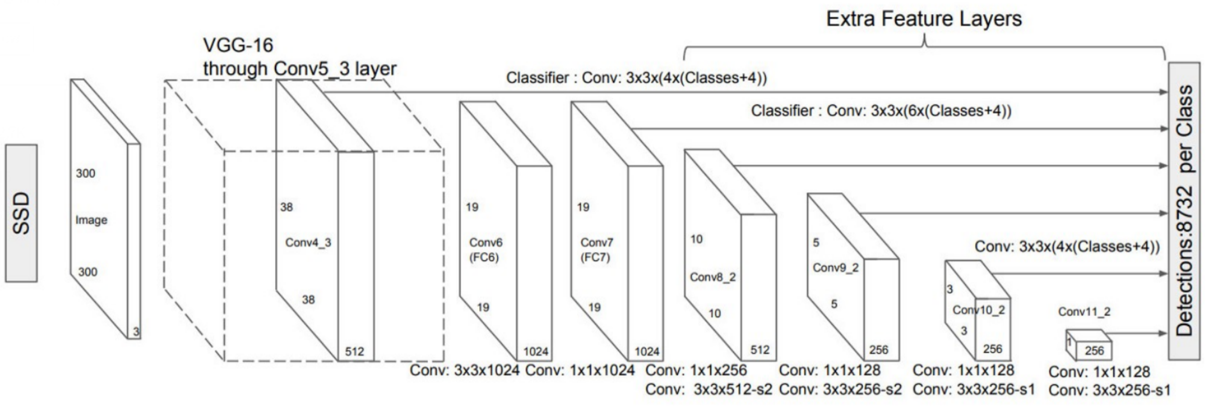


Figure 4.9: *The model of SSD* [81]

SSD uses VGG16 network as a feature extractor (the same CNN is used in Faster R-CNN). Then it adds custom convolution layers afterward. However, each of these layers reduce the spatial dimension and resolution. To overcome this problem it does independent object detection from multiple feature maps as it can be seen in Figure 4.9.

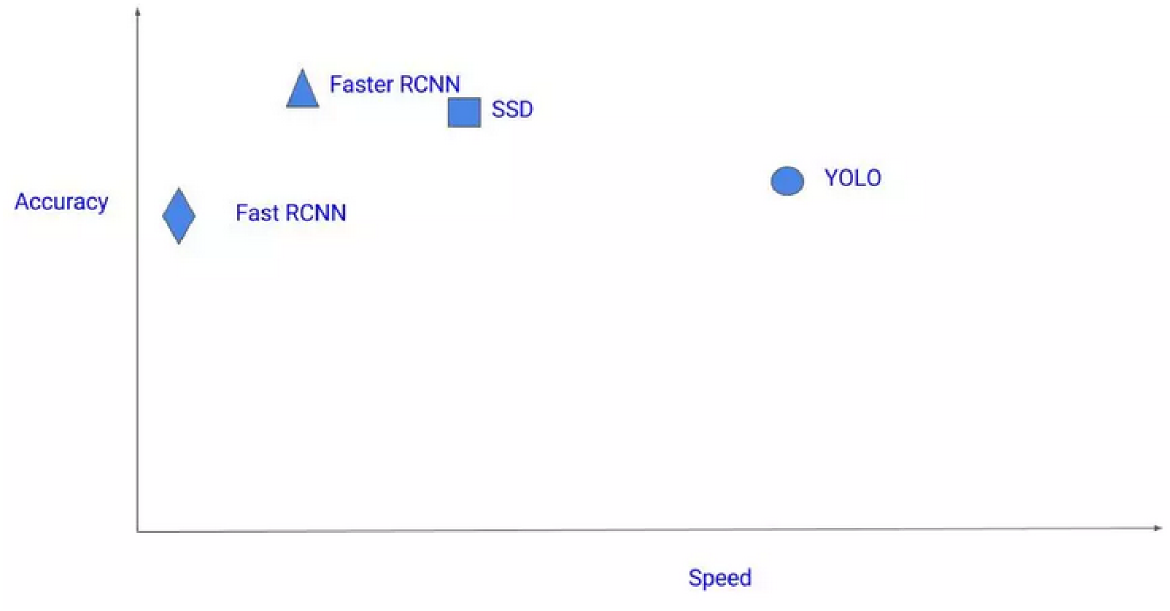


Figure 4.10: *A comparison between different frameworks* [79]

As it was already stated, single shot frameworks are superior to region proposal-based methods in terms of execution speed. However, it is important to mention that the price for the speed must be payed, and it costs accuracy. As Figure 4.10 shows each algorithm has its pros and cons [79]. Altough YOLO runs the fastest, it provides the worst accuracy amongst these methods, while Faster R-CNN gives the best. As for SSD, it seems to be a good solution as it is able to run on a video with little accuracy trade-off [82].

In conclusion, there is no golden rule when it comes to choosing the right object detection method because it depends on the problem one is facing.

4.2 Methodology

In the current valet parking scenario, our focus was on detecting people moving around the parking spaces and avoiding collision with them. A collision avoidance system using two (a front- and a back-facing) LIDARs have already been developed for the autonomous vehicle, which uses the point cloud created from the fusion of the two sensors perceptions as an input. However, these sensors could only detect objects at a given height, which caused failures in some scenarios. To overcome this problem we expanded this solution with the help of image-based object detection and depth image segmentation, where the dynamic object detection algorithm also contributes to the input of the collision avoidance system by adding the recognised people to the point cloud. It is important to emphasize, that the implemented method is modular and can be used in various other applications.

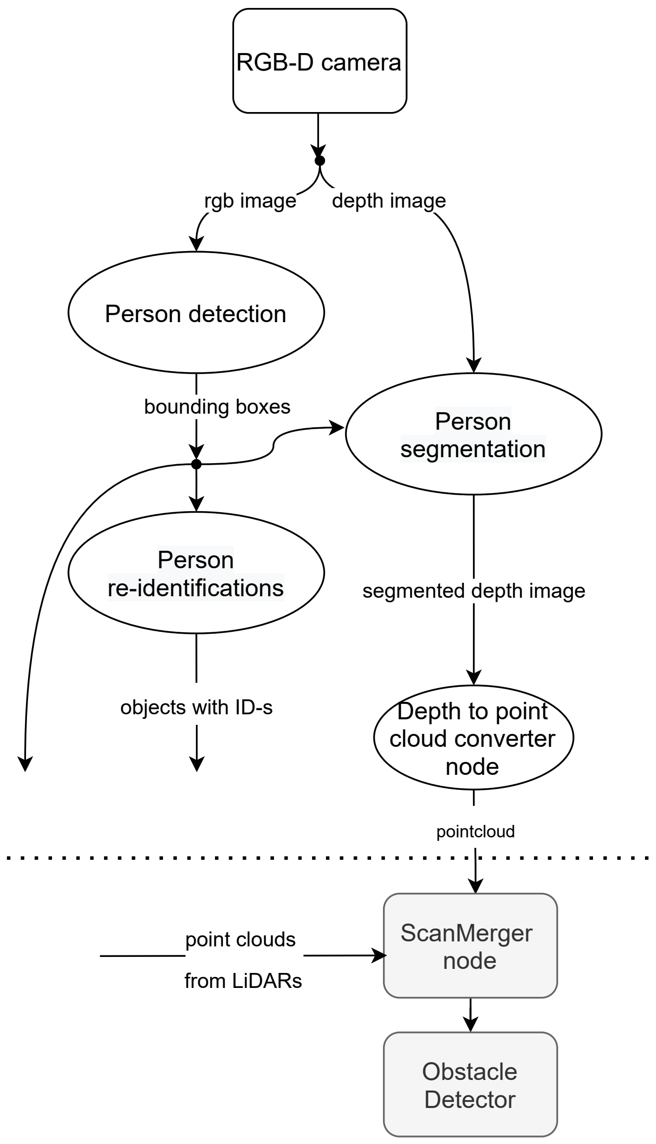


Figure 4.11: *The process of dynamic object detection*

As it was already stated, an RGB-D camera provides the input to the dynamic object detection algorithm. In practise, the inputs are two ROS Topics, one stores the RGB image with the bit depth of 8 on each channel, while the other contains the depth information on 16 bits, where each pixel value stores its the distance from the camera in millimetres. The general view of the process can be seen in Figure 4.11. After the first step, which is person detection the output is used for re-identification and depth image segmentation. The outputs of the re-idenetification part are yet to be utilized, while image segmentation is used for collision avoidance. To achive this the segmented depth image is converted into point cloud, then merged together with point clouds from other sources (LIDARs) from which the Obstacle Detector make decisions during motor control.

The gray colored nodes were already implemented and their function is the following:

- **ScanMerger node:** This node is able to merge point clouds from different sources into one single set.
- **Obstacle Detector:** This node is able to control the motor and limit its maximum speed if necessary. The decision is affected by the point cloud provided on its input.

All the communication is implemented using ROS topics, and the implemented solutions are described in the following part of this section excluding the already implemented ScanMergerNode and CollisionAvoidanceNode.

4.2.1 Object detection implementation

Object detection is realised with the help of OpenVINO Toolkit (Open Visual Inference and Neural Network Optimization) developed by Intel as it has modular and powerful solutions for vision-based applications. The Toolkit also provides pre-trained models for several applications, including person detection as well [83]. We chose this solution because in the future we plan to use an Intel RGB-D camera.

Unfortunately, the toolkit does not support depth image segmentation with neural networks, so that part of the task was implemented using traditional techniques.

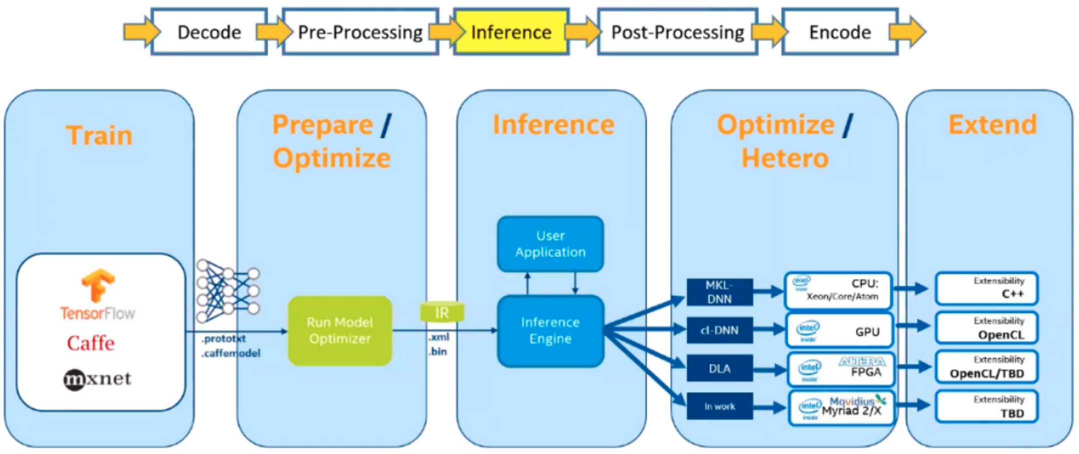


Figure 4.12: *Overview of OpenVINO Toolkit* [83]

4.2.1.1 OpenVINO Toolkit

OpenVINO toolkit is a comprehensive toolkit for quickly developing applications and solutions that solve a variety of tasks including emulation of human vision, automatic speech recognition, natural language processing, recommendation systems, and many others. Based on latest generations of artificial neural networks, including the already mentioned Convolutional Neural Networks (CNNs), recurrent and attention-based networks, the toolkit extends computer vision and non-vision workloads across Intel hardware, maximizing performance [83].

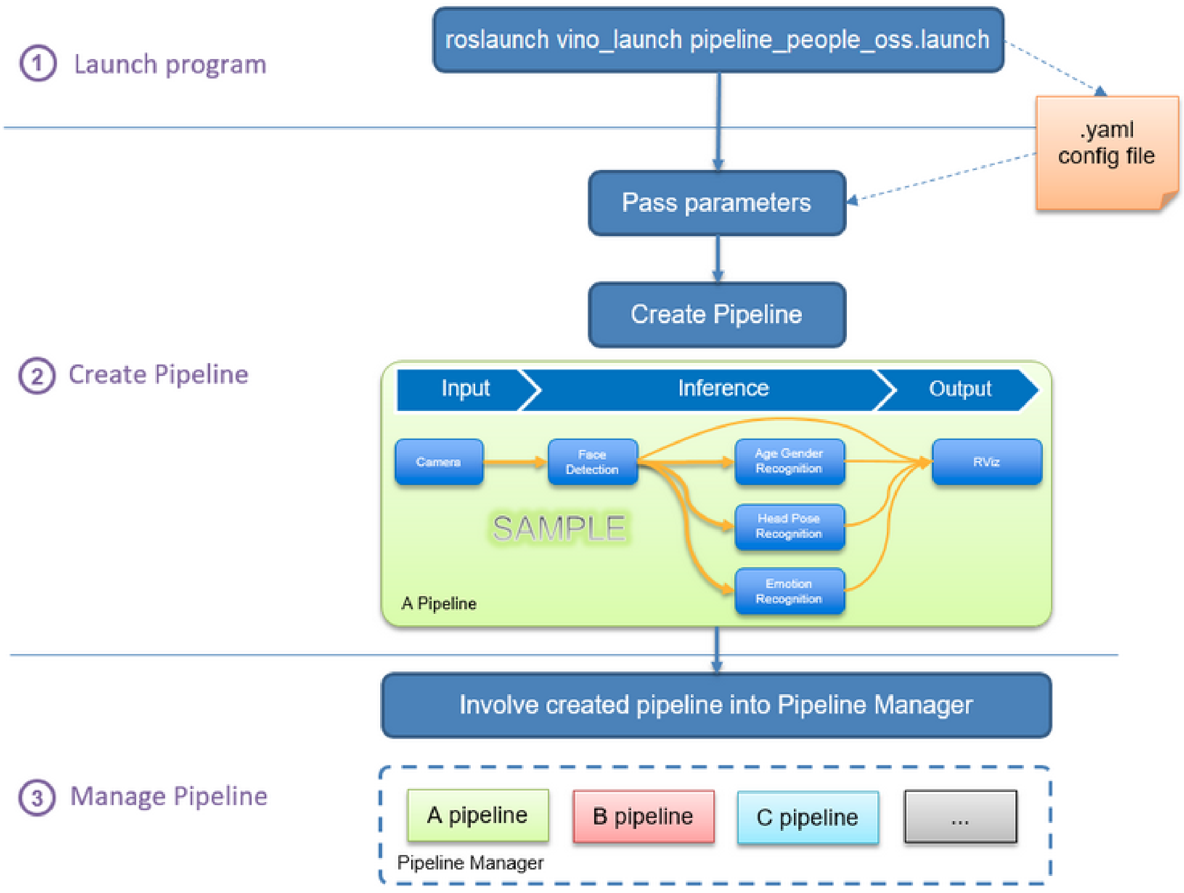


Figure 4.13: *Logic flow presented in ROS OpenVINO Toolkit* [84]

Figure 4.12 shows an overview of the toolkit. It is extremely versatile as it is able to use pre-trained models from almost all sources (Caffe, TensorFlow, MXNet etc.) using its Model Optimizer, which optimizes the model and converts it into its intermediate representation. Besides, the Inference Engine helps in the proper execution of the model on different devices such as CPU, GPU, FPGA and the Intel Movidius Neural Compute Stick [83].

Fortunately Intel created a ROS wrapper called ROS OpenVINO Toolkit, which integrates the capabilities of OpenVINO into ROS, which makes it easy to use in our system [84, 84]. In addition, it also provides a logic implementation, by introducing the definitions of parameter manager, pipeline and pipeline manager. Figure 4.13 depicts how these entities co-work together when the corresponding program is launched.

Considering the needs of the current application the toolkit is configured so that it takes an RGB camera topic as an input and performs person detection and re-identification with the help neural networks. The toolkit provides pre-trained networks for both purposes [84].

For person detection we chose a model that uses SSD, because it gives a good compromise between speed and accuracy, as it was discussed previously in Section 4.1.3. It takes the provided RGB image in a lower resolution (320x544) and gives the detected people with their bounding boxes, label and confidence [84].

The re-identification model takes a whole body image as an input and outputs an embedding vector to match a pair of images by the cosine distance. The model is based on the OmniScaleNet backbone developed for fast inference. A single re-identification head from the 1/16 scale feature map outputs an embedding vector of 256 floats [84].

All the outputs of the mentioned neural networks are available by subscribing to the corresponding topics and the results are visualized and an output image can be viewed using RViz.

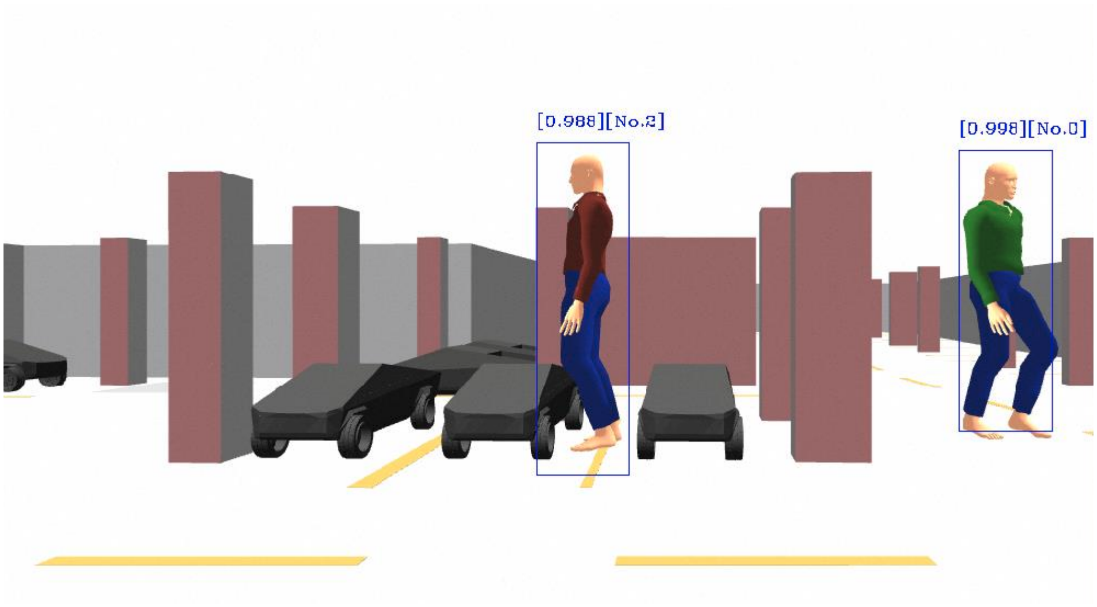


Figure 4.14: *Output of the neural networks visualised in RViz*

The output of the neural networks (inclouding the bounding boxes, the ID given during re-identification and the confidence value) can be seen in Figure 4.14.

4.2.1.2 Depth image segmentation

To utilise the extra information provided by the camera, in this step the position of the objects is specified with the help of the depth image.

Figure 4.15 shows the steps of this process segmentation. In the first part the mask of each object is created separately and in the final step these masks are merged together.

The creation of the mask involves the following phases: first, the depth image is cropped, with the help of ROIs (region of interests or bounding boxes - as they were referred earlier) which are provided by the neural network. Then the distance of the object is predicted. The prediction is based on the assumption, that the pixels in the middle of the bounding box are part of the objects and the predicted distance is the mean value of these pixels. After that, the cropped image is divided into clusters with the help of k-means. This step makes sure that the position of the pixels relative to each other is taken into account during segmentation. Finally, the mean value of the clusters is calculated and if this value is in a given range from the predicted one, the cluster is considered part of the object. Figure 4.16 shows the output mask image.

Depth to point cloud converter node Fortunately ROS provides several packages to manipulate data on ROS topics and one of them is called *depth image proc* which provides basic

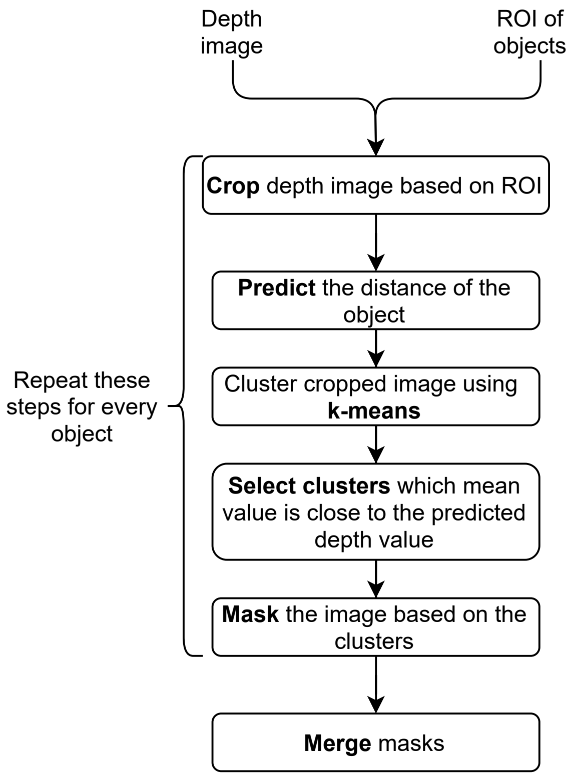

Depth image
ROI of objects
Crop depth image based on ROI
Predict the distance of the object
Cluster cropped image using k-means
Select clusters which mean value is close to the predicted depth value
Mask the image based on the clusters
Merge masks
Repeat these steps for every object

Figure 4.15: *Steps of segmentation*

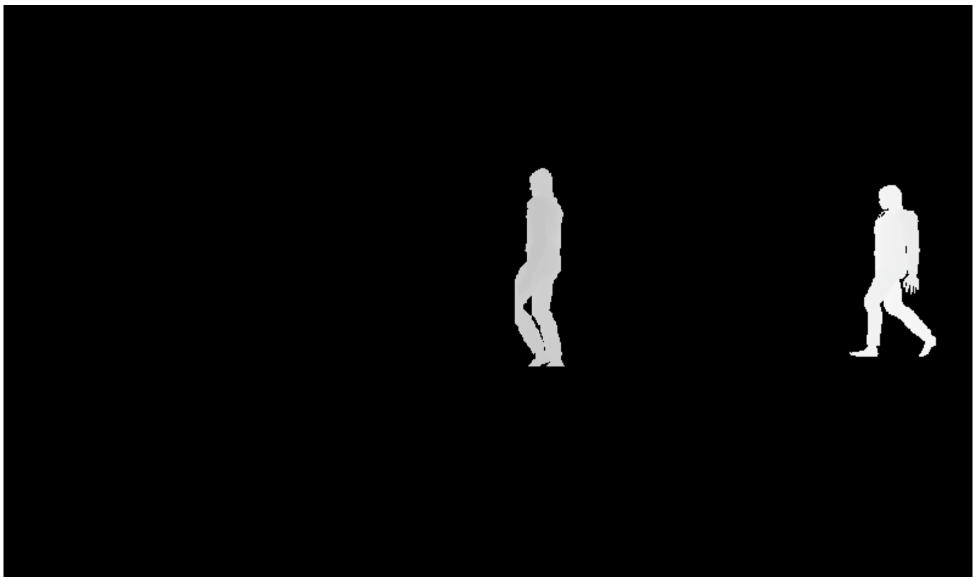
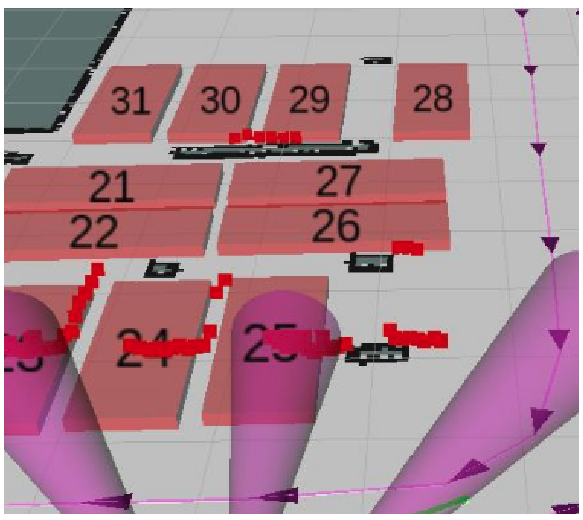
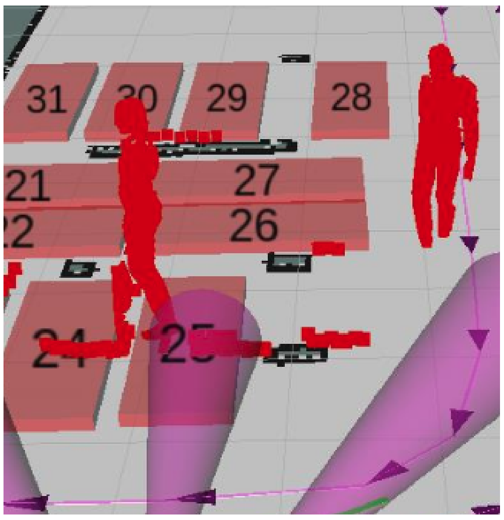


Figure 4.16: *Mask of the segmentad image*

processing for depth images. It includes converting the depth image to point cloud, so the application utilizes this package to provide *depth image proc* output to the previously mentioned ScanMerger node. The only configuring it needs is to provide the name of the input and output topics.



(a) The point cloud created from the LIDARs.



(b) Detected people become part of the point cloud.

Figure 4.17: *Output of ScanMerger before and after the implementation of dynamic detection*

In Figure 4.17, with red colour, the point cloud of the merged sets can be seen before and after the people are added. On the right you can see they contain the surface of the person, as it is seen from the view of the camera.

Chapter 5

Valet Parking System

The last chapter of our paper demonstrates the top level application of the system and shows the experimental results. Finally, opportunities for further improvements will be proposed.

5.1 Valet Parking Manager

Section 1.1 gave a general overview about the problem we aimed at solving, while Chapter 2, Chapter 3, and Chapter 4 described in detail the employed means of achieving it. This Section focuses on the Valet Parking Manager Node which is the integrator part of our system. It can be considered as a state machine, the state of which can be seen in Figure 5.1.

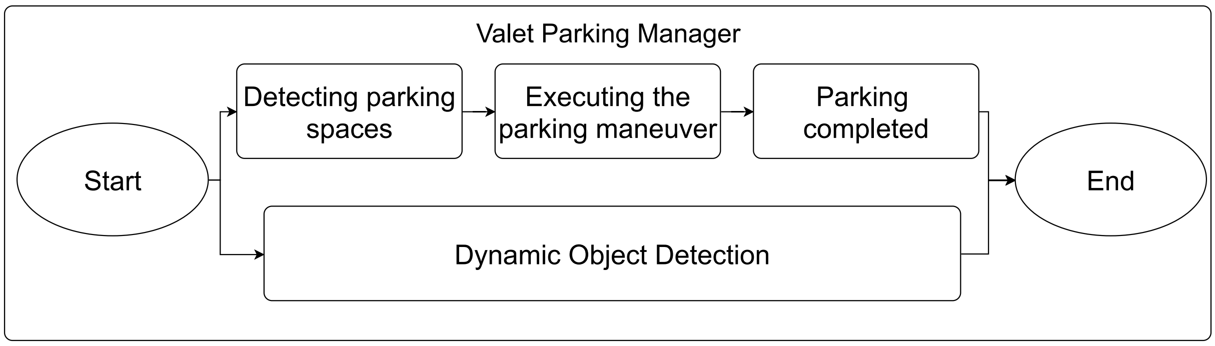


Figure 5.1: *Flowchart of Valet Parking Manager Node*

The initial position of the car is the driveway to the parking garage. Valet Parking Manager Node provides *2D Navigation Goals* which purpose is to ensure that the car is circling in the parking lot. These goals are stored in the same YAML file where the parking spaces are stored as well presented in Section 2.2.5. The following values are stored related to the navigation goals:

- **id:** An identification number that uniquely determines the navigation goal.
- **position:** 2D coordinates (x, y) of the goal.
- **orientation:** Orientation in quaternions (x, y, z, w). The reason for using quaternions is the same as explained in Section 2.2.5.

Similar to Parking Space Detection Node in Section 2.2.5, Valet Parking Manager Node reads this YAML file and interprets its content. If one of the data is corrupted (for example there is an invalid quaternion value), it warns the user and aborts the program. In this way it is ensured that navigation goals are properly set. All in all, there are a total of eight navigation goals, two-two in each corner of the parking garage, one before the corner and one after the corner. This is significant for the following reason. At first, a path is planned from the starting pose of the car to the first navigation goal which is designed by the Hybrid A* planner. A TrackGraph can be drawn on the map in advance which is taken into account when planning a path using Hybrid A*. This TrackGraph was introduced in Section 3.1.3.1. With proper tuning of the parameters, the planned path is nicely aligned with the lanes. As for curves in corners, it is especially important that the planned path does not deviate completely from the road we have defined in advance. That is the reason why there is a navigation goal before and after every corner as it can be seen in Figure 5.2.

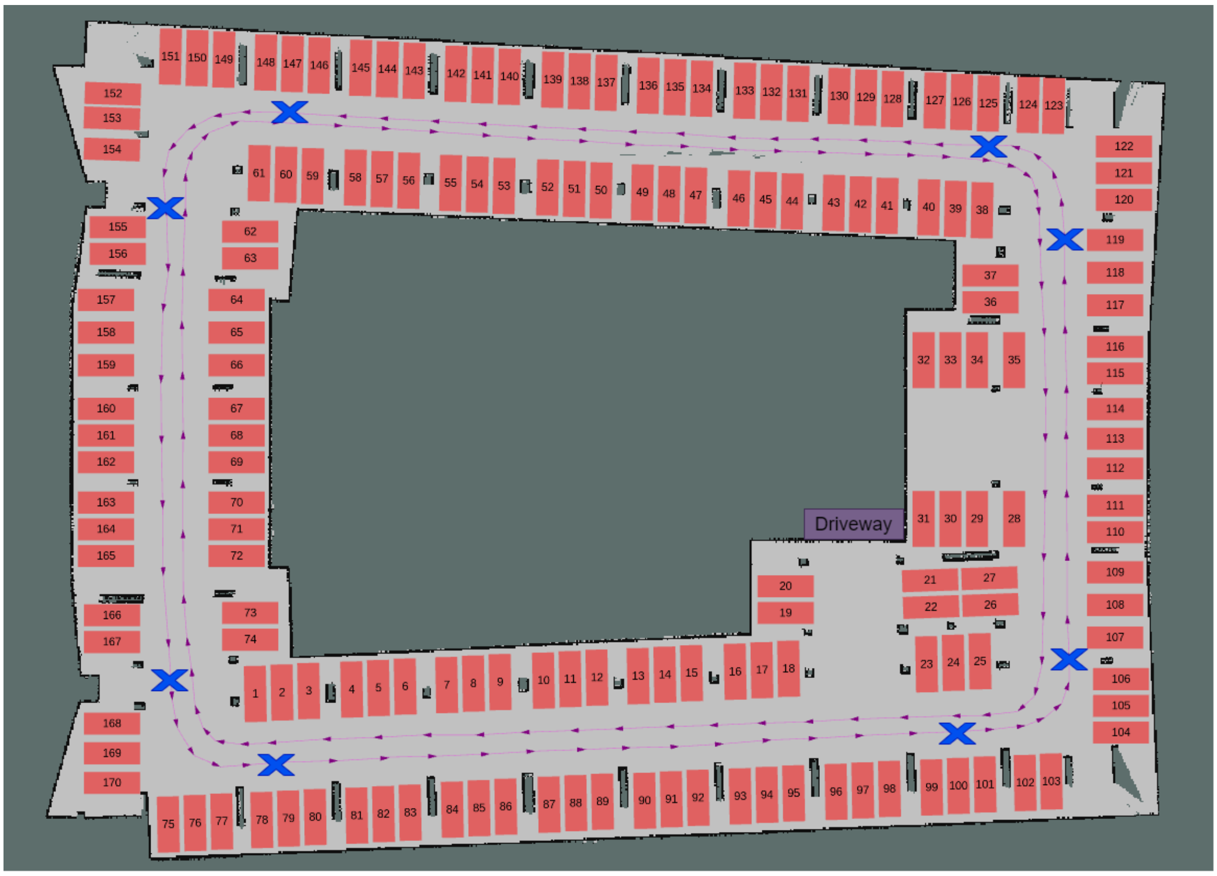
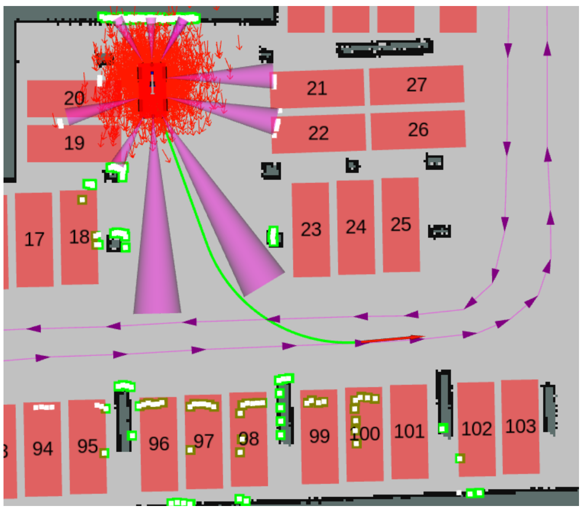
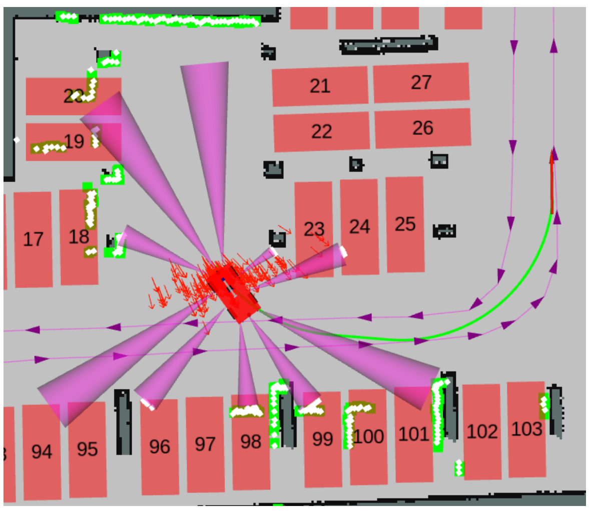


Figure 5.2: *The parking garage with the parking spaces. The purple box indicates the driveway from the upper level while the blue crosses denote the eight navigation goals. The small arrows constitute the TrackGraph.*

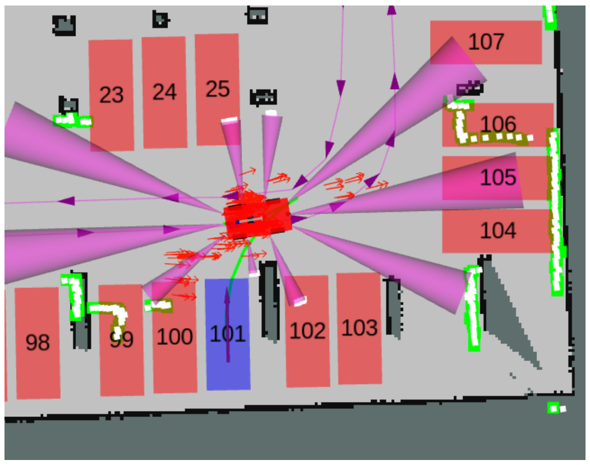
Simultaneously with circling, the car also detects the parking spaces and determines whether they are vacant or occupied. If the previously set navigation goal is approached, then the next goal is provided by Valet Parking Manager Node. If there is currently no vacant parking spots in the parking garage, the car will keep circling around the eight navigation goals until finding a vacant space. Once a vacant spot is detected, two things happen. First, if it is the only one free parking space in the close environment of the car, it will not just be a simple vacant spot, but also the preferred parking space denoted by Parking Space Detection Node. If there are multiple free spaces, then a preferred space will be selected as described in Section 2.2.5. Parking Space Detection Node sends the preferred parking space to Valet Parking Manager Node. Thereafter, Valet Parking Manager Node signals back to Parking Space Detection Node to turn off determining another preferred parking space. Second, it also changes the planner type from Hybrid A* to RTR+CCRS. This communication happens by using ROS Message and ROS Service, respectively. Once this change happened, the path to the parking space will be planned by RTR+CCRS, and the car executes the parking maneuver. Although the planned path must be followed by the car, the algorithm of the used Path Follower is out of the scope of the current work. We only note that the task of path following was solved by employing MPC (Model Predictive Control). While executing the parking maneuver, Valet Parking Manager compares the current pose of the car with the pose of the preferred space so it is known when the parking operation has just finished. Figure 5.3 demonstrates the above described operation of Valet Parking Manager.



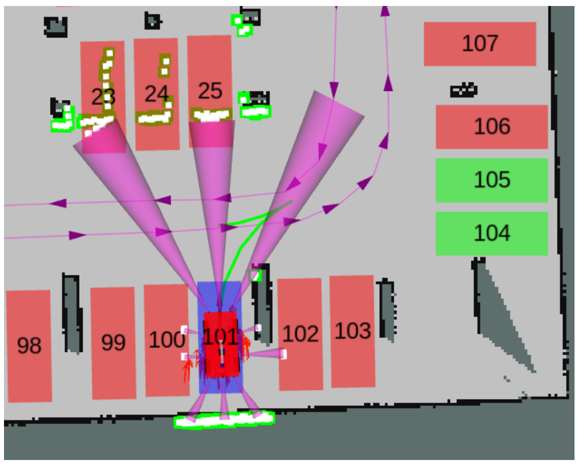
(a) Initial pose of the car and the planned path to the first navigation goal



(b) As the car gets close enough to the current goal, a new goal will be set if there is no preferred parking space



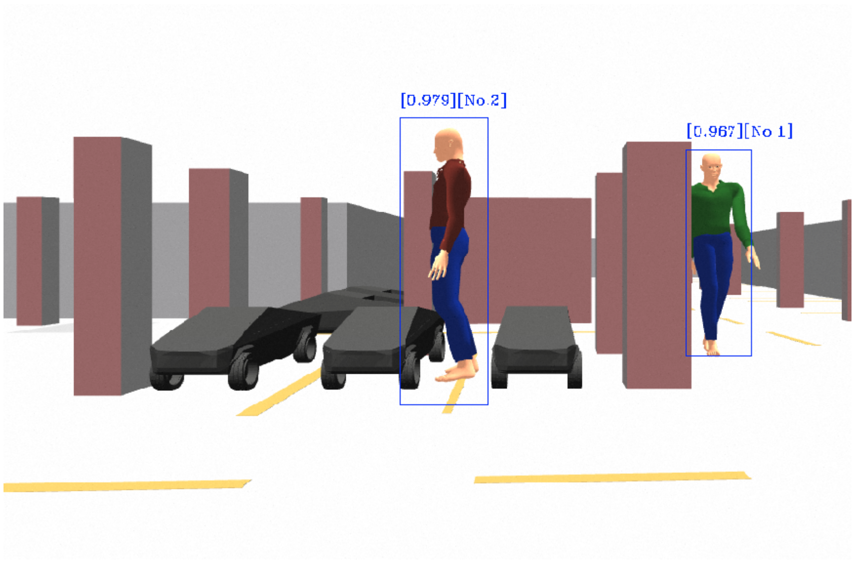
(c) If there is a preferred parking space, path planner will be changed from Hybrid A* to RTR+CCRS



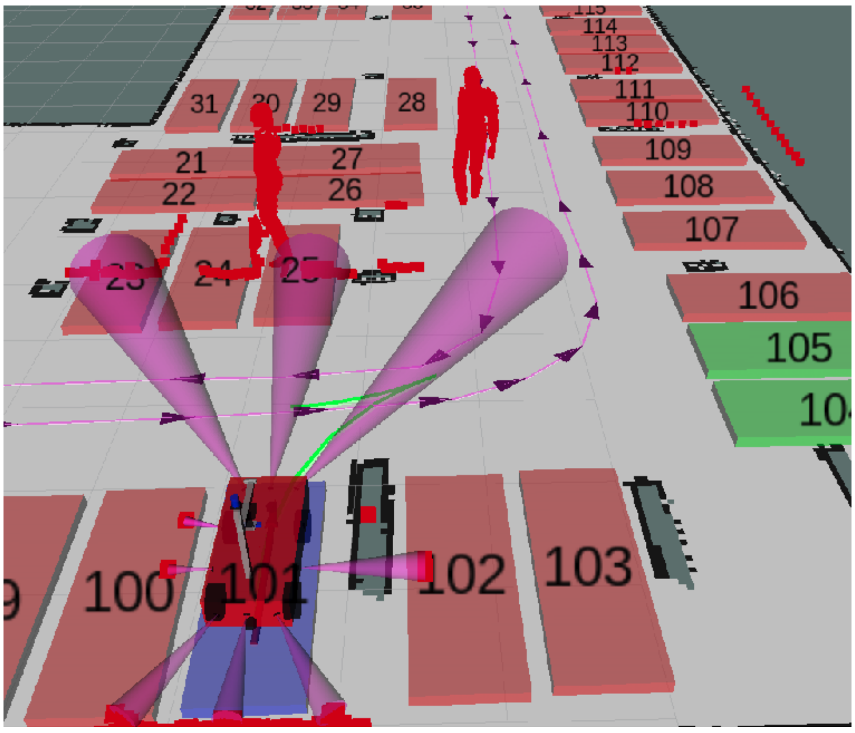
(d) Car parked to the preferred parking space and other parking spaces were successfully detected (spot 104 and 105) as well

Figure 5.3: *Valet Parking Manager Node in operation*

All along the process, people are detected and re-identified with the help of neural networks. Exploiting the extra depth data from the RGB-D camera a pointcloud containing the detected objects is also created. The collision avoidance system makes use of this pointcloud, while the other outputs are yet to be utilized in the process of improving the solution.



(a) The neural networks are able to detect and re-identify multiple people



(b) The detected people appear in the pointcloud as well

Figure 5.4: *Dynamic Object Detection in operation*

Figure 5.5 shows the simulated environment in Gazebo and also verifies the correct work of our system based on Figure 5.3 and Figure 5.4. It can be seen that parking spaces were correctly identified (for example spot 104 and 105 are vacant). Furthermore a standing and a walking person was detected as well.

It should also be noted that it can be influenced in advance how the car will pose itself within the parking spot. This is achieved by a ROS Parameter called *reverse_parking*. If it is set to true, then the back of the car will be posed towards the wall when parked. If it is set to false then its opposite happens. Also, parameters of the planners can be set in the same way. The most important ones for the Hybrid A*: length of the primitives, maximum deviation from the TrackGraph, weights of the distance from the graph points and the obstacle, and iteration limit. For RTR+CCRS planner, the most important is the maximum curvature change rate σ from Equation 3.8 which ensures that the path is feasible.

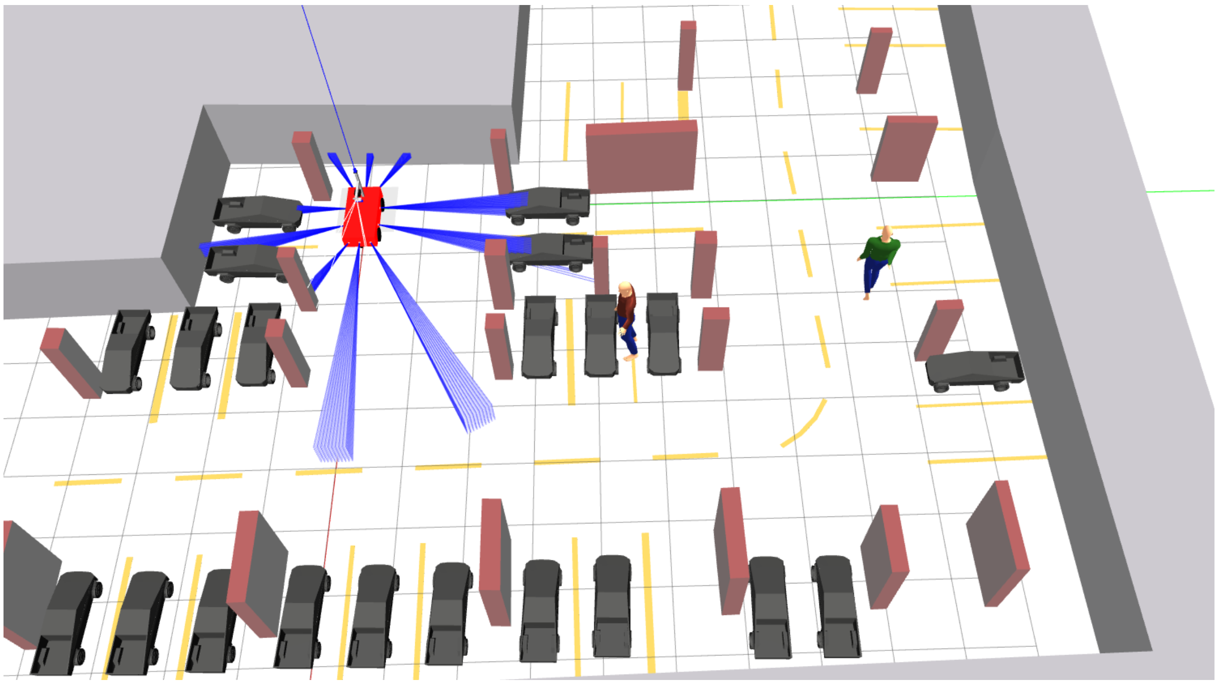


Figure 5.5: *The car in the simulated environment which was modeled by Gazebo*

Our progress can be followed at a YouTube channel[1] where we also demonstrate what could be seen in Figure 5.3 and Figure 5.4 in a form of a video.

5.2 Future Improvements

In the current implementation, a map was generated as described in Section 2.2.1, made available by the Map Server presented in Section 2.2.1.1 and the car was localized by employing AMCL explained in Section 2.2.2. We are aware of an issue, that since only localization is currently happening, the static map is never updated, therefore the cars in the detected occupied parking spaces will not be appeared in the real map. To overcome this hurdle, we propose to use the SLAM algorithm called Cartographer. As explained in Section 2.2.1, compared to GMapping, Cartographer is capable of adapting to the fast changes in the dynamic environment.

As in the case of path planners, there is a recent paper about further increasing the smoothness of the path [85]. It is achieved by introducing an upper limit not only to the maximum curvature and maximum curvature rate but for the maximum curvature acceleration as well. This approach fully satisfies the dynamic constraints of the steering system therefore results

[1] `https://www.youtube.com/channel/UCethKCG_CuYYY_Mb5pV8jIg`

in a better tracking performance. This local planner could be used as an improvement of the CCRS.

It can be noticed in Figure 5.4a that the people in the simulated environment are not scaled properly. We tried to change it with a couple of programs already, but due to their animation the simulator could no longer load it. In the future, we would like to align their sizes with the rest of simulated objects.

As for Dynamic Object Detection, there are plenty ways for improvement. By having the depth data and the detected people their movements can be predicted which information can be utilized by the path planner. Apart from person detection the current method can easily be used to detect other object, for example cars or road signs. With these information more complex decisions can be made.

Our team is already developing a 1:3 scaled model car at our department which is equipped with all the sensors used in the simulation. The ultimate goal is to operate the system on this real robot car in the future.

Bibliography

[1] J. Nyambal and R. Klein, “Automated Parking Space Detection using Convolutional Neural Networks,” in *2017 Pattern Recognition Association of South Africa and Robotics and Mechatronics (PRASA-RobMech)*, pp. 1–6, 2017.

[2] J. K. Suhr and H. G. Jung, “Automatic Parking Space Detection and Tracking for Underground and Indoor Environments,” *IEEE Transactions on Industrial Electronics*, vol. 63, no. 9, pp. 5687–5698, 2016.

[3] C. W. Hsu, C. F. Lin, C. Yao, M. K. Ko, and K. J. Chang, *Development of Full-Ultrasonic Positioning and Multi-Turn Control for Advanced Parking Guidance System in Parallel Parking*. PhD thesis, Thesis. China, 2010.

[4] Wan-Joo Park, Byung-Sung Kim, Dong-Eun Seo, Dong-Suk Kim, and Kwae-Hi Lee, “Parking Space Detection using Ultrasonic Sensor in Parking Assistance System,” in *2008 IEEE Intelligent Vehicles Symposium*, pp. 1039–1044, 2008.

[5] ROS Documentation, “Introduction.” `http://wiki.ros.org/ROS/Introduction`. Online; Accesssed on 28 October 2020.

[6] ROS Documentation, “Creating a ROS Package.” `http://wiki.ros.org/ROS/Tutorials/CreatingPackage`. Online; Accesssed on 28 October 2020.

[7] ROS Documentation, “Understanding ROS Nodes.” `http://wiki.ros.org/ROS/Tutorials/UnderstandingNodes`. Online; Accesssed on 28 October 2020.

[8] ROS Documentation, “ROS Master.” `http://wiki.ros.org/Master`. Online; Accesssed on 28 October 2020.

[9] ROS Documentation, “Understanding ROS Topics.” `http://wiki.ros.org/ROS/Tutorials/UnderstandingTopics`. Online; Accesssed on 28 October 2020.

[10] ROS Documentation, “Creating ROS Messages and Services.” `http://wiki.ros.org/ROS/Tutorials/CreatingMsgAndSrv`. Online; Accesssed on 28 October 2020.

[11] ROS Documentation, “RViz Summary.” `http://wiki.ros.org/rviz`. Online; Accesssed on 28 October 2020.

[12] Open Source Robotics Foundation, “Gazebo.” `http://gazebosim.org/`. Online; Accesssed on 26 October 2020.

[13] N. Bibi, M. N. Majid, H. Dawood, and P. Guo, “Automatic Parking Space Detection System,” in *2017 2nd International Conference on Multimedia and Image Processing (ICMIP)*, pp. 11–15, 2017.

[14] F. Abad, R. Bendahan, S. Wybo, S. Bougnoux, C. Vestri, and T. Kakinami, “Parking Space Detection,” *14th World Congress on Intelligent Transport Systems, ITS 2007*, vol. 2, 2007.

[15] H. Huang, Y. He, and F. Lin, “A Vehicle Transverse Automatic Parking Auxiliary System,” in *2015 International Conference on Machine Learning and Cybernetics (ICMLC)*, vol. 2, pp. 789–792, 2015.

[16] M. A. Fairus, S. N. S. Salim, Irma Wani Jamaludin, and M. Nizam Kamarudin, “Development of an Automatic Parallel Parking System for Nonholonomic Mobile Robot,” in *International Conference on Electrical, Control and Computer Engineering 2011 (InECCE)*, pp. 45–49, 2011.

[17] F. Bormann, E. Braune, and M. Spitzner, “The C2000 Autonomous Model Car,” in *4th European Education and Research Conference (EDERC 2010)*, pp. 200–204, 2010.

[18] H. Ahmad, A. Khan, W. Noor, G. Sikander, and S. Anwar, “Ultrasonic Sensors Based Autonomous Car Parking System,” in *Professional Trends in Industrial and Systems Engineering (PTISE)*, 2018.

[19] ROS Documentation, “Map Server Package Summary.” `http://wiki.ros.org/map_server`. Online; Accesssed on 28 October 2020.

[20] ROS Documentation, “AMCL Package Summary.” `http://wiki.ros.org/amcl`. Online; Accesssed on 28 October 2020.

[21] S. Thrun, D. Fox, and W. Burgard, *Probabilistic Robotics*. MIT Press, 2005.

[22] H. Barki, F. Denis, and F. Dupont, “A New Algorithm for the Computation of the Minkowski Difference of Convex Polyhedra,” in *2010 Shape Modeling International Conference*, pp. 206–210, 2010.

[23] Z. R. Gabidullina, “The Minkowski Difference for Convex Polyhedra and Some its Applications,” *arXiv: Mathematics - Optimization and Control*, 2019.

[24] P. Lindemann, “The Gilbert-Johnson-Keerthi distance algorithm,” *Algorithms in Media Informatics*, 2009.

[25] William Bittle, “GJK (Gilbert–Johnson–Keerthi).” `http://www.dyn4j.org/2010/04/gjk-gilbert-johnson-keerthi/`. Online; Posted on 13 April 2010; Accesssed on 28 October 2020.

[26] F. Zheng, A. R. Simpson, A. C. Zecchin, and J. W. Deuerlein, “A graph decomposition-based approach for water distribution network optimization,” *Water Resources Research*, pp. 2093–2109, 2013.

[27] X. Cui and H. Shi, “Direction oriented pathfinding in video games,” *International Journal of Artificial Intelligence and Applications*, vol. 2, 10 2011.

[28] R. Mosayebi and F. Bahrami, “A modified particle swarm optimization algorithm for parameter estimation of a biological system,” *Theoretical Biology and Medical Modelling*, 2018.

[29] E. W. Dijkstra, “A note on two problems in connexion with graphs,” *Numer. Math.*, vol. 1, p. 269–271, Dec. 1959.

[30] P. E. Hart, N. J. Nilsson, and B. Raphael, “A formal basis for the heuristic determination of minimum cost paths,” *IEEE Transactions on Systems Science and Cybernetics*, vol. 4, no. 2, pp. 100–107, 1968.

[31] A. Stentz, “The d* algorithm for real-time planning of optimal traverses,” 04 2011.

[32] Y. K. Ever, “Using simplified swarm optimization on path planning for intelligent mobile robot,” *Procedia Computer Science*, pp. 83 – 90, 2017. 9th International Conference on Theory and Application of Soft Computing, Computing with Words and Perception, ICSCCW 2017, 22-23 August 2017, Budapest, Hungary.

[33] Yanrong Hu and S. X. Yang, “A knowledge based genetic algorithm for path planning of a mobile robot,” in *IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA ’04. 2004*, vol. 5, pp. 4350–4355 Vol.5, 2004.

[34] M. Alajlan, A. Koubâa, I. Châari, H. Bennaceur, and A. Ammar, “Global path planning for mobile robots in large-scale grid environments using genetic algorithms,” in *2013 International Conference on Individual and Collective Behaviors in Robotics (ICBR)*, pp. 1–8, 2013.

[35] V. Kroumov and J. Yu, *Neural Networks Based Path Planning and Navigation of Mobile Robots*. 2011.

[36] D. Kiss and G. Tevesz, “Autonomous path planning for road vehicles in narrow environments: An efficient continuous curvature approach,” *Journal of Advanced Transportation, Hindawi*, 2017.

[37] D. Dolgov and S. Thrun, “Autonomous driving in semi-structured environments: Mapping and planning,” in *Proceedings of the 2009 IEEE International Conference on Robotics and Automation (ICRA-09)*, (Kobe, Japan), 2009. To Appear.

[38] S. Craw, *Manhattan Distance*, pp. 790–791. Springer US, 2017.

[39] *Path plannin - grid based search*, accessed October 23, 2020. https://pythonrobotics.readthedocs.io/en/latest/modules/path_planning.html.

[40] M. Montemerlo, J. Becker, S. Bhat, H. Dahlkamp, D. Dolgov, S. Ettinger, D. Haehnel, T. Hilden, G. Hoffmann, B. Huhnke, D. Johnston, S. Klumpp, D. Langer, A. Levandowski, J. Levinson, J. Marcil, D. Orenstein, J. Paefgen, I. Penny, A. Petrovskaya, M. Pflueger, G. Stanek, D. Stavens, A. Vogt, and S. Thrun, “Junior: The stanford entry in the urban challenge,” *Journal of Field Robotics*, 2008.

[41] D. Dolgov, S. Thrun, M. Montemerlo, and J. Diebel, “Practical search techniques in path planning for autonomous driving,” *AAAI Workshop - Technical Report*, 01 2008.

[42] D. Dolgov, S. Thrun, M. Montemerlo, and J. Diebel, "Path planning for autonomous vehicles in unknown semi-structured environments," *I. J. Robotic Res.*, vol. 29, pp. 485–501, 04 2010.

[43] P. Giordano, M. Vendittelli, J.-P. Laumond, and P. Souères, "Nonholonomic distance to polygonal obstacles for a car-like robot of polygonal shape," *Robotics, IEEE Transactions on*, vol. 22, pp. 1040 – 1047, 11 2006.

[44] L. E. Dubins, "On curves of minimal length with a constraint on average curvature, and with prescribed initial and terminal positions and tangents," *American Journal of Mathematics*, vol. 79, no. 3, pp. 497–516, 1957.

[45] J. A. Reeds and L. A. Shepp, "Optimal paths for a car that goes both forwards and backwards.," *Pacific J. Math.*, vol. 145, no. 2, pp. 367–393, 1990.

[46] J.-D. Boissonnat, A. Cérézo, and J. Leblond, "Shortest paths of bounded curvature in the plane.," pp. 3–18, 01 1991.

[47] B. J. Souères P., "Optimal trajectories for nonholonomic mobile robots.," in *Robot Motion Planning and Control. Lecture Notes in Control and Information Sciences*, pp. 91–170, 1998.

[48] K. Komoriya and K. Tanie, "Trajectory design and control of a wheel-type mobile robot using b-spline curve," *The Autonomous Mobile Robots and Its Applications*, pp. 398–405, 1989.

[49] T. Fraichard and A. Scheuer, "From reeds and shepp's to continuous-curvature paths," 01 2004.

[50] R. Liscano and D. Green, "Design and implementation of a trajectory generator for an indoor mobile robot," *The Autonomous Mobile Robots and Its Applications*, pp. 380–385, 1989.

[51] A. Piazzi, C. Guarino Lo Bianco, M. Bertozzi, A. Fascioli, and A. Broggi, "Quintic g2-splines for the iterative steering of vision-based autonomous vehicles," *Intelligent Transportation Systems, IEEE Transactions on*, vol. 3, pp. 27 – 36, 04 2002.

[52] D. K. Wilde, "Computing clothoid segments for trajectory generation," *In Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 2440 – 2445, 2009.

[53] T. Fraichard and A. Scheuer, "From reeds and shepp's to continuous-curvature paths," 01 2004.

[54] P. Dávid, "Önvezető autók pályatervezése szűk környezetben, folytonos görbületű pályaelemekkel," 2016.

[55] N. Correll, *Introduction to Autonomous Robots*, v1.9, March 6, 2020. `https://github.com/correll/Introduction-to-Autonomous-Robots/releases`.

[56] *High accuracy calculation of Fresnel-Integrals*, accessed October 20, 2020. `https://keisan.casio.com/exec/system/1180573479`.

[57] S. Sirouspour, “Advanced engineering and computational methodologies for intelligent mechatronics and robotics,” pp. 1–363, 2013.

[58] K. András, “Objektum felismerés autonóm jármű esetén [object detection for autonomous vehicle],” 2019.

[59] N. O’Mahony, S. Campbell, A. Carvalho, S. Harapanahalli, G. V. Hernandez, L. Krpalkova, D. Riordan, and J. Walsh, “Deep learning vs. traditional computer vision,” in *Advances in Computer Vision* (K. Arai and S. Kapoor, eds.), (Cham), pp. 128–144, Springer International Publishing, 2020.

[60] J. Kocić, N. Jovičić, and V. Drndarević, “Sensors and sensor fusion in autonomous vehicles,” in *2018 26th Telecommunications Forum (TELFOR)*, pp. 420–425, 2018.

[61] S. Sivaraman and M. M. Trivedi, “A review of recent developments in vision-based vehicle detection,” in *2013 IEEE Intelligent Vehicles Symposium (IV)*, pp. 310–315, 2013.

[62] A. Mukhtar, L. Xia, and T. B. Tang, “Vehicle detection techniques for collision avoidance systems: A review,” *IEEE Transactions on Intelligent Transportation Systems*, vol. 16, no. 5, pp. 2318–2338, 2015.

[63] A. A. Ali and H. A. Hussein, “Distance estimation and vehicle position detection based on monocular camera,” in *2016 Al-Sadeq International Conference on Multidisciplinary in IT and Communication Science and Applications (AIC-MITCSA)*, pp. 1–4, 2016.

[64] A. S. Huang, A. Bachrach, P. Henry, M. Krainin, D. Maturana, D. Fox, and N. Roy, *Visual Odometry and Mapping for Autonomous Flight Using an RGB-D Camera*, pp. 235–252. Cham: Springer International Publishing, 2017.

[65] H. Peng, B. Li, W. Xiong, W. Hu, and R. Ji, “Rgbd salient object detection: A benchmark and algorithms,” in *Computer Vision – ECCV 2014* (D. Fleet, T. Pajdla, B. Schiele, and T. Tuytelaars, eds.), (Cham), pp. 92–109, Springer International Publishing, 2014.

[66] I. R. Ward, H. Laga, and M. Bennamoun, *RGB-D Image-Based Object Detection: From Traditional Methods to Deep Learning Techniques*, pp. 169–201. Cham: Springer International Publishing, 2019.

[67] M. Szemenyei, “Lecture notes in computer vision systems [számítógépes látórendszerek jegyzet],” 2020.

[68] R. Lienhart and J. Maydt, “An extended set of haar-like features for rapid object detection,” in *Proceedings. International Conference on Image Processing*, vol. 1, pp. I–I, 2002.

[69] P. F. Felzenszwalb, R. B. Girshick, D. McAllester, and D. Ramanan, “Object detection with discriminatively trained part-based models,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 32, no. 9, pp. 1627–1645, 2010.

[70] U. Karn, “An intuitive explanation of convolutional neural networks.” `https://ujjwalkarn.me/2016/08/11/intuitive-explanation-convnets/`, 2016.

[71] Z. Zhao, P. Zheng, S. Xu, and X. Wu, "Object detection with deep learning: A review," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 30, no. 11, pp. 3212–3232, 2019.

[72] U. Karn, "A quick introduction to neural networks." `https://ujjwalkarn.me/2016/08/09/quick-intro-neural-networks/`, 2016.

[73] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," 2014.

[74] R. Girshick, "Fast r-cnn," in *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, December 2015.

[75] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," in *Advances in Neural Information Processing Systems 28* (C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, eds.), pp. 91–99, Curran Associates, Inc., 2015.

[76] K. He, X. Zhang, S. Ren, and J. Sun, "Spatial pyramid pooling in deep convolutional networks for visual recognition," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 37, no. 9, pp. 1904–1916, 2015.

[77] R. Gandhi, "R-cnn, fast r-cnn, faster r-cnn, yolo — object detection algorithms." `https://towardsdatascience.com/r-cnn-fast-r-cnn-faster-r-cnn-yolo-object-detection-algorithms-36d53571365e`, 2018.

[78] C. Cortes and V. Vapnik, "Support vector machine," vol. 20, no. 3, p. 273–297, 1995.

[79] K. Sinhal and A. Sachan, "Zero to hero: Guide to object detection using deep learning: Faster r-cnn,yolo,ssd." `https://cv-tricks.com/object-detection/faster-r-cnn-yolo-ssd/`, 2019.

[80] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," 2016.

[81] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, "Ssd: Single shot multibox detector," *Lecture Notes in Computer Science*, p. 21–37, 2016.

[82] J. Hui, "Object detection: speed and accuracy comparison (faster r-cnn, r-fcn, ssd, fpn, retinanet and yolov3)." `https://medium.com/@jonathan_hui/object-detection-speed-and-accuracy-comparison-faster-r-cnn-r-fcn-ssd-and-yolo-5425656ae359`, 2018.

[83] "Openvino toolkit." `https://docs.openvinotoolkit.org/2018_R5/index.html`. Accessed: 2020-08-25.

[84] "Ros openvino toolkit." `https://github.com/intel/ros_openvino_toolkit`. Accessed: 2020-09-01.

[85] H. Banzhaf, N. Berinpanathan, D. Nienhüser, and J. M. Zöllner, "From g2 to g3 continuity: Continuous curvature rate steering functions for sampling-based nonholonomic motion planning," *2018 IEEE Intelligent Vehicles Symposium (IV)*, pp. 326–333, 2018.