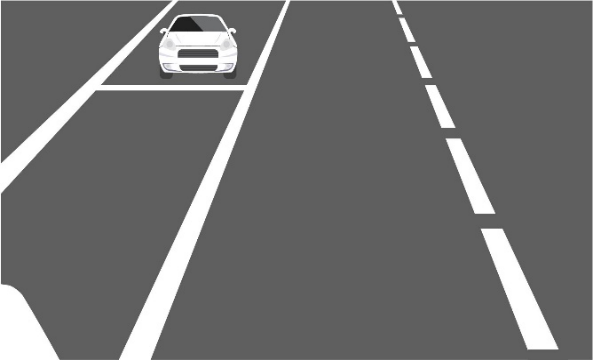
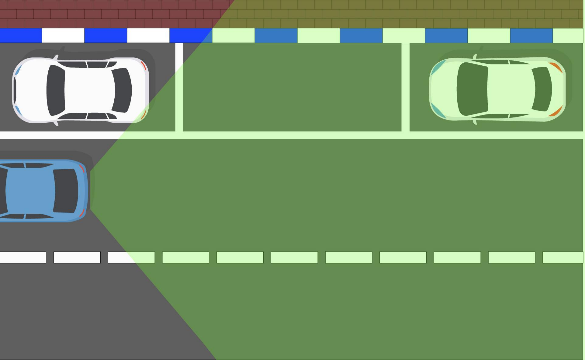


Software Engineering Department

Ort Braude College

Course 61771: Extended Project in Software Engineering

**Car parking assistance system using reverse camera**



In Partial Fulfillment of the Requirements for

Final Project in Software Engineering (Course 61771)

Karmiel – July 2019

Ahmad Saeed 313425118

George Hanna 203133673

**Supervisor(s)**:

Dr. Zeev Frenkel

Contents

1. Introduction 1
2. Theory 2
   1. History 2
   2. Existing techniques and algorithms 2
   3. Basic idea 3
   4. Algorithm 3
   5. Recurrent algorithm 7
3. Software engineering documents 7
   1. Use Case 7
   2. GUI 11
   3. Design 14
   4. Testing 14
4. Results and conclusions 15
   1. Results 15
      1. Calculation of position on the map 15
      2. Driving maneuvers in simple situations 16
      3. Study the number of maneuvers 18
   2. Conclusions 19

References 19

1. **INTRODUCTION**

Rear camera can provide more information than mirrors. It can help to plan safe maneuvers by considering information on vehicles going from the back. It is especially useful in backward movement when area next to the car from the back is about invisible from the driver place.

In regular road backward movement can be not very accurate. In the parking zones room is usually quite restricted. Movements in such places should be very careful and accurate. Such maneuvers are difficult for not enough trained drivers. They can’t estimate how need to drive car in such complicated situations.

Modern cars have quite good solutions for these problems, including about fully automated parking systems. Indeed, a lot of people still prefer cheaper older cars where installation of such systems is quite problematic and expensive.

Nevertheless, simple cameras with Wi-Fi/Bluetooth are cheap and it is very simple to install it. Now about everybody have smartphone, iPhone or similar device with large enough screen, powerful computer and possibility to connect with external devices by various ways. Hence, it is very simple to get real time picture from rear camera to smartphone. In contrast to simple mirror, iPhone can make calculations. Recommendations based on such calculations can be provided for driver in very intuitive form. Our project is concentrated on construction of Parking assistance application based on this idea.

The most important parameters for such system are:

1. Parking methods (Angle Parking, Perpendicular Parking, Parallel Parking).
2. Car measurements (Length, Width).
3. Measuring distances between cars to sidewalk (Distance between our car and the other cars,

Distance between our car and the sidewalk).

In our project the application installed on the driver’s smartphone estimates distances by using changes of the sizes of various objects in the images taken by the rear camera. Such changes are resulted by changing of view angle. The distances are calculated based on theorems of projection geometry [1].

Information on distances and angles enable to reconstruct a geographic map of situation on the parking zone as well as estimate position and orientation of driving car.

To calculate optimal driving for the car we first calculate optimal car trajectory. Such trajectory is searched with considering estimated positions of other objects in the parking zone, sizes of driving car, maximal curvature of trajectory and what wheels are driving. We formulate the problem of optimal trajectory searching in terms of shortest path searching in the net of all possible car states (defined by two coordinates of center and by the azimuth angle). In such formulation the problem can be solved by Dijkstra’s algorithm [2]. Based on the resulting optimal trajectory our application generates recommendations for the driving. It is also controlling the correspondence of real car moves to proposed ones. If need, application in real time regime recalculate the trajectory and propose the driving with maximization of similarity to on (to protect driver from the confuse). To test effectiveness of our algorithm we develop computer simulator for our system. develop user friendly interface for application and simulator to test it.

1. **THEORY**
   1. **History**

Intelligent Parking Assist System started to surface back in 1992 by Volkswagen. Volkswagen proposed an automatic parking technology by steering all four wheels, allowing the car to move sideways for parallel parking, though they did not announce it or ran or publish the idea. German Aerospace Center took the four-wheel steering idea and used it in their electric car “Robo mobile”. The idea was that the car can rotate its four wheels in the perpendicular direction to prepare itself for the sideward motion [3].

In 2003, the hybrid vehicle Toyota Prius was released by Toyota, which included an optional automatic parallel parking system that was called “Intelligent Parking Assist” [4].

In 2004, the Evolve car was developed by a group of Linkoping University students, who worked with Volvo. The Evolve car could perform parallel parking automatically using sensors and a computer to control the acceleration, braking and steering of the Volvo S60 [3].

In 2006, self-parking system was added to Lexus LS sedan by Lexus. The car could parallel park itself automatically and perform perpendicular parking as well [3].

in 2009, the Active Park Assist was introduced by Ford for their Lincoln models [5].

In 2010, A system called “Parking Assistant” was introduced by BMW on their redesigned 5 series to perform parallel parking [6].

Over the years, parking assist systems kept improving, until fully automated car parking systems were introduced, a revolutionary system that can help the uninitiated driver, disabled people or amateur drivers to park their car perfectly [7, 8]. In the past decades many companies are developing Automated Car Driving, not only Automated Parking [9-15]. The main reasons that Automated Parking was the first product in this field are demand and simplicity to provide safety: parking manures are sometimes complicated for the driver, and even in case of a slight malfunction, the system is safe enough because of low speed in parking manures [7, 12, 14].

* 1. **Existing techniques and algorithms**

Existing Intelligent Parking Assist Systems (IPAS) use several devices to function properly: sensors of distance, cameras, sometimes hydroscopes, GPS, tachograph etc. [10, 12]. Typically, intelligent parking systems use normal computer processors. These processors are connected to the car’s warning system. The warning system is fed with data from various devices. That way, when a sensor detects a near object, it triggers the car’s warning system in order to alert the driver [3, 6, 7, 11, 13]. In most of technologies, sensors are installed only in the back bumper of a car [3]. By using ultrasonic technology, it can measure the distances between the car and its surrounding objects. The moment the driver shifts into reverse gear, a warning beeping sound is sounded, which gets faster as the car gets closer to an object [8]. Another crucial element in parking systems, is the “Rear View Camera” [3]. Combing cameras vision with ultrasonic sensors, the drivers can get more accurate image of what’s happening in the “Dead Zone” of the car. Usually, drivers can see markings over the video feed by the rear-view camera to help them steer into the parking spot perfectly. Typically, mounting a rear-view camera meant installing a video screen on the dashboard inside the car, until the whole system of cameras, sensors and screens were introduced as standard equipment in modern cars [7, 8].

The first version of park assist system was a bit limited: it faces difficulties detecting small objects, or any other object that is smaller than a regular car [7]. Over the years, parking systems became smarter, hence, parking sensors became more accurate than before, data from the parking sensors is used to steer the car in perfect turn angles to perform a parallel parking [7, 8]. All the data collected by the sensors help the computer calculate distances and sound an alarm in case of a close object, is displayed on a touch screen, mounted on the dashboard. cameras resolution improved; they became capable of recognizing the markings on the streets [8]. Nowadays, Intelligent Parking Assist Systems can notify the driver when it detects a compatible parking spot. Some assist systems can steer the car autonomously, but the driver must control the brakes, clutch and accelerator. More improved versions of the systems have control over everything in the car [3, 7, 8, 11, 14]. Safety of the systems is provided by automated stop if sensors detect unexpected obstacles [12, 14].

* 1. **Basic idea**

As we mentioned in introduction, the main idea of our project is to use smartphone application and cheap camera in situation where standard parking assistance systems are absent. Connection between camera and smartphone is provided via Bluetooth. Standard cameras can detect items in a 90-165 degrees angle range, and the smartphone can show a real-time video with instructions to help a person park his car. These instructions are calculated using image processing that we will discuss in the next subsection.

* 1. **Algorithm**

Let’s say we have this kind of situation on the street:

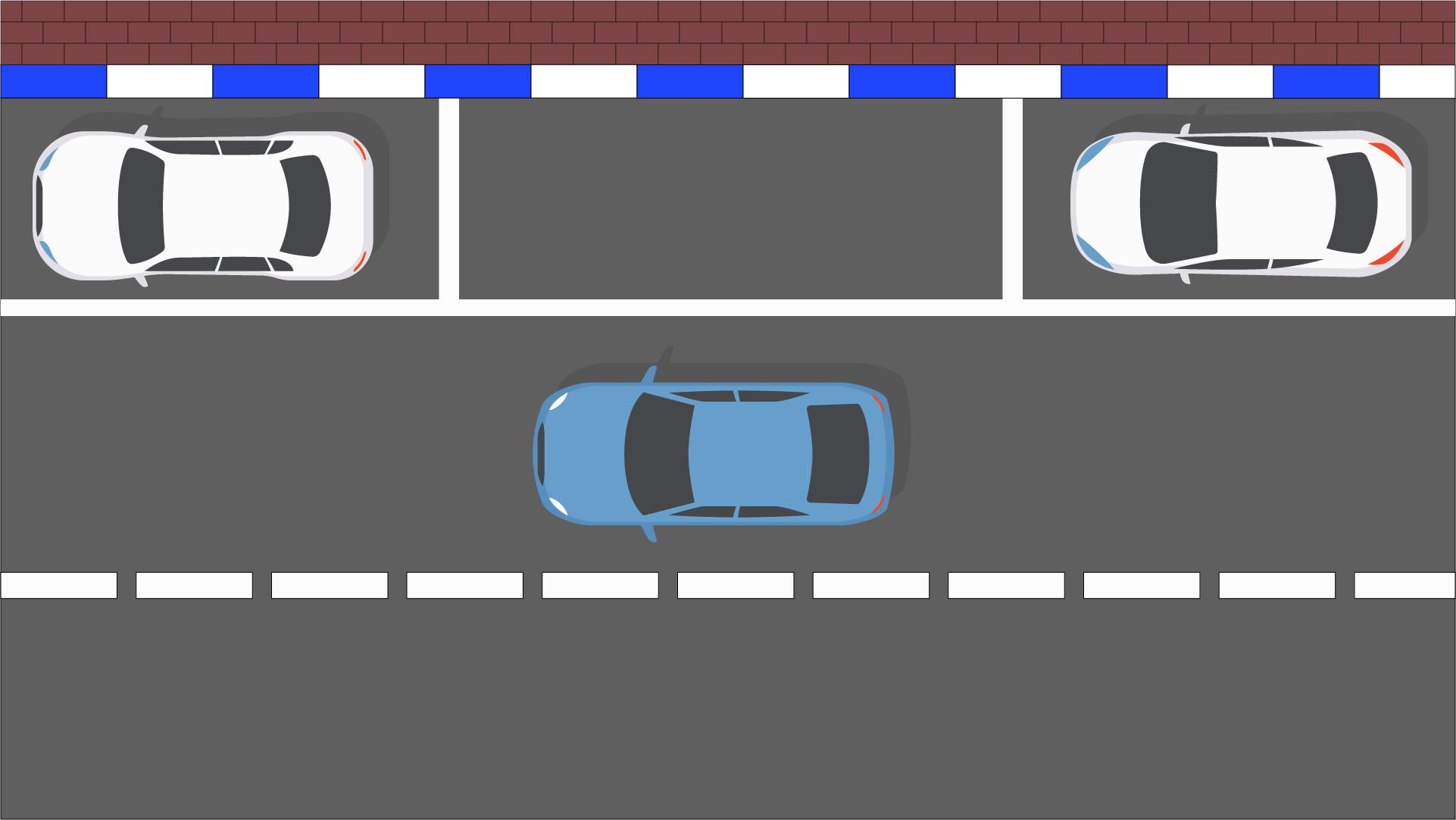


Fig.1: Initial situation for the parking assistant activation. The driver visually finds a potential parking spot (in the presented example, restricted from two sides).

The driver in the blue car needs to park between the two white cars. Driver activates the application on his smartphone. Smartphone starts to obtain images from rare camera (Figure 2ab).

(a) (b)

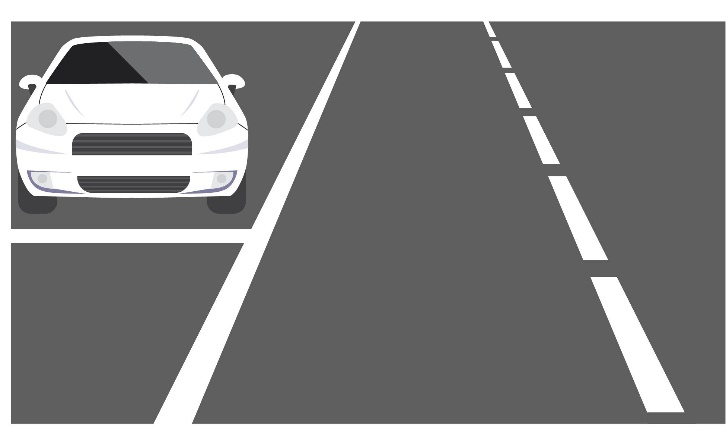
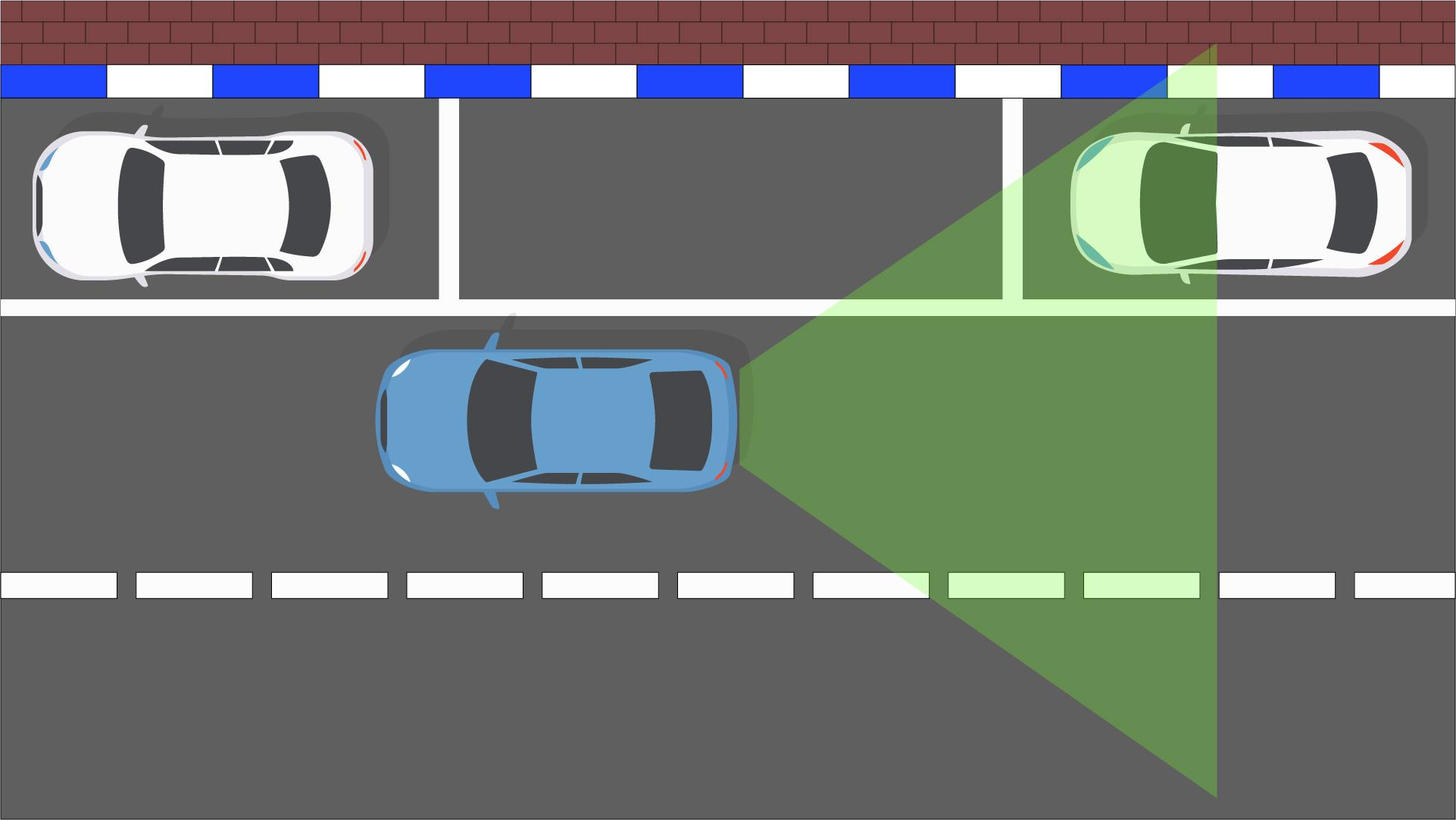


Fig. 2: Activation of application on smartphone. (a) Map of situation and viewpoint of rare camera; (b) Camera’s Perspective.

The application then asks the driver to move forward until getting a notification to stop (till some piece of the front car is seen). By moving forward, the camera can capture all the relevant points to get almost accurate distances between our car and the parking car (Figure 3ab).

(a) (b)

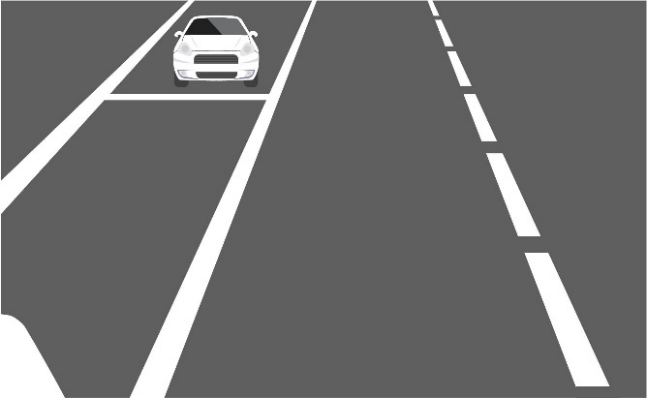
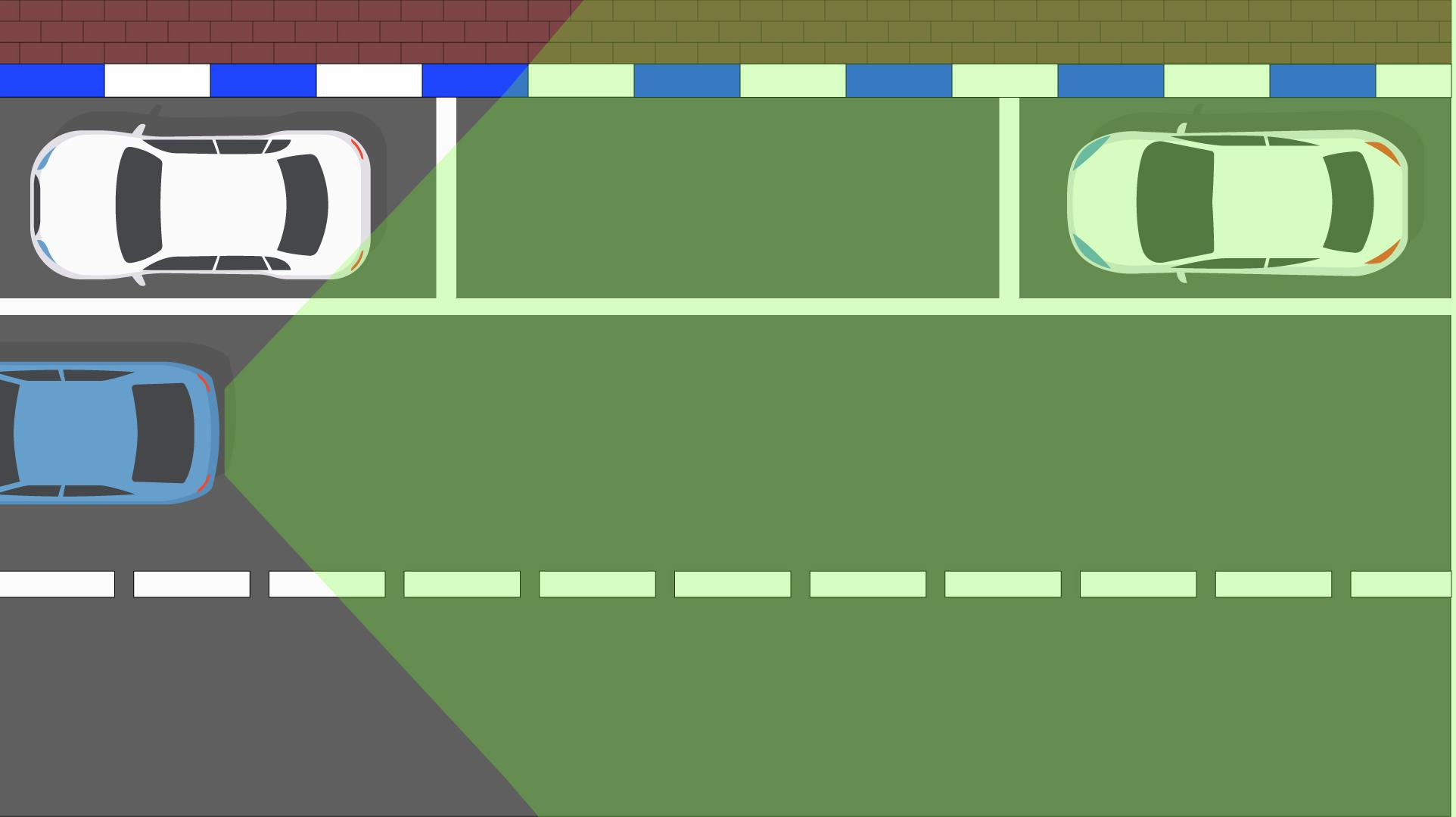


Fig. 3: Stopping Point. (a) Map of situation and viewpoint of rare camera; (b) Camera’s Perspective. Now size of white car on the image is less than in Fig 2b. Such changes of size are used for estimation of distance. Note that perspective is depend on the focal distance of the camera that is considered as given.

After capturing all the needed points, the application measures the distances between our car, the back white car and other items. As shown in “Figure 4ab” below, let’s assume that the black rectangle is the car’s license plate (with a standard fixed height and width (~12 x 50 cm, depending on the country) (. Let *S* be the plate’s real-life width, *h* is the plate’s width on the screen, *w* is the screen’s width and *z* is the distance between the parking car, and the standing one. Let camera can capture in a *ψ=120֯* angle, and the *α* angle, is the camera capturing angle of the license plate [1].

(a) (b)

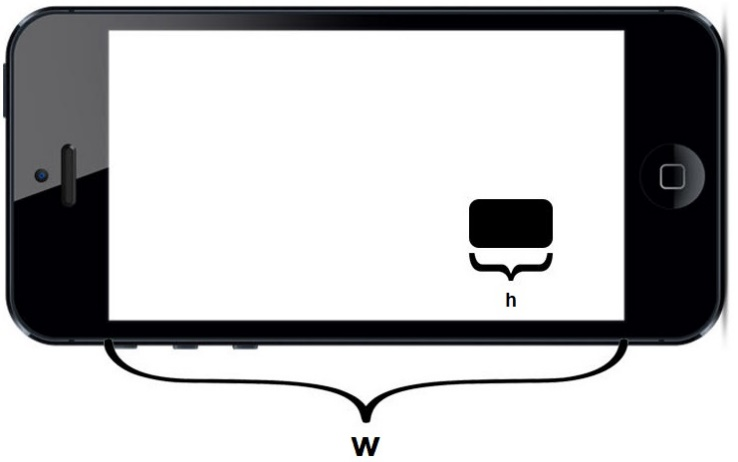
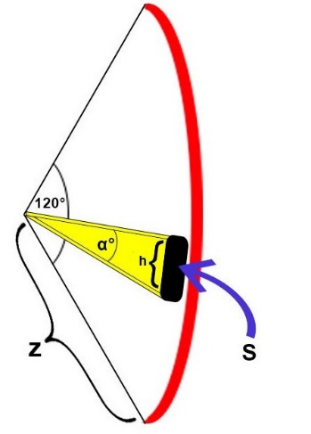


Fig. 4: All the parameters shown will help us calculate the z, which is the distance between the parked car, and the other car in the back.

Finding the *α* angle, is the first step for calculating the *z*:

After finding *α*, using the Tangent equation (Figure 5), the application can calculate *z*:

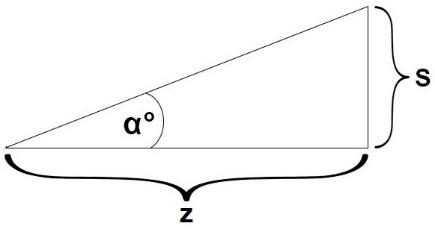


Fig. 5: The tangent equation can be taken from this right triangle.

Based on detected objects and estimated distances a graph of all possible states (positions of centre *(x, y)* and orientation defined by the azimuth angle *φ*). In such graph nodes are possible states and edges are possible simple movements (Figure 6). Connecting all these points forms a path diagram which will be suggested as possible parking path. Note that shortest path in 2D can be impossible in 3D: car can’t turn immediately on the large angle. There is some minimal possible radius of turn ([12], Figure 7).

(a) (b)

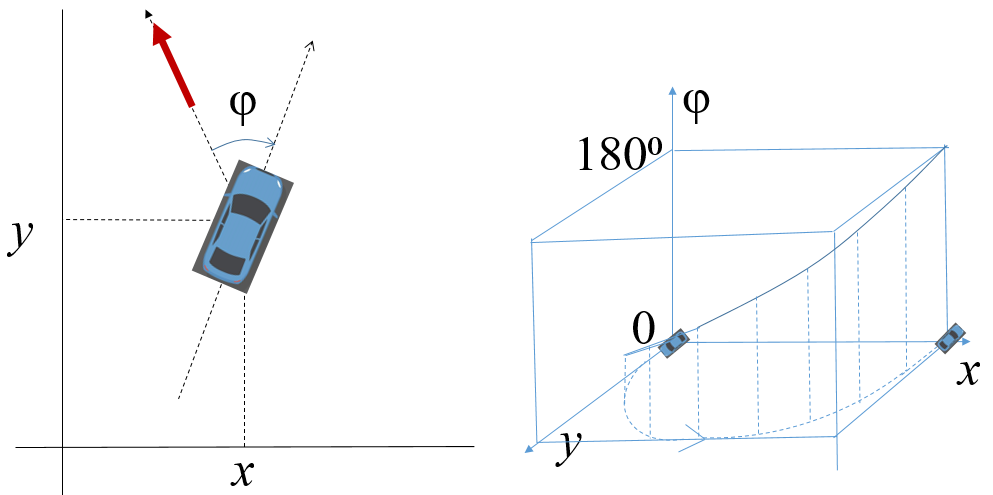
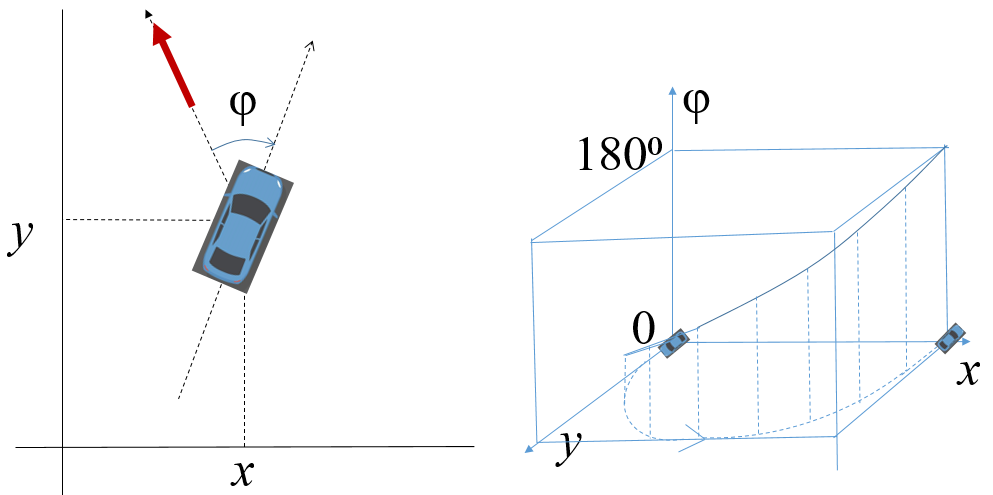


Fig. 6: Trajectory of moving in the terms of (x, y, φ): (a) translation of the car’s position and orientation onto point in coordinate space (x, y, φ). (b) Correspondence of trajectories in 2D and 3D spaces.

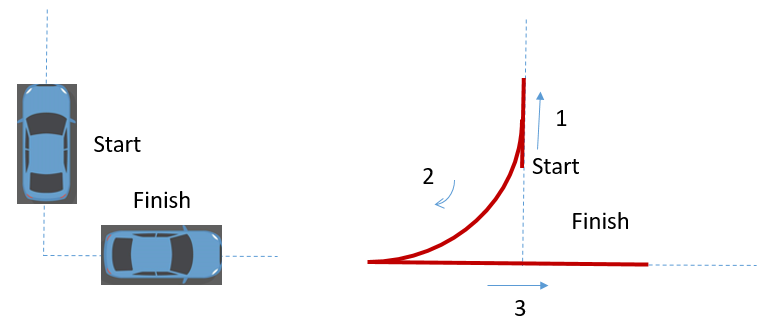


Fig. 7: Example of possible movement (2D). Direct movement from state Start to state Finish is impossible.

When the graph of all possible states connected by all possible movements is constructed, the problem of optimal trajectory searching is reduced to the problem of searching for the shortest path on the nodes of such graph. We solve this problem by using Dijkstra’s Algorithm [2]. We illustrate this algorithm by the example presented in Figure 8.

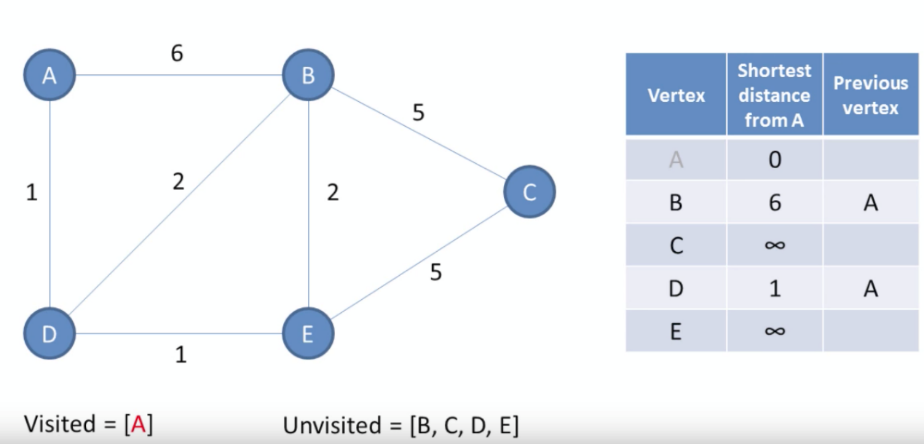


Fig. 8: Illustration of Dijkstra’s algorithm in simple case, First select a node n from the unvisited nodes which has the least weight, and then for each neighbor x of n, compare weight[x] to weight[n] + distance[n,x], This is determining, can we get a path from the root x with smaller distance by going through n instead of the current best path? if yes, update the weight of x to the new distance.  
And then remove n from the set of unvisited nodes [2, 16].

We are looking for the shortest path starting from the car’s current position (which will be our starting point) to the parking position. To find the shortest path, car should move by the according nodes included to the best solution [9]. In case that the driver didn’t follow the instructions correctly, the application recalculates the shortest path starting at the point where he stopped.

* 1. **Recurrent algorithm**

To improve performance, we developed a faster algorithm to calculate the path. We built a path from straight movements and curves of circle. Minimal radius of the circle is defined by technical characteristics of the car.

To achieve a parallel parking spot situated in coordinates *(x,y)* from current position *(x0,y0)* and limitations *xmin xmax*, do the following:

Case 1: If parking spot position situated in the straight line going through the center of our car, and directed by the main line of our car, then go straight to the parking slot.

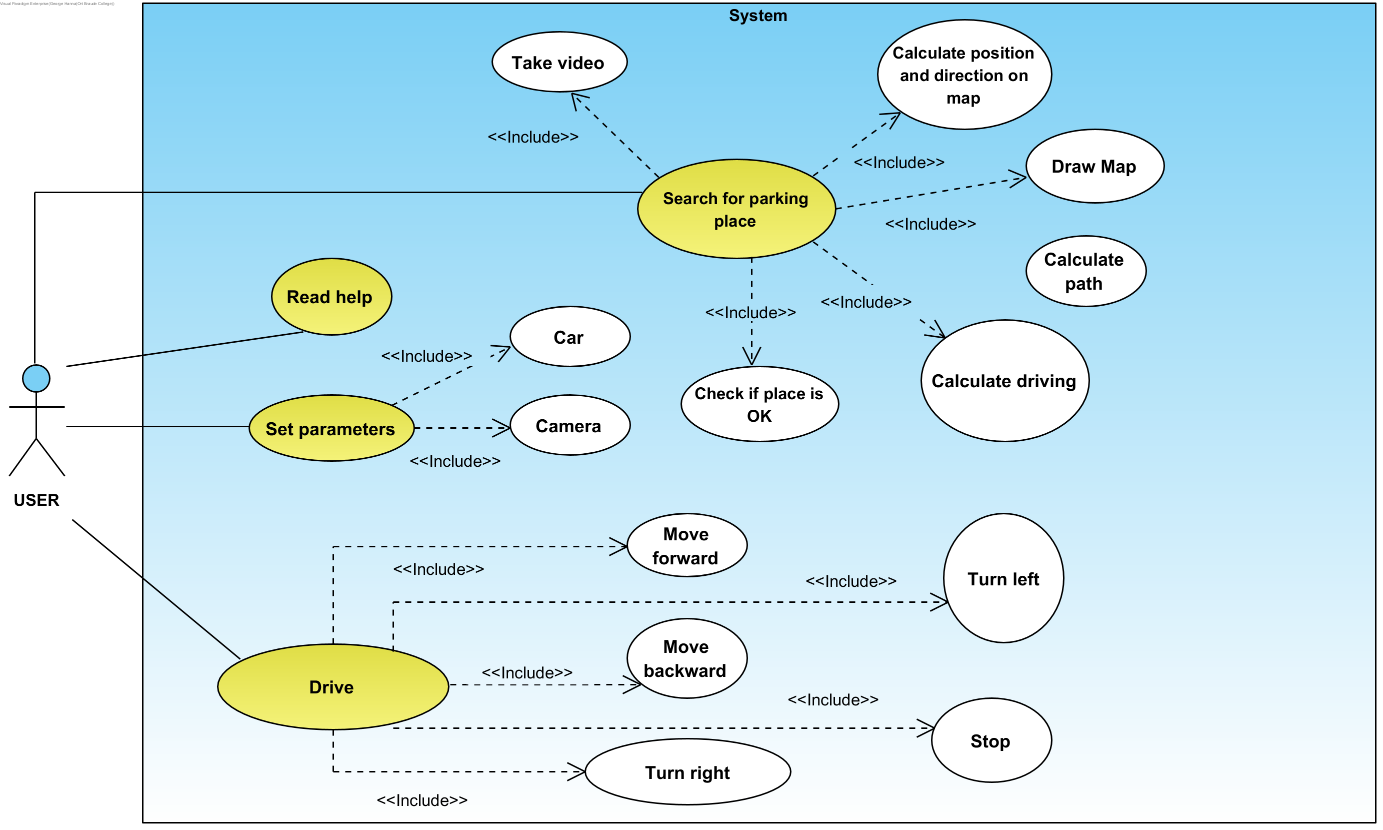
Case 2: If it is possible to achieve parking spot by 2 opposite arches which are equal in length, so go through these arches.

Case 3: If it is possible to achieve parking spot by 2 opposite arches equal which are in length, and a straight line, so go through this path.

Case 4: If it is possible to achieve depth of a parking spot by 2 opposite arches which are equal in length, separated by a straight line, then, go to depth *y*and finish the parking by straight movement.

Case 5: Move to the most possible depth *ymax*. Let such maneuvers are finished in the point *(x’,ymax)*. Now we solve the same problem with initial point *(x’,ymax)*.

1. **SOFTWARE ENGINEERING DECUMENTS**
   1. **Use Case**



UC1: Read Help

* Goal: Help the user understand any functionality in the application.
* Preconditions: No preconditions.
* Possible user errors: No possible user errors.
* Limitations: No limitations.
* Pseudo code Flow:

|  |  |
| --- | --- |
| Actor | System |
| 1. Click 'Help' button. | 1. Display the 'Help' page. |

UC2: Set car parameters

* Goal: Input values for parameters to calculate the path.
* Preconditions: Vehicle should be selected.
* Possible user errors: User enters unfeasible values/leaves any fields empty.
* Limitations: It should contain only digits in a specific range for each parameter.
* Pseudo code Flow:

|  |  |
| --- | --- |
| Actor | System |
| 1. Click 'Settings' button. | 1. Display the 'Settings' page with the default parameters. |
| 1. Click ‘Save’ button. | 2. If inputs are illegal: the system displays an error message with the correct range near each illegal input. 3. If inputs are legal: the system returns to the main menu and saves the values in an array. |
| 1. Click ‘Back’ button. | 1. Return to the main menu without saving any parameters. |

UC3: Car

* Goal: Input values for car parameters to calculate the path.
* Preconditions: No preconditions.
* Possible user errors: User enters unfeasible values/leaves fields empty.
* Pseudo code Flow:

|  |  |
| --- | --- |
| Actor | System |
| 1. Set car parameters. | 1. Show the values inside the text fields. |

UC4: Camera

* Goal: Input values for camera parameters to calculate the path.
* Possible user errors: No preconditions.
* Limitations: User enters unfeasible values/leaves fields empty.
* Pseudo code Flow:

|  |  |
| --- | --- |
| Actor | System |
| 1. Set camera parameters. | 1. Show the values inside the text fields. |

UC5: Search for parking place

* Goal: Search for the parking place.
* Possible user errors: All inputs in the Settings page are legal.
* Limitations: Car in the back should be existing (the system calculates the distances using the license plate of that car).
* Pseudo code Flow:

|  |  |
| --- | --- |
| Actor | System |
| 1. User runs the application | 1. Turns on the camera, loads the parameters. |
| 1. User clicks on “Parking Search”. | 1. Program starts taking pictures from the camera, and search for a License Plate, and calculate path and driving commands. |

UC6: Take video

* Goal: Calculate the map and our position in the map.
* Possible user errors: Camera not connected to Bluetooth.
* Limitations: Should be enough light.
* Pseudo code Flow:

|  |  |
| --- | --- |
| Actor | System |
|  | 1. Take video, and search for the License plate, and calculate its size on the screen. |

UC7: Calculate position and direction on map

* Goal: Calculate position and direction on map.
* Possible user errors: Back car doesn’t have a License Plate, or License Plate size doesn’t have the standard size.
* Limitations: Back car should have a License Plate, and it should have a standard size.
* Pseudo code Flow:

|  |  |
| --- | --- |
| Actor | System |
| 1. Drives forward along the parking places. | 1. Calculates the distance between our car and the car in the back and builds the map. |

UC8: Draw map

* Goal: Study the territory.
* Preconditions: Video is taken, and distance is calculated.
* Limitations: No objects in the camera view (excluding back and forward cars).
* Pseudo code Flow:

|  |  |
| --- | --- |
| Actor | System |
|  | 1. Draws the results of the calculations on the screen. |

UC9: Calculate path

* Goal: Find optimal maneuvers.
* Preconditions: Map exists, car’s position and direction are known.
* Limitations: Feasible path exists.
* Pseudo code Flow:

|  |  |
| --- | --- |
| Actor | System |
|  | 1. Runs our algorithm. |

UC10: Calculate driving

* Goal: Give instructions for the driver.
* Preconditions: Car path is calculated.
* Limitations: Feasible driving instructions.
* Pseudo code Flow:

|  |  |
| --- | --- |
| Actor | System |
|  | 1. Calculates instructions and presents it on the screen. |

UC11: Check if place is OK

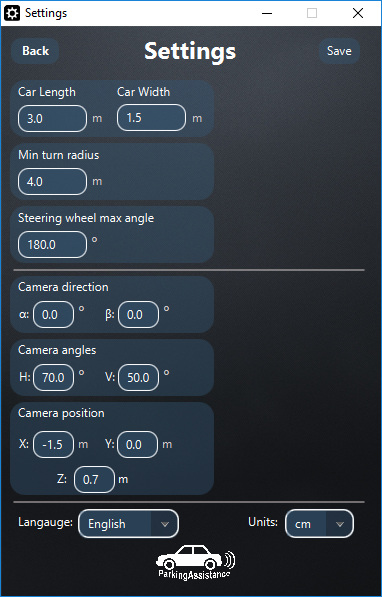
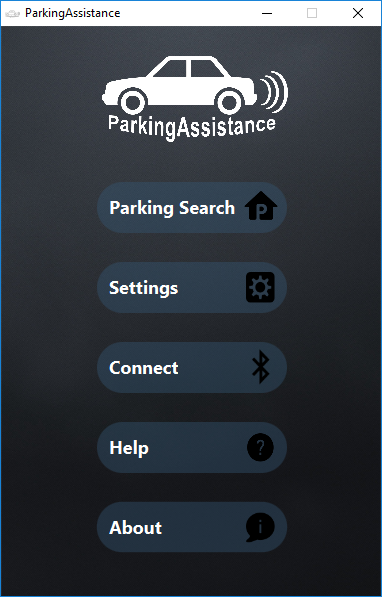
* Goal: Tells the driver that parking process is finished.
* Preconditions: Car is situated in the parking spot.
* Limitations: System should know the map.
* Pseudo code Flow:

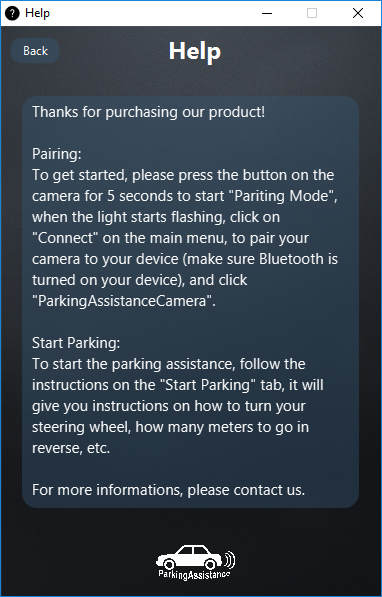
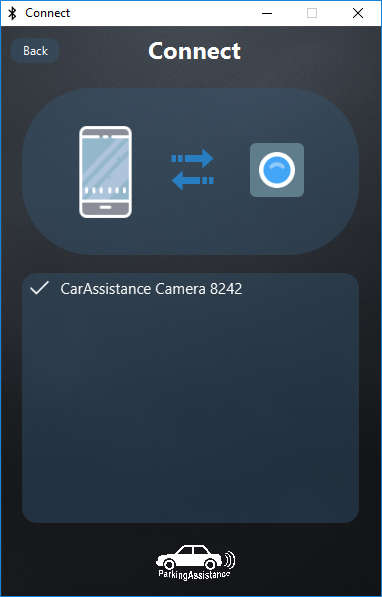
|  |  |
| --- | --- |
| Actor | System |
|  | 1. Presents a message to the driver that the parking procedure is finished. |

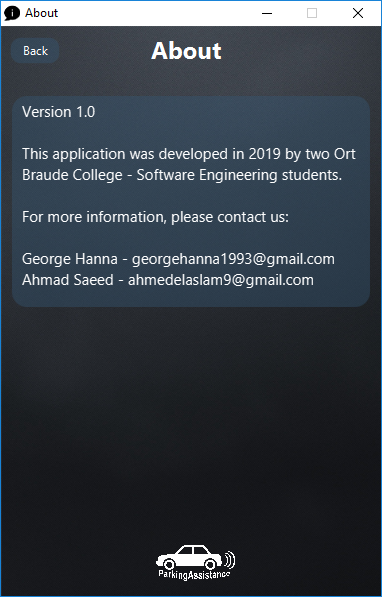
UC12: Drive

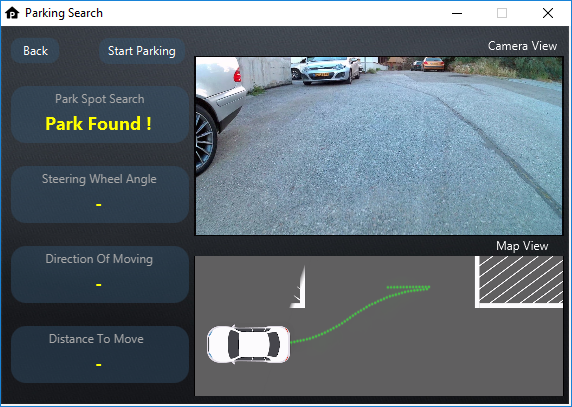
* Goal: Scan the territory, arrive to the parking spot.
* Preconditions: Application get the video and build the map.
* Limitations: Car doesn’t have any problems.
* Pseudo code Flow:

|  |  |
| --- | --- |
| Actor | System |
| 1. Drives the car following the instructions. | 1. Controls position of the car on the map, and recalculate path and instructions. |

* 1. **GUI** 

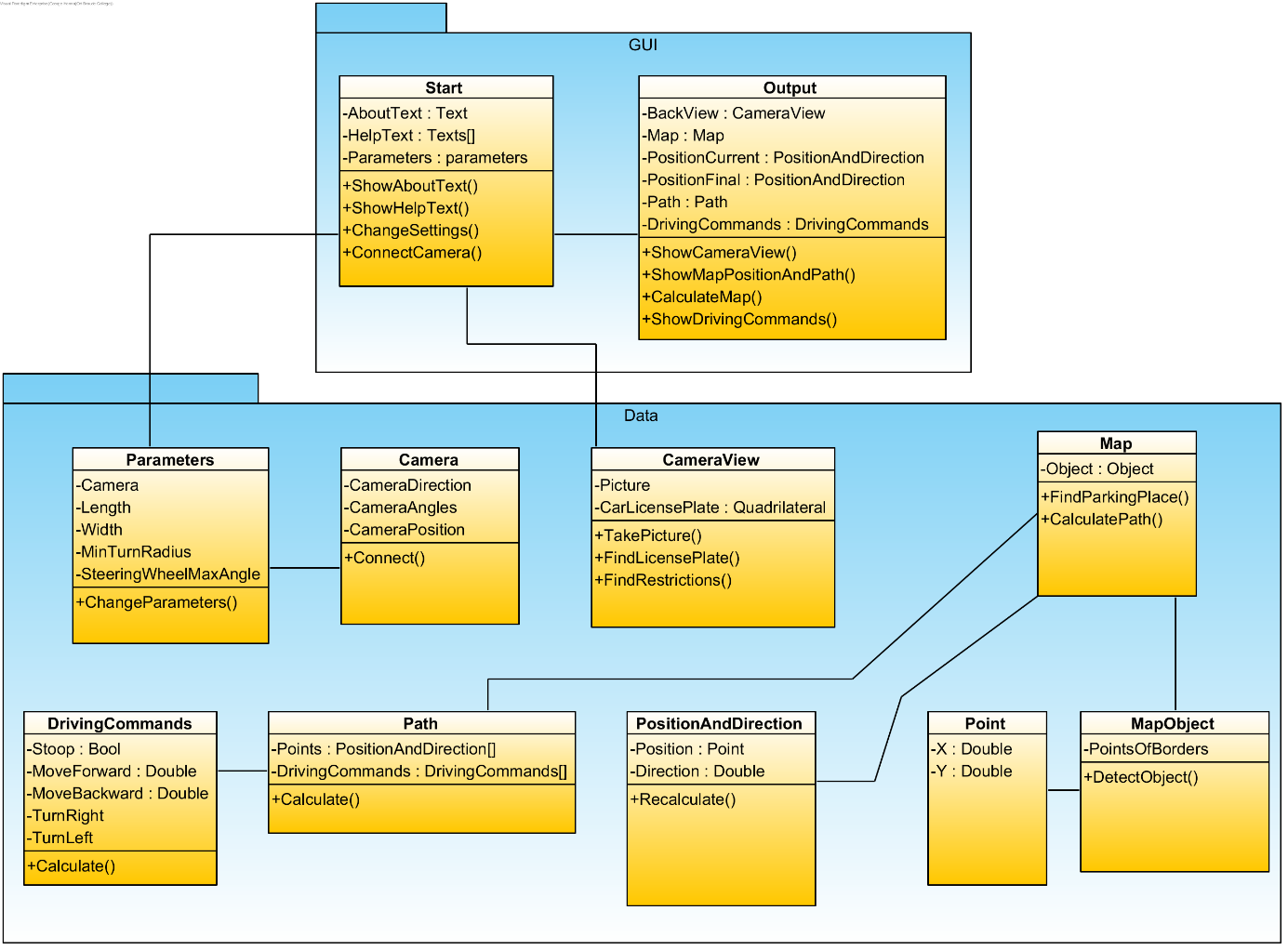








* 1. **Design**



* 1. **Testing**

In order to check out the system’s performance, we run the program on some significant inputs:

|  |  |  |  |
| --- | --- | --- | --- |
| **Test No.** | **Test subject** | **Expected result** | **Actual results** |
| **1.** | The user clicks on “Start Parking” without connecting to the Bluetooth. | Camera not showing, and an **error** message appears. | Not tested with real camera |
| **2.** | The user didn’t follow the instructions correctly while parking. | The application recalculates the route and gives the driver a new path. | pass |
| **3.** | The user tries to park in a tight parking spot where his car can’t fit. | An **error** message appears | pass |
| **4.** | The user is getting closer to the car in the back | A **warning** message appears. | pass |
| **5.** | The car is standing directly in front of the parking spot. | Proposed path is “go reverse” only (without driving wheel turning from the middle) | pass |
| **6.** | The car enters the parking spot. | Proposed path is empty, Message appears: “Stop, car parked successfully”. | pass |
| **7.** | Learn how number of maneuvers depend on parking place. | In simple case we expect small numbers from 1-3. In complicated case it can be large. | pass |

1. **RESULTS AND CONCLUSIONS**
   1. **Results**

In this section, we present results from the runs of our applications. First, we study how to calculate position of backward car on the map based on the size and position of the license plate on the screen (4.1.1). Second, we tested usability and performance of our algorithm in simple cases. We found that net-based algorithm is too slow to work in real-time regime, and we decided to change the algorithm to a recurrent one (see methods). Third, we present the results of calculations for specific simple situation (4.1.2). Fourth, we studied how the number of maneuvers depends on the distance for backward car, and depth of parking spot (4.1.3).

* + 1. ***Calculation of position on the map***

We calculated how to estimate position of the backward car based on the size and position of the license plate on the picture (in the smartphone). Supposing the real position of the license plate in *(x,y)* and the real size *b*, we calculate *(x,y)* by the image size *b’*, distance from the center *y’* and camera view angle by the equations:



Equation (1) Equation (2)

Using the Pythagoras rule, we can calculate the distance *h*:

A close up of a logo

Description automatically generated



Equation (3)

Fig. 15: Pythagoras Theorem

See example (Fig. 15):

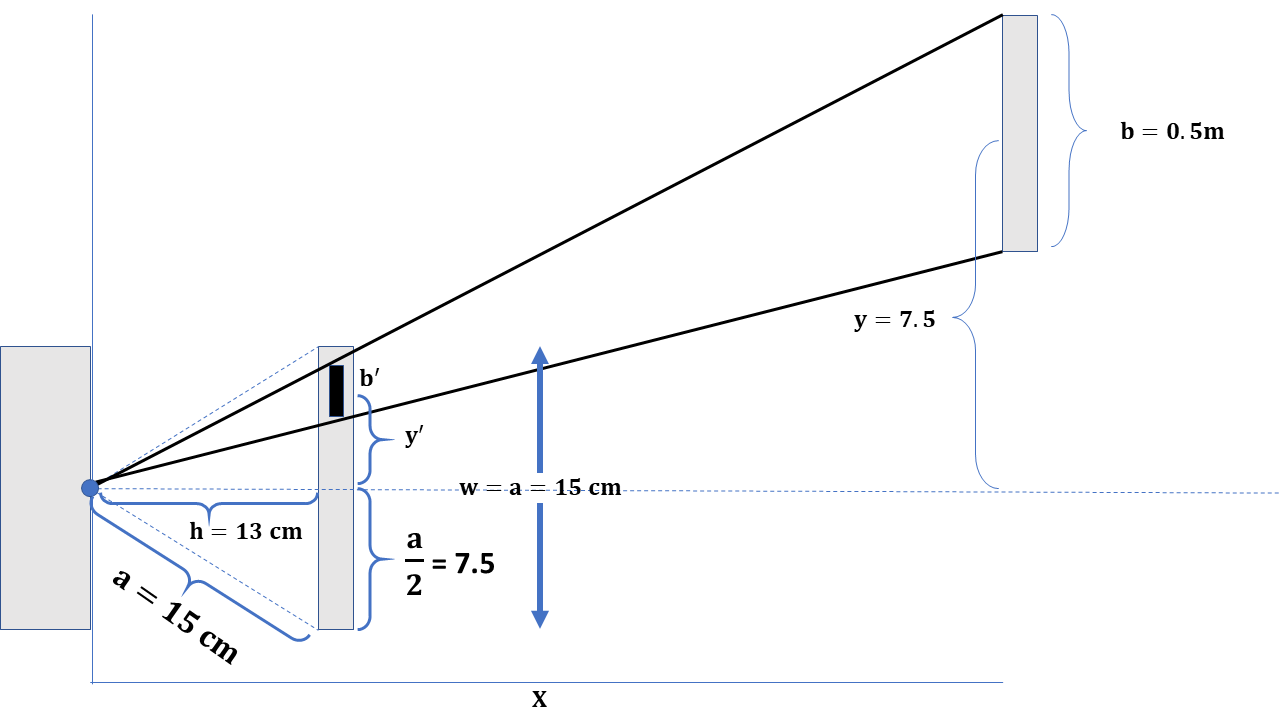


Fig. 15: Example of calculations of the backward car position based on the size and position of the license plate in the image. Here, we supposed that camera view angle is 60°, and the length of the screen *w* *=* 15cm. This means that the camera projects the view on the screen situated on the distance *h* of about 13cm. Hence if image size of license plate *b’* and the distance from the center is *y’*, (Equation (2)), and (Equation (1)) If image processing enables detecting license plate with image size at least 0.5cm, then maximum x is . Maximum distance from the center of the screen y’ is 7.5cm, hence, maximum y is .



* + 1. ***Driving maneuvers in simple situations***

In most simple cases, the driver needs 2 – 5 maneuvers to get to the parking spot. (Fig. 16) .In Fig. 17, and Table 1, we present path and driving commands for the case of 3 maneuvers. In the Fig. 18 we present real optimal path with 5 maneuvers.

A close up of a map

Description automatically generated

Fig. 16: Examples of maneuvers enabling arriving to the parking spot. (a) Only 2 maneuvers: Backward right arch + Backward left arch (red line). (b) 3 maneuvers: Backward right arch + Backward left arch + Forward straight (blue line). (c) 5 maneuvers: Backward right arch + Backward left arch + Forward right arch + Forward left arch + Backward straight (orange line). The number of maneuvers depends on the restrictions *xmin* and *xmax* which are calculated based on position of forward and backward cars *x1* and *x2*, length of the car, and minimal distance to keep.



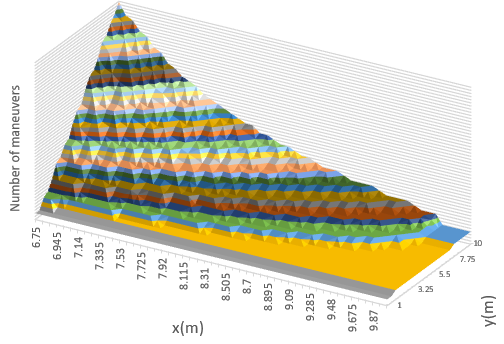
|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| # Maneuver | X (Start) | Y (Start) | Phi (Start) | X (Finish) | Y (Finish) | Phi (Finish) | Move (m) | Angle (Deg) | R | xC | yC |
| 0 | 0 | 0 | 180 | 3.65 | 1 | 210.64 | -3.82 | 100.54 | 7.16 | 0 | 7.16 |
| 1 | 3.65 | 1 | 210.64 | 7.3 | 2 | 180 | -3.82 | -100.54 | 7.16 | 7.3 | -5.16 |
| 2 | 7.3 | 2 | 180 | 5 | 2 | 180 | 2.3 | 0 | -1 | 0 | 0 |

Table. 1: Maneuvers for the situation presented in Fig. 17, rows correspond to maneuvers. Each maneuver is characterized by initial state, final state, moving distance, steering wheel angle. We also present corresponding radius and center of the circle.

* + 1. ***Study the number of maneuvers***

we studied how the number of maneuvers depends on the distance for backward car, and depth of parking spot. In Fig. 19a and Fig. 19b, we present such dependence for the car with default parameters in the case with forward car ended in *x0*=1m, stopping position is *x*=5m, and the start of the backward car *x1* is greater than 6 meters. We obtained that the number of maneuvers can be quite large if backward car is too close, we also found that if we have enough place, then parking spot can be reached with 4 maneuvers only, even under condition of the shortest path.

(a) (b)

A close up of a logo

Description automatically generated

Fig. 19: Dependence of maneuver’s number on the distance from backward car, and depth of parking spot. a) 3D plot, b) Contour lines, we see that the number of maneuvers decrease with increasing of the distance for backward car and increase with increasing of the depth of the parking spot. With distance greater than 2*R* = 8 m, depth of parking slot starts to play less role: Any depth can be reached with 4 maneuvers, Backward right + Backward straight + Backward left + Forward straight (See right upper corner of Fig. 19b).

* 1. **Conclusions**

We succeeded to build an application calculating an optimal path and a corresponding driving commands for parking in the case of a given map. Such application can be useful to calculate maneuvers for the parking of car, even without having a computer and distance sensors. We supposed that mapping information can be calculated based on the camera view only. We calculated the distance for backward car by the size and position of the license plate image on the captured video.

We tested several variants of the algorithm and decided that a recurrent one have a higher performance and provides good enough results, at least in an uncomplicated situations.

We believe that our application can help parking for non-trained drivers, increase safety, save time and reduce amount of stress.

**REFERENCES**

[1] D. Pomerleau, T. Jochem. “Rapidly adapting machine vision for automated vehicle steering”. *IEEE expert*. Apr;11(2):19-27, 1996.

[2] Dijkstra EW. A note on two problems in connexion with graphs. Numerische mathematik. 1(1):269-71, 1959.

[3] “Automatic parking - Wikipedia”: <https://en.wikipedia.org/wiki/Automatic_parking>

[4] “Toyota unveils car that parks itself”: <http://www.cnn.com/2003/TECH/ptech/09/01/toyota.prius.reut/index.html>

[5] “Up Close: Ford's Self-Parking System”: <http://blogs.cars.com/kickingtires/2009/01/fords-parallel-park-system.html>

[6] “BMW Debuts New Parking Assistant In 2011 5-Series”: <http://www.thecarconnection.com/news/1041977_bmw-debuts-new-parking-assistant-in-2011-5-series>

[7] “A brief history of car parking technology”: [https://www.confused.com/on-the-road/gadgets-tech/parking-technology-brief-history](https://www.confused.com/on-the-road/gadgets-tech/parking-technology-brief-history%20)

[8] “What is the Intelligent Parking Assist System ?”: <https://www.quora.com/What-is-the-Intelligent-Parking-Assist-System>

[9] SE. Shladover, CA. Desoer, JK. Hedrick, M. Tomizuka, J. Walrand, WB. Zhang, DH. McMahon, H. Peng, S. Sheikholeslam, N. McKeown. “Automated vehicle control developments in the PATH program”. *IEEE Transactions on vehicular technology*. 40(1):114-30, 1991.

[10] JD. Strong, inventor; Telxon Corp, assignee. “Automated vehicle return system”. *United States patent US 6,006,148*, 1999.

[11] JM. Anderson, K. Nidhi, KD. Stanley, P. Sorensen, C. Samaras, OA. Oluwatola. “Autonomous vehicle technology: A guide for policymakers”. *Rand Corporation*; 2014.

[12] "Automated Vehicle Technology." *King Coal Highway 292*: 23-29, 2014.

[13] NJ. Goodall. “Ethical decision making during automated vehicle crashes”. *Transportation Research Record*. 2424(1):58-65, 2014.

[14] D. Zhao, X. Huang, H. Peng, H. Lam, D.J. LeBlanc. “Accelerated evaluation of automated vehicles in car-following maneuvers”. *IEEE Transactions on Intelligent Transportation Systems*, 19(3), pp.733-744, 2017.

[15] S. Zhang, J. Chen, F. Lyu, N. Cheng, W. Shi, X. Shen. “Vehicular Communication Networks in the Automated Driving Era”. *IEEE Communications Magazine*. 56(9), pp.26-32, 2018.

[16] “Dijkstra's Algorithm for Shortest Path”: <http://rkpandey.com/AlgorithmHelper/shortest/path/2017/05/16/dijkstra.html>