

University Of West Bohemia
Faculty of Applied Sciences
Department of Cybernetics

MASTER'S THESIS

Driving Range Estimation and Trajectory Planning for Electric Vehicles



PORSCHE

Porsche Engineering Services s.r.o. - Radlická 714/113a - 158 00
Praha 5

Porsche Engineering
Services s.r.o.
Radlická 714/113a
158 00 Praha 5
Tel. +420 2350 911 51
Fax. +420 2350 911 98

Diploma thesis name:

Driving range estimation and trajectory planning for electric vehicles

1.) Research the optimization and HMI concepts available on the market

2.) Trajectory optimization

- research the function of trajectory optimization taking into account the location of charging stations and their power
- the algorithm may take the advantage of combining own algorithms with the API available by Google for trajectory planning
- implement the algorithm on Samsung Galaxy Tab (initial version of the java-based application will be given by Porsche Engineering Services s.r.o.)

3.) Extend the application for a map feature

- extend the application so that
 - the map (Google maps) with the current position is shown to the driver
 - the driving range is shown in the map as a "polygon"
 - application uses Google API to access additional map information (used for displaying on the map or for trajectory optimization)

3.) Testing

- test all developed algorithms using the Software in the loop approach in several use cases
- verify that developed application in several real time tests and experiments in the Demo vehicle

contact person:
Ing. Martin Rezac, Ph.D.

Porsche Engineering Services, spol. s.r.o.
Radlicka 714/113a
158 00 Praha 5

Tel. +42 02350 91183
Cell. +420734128567
E-Mail: martin.rezac@porsche-engineering.cz

Sídlo společnosti:
Praha
Městský soud v Praze
IČO: 26 47 41 15

Jednatel:
Doc. Ing. Miloš Polášek, Ph.D.

Bankovní spojení:
Komerční Banka, a.s.
Č. účtu:
43-2544090277/0100

Prohlášení

Prohlašuji, že jsem diplomovou práci vypracoval samostatně a výhradně s použitím citovaných pramenů.

V Plzni dne 23.05.2017

Bc. Robin Popelka

Declaration

I hereby declare that this master's thesis is completely my own work and that I used only the cited sources.

Anotace

Cílem této práce je prozkoumat způsoby a možnosti HMI nabízených v současné době na automobilovém trhu a následně navrhnout řešení pro plánování trasy s ohledem na umístění dobíjecích stanic a jejich výkonu pro elektrická vozidla. Dalším úkolem je naimplementovat mapové podklady a zobrazit na mapě dojezdovou vzdálenost ve tvaru polygonu s ohledem na aktuální kapacitu baterie v elektromobilu. Pro řešení zmíněných problémů je využito několik API od fy. Google. Navržené algoritmy byly následně implementovány do dodané Android aplikace, která zajišťuje komunikaci s elektromobilem. Jednolivé algoritmy byly následně otestovány (Software in the Loop) a poté také proběhlo testování na reálném Demo elektromobilu.

Klíčová slova

HMI, elektrická vozidla, nabíjecí stanice, plánování trasy, dojezdová vzdálenost, Google API

Abstract

The main objective of this study is to research the HMI concepts available on the market for electric vehicles and design a solution of trajectory planning, taking into account the location of charging stations and their power. The next objective is to implement Google maps and display driving range on the map as a polygon with respect to the vehicle's battery capacity. Several APIs from Google were used to solve the aforementioned problems. The designed algorithms were implemented to an Android app which communicates with an electric vehicle. All developed algorithms were tested using the Software in the loop approach for several use cases. After that, the developed application was verified in several real time tests and experiments in a Demo vehicle.

Keywords

HMI, electric vehicles, charging stations, route planning, driving range, Google API

List of Abbreviations

Abbreviation	The meaning of the abbreviation
AC	Alternating Current
API	Application Programming Interface
App	Application
CAN	Controller Area Network
CHAdemo	Trade name of quick charging method up to 62.5 kW
DC	Direct Current
EV	Electric Vehicle
GPS	Global Positioning System
HMI	Human Machine Interface
HTTP	The Hypertext Transfer Protocol
IEC	International Electrotechnical Commission
JSON	Javascript Object Notation
kW	Kilowatt (a unit of electric power)
kWh	The Kilowatt-Hour (a derived unit of energy)
SAE	Society of Automotive Engineers
SDK	Software Development Kit
SoC	State of Charge
TTF	Time To First Fix
UUID	Universally Unique Identifier

Contents

1	Introduction	1
1.1	Motivation	1
1.2	Target of this Thesis	1
1.3	Structure of this Thesis	2
2	HMI Concepts on the Market	4
2.1	Tesla Company	4
2.2	Nissan Company	5
2.3	Other Car's Producers and Conclusion	5
3	Charging Stations	6
3.1	Standards	6
3.2	Future of Charging Stations	7
4	Google APIs	8
4.1	Google Maps API	8
4.2	Google Maps Directions API	8
4.3	Google Maps Distance Matrix API	8
4.4	Google Maps Places API	9
4.5	Google Maps Web Service API	9
5	Driving Range Polygon	10
5.1	Satellite Navigation	10
5.2	Draw Current Location in the Map	11
5.3	Approaches How to Get Distance	11
5.3.1	Great-Circle Distance	11
5.3.2	Haversine Formula	12
5.4	WGS84 Standard	13
5.5	Designing Algorithm for Driving Range	14
5.5.1	Result from Google Maps Distance Matrix API	16
5.5.2	Making a JSON Request in Android	16
5.5.3	First Suggestion	17
5.5.4	Second Suggestion	18
5.5.5	Limit of API	19

6	Route Planning	20
6.1	Problem Definition	20
6.2	Basic Idea of an Algorithm	20
6.2.1	Reaching the Destination Without Charging	21
6.2.2	Need to Stop at Least Once on Charging Station	23
6.2.3	Can't Reach the Destination	23
6.3	Graph Theory	24
6.3.1	Path in Directed Graphs	25
6.3.2	The Shortest Path	25
6.4	Floyd-Warshall Algorithm	25
6.5	Dijkstra Algorithm	26
6.5.1	Priority Queue	27
6.6	Database of Charging Stations	27
6.7	Route Planning With Charging to 100 %	29
6.7.1	Disadvantage of this Approach	29
6.8	More Accurate Route Planning	30
6.8.1	More Diversified State of Charge	31
6.9	Time Spent on Battery Charging	32
6.10	Speed Performance of Both Used Algorithms	32
6.11	Heuristic Approach	33
7	Implementation into the Delivered Application	35
7.1	Reading Signals	35
7.2	Example of Implementation to the App	35
7.3	The Interface of Developed HMI	36
7.3.1	Options	36
7.3.2	Route Planning	38
7.3.3	Displaying Route Information	38
8	Testing	40
8.1	SW in the Loop	40
8.1.1	Test Driving Range Polygone	40
8.1.2	Test Route Planning Algorithm	40
8.2	HW in the Loop - Demo Vehicle	44
9	Conclusion	45
A	Google Maps API	48
A.1	How to Set Up the Google Maps API	48
A.2	Modify the Application's Manifest	48
A.3	How to Obtain a Google API Key	48
B	Application's Manifest	50

C The Class Diagram

51

1 Introduction

1.1 Motivation

The name of this thesis is "Driving Range Estimation and Trajectory Planning for Electric Vehicles". These days, electromobility is on the rise. With the current limited driving range of electric vehicles and the weak network of charging stations for them, arises a need for route planning through stations. Also because of the low driving range, it would be a good feature to display the driving range of the vehicle on the map in real time.

To be connected continuously to the Internet has become a standard in more and more vehicles, and because of this, there could be, according to the assignment of this thesis, used the Google environment and accomplished the route planning with the help of the Google APIs.

1.2 Target of this Thesis

There are several tasks that need to be done in this thesis.

- Research the optimization and HMI concepts available on the market.

There will be introduced two representative manufacturers of electric vehicles. The first one is the Tesla company and the second company will be Nissan. Two main problems connected to the topic of this thesis will be discussed. The intention will be focused on route planning problem and how these companies deal with displaying driving range on the map.

- Trajectory planning and optimization.

In this task, there will be researched the function of trajectory planning, taking into account the location of charging stations and their power. As mentioned before, there will be developed an algorithm that could plan the route from origin to destination with help of Google APIs. Discussed will be the limitations of the Google APIs, as well as several approaches to get the shortest path based on the shortest time.

- Display driving range as a polygon on the map.

With the help of Google APIs, again there will be displayed the actual driving range on the map as a polygon. This polygon will be updated according to changes of GPS position. The actual position will be displayed on the map as well.

- Add additional information using Google APIs

Besides using Google APIs to compute driving range and route planning, there will be used information about actual traffic conditions, and display this information alongside the route planning algorithm.

- Implement the algorithm on Samsung Galaxy Tab.

The developed algorithms and HMI will be implemented into an Android java-based application supplied by the Porsche Engineering s.r.o. company.

- Testing

Firstly, there will be tested all developed algorithms using the Software in the loop approach by running several examples and then our application will be additionally tested on the Demo vehicle in real tests to prove correctness of the developed algorithms and all functions of the developed application.

1.3 Structure of this Thesis

This thesis is divided into 8 chapters. This text is included in the first chapter of this thesis.

After this introductory chapter there will be described HMI concept available at this time on the market. There will be discussed how other manufacturers deal with the topic of this thesis.

The third chapter will be focused on charging stations. What standards are now in use, and what is the future of the charging stations' charging power.

The fourth chapter describes all used Google APIs in this thesis. There is described how to obtain an API key and how to implement Google API to our Android application. There will be described which APIs will be used. The attention will be focused also on the limitation of free of use these APIs.

The fifth chapter is about displaying driving range as polygon problem. There will be described how could be obtained GPS location in Android and there will be described also an algorithm for the polygon driving range problem. Doing this algorithm I dealt with a few problems and I will discuss the disadvantages of used approaches.

The sixth chapter is focused on the route planning through charging stations problem. There will be introduced two graph search path algorithms, the Floyd-Warshall and the Dijkstra with the priority queue. There will be designed an algorithm from scratch with help of these algorithms. There will be discussed advantages and disadvantages of each used algorithm and there will be done a comparison of them.

Chapter seven is about the implementation of all developed algorithms and the interface into the delivered java-based Android application.

The last chapter is about testing the developed algorithms in such cases, and also the application will be tested on real Demo vehicle to prove that all functions are correct.

2 HMI Concepts on the Market

This chapter will be focused on HMIs concepts available this time on the market for electric vehicles. There will be described my personal experiences with HMI of Tesla and Nissan company.

2.1 Tesla Company

The main leader and innovator of EVs is the Tesla company. Tesla offers amazing electric vehicles for daily use, largely because of the big battery capacity used in these cars. There is a current problem with the density of charging stations, especially in Eastern Europe, and including the Czech Republic. There is only one Tesla Supercharger station in Humpolec, and another one is in Prague planned. You can use other, slower charging stations as well but you would wait for a while for your car to be fully charged for the next trip.

I had an opportunity to drive a Tesla Model S P90D. This car is amazing and I focused my attention on HMI of this car. You can see at the first sight that this car was made completely from scratch as an EV. Tesla has created a different style of a dashboard, with one big touchscreen display. With this, you can control almost all functions of this car including planning the route [Tesla, 2017a]. One task of this thesis is to make a polygon of the driving range around current position. There is not such a feature in Tesla car. This is maybe because of its big driving range, which is more than 350 km. The map in Tesla is based on Google Maps. You can plan the route and the navigation take you through the necessary count of charging stations to reach the destination. The route planner creates a path only through Tesla Superchargers, so if you want to drive a Tesla car in the Czech Republic, this navigation system is, at this time, inadequate. In any case, it works perfectly, and the route planner tells you how much time you spend on every single charging station to reach the next station or the destination. There is another great feature in route planner: because Tesla's car is always online, the planner can detect real-time traffic conditions and reroutes you to the destination, if a better alternate route is available. There cannot be told much about how Tesla's planning algorithm works, because it's proprietary, and there cannot be told how the algorithm to calculate driving range works, or what kind of data is taken into account in their algorithm.

2.2 Nissan Company

The next EV I had an opportunity to drive is from the Nissan company. Nissan's popularity made it an obvious choice. The top-selling EV in the world is the Nissan LEAF, which has sold over 250,000 units since it was first introduced in 2010 [Paquet, 2017]. This EV has a smaller driving range and it is much cheaper than the car from Tesla. It is also built from the ground as an EV. The driving range is up to 250 km, and nowadays this car is suitable for daily use as well.

This vehicle has the HMI much simpler than in Tesla, and the maps are not based on Google Maps. You can see your current location with driving range displayed as a radius on the map. There are actually two circles. The bigger one is for eco-drive mode and the smaller one is for the normal drive mode. You can select what kind of driving style you prefer with regard to the range. I got information (according to salesman) that the real driving range is about 160 km. It depends of course on many factors. The radius shown on the map is not accurate enough because it is only straight line distance from current location and there is no information about the real road.

There is a possibility to plan a route in this car, but unfortunately, there is no information about real-time traffic conditions and the map data can be outdated as well. That is not the biggest disadvantage of this car's route planner. I was surprised how this planner works. The map included several charging stations but only for the Czech Republic (this was caused because it was a brand new demonstration car as told salesman). If you select your destination it turns on navigation, but only if your driving range is too low does it navigate you to the nearest charging station. There is no information about how much time you need to spend at the charging station, and also there is no summary information about the road, including charging stations and time spent on charging. So if you plan a long trip you can get into trouble.

2.3 Other Car's Producers and Conclusion

I had deliberately chosen two different EVs and researched their HMIs. Other vehicle manufacturers are providing more or fewer features as mentioned above. All major vehicle manufacturers nowadays invest a lot of money in the development of EVs. It is still a small market in comparison to conventional vehicles, but the vehicle manufacturers realize that EVs are definitely the future in the automotive industry. I looked for other vehicle manufacturers HMIs information available on the internet and it is more or less similar to mentioned features.

3 Charging Stations

Charging station infrastructure is already built. It is much better in Western Europe, but it is only a matter of time until there are more stations in the Czech Republic and other eastern European countries [KELAG, 2017]. The main problem of the current charging stations is their charging power. Today the best high-current charging stations offer 50 kW of charging power in the Czech Republic. On the other hand, the Tesla Supercharger offers 120 kW of charging power, but for now, it cannot be used by other EVs [Tesla, 2017b]. There are many low-power charging stations in our territory and they offer charging power of 22 kW.

3.1 Standards

There are two main charging options in SAE terminology to be distinguished. The first one is 240 volt AC charging used at home or on public place by companies. This AC charging is known as Level 2. The second one is 500 volt DC charging known as high-current or DC Fast Charge [Wikipedia, 2017; Bakker, 2013].

The International Electrotechnical Commission modes definition (IEC 62196):

- Mode 1 – slow charging from a regular electrical socket (single- or three-phase)
- Mode 2 – slow charging from a regular socket but with some EV specific protection arrangement
- Mode 3 – slow or fast charging using a specific EV multi-pin socket with control and protection functions (e.g., SAE J1772 and IEC 62196)
- Mode 4 – fast charging using some special charger technology such as CHAdeMO

There are three connection cases:

- Case A is any charger connected to the mains (the mains supply cable is usually attached to the charger) usually associated with modes 1 or 2.
- Case B is an on-board vehicle charger with a mains supply cable which can be detached from both the supply and the vehicle – usually mode 3.
- Case C is a dedicated charging station with DC supply to the vehicle. The mains supply cable may be permanently attached to the charge-station such as in mode 4.

There are four plug types:

- Type 1 – single-phase vehicle coupler – reflecting the SAE J1772/2009 automotive plug specifications
- Type 2 – single- and three-phase vehicle coupler – reflecting the VDE-AR-E 2623-2-2 plug specifications
- Type 3 – single- and three-phase vehicle coupler equipped with safety shutters – reflecting the EV Plug Alliance proposal
- Type 4 – fast charge coupler – for special systems such as CHAdeMO

In this thesis, I used a database of the Android application UEmap of Electromobility Union [Union, 2016]. There is information about actual charging stations in the Czech Republic with power information and location. I omitted stations with charging power of 3.6 kW because of charging speed, so included are all stations with the power of 50 kW and 22 kW.

3.2 Future of Charging Stations

According to website hybrid.cz there will be charging stations with power up to 350 kW in the future [Horčík, 2017]. With this performance can be battery charged in minutes. For test purpose and for simulation of the future charging stations I used several more powered stations for example with charging power of 150 kW.

4 Google APIs

In this thesis, it is necessary to implement Google Maps as a source of maps. The reason for this choice is obvious, as Google Maps is used worldwide and nowadays provides one of the best map interfaces and APIs in the world, especially for Android platform.

Several APIs from Google are used for the purpose of this thesis. The most important is to implement Google Maps API to display map data on the screen. All of the other used APIs from Google return JSON objects, so I had to use JSON parser to work with it [Google, 2017a]. Before you use any API from Google you need to obtain a Google API key and add it to the app. In appendix A is described how to set up the Google Maps API, how to modify the application's manifest file and how to obtain a Google API Key.

4.1 Google Maps API

The Google Maps API is very well documented, and for the Android platform, it is the best I can interact with. It is free of use with query limitations, but for the purpose of this thesis it is sufficient, and I count on it when implementing algorithms. With this API I can add to the map for example markers, polygons, and overlays.

4.2 Google Maps Directions API

This API provides the ability to find directions between locations. There is a very useful feature named Time in Traffic. It returns the time required to get to the destination with respect actual traffic conditions. Besides that, it returns important information about the path. Included are time, time in traffic, distance, individually legs along the path etc. There is the possibility to get requests with more waypoints along the path. It is limited to maximum 23 waypoints in each request. There is a usage limit for this API. There are 2,500 requests per day for free, and speed limitation is declared as 50 requests per second. After exceeding this limit the API returns an empty JSON object [Google, 2017b].

4.3 Google Maps Distance Matrix API

Google's Distance Matrix service computes travel distance and journey duration between multiple origins and destinations using a given mode of travel. This service does not return

detailed route information. Route information, including polylines and textual directions, can be obtained by passing the desired single origin and destination to the Directions Service [Google, 2017c].

I use this API to compute distances needed for displaying the polygon of the driving range. I can use the Google Distances API as well, but in this case, I would share one API key and hence one request limit for polygon and route planning algorithm, so at the beginning of designing the driving range algorithm, I decided to use this API because there is no need for other information besides distance.

Because of request limit and application speed, I decided to use a Haversine formula to compute a great-circle distance between two points on the earth. This formula will be described in subsection 5.3.2 of this thesis.

4.4 Google Maps Places API

For a user-friendly interface of the app, I decided to use Google Places API [Google, 2017d]. It provides type-ahead predictions like on Google Search Engine and it prevents the wrong address from being typed.

The usage limit for this API is declared as 1,000 requests per day.

4.5 Google Maps Web Service API

This API must be enabled in the Google API Console before the Google Places API is used. Otherwise, the Google Places API won't work.

The usage limit for this API is declared as 1,000 requests per day as well.

5 Driving Range Polygon

One of the goals this thesis is to draw driving range in the map as a polygon. This goal was chosen because today's vehicle market provides, in most cases, only a circle to show driving range. This radius does not provide an accurate solution. It is computed as great-circle distance and it does not account for the diversity of the roads, so by using the help of the Google APIs there could be obtained a more accurate solution.

The first thing need to be obtained is the current location using the satellite navigation and using Network. There will be described how to obtain this location in Android application and how to draw it in the Google Map.

There will be explained a couple of approaches how to get a distance between two geographical points in this chapter. Due to the Google API limitation mentioned in chapter 4, I was forced to get distances by another way. For this purpose, there will be described a great-circle [Rodrigue] and Haversine formula [Rick, 1999] approach. There will be described also World Geodetic System 1984 known as WGS84 [Verheijen, 2016] to get the more accurate distance in computation.

After this assumption mentioned above there will be described an algorithm's approach how to solve driving range polygon problem. At the end of this chapter, there will be summarized API limitation in the designed algorithm.

5.1 Satellite Navigation

Satellite navigation is a system that allows us to get the current location, given by the receiver, in our case Android tablet. From satellite navigation, there could be obtained longitude, latitude and elevation level with an accuracy of meters. There could be obtained the current local time with high precision as well. The signals from the satellite are obtainable anytime and anywhere on the earth. There is only one requirement for obtaining the location. The device must be placed outdoors to obtain a position. It can take a while to get the location fix. If an internet connection is available, there is the ability to obtain location much faster using Assisted GPS which improves TTFF.

The position can be obtained from the Network provider as well. This provider determines location based on WiFi access points and cell tower with lower accuracy compared with satellite location. The TTFF is much faster in this case.

In summary, there are used both providers to get the device's location. If the location cannot be obtained from the satellite then it is taken coarse location from the Network

provider. In appendix B are described all permissions needed to be declared in the manifest file for the running the algorithm.

5.2 Draw Current Location in the Map

From the location listener, there is obtained, among other information, the current location defined as an object with longitude and latitude attributes. These values will be applied to display current location. Google API offers us to use Marker object to show location on the map. With following code we draw new Marker of current location to Google Maps:

```
@Override
public void onMapReady(GoogleMap map) {
    map.addMarker(new MarkerOptions()
        .position(new LatLng(latitudeValue , longitudeValue))
        .title("Current location"));
}
```

The text "Current location" is an info window when clicked on the Marker. There could be added additional options to the marker such as color, shape etc.

5.3 Approaches How to Get Distance

For purpose of a driving range polygon algorithm, there is a need to get a great-distance between two geographical points. This distance is a starting point and it will be next improved with iteration algorithm approach to get the more accurate distance between two points. It will be used also for displaying driving range distance as a circle. This distance can be obtained with a few approaches. There will be described two similar functions to get this distance without using Google APIs.

5.3.1 Great-Circle Distance

For long distances can be used a simple great-circle equation known also as orthodromic distance. This approach simplifies the model of the earth and supposes that the Earth is a perfect sphere.

Formulas

Let

$$\phi_1, \lambda_1, \phi_2, \lambda_2$$

be the geographical latitude and longitude of two points 1 and 2, and

$$\Delta\phi, \Delta\lambda$$

their absolute differences; then $\Delta\sigma$, the central angle between them, is given by the spherical law of cosines:

$$\Delta\sigma = \arccos(\sin \phi_1 \cdot \sin \phi_2 + \cos \phi_1 \cdot \cos \phi_2 \cdot \cos(\Delta\lambda))$$

The distance d , i.e. the arc length, for a sphere of radius r and $\Delta\sigma$ given in radians

$$d = r \Delta\sigma$$

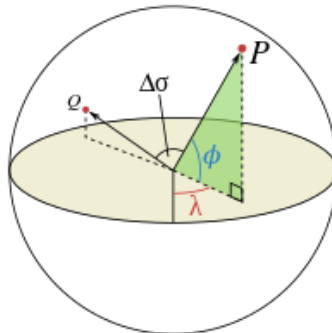


Figure 5.1: An illustration of the central angle, $\Delta\sigma$, between two points, P and Q. λ and ϕ are the longitudinal and latitudinal angles of P respectively [Wikipedia, 2016].

This method does not give us accurate results for short distances and we can use a better method where rounding error is smaller.

5.3.2 Haversine Formula

With this formula can be obtained the great-circle distance with more accurate result. There is implemented the WGS84 Standard to get better results than mentioned in subsection 5.3.1. The distance between any two points on the earth can be computed by the following equation.

$$\text{hav}\left(\frac{d}{r}\right) = \text{hav}(\varphi_2 - \varphi_1) + \cos(\varphi_1) \cos(\varphi_2) \text{hav}(\lambda_2 - \lambda_1),$$

where hav is the haversine function defined as

$$\text{hav}(\theta) = \sin^2\left(\frac{\theta}{2}\right) = \frac{1 - \cos(\theta)}{2} \quad \text{hav}(\theta) = \sin^2\left(\frac{\theta}{2}\right) = \frac{1 - \cos(\theta)}{2},$$

where d is the distance between the two points,

r is the radius of the sphere,

φ_1, φ_2 : latitude of point 1 and 2, in radians,

λ_1, λ_2 : longitude of point 1 and 2, in radians.

The distance d could be obtained applying the inverse haversine as follows.

$$\begin{aligned} d &= r \text{hav}^{-1}(h) = 2r \arcsin\left(\sqrt{h}\right) \\ d &= 2r \arcsin\left(\sqrt{\text{hav}(\varphi_2 - \varphi_1) + \cos(\varphi_1) \cos(\varphi_2) \text{hav}(\lambda_2 - \lambda_1)}\right) = \\ &= 2r \arcsin\left(\sqrt{\sin^2\left(\frac{\varphi_2 - \varphi_1}{2}\right) + \cos(\varphi_1) \cos(\varphi_2) \sin^2\left(\frac{\lambda_2 - \lambda_1}{2}\right)}\right) \end{aligned}$$

5.4 WGS84 Standard

The World Geodetic System is a standard established in 1984. It allows us to estimate the ellipsoid (in our case the Earth) with a precision of centimeters. This standard is used by GPS and it is implemented in the Haversine formula used in the final source code. This standard defines the coordinates system, geoid, and referenced ellipsoid. It was developed by the U.S. Army.

The geoid is defined as an ideal model of the Earth but this model is hard to describe mathematically and it is not suitable for geographical and cartographical tasks. Because of this, it is common to use reference model WGS84 instead. The origin of this standard is meant to be located at the Earth's center of mass. It is computed with help of satellite measurement and it was last revised in 2004. The coordinate system is based on this standard. Each geographic point is defined with its longitude, latitude and elevation level [Verheijen, 2016].

5.5 Designing Algorithm for Driving Range

In this section, there will be described an algorithm for driving range shown as a polygon on the map. There will be discussed the problems that arose during the design of an algorithm associated with Google APIs. The goal of this effort is to display polygon on the map as shown in the Fig 5.2.

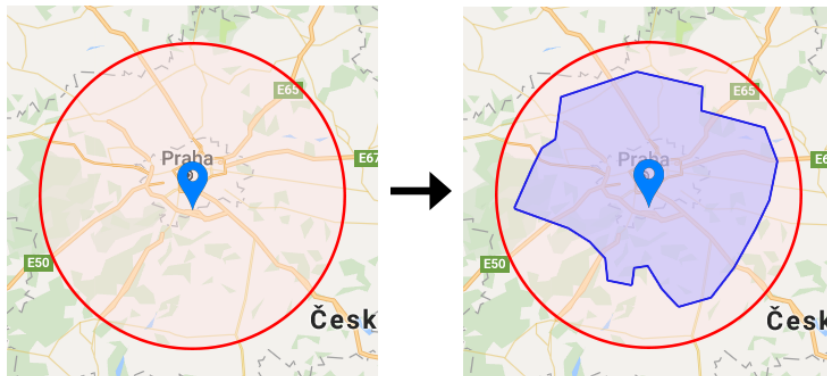


Figure 5.2: The goal visualization of driving range polygon algorithm. On the left side is a commonly used circle with great-circle driving range distance and on the right side is the goal polygon shape which the algorithm want to achieve. The red circle shows us the great-circle distance. The blue colored area shows us real driving range with real road's paths.

The main problem associated with displaying range as polygon is that none of Google APIs provides for returning the coordinates of the point with distance, direction and start point coordinates given as an input parameter. There must be given two points coordinates (the origin point - usually current location, the destination point - any point around the current location in driving range distance) to get great-circle or road's distance. The problem is that the real road's destination point is unknown. It means that on the beginning can be obtained only great-circle distance. Using the Haversine formula can be obtained the point's coordinates with driving range distance as a parameter in each direction. This point means the starting point (only great-circle distance) and now it can be returned real road's distance using Google API between the current location and starting point. Simplify, the algorithm starts with the starting point and with each iteration reduces the difference between great-circle distance and the real road's distance.

The red circle represents only 90 percent of the driving range distance because the red circle shows us the great-circle distance and there was proven that anytime is the real road's distance inside the red circle. According to the development of the algorithm was proven that this statement is true. If it were, in any case, not right, it would still be more

accurate than considering 100 percent.

With the help of the Google API the circle could be displayed as follows:

```

GoogleMap map;
Circle circle = map.addCircle(new CircleOptions()
    .center(new LatLng(latitudeValue , longitudeValue))
    .radius(10000) // in meters
    .strokeColor(Color.RED)
    .strokeWidth(4)
    .fillColor(Color.argb(140, 192, 192, 192)));

```

The following code is to create a polygon on the map. It consists of several location points connected by a closed curve to create a polygon shape.

```

GoogleMap map;
Polygon polygon = map.addPolygon(new PolygonOptions()
    .add(new LatLng(0, 0), new LatLng(0, 5), new LatLng(0, 0))
    .strokeColor(Color.BLUE)
    .fillColor(Color.BLUE));

```

In the algorithm is considered the driving range from the Demo vehicle. This driving range is counted as a SoC divided by an average consumption for last 20 minutes of driving. Following is the equation how the driving range is counted in the vehicle.

$$d = \frac{\text{SoC}(kWh)}{\text{avg. consumption for last 20 min}(kWh)} \cdot 100$$

During the real test on Demo vehicle wasn't, unfortunately, the algorithm correctly calibrated. Due to this fact, I was forced to set the average consumption for last 20 minutes as a constant parameter (20 kWh/100km). For any other additional information, I don't use in the algorithm because the Demo vehicle does not provide any.

When designing an algorithm the idea is based on current location which could be obtained from a provider. The Google API does not return distance as the polygon. At the beginning, I used Google Distances Matrix API because the purpose is to get the only value of the distance between points. I use Haversine formula to count great-circle distance in the algorithm.

5.5.1 Result from Google Maps Distance Matrix API

Google Maps Distance Matrix API return a JSON object with distance and duration value as follows:

```
{
  "destination_addresses" : [ "Praha, Cesko" ],
  "origin_addresses" : [ "Plzen, Cesko" ],
  "rows" : [
    {
      "elements" : [
        {
          "distance" : {
            "text" : "94,4 km",
            "value" : 94394
          },
          "duration" : {
            "text" : "1 hodin, 15 minut",
            "value" : 4472
          },
          "status" : "OK"
        }
      ]
    }
  ],
  "status" : "OK"
}
```

The JSON object must be parsed to use it in our application. I use org.json Parser [26] for these needs.

5.5.2 Making a JSON Request in Android

In order to make a request to the Google API servers I need to use HTTP request. First, I created classical HTTP request approach with HttpURLConnection [Google, 2017e] and HttpClient in the Android application[Burnette, 2009]. It worked just fine but the requests' tasks are returning slowly from the server and in this case, are returning back one after another task. This means that every task waits on finishing the previous task before being launched. This causes a problem with speed performance. If I consider, for example, 24 requests for the first iteration of the algorithm, it will last tens of seconds before all tasks are finished. Despite the fact that this solution worked well, I had to use an asynchronous HTTP request approach instead.

For the asynchronous HTTP request, I decided to use Volley library [Chakraborty, 2016]. With this library, tasks can be executed asynchronously on a different thread which won't block the "main application's thread". In our algorithms, I create one common Request queue and the library executes them all at once. The responses come independently of each other and the speed performance is limited only with internet connection speed and Google API's speed limitation which is declared as 50 requests per second. Moreover, the Volley library allows a JSON object to return instead of a String.

5.5.3 First Suggestion

At the very beginning, I created a red circle to display the great-circle distance from the current location. As said before, it is just 90 percent of the range value. For testing purposes, I decided to create a hexagon in this circle with the center of the current location as well. The hexagon is finally replaced by a more accurate twenty-four-sided polygon.

There is a need to get a location of each point in the hexagon. It means the distance need to be set which is the actual range value and bearing from the current location. I use the following formula to get it.

Formula:

$$\varphi_2 = \text{asin}(\sin\varphi_1 \cdot \cos\delta + \cos\varphi_1 \cdot \sin\delta \cdot \cos\theta),$$

$$\lambda_2 = \lambda_1 + \text{atan2}(\sin\theta \cdot \sin\delta \cdot \cos\varphi_1, \cos\delta - \sin\varphi_1 \cdot \sin\varphi_2),$$

where φ is latitude, λ is longitude, θ is the bearing (clockwise from north), δ is the angular distance d/R ; d being the distance travelled, R the earth's radius [Veness].

Each of hexagon's point lies on the circle and with each iteration is each point refined until stop condition is reached. The stop condition is set up as 3 kilometers.

The following are the described algorithm's steps:

1. Create circle with a great-circle based radius
2. Create a hexagon in this circle
3. For each point of the hexagon, get the driving distance returned by Google Maps Distances Matrix API
4. With each iteration reduce the deviation of each point as long as the stop condition is reached (3 km deviation)
5. Draw polygon.

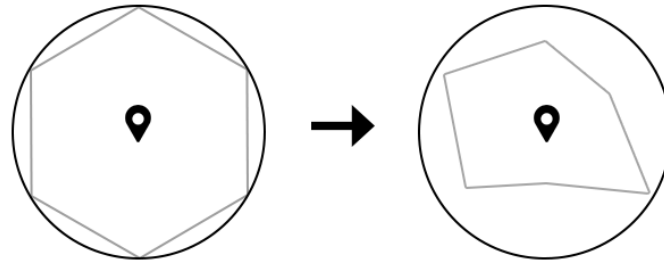


Figure 5.3: Example of hexagon driving range visualization.

While designing this algorithm a problem occurred. When reducing deviation there comes unreliable result from Google server. Although I want to get driving distance on potential same great-circle line I could obtain a different route from each request. It means that the great-circle distance can be, in the next step, farther from our location but the result could be closer than before because of alternative Google route response.

This problem means that I can get, with iteration, to the goal deviation, but it can take many more requests and the algorithm gets slower with each request.

5.5.4 Second Suggestion

With regards to the above-mentioned problem, I deal with it using the Google Maps Directions API. This API returns JSON with more information such as individual steps in the route. Every step contains distance value and its location.

To avoid the deviation problem in the first suggestion, I iterate, in this approach, through steps in response and the steps' distances are added until the driving range is overcome. After that, a new request is called from the last added step's location and it is iterated to the required driving range distance.

With this suggestion, I avoid the deviation problem, so the good results could be obtained with fewer requests but a new problem could also have occurred. In some cases, it could happen that close points in multiple-side-polygon are returning a similar JSON object back. For example, the JSON object returns almost the same route for close points, and if an algorithm stops too early with the stop condition, then we get the same location for both points and it means we get a less accurate solution in some cases.

The following are the described algorithm's steps:

1. Create a circle with a great-circle based radius

2. Create a hexagon in this circle
3. For each point of the hexagon, get the driving distance returned by Google Maps Directions API
4. Add the steps' distances until the driving range is overcome
5. Call a new request from the last added step's location and get a more accurate solution
6. Repeat step 5 until deviation limit is reached (3 km)
7. Draw polygon.

5.5.5 Limit of API

For the first suggestion, I use the Google Maps Distances Matrix API. I don't use this API for the route planning described in this thesis, but I must take into account that with each API request I can obtain an unexpected result and I must ask more times to get a sufficient solution.

The second suggestion uses the more detailed Google Directions API. This API is used for trip planning as well. Despite this, I can get a result with much fewer requests to the Google servers, although it could merge close points to one location.

6 Route Planning

The biggest problem today's EVs is limited driving range and with that connected long charging time. The charging stations network is also not well built in Czech Republic especially there are only a few high-current charging stations.

The charging stations network is weak and yet there are differences based on their charging power. These facts have to be considered into account when planning route and there has to be done an algorithm which can produce optimal time results.

On the beginning will be adumbrated the problem definition and how should work the algorithm with help of Google APIs. The graph problem will be described and what algorithm's I use to search the best time route through stations. Because of Google API's limitation there will be described how was made an offline charging stations database.

I describe several route planning approaches in our algorithm and then I compare them all in several examples.

6.1 Problem Definition

The idea of the algorithm is to find the path between origin and the destination through charging stations with respect the optimal time results. On each station may be or not the battery charged to the full capacity.

The algorithm is divided into two parts. The first part is based on dynamic programming [Park, 2015] which finds the shortest time path through charging stations. In the second part of this algorithm is the founded path check with Google API. There is returned more accurate time result with consideration of actual traffic conditions taken into account.

6.2 Basic Idea of an Algorithm

The strategy of route planning is designed with Google Distance API together. The Google API return route between origin and destination and the returned JSON object is including time and each step information. Because of Google API daily limitation I had to create database of charging stations and paths between them.

If there is database between all charging stations and their paths, the Google servers don't have to ask too many times on each route planning and the algorithm can run

simultaneously much faster. In each route planning task, the Google servers need to be asked for origin and destination surrounding charging stations in the distance of actual driving range.

Three basic situations may be considered:

1. The vehicle can reach the destination without recharging battery.
2. The vehicle must stop to charge batteries at least once on the route.
3. The vehicle is too far away from reaching destination or charging station.

The notations are defined for above description as follows:

r - vehicle's range

s - charging station

d - distance between start and destination

d_1 - distance between start and the nearest charging station

d_2 - distance between destination and the nearest charging station

d_3 - distance between start and charging station

d_4 - distance between destination and charging station

d_5 - distance between charging stations

6.2.1 Reaching the Destination Without Charging

In this situation is considered that the vehicle doesn't need to charge battery to reach the destination. The assumption must be fulfilled that the vehicle's range is smaller than distance between start and destination as follows:

$$r > d$$

There must taken into account that the vehicle's battery can be out of its power in destination. There will be considered that the vehicle can be charged in destination from 240-volt source of electricity. In the app can be set up the level of battery's capacity need to be charged in destination assuming slow charging speed and this parameter will be considered

in the algorithm because the following situation may occur. The level of battery capacity will be set up in destination as 70 percent. Near by the destination would be charging station with power of 50 kW so that is faster charge at this charging station than let it be charging in destination although the destination will be reached in fact faster if we don't stop at charging station.

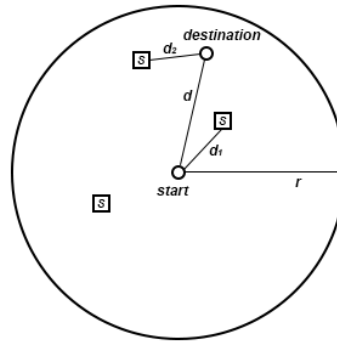


Figure 6.1: Reaching the destination without charging.

6.2.2 Need to Stop at Least Once on Charging Station

In situation that the vehicle's range can't reach the destination the vehicle must stop at charging station to charge battery. If it is from this charging station range smaller then distance to the destination we mustn't charge battery and we can go straight to destination. The following equation is:

$$r < d,$$

$$r > d_3$$

The following Fig. 6.2 shows the situation.

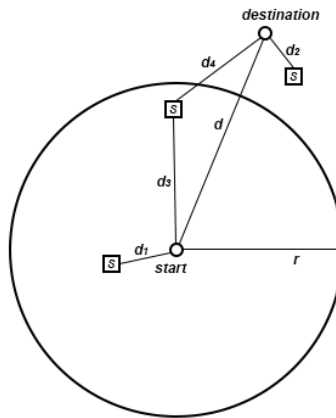


Figure 6.2: Charge battery to reach the destination.

In the algorithm must be considered the charging station's power, so the algorithm must choose more powered station and simultaneously the station can't be too far to pay off to go over it. It can happen that it is faster to stop on several high-current charging stations than stop on one with low power speed. This situation will be described later in this thesis.

6.2.3 Can't Reach the Destination

There can occur several situations, where the destination can't be reached. They are described with following picture in Fig. 6.3.

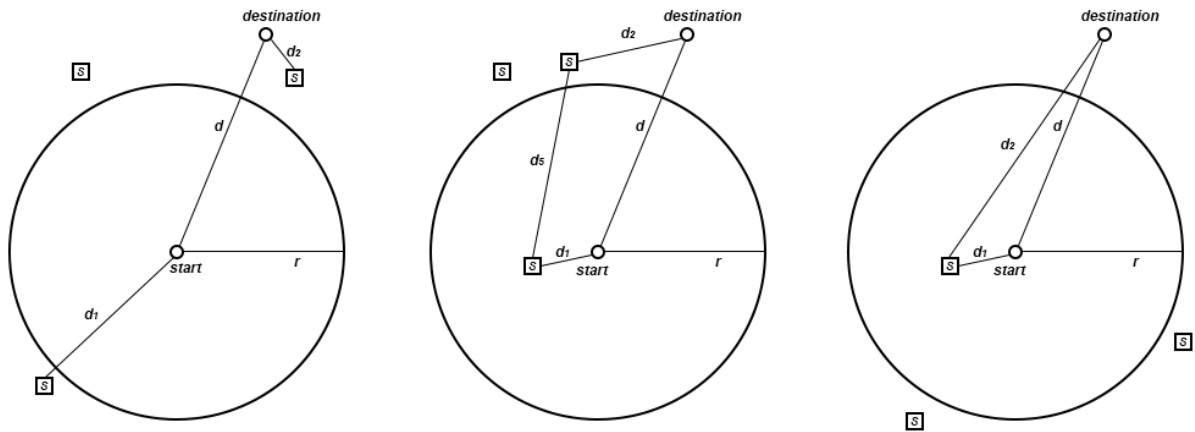


Figure 6.3: 3 situations that vehicle can't reach the destination.

In Fig. 6.3 on the left is shown that the charging station is too far from the start position and the vehicle can't continue in the trip.

$$r < d_1$$

In Fig. 6.3 in the middle can be seen that there could be reached charging station from start and destination, but the destination between them is too far and the vehicle can't continue.

$$r < d_5$$

In Fig. 6.3 on the right shows us that there is no charging station near by destination.

$$r < d_2$$

6.3 Graph Theory

Graph is a set of finite vertices and edges between them. We can define graph as following equation:

$$G = (V, E)$$

Let's define that the vertices represent the charging stations and the edges are paths between them. The edges are in this route planning problem oriented. There has to be considered that the path from A to B can be quite different than path from B to A although there is the same road and yet the existence of edge between A and B does not imply edge between B and A as well. The duration can be often different. The graph with oriented edges is called as directed graph [Hliněný, 2010].

6.3.1 Path in Directed Graphs

Path in directed graph is a sequence of vertices defined as $P = (v_1, \dots, v_n)$ and it begins from vertex $v_1 \in V$ to vertex $v_n \in V$. There exist an edges $e_i = (v_i, v_{i+1}) \in E$ for $i = 1, \dots, n - 1$. This path is defined by start and destination vertices and tuple of vertices between them with oriented edges [Hliněný, 2010].

6.3.2 The Shortest Path

The problem in route planning leads to the shortest path problem. It is about getting from origin to the destination with the shortest path across necessary count of charging stations. The attention is focused on the minimum value of the duration of all possible paths.

There are two approaches of shortest path problem used in this thesis:

- The All-Pairs Shortest Path problem. It finds shortest path between all vertices in our graph. (Floyd-Warshall algorithm)
- The Single-Pair Shortest Path problem. It finds shortest path between given vertices in our graph. (Dijkstra algorithm)

6.4 Floyd-Warshall Algorithm

Floyd-Warshall algorithm is an algorithm based on dynamic programming [Kolář, 2004]. It was first approach I used in route planning in this thesis. It is simple to implement it in Java at the expense of its asymptotic complexity which is $O(|N^3|)$, where $|N|$ is number of vertices in the graph.

Java implementation of an algorithm:

```
/**
```

```

* @param d matrix of distances
* @param p matrix of predecessors
*/
for (int k = 0; k < d.length; k++) {
    for (int i = 0; i < d.length; i++) {
        for (int j = 0; j < d.length; j++) {
            if (d[i][k]==Integer.MAX_VALUE||d[k][j]==Integer.MAX_VALUE){
                continue;
            }
            if (d[i][j] > d[i][k] + d[k][j]) {
                d[i][j] = d[i][k] + d[k][j];
                p[i][j] = p[k][j];
            }
        }
    }
}

```

The algorithm consists of three for loops and it compares all paths in graph between each vertices. It works well but with more density graph the asymptotic complexity rises up and this may causes that the algorithm is too slow to solve the shortest path problem.

The algorithm returns shortest path value among all pairs of vertices and there must be constructed the matrix of predecessors to get the output shortest path.

6.5 Dijkstra Algorithm

The Dijkstra algorithm was published in 1959 by dutch scientist Edsger W. Dijkstra. This algorithm can find the single-pair shortest path between vertices in graph and it exists in many kind of implementation. The edges can't be negative length.

We can say that this algorithm is basically kind of form of breadth-first search algorithm, in which the order of traversed noded is not determined by number of edges from the root, but as a distance from the root. As a consequence Dijkstra's algorithm process only those node, for which the shortest path was already discovered. The algorithm stores all nodes in a priority queue ordered by distance of the node from the root – in the first iteration of the algorithm, only root has distance set to 0, distance of all other nodes is equal to infinity. Than in each step Dijkstra's algorithm picks from the queue a node with the highest priority (least distance from the root) a processes it and reevaluates distances of all unprocessed descendants of the node[23; Kolář, 2004].

The algorithm has without modification an asymptotic complexity as $O(|V|^2)$, where V is

number of vertices. If there will be used binary heap in implementation of priority queue then the complexity will be as $O(|E| \cdot \log_2|N|)$, where E is number of edges so I decided to use binary heap to solve shortest path problem in our application.

6.5.1 Priority Queue

Dijkstra using a Heap is one of the most powerful techniques. It essentially allows you to write a Breadth First search, and instead of using a Queue you use a Priority Queue and define a sorting function on the nodes such that the node with the lowest cost is at the top of the Priority Queue. This allows us to find the best path through a graph in $O(m * \log(n))$ time where n is the number of vertices and m is the number of edges in the graph [Gladius].

The fundamental operations on a Heap are:

Add - Inserts an element into the heap, putting the element into the correct ordered location.

Pop - Pops the top element from the heap, the top element will either be the highest or lowest element, depending on implementation.

Top - Returns the top element on the heap.

Empty - Tests if the heap is empty or not.

6.6 Database of Charging Stations

For speed performance and limitation of Google APIs there was created a database of charging stations and paths between them. If we consider actual state in Czech Republic, there are more than 65 charging stations (with charging power more than 20 kW). To count all paths between all stations then must be asked $(65^2) - 65 = 4,160$ queries to Google's servers but the daily limit is set up to 2,500 tasks. There could occur a question what happens if the count of charging stations is not 65 but 1,000. It would work anyway because the algorithm wouldn't take into account all of them. This count depends on driving range and also it depends on the distribution of charging stations. If exists a large number of charging stations then there could be considered only stations with high-current charging power.

The database is in JSON format and saved to the external storage on Android device. Following is shown the structure of JSON database:

```
{
  "0": {
    "name": "Vystrkov u Humpolce",
    "location": {
      "lat": "49.525250",
      "long": "15.347449"
    },
    "power": "50",
    "distances": {...}
  },
  "1": {
    "name": "MC Praha 16 CEZ",
    "location": {
      "lat": "49.984975",
      "long": "14.363754"
    },
    "power": "50",
    "distances": {...}
  }
}
```

Each JSON object has UUID defined as number from 0 increased by 1 with each new station. This object has attribute name of station, location defined by latitude and longitude coordinates, charging power defined in kW and object named distances which contains routes between this station to all other besides itself and besides stations with distance more than 250 km from this station because if we consider maximal vehicle's range set up to 250 km then it does not make sense to get path between stations more than maximal range nevertheless this is only parameter which can be changed as required.

There could be added new station by adding that at the end of JSON file. In the application there will be a button "Update DB" which update all new added stations to each other in database.

Now there is a need for asking Google server in the algorithm only for the nearest stations around start and the destination. The database will be asked for stations between start and the destination with much faster speed response.

6.7 Route Planning With Charging to 100 %

The first approach with good speed performance is route planning through stations with charging to the full state of charge on each of them no matter what for power of charging the station provides. There will be focused always on the the shortest time path planning. The following Fig. 6.4 shows us how the path can be discovered.

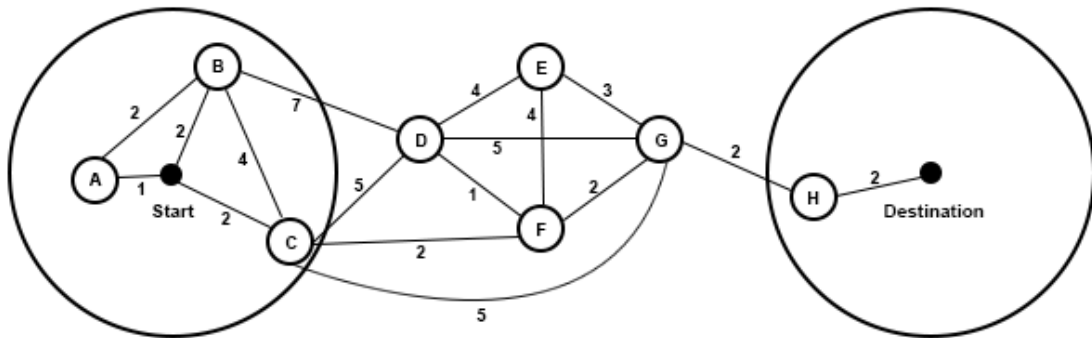


Figure 6.4: The circles indicate vehicle's driving range. We can get from start to stations named A, B and C. The only available station in this example near by driving range of destination is the only one named H. The algorithm must in this situation ask Google's servers four times (station A, B, C and H) and then could be obtained the path between all possible stations with help of charging stations database. All displayed edges in the graph are in the distance up to the maximum driving range. The cost of each station depends on edge which leads into this station plus cost of station where the edge starts. Every edge has different length of distance to the station and it means that each edge discharges the battery differently so the cost function on station depends on each edge.

Cost function on each station vertex:

$$\text{cost}(\text{station}) = \text{edgeDistance} + \text{stationWeight}$$

To get the best path from start to destination was used above described Floyd-Warshall algorithm. This algorithm can't deal with any cycles of negative edges. This assumption is fulfilled in the route planning problem.

6.7.1 Disadvantage of this Approach

This algorithm provides good speed results but the problem is that the result can be more accurate. We don't need to charge the battery to the full state of charge always on the

road and this is the biggest problem of this approach and this will be discussed in the next chapter.

6.8 More Accurate Route Planning

Let's consider the following situation. The best path has first charging station with low-current charging power and the vehicle will be charged to the full state of charge. It takes too long time although there could be possible to get from this station with lower SoC to the next charging station. The next station could be a high-current charging station and in this situation can be the battery loaded much faster. There cannot be told, how much exactly should be the battery loaded to get the best result and this parameter is unknown.

On the beginning there will be considered only three possible of charging states (100 %, 75 % and 50 % state of charge) on each station in this approach. Each vertex will be divided into three different vertices with the same station's parameter but with different state of charging. There could happen that the edge may not exist because of lower SoC and the algorithm must take this information into account.

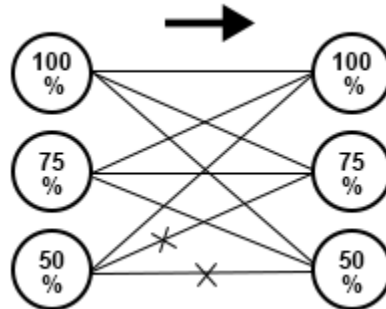


Figure 6.5: Edges may not exist.

In Fig. 6.6 is shown an example of three different charging states. It is obvious that the complexity rises up. There are only four stations but 12 vertices and 78 edges (both directions must be considered besides edges leading from start and to destination vertex). It leads to more complicated graph and the asymptotic complexity rises up exponentially.

The number of edges can be computed as follows:

n = number of all possible states of charge

E = number of edges

S = number of stations in driving range distance from start

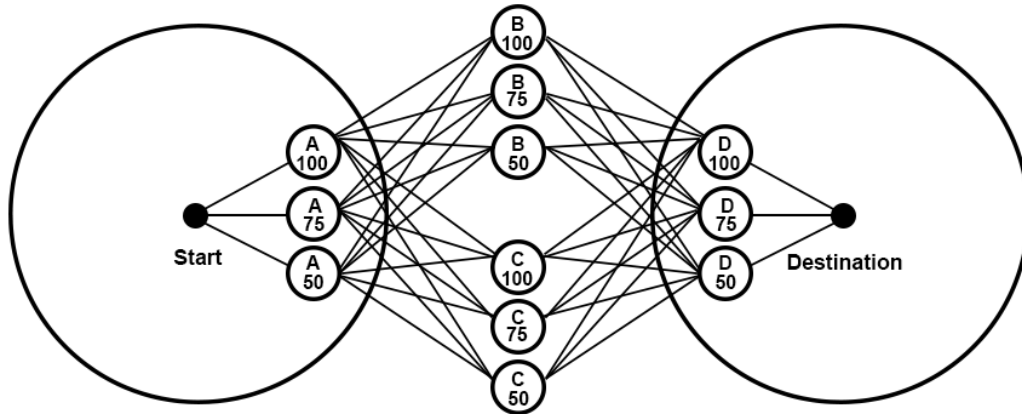


Figure 6.6: Route planning with 3 different states of charge.

D = number of stations in driving range distance from destination

V = number of possible vertices (stations)

$$E = (n * S) + (V * n^n) + (n * D)$$

With greater asymptotic complexity I had to use other algorithm approach because the speed of Floyd-Warshall is not fast enough for this complexity if were considered more than 65 possible stations in the Czech Republic and 3 states of charge on each station. The best algorithm I can use is already described Dijkstra algorithm. For best results I use priority queue to get the path fast.

6.8.1 More Diversified State of Charge

For more accurate solution I tried to increase the number of states of charge of the battery. The minimum state of charge is defined as 15 percent. We have experimented with different settings and in conclusion, I use following set of states of charge:

$$SoC = \{0.15, 0.2, 0.25, 0.3, 0.35, 0.4, 0.45, 0.45, 0.5, 0.55, 0.6, 0.65, 0.7, 0.75, 0.8, 0.85, 0.9, 0.95, 1\}$$

The value is multiplied with 100 to get value in percent.

6.9 Time Spent on Battery Charging

The battery needs to be charged on the station and time necessary to charge it depends on charging power of station and the amount of energy needed to be charged. Following will be defined equation of charging the battery. Let's consider that the progress of charging has linear progression.

CH = charging power of station (kW)

T = time spent on the station (hr)

B = capacity of battery (kWh)

A = actual state of charge (kWh)

C = average consumption (kWh/100 km)

D = distance (m)

E = kWh needs to be charged

P = state of charge to be load (%)

INPUT = actual SoC (A), required SoC (P)

OUTPUT = time spent on station (T)

$$E = (B \cdot P) - (A - ((C/100000) \cdot D))$$

$$T = \frac{E}{CH}$$

6.10 Speed Performance of Both Used Algorithms

Both algorithms will be compared on real graph shortest path problem. The main advantage of Dijkstra algorithm is that it search only for single-pair vertices path instead of all-pair vertices path that are provided by Floyd-Warshall algorithm. By using priority queue in Dijkstra algorithm, the algorithm runs yet much faster.

In Fig. 6.7 are compared both algorithms in three situations. The first situation is charging to 100 percent, the second situation is charging to 50, 75 and 100 percent and the last one (the most time-consuming) is charging from 15 to 100 percent with a 5 percent step.

Each result shown in the graph is the average time needed to get the shortest path for 10 real route planning results in which there are more than 65 charging stations in the graph and the test was tested on Samsung Tab S2 T-815 with Android OS version of 6.0.1.

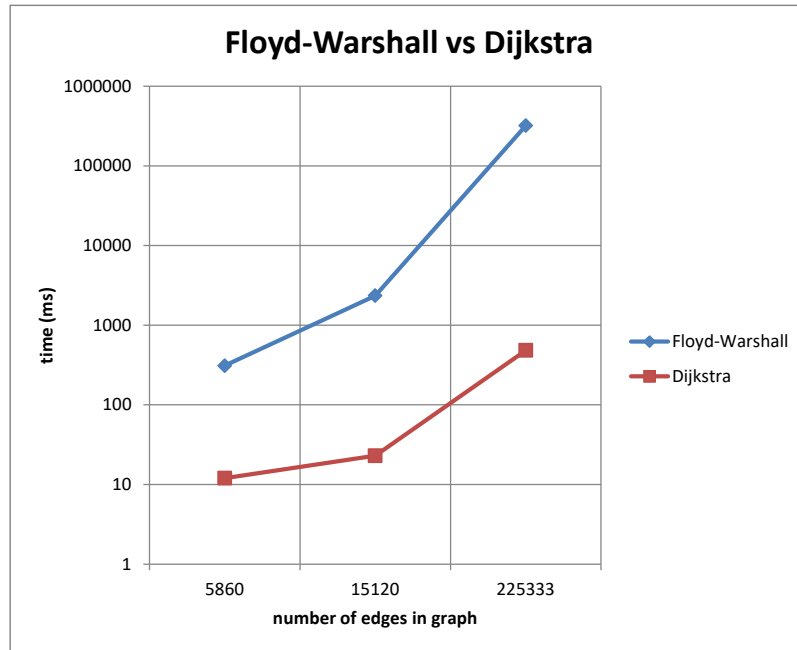


Figure 6.7: Route planning with 3 different states of charge.

In Fig. 6.7 is on the left side logarithmic scale. We can see that the biggest difference rises up with bigger number of edges and the Floyd-Warshall algorithm is inappropriate for the third situation of accuracy. The result time using Floyd-Warshall algorithm in third situation takes several minutes to get the result so in the final application I use only Dijkstra algorithm with priority queue.

6.11 Heuristic Approach

A heuristic function can be used in this problematic, especially if there is a need to reduce the number of queries to Google servers. From the principle of route planning, I can consider in which direction, I mean by angle, is the destination and I do not have to consider using stations which are in the opposite direction, and that is true for the direction from the destination to the start position. In the following Fig. 6.8 is described this aforementioned thought.

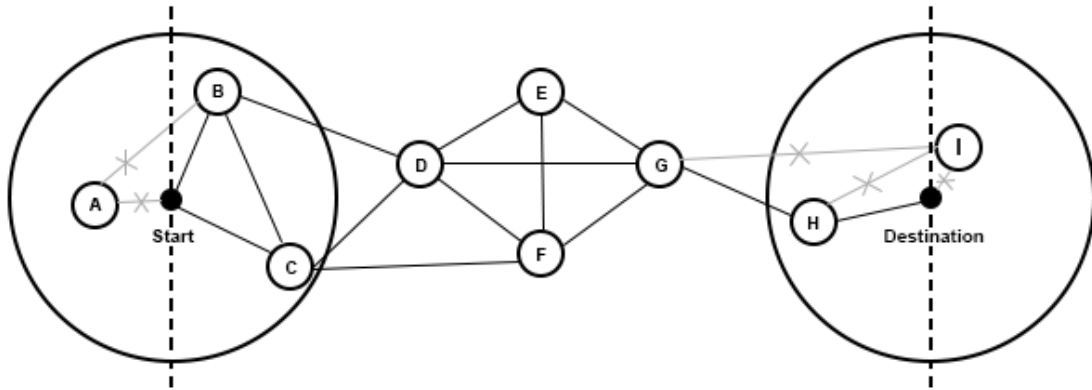


Figure 6.8: Heuristic approach.

The concept of this approach is to split the driving range circle into two sides split by the angle of 180 degrees. There are considered all stations if, and only if, they are within an angle of 180 degrees to the destination and vice versa. The charging stations which are behind this imaginary line, will be not used. With this approach many queries can be reduced to Google servers which could be equally unnecessary to get the shortest path and because the queries to the remote servers take a while, there will be reduced also the time of running the algorithm.

I tried this approach in the already developed algorithm, and it significantly reduced the number of queries to Google servers, but I came across a serious problem in this approach. There can occur several problems. If I consider the start position, then the vehicle's battery may be discharged, and if there is a station behind the imaginary line which will be not accepted, then this algorithm says that there is no charging station nearby to the start position, and the vehicle must charge the battery at home with the low-current station. The other problem can be that there might be a high-current charging station behind the imaginary line and the algorithm will select another with a lower charging power instead. This problem could be solved if I were to consider all high-current stations as inclusive space behind the line. There can, of course, occur a problem in reaching the destination. The outcome of the algorithm can be that the destination wouldn't be reached, despite the fact that behind the imaginary line could be a charging station which I could have used to reach the goal. Another problem could be, that in this algorithm I consider the parameter SoC in destination, and if I do not take the route through the high-current station, I will have to charge the battery with the low-current station at the destination.

In summary, I decided not to use this approach, although the number of queries would be reduced. We could have used this approach in the situation where the network of stations rises up and we must search in more depth to graph them, but for now the improvement would be at the expense of the accuracy our algorithm.

7 Implementation into the Delivered Application

The initial application, from Porsche Engineering Services s.r.o., provides the necessary java-based code which already works well. The connection between the tablet and this app is done by reading CAN signals in the vehicle, and transmitting them to the tablet per Bluetooth communication protocol. The class diagram of the application, with developed code, is displayed in appendix C of this thesis.

7.1 Reading Signals

For the purpose of this thesis, the app reads only two CAN signals and those are the "Remaining driving range" and the "Average Consumption". The following example of code was used to read these values in the algorithm.

```
SignalValue drivingRange = signalMgr.getSignalValue("Energy",  
NiceNameStorage.CUSTOMREMAINING_DRIVING_RANGE);
```

```
SignalValue avgConsumption = signalMgr.getSignalValue("Energy",  
NiceNameStorage.CUSTOMAVERAGE_CONSUMPTION);
```

During the implementation, there was a problem with this value because it didn't correspond to the reality (the driving range algorithm wasn't calibrated) so, I was forced to edit this signal so that the average consumption of the vehicle was statically defined as 20 kWh/100 km, and the algorithm responsible for getting this signal value was corrected.

7.2 Example of Implementation to the App

The developed application was implemented as Google Map Fragment with widget properties. The controls are inside the fragment and it inherited the Graph object from the delivered app. This Graph object has built functions and these functions allow that the object can be resized and moved all over the screen. It means that the Google Map Fragment is not implemented as a static object. It is a more user-friendly solution. There are several layouts in the app, and there was created the new one named Google Map. An example is shown in Fig. 7.1 .

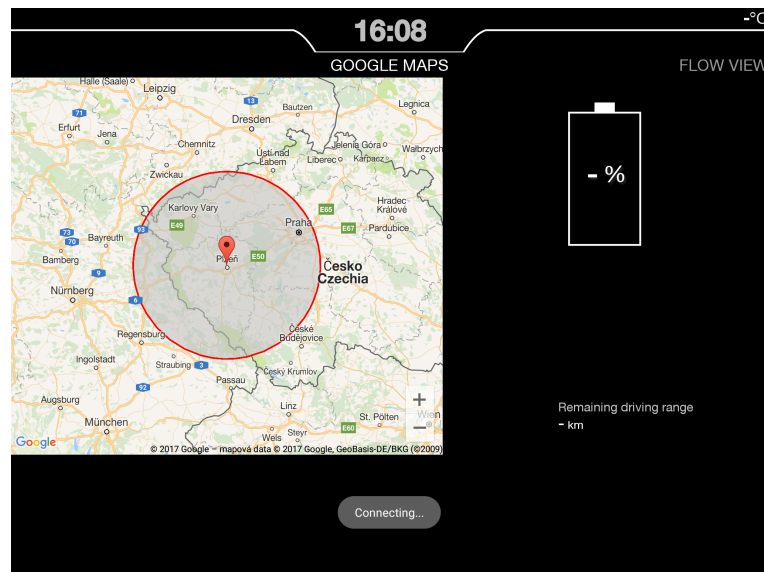


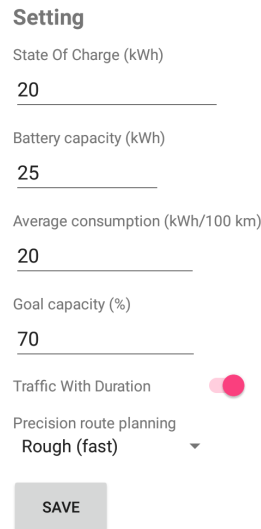
Figure 7.1: Interface of the developed app and its implementation to the delivered app. It shows us how the app looks after implementation of the Map Fragment into it. This is only a basic screenshot without a connection to the Demo vehicle, so there cannot be seen any data on the map or from the other widgets next to it.

7.3 The Interface of Developed HMI

Now, there will be described how the interface looks and how can it be controlled.

7.3.1 Options

In Fig. 7.3 on the top left of the Map Fragment, there is a button with an icon gear which leads to the “Setting” popup window as follows.



Setting

State Of Charge (kWh)
20

Battery capacity (kWh)
25

Average consumption (kWh/100 km)
20

Goal capacity (%)
70

Traffic With Duration

Precision route planning
Rough (fast) ▼

SAVE

Figure 7.2: Settings for the Developed Application.

From the top in Fig. 7.2, we can see the value for the SoC. This value is here for test purposes and will be inactive when connected to the Demo vehicle because this value will be obtained from the CAN signal of the vehicle. The next value defines the battery capacity. This value can be changed according to the vehicle we are connected with. The average consumption value will be also inactive when connected to the vehicle and we also get it from the CAN signal of the vehicle. The next parameter is Goal capacity. This parameter means how much energy should be once the destination will be arrived, which the algorithm takes it into account. I consider as possible that the vehicle can always charge the battery at the destination, but with a low-current charger (22kW or less).

The next parameter is a switch named, "Traffic With Duration". This parameter allows us to return actual traffic conditions through a planned route, and we can obtain a more accurate result using it, because the database of charging stations was updated at different times and without traffic information included. Unfortunately, we don't obtain exact information about possible delays but only time information about how long the trip would take. Google does not provide such detailed information nowadays.

Finally, there can be chosen to configure how fast and accurate we want to obtain the results using the developed algorithm. There are three options (fast, medium, or very accurate), as described already in chapter chapter 6 of this thesis.

7.3.2 Route Planning

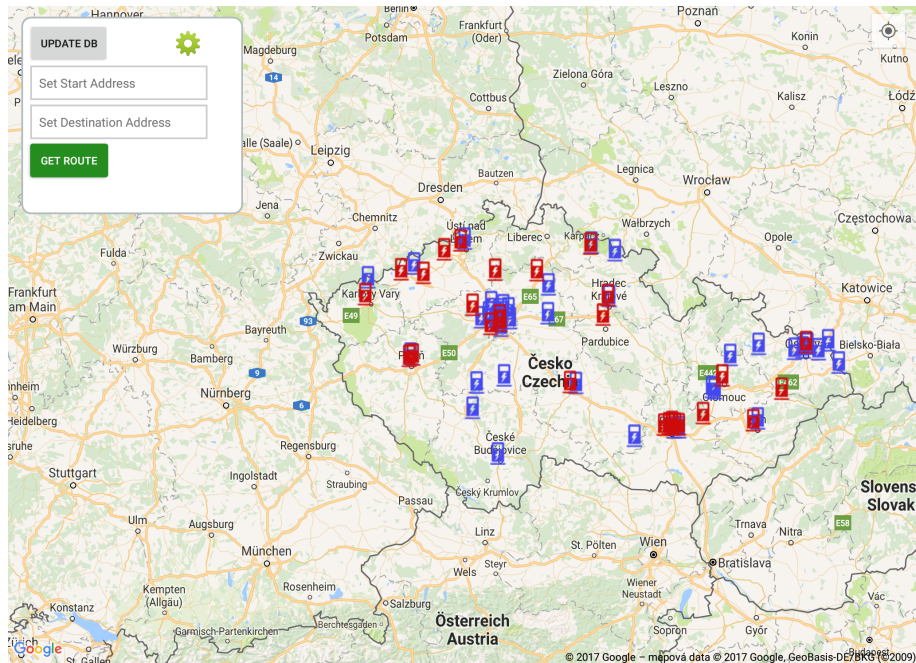


Figure 7.3: The basic appearance of the developed application.

In Fig. 7.3 there is a basic screen of how the developed application looks. In the top left corner, we can see a small window where we can get into the settings and update the charging stations database. Below that, we have got the origin and destination position which may be written by the user. The green button runs the algorithm to get the route.

Charging stations are also displayed on the map. These are all stations that are currently publicly available in the Czech Republic. The stations with blue color dispense 22 kW charging power and the red ones have charging power of 50 kW. For test purposes, there will be some stations with black color and with charging power of more than 50 kW. These stations will be introduced as dummy to show the key features of the algorithm.

7.3.3 Displaying Route Information

If the "Get Route" button will be clicked, then detailed route information will be obtained on the map and it also draws the path from origin to the destination with red color, as shown in Fig. 7.4

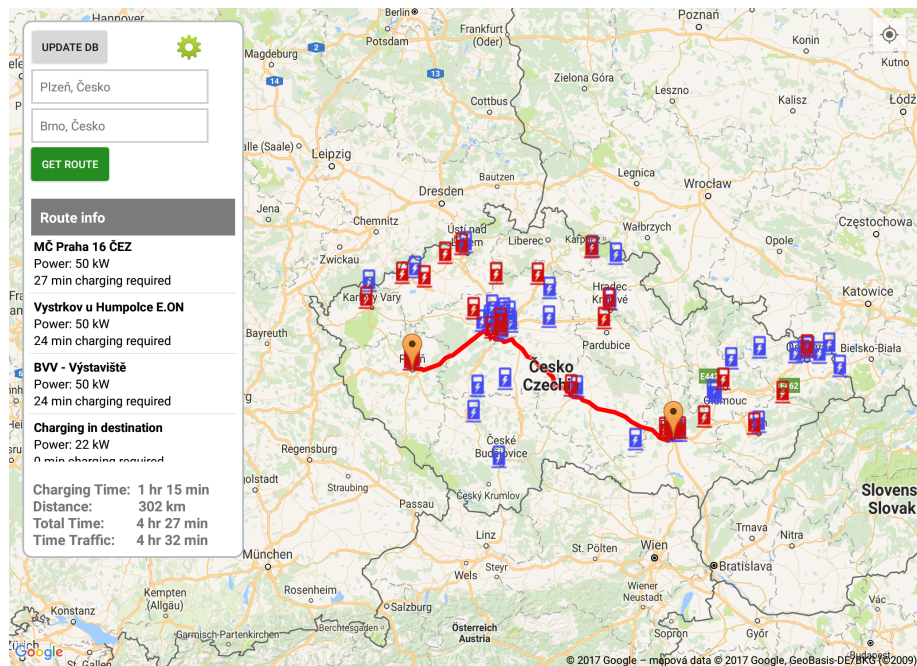


Figure 7.4: On the left column, we can read detailed information about the path. There is a list of charging stations which need to be visited on the road with information such as name, charging power and how long is needed to stop to charge, in order to be charged enough to reach the next stop. On the bottom, there is a summary of the trip. The first information is how long the charging takes on the whole trip. The distance is shown as well. The next information is “Total Time,” including charging time, and the last shown information is the already mentioned “Time with Traffic.”

8 Testing

In this chapter, all developed algorithms will be tested and also the implementation will be tested in the delivered application. The first tests will be only locally, with Software in the loop approach, and then the algorithms will be tested also on the real Demo vehicle.

8.1 SW in the Loop

8.1.1 Test Driving Range Polygon

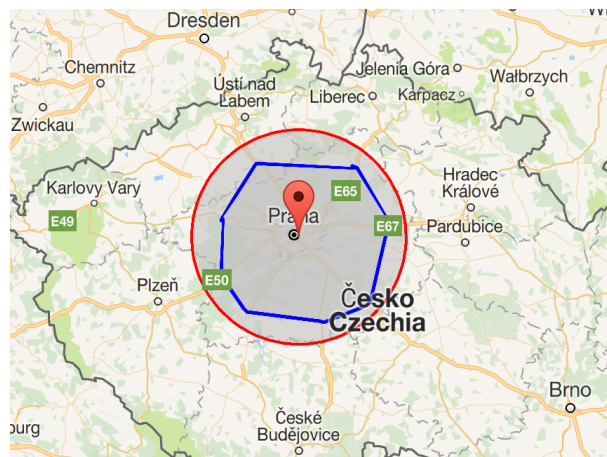


Figure 8.1: Driving range polygon.

Several tests were performed to prove that the algorithm for displaying the driving range as the polygon works. The picture above shows how the polygon can look. If the API limit is reached, then the red circle will be displayed with great-circle distance driving range only.

8.1.2 Test Route Planning Algorithm

In this section, there are shown several examples of route planning with a different setting. We demonstrate interesting examples to prove flawlessness of the algorithm. We also compare the accuracy of the algorithm.

Table 8.1: Plzeň -> Brno (Rough Accuracy)

SoC	Battery Capacity	Avg. consumption	Goal capacity
20 kWh	25 kWh	20 kWh	70 %
Name of charging station	Charging power	Charging required	
MČ Praha 16 ČEZ	50 kW	27 minutes	
Vystrkov u Humpolce E.ON	50 kW	24 minutes	
BVV - Výstaviště ČEZ	50 kW	24 minutes	
Charging time	Distance	Total Time	Traffic Time
1 hr 15 min	302 km	4 hr 27 min	4 hr 28 min

Table 8.2: Plzeň -> Brno (Very Accurate)

SoC	Battery Capacity	Avg. Consumption	Goal Capacity
20 kWh	25 kWh	20 kWh	70 %
Name of Charging Station	Charging Power	Charging Required	
MČ Praha 16 ČEZ	50 kW	23 minutes	
Vystrkov u Humpolce E.ON	50 kW	24 minutes	
BVV - Výstaviště ČEZ	50 kW	21 minutes	
Charging time	Distance	Total Time	Traffic Time
1 hr 8 min	302 km	4 hr 20 min	4 hr 22 min

Table 8.3: Plzeň -> Brno (Very Accurate)

SoC	Battery Capacity	Avg. Consumption	Goal Capacity
20 kWh	25 kWh	20 kWh	70 %
Name of Charging Station	Charging Power	Charging Required	
Testing charger	150 kW	9 minutes	
Vystrkov u Humpolce E.ON	50 kW	17 minutes	
BVV - Výstaviště ČEZ	50 kW	21 minutes	
Charging time	Distance	Total Time	Traffic Time
47 min	304 km	4 hr 2 min	4 hr 4 min

Table 8.4: Plzeň -> Jindřichův Hradec (Medium Accuracy)

SoC	Battery Capacity	Avg. Consumption	Goal Capacity
25 kWh	25 kWh	20 kWh	0 %
Name of Charging Station	Charging Power	Charging Required	
MČ Praha 16 ČEZ	50 kW	21 minutes	
Vystrkov u Humpolce E.ON	50 kW	9 minutes	
Charging time	Distance	Total Time	Traffic Time
30 min	250 km	3 hr 24 min	3 hr 25 min

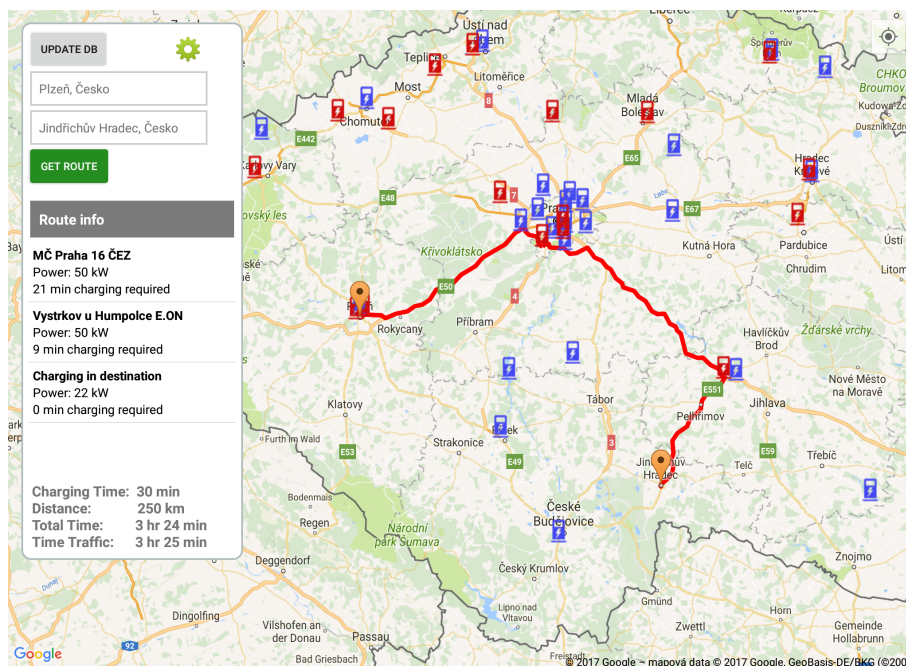


Figure 8.2: An example of route planning.

In Fig. 8.2 can be seen that the fastest route leads through Prague. This is caused because of highways along most of the path, and nothing less than high-current charging stations we stop over. If we drive across Pisek city, the path would be about 80 km shorter but it wouldn't be the fastest path, due to charging.

Table 8.5: Krnov -> Plzeň (Rough Accuracy)

SoC	Battery Capacity	Avg. Consumption	Goal Capacity
10 kWh	25 kWh	15 kWh	0 %
Name of Charging Station	Charging Power	Charging Required	
Lipina Luky system	22 kW	64 minutes	
Brno ČEZ	50 kW	17 minutes	
Vystrkov u Humpolce E.ON	50 kW	20 minutes	
Burger King Praha	50 kW	16 minutes	
Charging time	Distance	Total Time	Traffic Time
1 hr 57 min	461 km	7 hr 11 min	7 hr 36 min

Table 8.6: Krnov -> Plzeň (Very Accurate)

SoC	Battery Capacity	Avg. Consumption	Goal Capacity
10 kWh	25 kWh	15 kWh	0 %
Name of Charging Station	Charging Power	Charging Required	
Lipina Luky system	22 kW	9 minutes	
Oloumouc - OC Haná	50 kW	12 minutes	
Brno ČEZ	50 kW	19 minutes	
Vystrkov u Humpolce E.ON	50 kW	17 minutes	
Burger King Praha	50 kW	18 minutes	
Charging time	Distance	Total Time	Traffic Time
1 hr 15 min	462 km	6 hr 34 min	6 hr 58 min

In table 8.1.2 and 8.1.2, we can see the difference between the Rough Accuracy and the Very Accurate approach, where the Very Accurate provides a better result. We spent only the absolutely necessary time at the low-current charging station and then we drive to the faster charging station, which optimizes charging time and also demonstrates the power of the developed algorithm.

8.2 HW in the Loop - Demo Vehicle

At the beginning of the project, I visited the Demo vehicle, and together with my supervisor we defined the problematics and tasks needed to be done in this thesis. I was introduced to the Demo vehicle, especially how it connects to the Tablet device, and what information can we obtain from the CAN interface.

One thing is to do the tests locally at home, and very important is to do tests on the real vehicle to prove the flawlessness of developed algorithms. We had done several vehicles' visitations, in which we discovered a few programming bugs and subsequently I repaired them. There was a problem with the aforementioned problem of wrong value(s) in the Remaining Driving Range signal. The algorithm didn't count this value properly, and I was forced to set it as a static variable in such a way, so that the app was still reading the signal.

9 Conclusion

In this thesis, we faced several problems related to drawing driving range as a polygon on the map, designing and planning routes while considering charging stations and their properties, implementing all developed algorithms into the delivered application from Porsche Engineering Services s.r.o., and testing the algorithms and implementation on real Demo vehicles. At the beginning, we got to know the electromobility problematics through the researching of HMIs available on the market. We tried two EVs from different companies and we compared the HMIs they include. We also went into charging stations' problematics and we carried out our actual situation in the Czech Republic.

In next chapter we focused on the problem of displaying driving range on the map as a polygonal shape. We went through the Google APIs and described how they work and how they can be used for our purpose. After that, there were discussed the limitations of this approach.

Chapter number six was about designing the route through charging stations. We implemented and tested two different algorithms and the problem was designed from scratch to the required state, with consideration of the varying charging power of the stations.

The next chapter, we dealt with implementing all designed algorithms and interface into delivered application based on Android application. We created our own interface and connected it with the mentioned app.

In the last chapter, we accomplished several tests locally and also with the Demo vehicle, and we connected needed signals to our algorithms. We created simulations and we proved the flawlessness of our developed algorithms.

There could be done also several extensions in the future work such as adding elevation profile to get more accurate driving range. The next upgrade could be about creating driver's profile to take into account what style he uses and nevertheless there could be upgraded the heuristic function in route planning.

In summary, all tasks in this thesis have been successfully accomplished and I regard this work as successful.

Bibliography

- [23] Dijkstra's algorithm. . – URL <https://www.programming-algorithms.net/article/45514/Dijkstra's-algorithm>
- [26] : *JSON In Java*. – URL <https://mvnrepository.com/artifact/org.json/json>
- [Bakker 2013] BAKKER, Sjoerd: Standardization of EV Recharging Infrastructures. (2013). – URL http://archive.northsearegion.eu/files/repository/20140805153226_StandardizationofEVRecharginginfrastructure.pdf
- [Burnette 2009] BURNETTE, Ed: *Hello, Android introducing Google's mobile development*. Dallas, Texas : The Pragmatic Bookshelf, 2009
- [Chakraborty 2016] CHAKRABORTY, Arnab: Asynchronous HTTP Requests in Android Using Volley. (2016). – URL <http://arnab.ch/blog/2013/08/asynchronous-http-requests-in-android-using-volley/>
- [Gladius] GLADIUS: Section 3: Finding the Best Path through a Graph. . – URL <https://www.topcoder.com/community/data-science/data-science-tutorials/introduction-to-graphs-and-their-data-structures-section-3/>
- [Google 2017a] GOOGLE: Google Maps APIs - documentation. (2017). – URL <https://developers.google.com/maps/android/>
- [Google 2017b] GOOGLE: *Google Maps Directions API*. 2017. – URL <https://developers.google.com/maps/documentation/directions/>
- [Google 2017c] GOOGLE: *Google Maps Distance Matrix API*. 2017. – URL <https://developers.google.com/maps/documentation/distance-matrix/>
- [Google 2017d] GOOGLE: *Google Maps Places API*. 2017. – URL <https://developers.google.com/places/android-api/>
- [Google 2017e] GOOGLE: HttpURLConnection. (2017). – URL <https://developer.android.com/reference/java/net/HttpURLConnection.html>
- [Hliněný 2010] HLINĚNÝ, Petr: *Základy teorie grafů*. Brno : Masarykova Univerzita, 2010

- [Horčík 2017] HORČÍK, Jan: V Evropě se objevují první nabíječky 350 kW CCS Combo. (2017). – URL <http://www.hybrid.cz/v-evrope-se-objevuji-prvni-nabijecky-350-kw-ccs-combo>
- [KELAG 2017] KELAG: *Charging stations*. 2017. – URL <https://ev-charging.com/at/en/elektrotankstellen>
- [Kolář 2004] KOLÁŘ, Josef: *Teoretická informatika. 2. ed.* Praha : Česká informatická společnost, 2004
- [Paquet 2017] PAQUET, Josée: Alliance Renault-Nissan World's Top Maker of EVs. (2017). – URL <https://www.auto123.com/en/news/nissan-leaf-best-selling-electric-car/63342/>
- [Park 2015] PARK, Jaehyun: Dynamic programming. (2015). – URL <https://web.stanford.edu/class/cs97si/04-dynamic-programming.pdf>
- [Rick 1999] RICK: Deriving the Haversine Formula. (1999). – URL <http://mathforum.org/library/drmath/view/51879.html>
- [Rodrigue] RODRIGUE, Dr. Jean-Paul: The Great Circle Distance. . – URL <https://people.hofstra.edu/geotrans/eng/ch1en/conc1en/greatcircle.html>
- [Tesla 2017a] TESLA: *Tesla Model S*. 2017. – URL <https://www.tesla.com/models/>
- [Tesla 2017b] TESLA: *Tesla Supercharger*. 2017. – URL <https://www.tesla.com/supercharger>
- [Union 2016] UNION, Electromobility: *EUmap Android App*. 2016. – URL <https://play.google.com/store/apps/details?id=cz.shmoula.uemap>
- [Veness] VENESS, Chris: Calculate distance, bearing and more between Latitude/Longitude points. . – URL <http://www.movable-type.co.uk/scripts/latlong.html>
- [Verheijen 2016] VERHEIJEN, John: World Geodetic System 1984 (WGS84). (2016). – URL <https://confluence.qps.nl/pages/viewpage.action?pageId=29855173>
- [Wikipedia 2016] WIKIPEDIA: Great Circle Distance. (2016). – URL https://en.wikipedia.org/wiki/Great-circle_distance
- [Wikipedia 2017] WIKIPEDIA: Charging station - Standards. (2017). – URL https://en.wikipedia.org/wiki/Charging_station#Standards

A Google Maps API

A.1 How to Set Up the Google Maps API

- Install the Android SDK
- Install and configure the Google Play services SDK, which includes the Google Maps Android API. Note: If you use the Google Maps Android API with a Google Maps APIs Premium Plan license, you must download and configure the Premium Plan SDK instead.
- Get an API key. To do this, you will need to register a project in the Google API Console, create an API key, and restrict the API key using your app's signing certificate.
- Add the required settings to your application's manifest.

A.2 Modify the Application's Manifest

It is necessary to modify the application's manifest file to run the Google Maps in our app. Specified must be the Google Play services version number, so adding the following code in the manifest file.

Add the following declaration within the <application> element of AndroidManifest.xml. This embeds the version of Google Play services that the app was compiled with.

```
<meta-data
    android:name="com.google.android.gms.version"
    android:value="@integer/google_play_services_version" />
```

A.3 How to Obtain a Google API Key

- First you must create an Google account. Then after logged in you must register your app project on the Google API Console (<https://developers.google.com>).
 - 1) Go to the Google API Console
 - 2) Create or select a project
 - 3) Click Continue to enable the Google Maps Android API
 - 4) On the Credentials page, get an API key

- You need to include your API key in application's manifest, contained in the file AndroidManifest.xml
 - 1) In AndroidManifest.xml, add the following element as a child of the <application> element, by inserting it just before the closing </application> tag:

```
<meta-data
  android:name="com.google.android.geo.APIKEY"
  android:value="YOUR_APIKEY"
/>
```

- 2) Save AndroidManifest.xml and re-build your application.
- Before use any API you need to enable it first in Google API Console.

B Application's Manifest

For use of GPS provider, the permission need to be declare for precision location as follows:

```
<manifest ... >
  <uses-permission
    android:name="android.permission.ACCESS_FINE_LOCATION" />
  <uses-feature
    android:name="android.hardware.location.gps" />
  ...
</manifest >
```

For use of Network provider, the permission need to be declared for coarse location in app's manifest file as follows:

```
<manifest ... >
  <uses-feature
    android:name="android.hardware.location.gps" />
  ...
</manifest >
```

C The Class Diagram

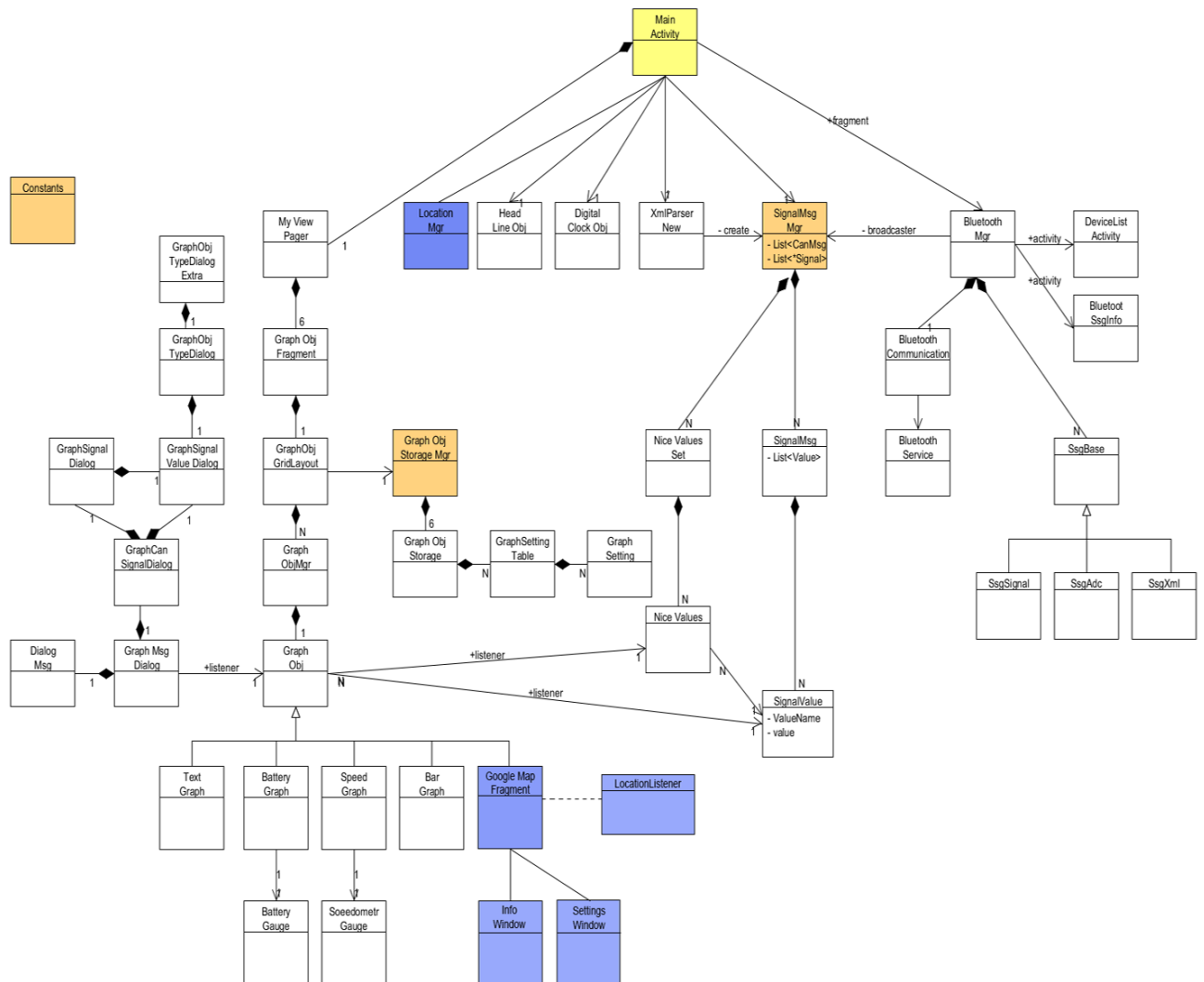


Figure C.1: The class diagram of the application. The blue marked objects are representing new code added to the app.