e.GO Digital Hackathon

Team: Hugs for bugs

2019-05-24

Contents

Chapter 1

Task

The e.GO life is generating, collecting and processing serveral data per second. Those data could be very valuable for us. Especially when you think of developing new innovative business models. Your task is to develop value-adding services for the e.GO Life and its user considering the provided data.

Chapter 2

Value-adding services

Our services should support at least the following functions:

- Intelligently choose the parking lot
  - location convenient
  - cheap price
  - support egoPay
- Shortly reservation for the parking lot
  - estimating the arrival time
  - automatically reserve the parking space in advance
  - message notification
- Immediately payment – egoPay
  - no worry about parking time
  - automatic charging fees while leaving
- Ego drive-thru
  - food reservation from supermarket
  - food pick up from parking lot
  - food payment by egoPay

Chapter 3

Dataset Overview

3.1 We prepare a dataset, as shown below

```
import pandas as pd
df = pd.read_csv('location.csv', sep='\t', index_col=0)
df
```

```
##   location_name   latitude  longitude  price/hour  is_outdoor  ego_pay
## 0          e.Go  50.781624   6.046581           2           1        1
## 1       Super C  50.778430   6.078702           3           1        0
## 2        Bushof  50.777557   6.090270           4           1        0
## 3  Hauptbahnhof  50.767911   6.091582           4           0        1
## 4    Euregiozoo  50.763672   6.115565           3           1        1
```

- location_name: the parking plot location
- latitude: latitude of the parking plot
- longtitude: longtitude of the parking plot
- price/hour: price to be charged per hour
- is_outdoor: is the parking plot outdoor or inside
- ego_pay: does the parking plot support "EgoPay"

3.2 Get current location, update location

```
import requests
class EgoCar:

    def __init__(self, api_url):
        self.api_url = api_url
        self.api_key = '7443363c-4304-4c56-9df0-6af4af40c613'
        self.header = {'Content-type': 'application/json', 'X-Api-Key': self.api_key}

    def get_location(self):
        res = requests.get(self.api_url, headers=self.header)
        location = res.json()['location']
        return location
```

```python
    def update_location(self, new_location):
        x, y = new_location.split(',')
        x, y = float(x), float(y)
        if x >= -90 and x <= 90 and y >= -90 and y <= 90:
            data = {'location': new_location}
            requests.patch(self.api_url, json=data, headers=self.header)
            print('new location: ', new_location)
        else:
            print('[Error] Latitude value must be between -90 and 90')
```

- get current location

```python
api_url = 'https://ego-vehicle-api.azurewebsites.net/api/v1/vehicle/signals'
ego_car = EgoCar(api_url)
print('current location: ', ego_car.get_location())
```

```
## current location:  50.00, 6.00
```

- update location

```python
ego_car.update_location('50.00, 6.00')
```

```
## new location:  50.00, 6.00
```

Chapter 4

Intelligent choose parking lot

Intelligent, we mean that we should use smart algorithms to choose the most appropriate parking lot, considering the distance, price and etc.

- Calculate Geographical distance

```
from geopy.distance import geodesic
from geopy.geocoders import Nominatim
class GeoDistance:

    def __init__(self):
        self.geolocator = Nominatim(user_agent="demo")

    def get_geo_position(self, location):
        addr = self.geolocator.geocode(location)
        return (addr.latitude, addr.longitude)

    def get_location(self, geo_position):
        location = self.geolocator.reverse(geo_position)
        try:
            addr = location.raw['address']
            return '{road}, {city_district}, {city}, {postcode}'.format(**addr)
        except:
            return location.address

    def calc_geo_distance(self, origin, destination):
        distance = geodesic(origin, destination).km
        return distance
```

- First get the current location

```
api_url = 'https://ego-vehicle-api.azurewebsites.net/api/v1/vehicle/signals'
ego_car = EgoCar(api_url)
curr_loc = ego_car.get_location()
print('current location: ', curr_loc)
```

```
## current location:  50.00, 6.00
```

```
geo_dis = GeoDistance()
addr = geo_dis.get_location(curr_loc)
```

```
print('current address: ', addr)
```

```
## current address:  Kiischpelt, Canton Wiltz, 9776, Lëtzebuerg
```

- Calculate distance to parking lot

```
pd.options.mode.chained_assignment = None
df_dis = df[['location_name', 'latitude', 'longitude']]
origin = ego_car.get_location()
destination = zip(df_dis['latitude'].values, df_dis['longitude'].values)
df_dis['distance/km'] = [geo_dis.calc_geo_distance(origin, des) for des in destination]
df_dis.sort_values('distance/km', ascending=True)
```

```
##   location_name   latitude  longitude  distance/km
## 4    Euregiozoo  50.763672   6.115565    85.344859
## 3  Hauptbahnhof  50.767911   6.091582    85.667684
## 2        Bushof  50.777557   6.090270    86.730648
## 1       Super C  50.778430   6.078702    86.770605
## 0          e.Go  50.781624   6.046581    87.008247
```

- Smart choice (toy example, with different weight on attributes)
  - distance: 0.5
  - price: 0.2
  - is_outdoor: 0.1
  - egoPay: 0.2

```
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()

df_new = df[['price/hour', 'is_outdoor', 'ego_pay']]
df_new['is_outdoor'] = df_new['is_outdoor'].apply(lambda x : x ^ 1)
df_new['ego_pay'] = df_new['ego_pay'].apply(lambda x : x ^ 1)
df_weight = df_new.join(df_dis['distance/km'], how='inner')
# max-min scaler
norm_values = scaler.fit_transform(df_weight.values)
norm_values
```

```
## array([[0.        , 0.        , 0.        , 1.        ],
##        [0.5       , 0.        , 1.        , 0.85713399],
##        [1.        , 0.        , 1.        , 0.83311221],
##        [1.        , 1.        , 0.        , 0.19407646],
##        [0.5       , 0.        , 0.        , 0.        ]])
```

- the smaller value, the better choice

```
import numpy as np
cost = np.sum(norm_values * [0.2, 0.1, 0.2, 0.5], axis=1)
df_weight['cost'] = cost.T
df_weight.sort_values('cost', ascending=True)
```

```
##    price/hour  is_outdoor  ego_pay  distance/km      cost
## 4           3           0        0    85.344859  0.100000
## 3           4           1        0    85.667684  0.397038
## 0           2           0        0    87.008247  0.500000
## 1           3           0        1    86.770605  0.728567
## 2           4           0        1    86.730648  0.816556
```

It seems the best choice is **Euregiozoo**

Chapter 5

Google Map Visulization

- Code to add marker and direction

```
class GeoMaps:

    def __init__(self, curr_location):
        self.key = 'GoogleAPIKey'
        gmaps.configure(api_key=self.key)
        self.curr_location = curr_location
        self.fig = gmaps.figure(center=self.curr_location, zoom_level=12)

    def add_marker(self, marker_pos):
        markers = gmaps.marker_layer(marker_pos)
        self.fig.add_layer(markers)
        return self.fig

    def add_direction(self, origin, destination):
        to_destination = gmaps.directions_layer(origin, destination)
        self.fig.add_layer(to_destination)
        return self.fig

    def get_curr_location(self):
        return self.curr_location

    def update_location(self, new_location):
        self.curr_location = new_location
```

- Mark current location

```
knitr::include_graphics("img/google_marker.png")
```

- Draw car driving direction

```
knitr::include_graphics("img/car_direction.png")
```

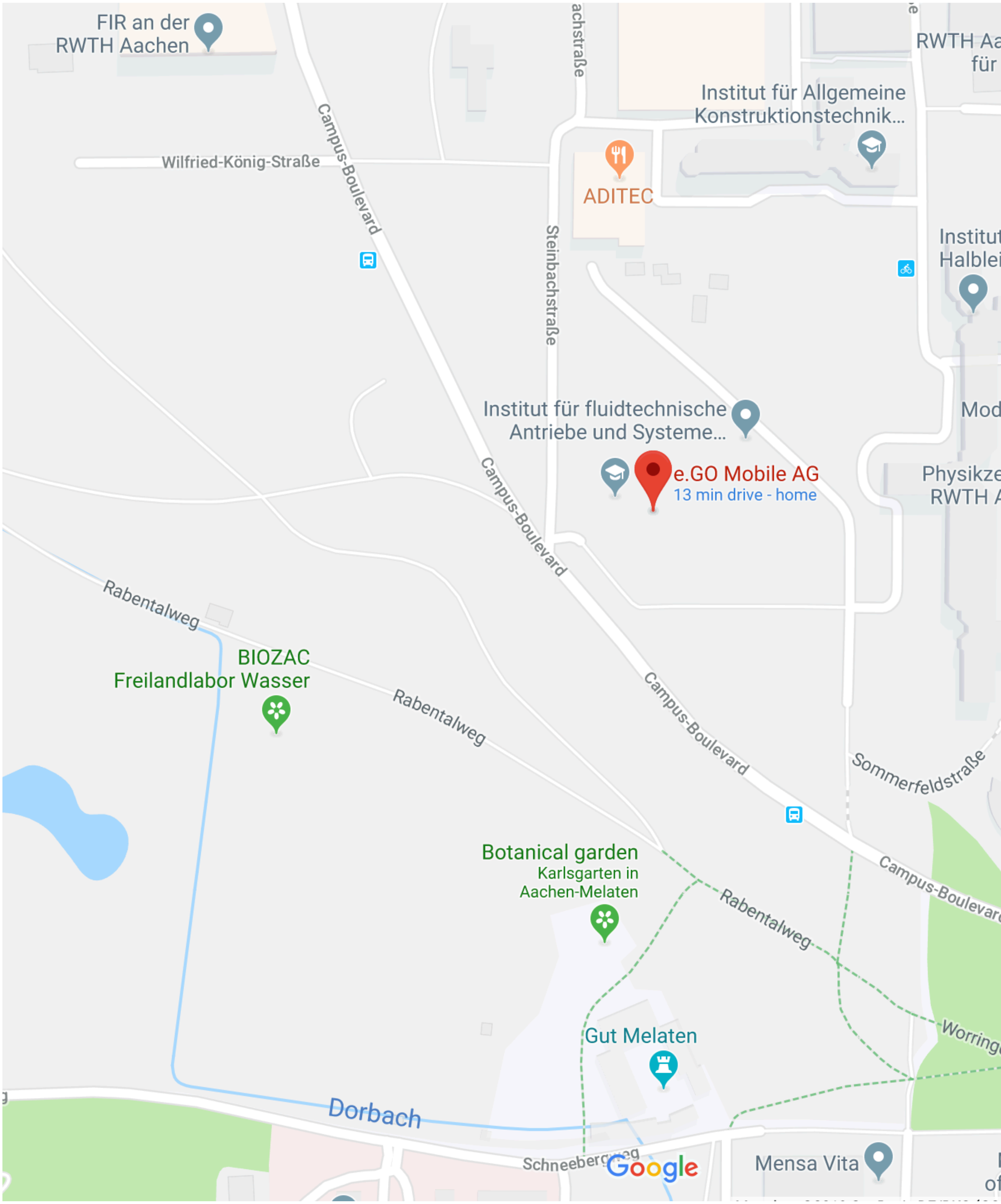

Figure 5.1: google map with marker

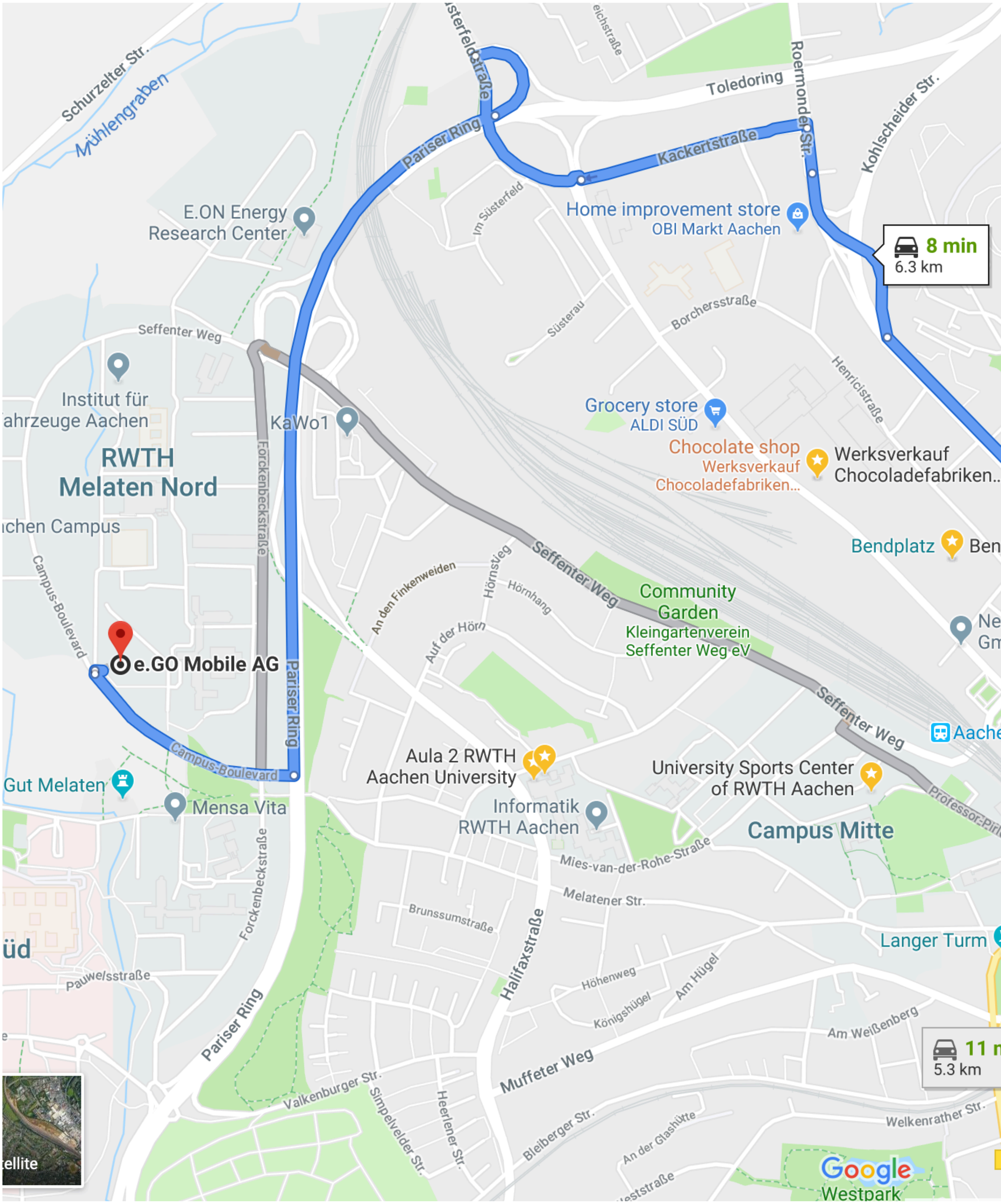

Figure 5.2: google map with car driving direction