Nemyrovska Maryna

maryna.nemyrovska@gmail.com

The visualization of the RSSI value for each sensor unit at all receivers over time

August 20, 2018

Contents

1 General information

- The amount of time spent on the challenge: 24 hours.
- Used libraries in *Python*:
  - json;
  - pandas;
  - numpy
  - re;
  - datetime;
  - dash, dash core components, dash html components;
  - networkx;
  - plotly.

2 Problem solution

Explanation of some notations:

readFiles()	the name of a function
all_files	the name of a variable

In order to see described data manipulation below (in subsections 2.1 - 2.3), please, open attached python notebook *'Sensors.ipynb'*.

2.1 Data processing

Since given data were of 'json' type, but wasn't 'json' file extension I decided to firstly read all files with the help of function **readFiles()** that returned the list of all files.

After that for every element of the **all_files** list, which is the whole file itself, e.g."01-003-0038cl", I applied function **sparseJSON()** to have a list of json files. This function uses regular expressions in order to find a pattern of a file of json type.

Following this, there was made a decision to use *DataFrame* to retrieve the tables by means of queries (to have an insight of the data). When all the data was already saved into the data frame I renamed all columns so that I can access the fields without errors, for example, directly after normalizing json files I had **message.payload.rssi** that was transformed into **message_payload_rssi**. In addition to this, I changed the data of the column 'sensors' in order to have a better string format, that will coincide further with the variable **message_paylod_suID**. As a result, I have obtained this kind of table:

	message_destination	message_payload_rssi	message_payload_suData_bat	message_payload_suData_sensor_config	message_payload_suData_temp	message_payload_suID	receiverid	request_responseTime	request_status	sensors	timestamp
0	v1/rec/02-004-0001/suStatus	-6	2960	192	23	01-003-0052	02-004-0001	92.814207	200	01-003-0052	1528701000.00
1	v1/rec/02-004-0001/suStatus	-6	3007	192	23	01-003-0052	02-004-0001	106.623888	200	01-003-0052	1528702000.00
2	v1/rec/02-004-0001/suStatus	-16	2960	192	23	01-003-0052	02-004-0001	99.778414	200	01-003-0052	1528697000.00
3	v1/rec/02-004-0001/suStatus	-12	3007	192	20	01-003-0052	02-004-0001	108.108044	200	01-003-0052	1528697000.00
4	v1/rec/02-004-0001/suStatus	-3	3007	192	23	01-003-0052	02-004-0001	92.387676	200	01-003-0052	1528700000.00
5	v1/rec/02-004-0001/suStatus	-6	2960	192	23	01-003-0052	02-004-0001	95.406294	200	01-003-0052	1528701000.00
6	v1/rec/02-004-0001/suStatus	-18	2960	192	23	01-003-0052	02-004-0001	93.272448	200	01-003-0052	1528697000.00
7	v1/rec/02-004-0001/suStatus	-15	3007	192	20	01-003-0052	02-004-0001	108.892679	200	01-003-0052	1528699000.00
8	v1/rec/02-004-0001/suStatus	-4	2960	192	23	01-003-0052	02-004-0001	106.69589	200	01-003-0052	1528701000.00
9	v1/rec/02-004-0001/suStatus	-15	2960	192	23	01-003-0052	02-004-0001	100.328922	200	01-003-0052	1528699000.00

2.2 Insight in data

There was implemented the function **printStats()**. After that I had the insight in data frame columns of type *float*, particularly on its range, mean, standard deviation. With regard to quantitative data, in other words the data of type *object*, the function gives information about its unique values.

2.3 Queries over data

I have implemented the query by means of pandas library that shows all the necessary information about received signal strength indicator (rssi) partitioned by **sensors, receiverid**.

The fetched table covers max, min, median and mean of **rssi**, the started value of a **timestamp**, the end value of a **timestamp** for every sensor at the receiver. (To see these tables, please, look into *'Sensors.ipynb'*.)

According to the fact that **timestamp** represents the real time when data was received at backend, I wrote a function **findTimeDifferenceDataFrame()**, which returns the data frame that has an additional column with the time difference.

To have more representative data with regard to its time columns, I implemented the following functions:

- Functions that changes **timestamp** value into the date format *"Y-m-d H:M:S"*
  - **getTime()**
  - **differenceTime()**
- Functions that add an additional column of a human friendly time value into the available data frame.
  - **changeTimestamp()**
  - **changeDifferenceInTime()**

  Also next functions represents the queries over data:
- **minMaxTimeDataFrame()** - includes only information about a **sensor, receiverid, timestamp_min, timestamp_max** and their representations in the standard time format.
- **rssiDataFrame()** - includes all the information about rssi changes over time for each sensor unit at a receiver that will be used to represent the data.

2.4 Suggestions. Data Manipulation

There was a requirement in the task to use python only, in addition to little amount of data, so I came up with the idea described above how to store and process data.

In case there is a comparably big amount of data, which still can be stored, I would choose using SQL data base with embedded there python. Firstly, because it is easy to parse json files into a database, secondly, because it is efficient to write a queries there since a database uses hashing and indexation.

If an amount of data was increasing enormously during time, I would think about some online algorithm that will smartly add new information into an available data frame and this action will be followed by some processing of data, occasional transformation of data etc. Under these circumstances, we will be forced to control the capability of memory storage so that no collisions will occur.

Personally, I believe that the third scenario of a development of the online algorithm is more likely to be used in real life.

3 Visualization

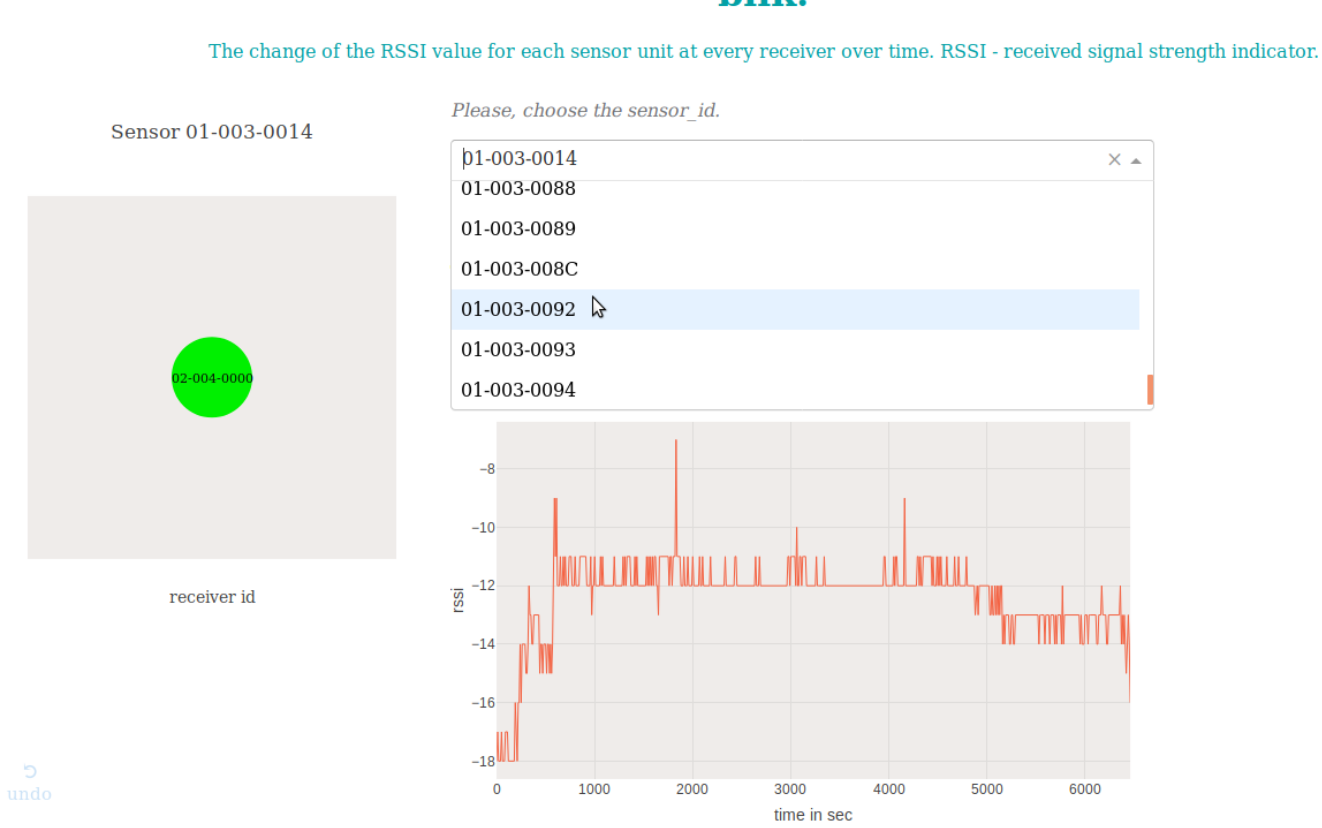
I chose *dash* and *plotly* visualization tools to represent the data due to several reasons.

Firstly, both these libraries have nice instruments that allow a customer to feel comfortable while reading the data.

Secondly, they are made in a way a customer can easily interact with data.

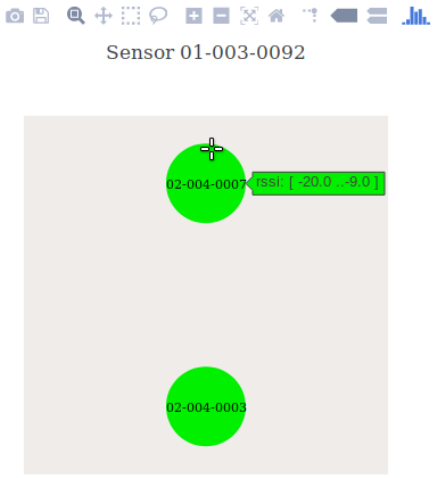
3.1 Application

The application has the following view:



blik.

The change of the RSSI value for each sensor unit at every receiver over time. RSSI - received signal strength indicator.



Please, choose the sensor_id.

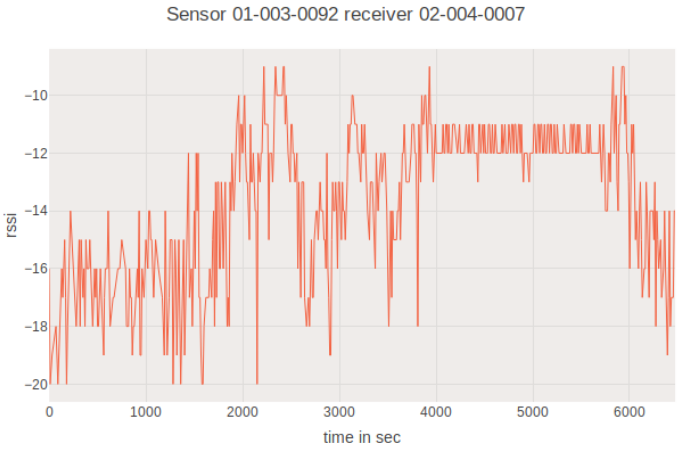
01-003-0092

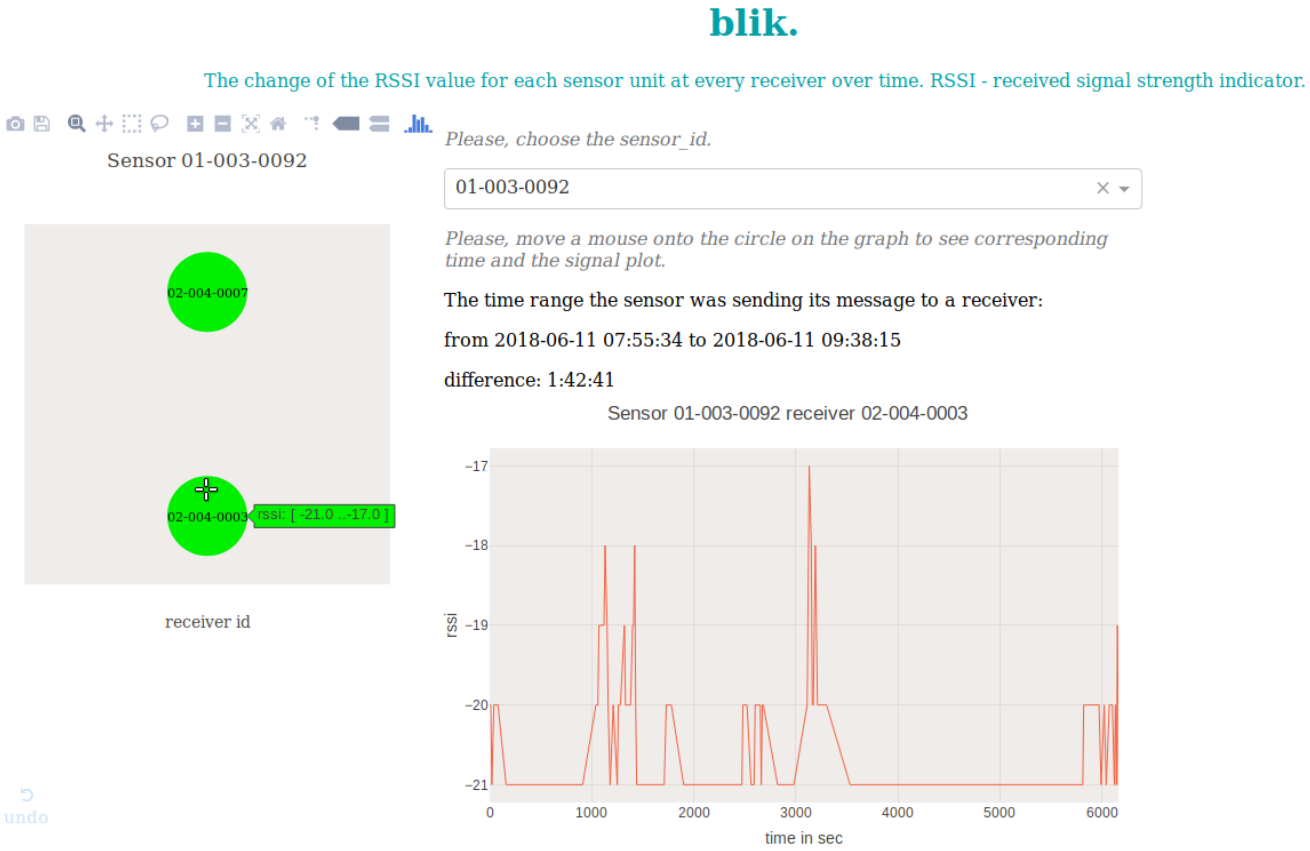
Please, move a mouse onto the circle on the graph to see corresponding time and the signal plot.

The time range the sensor was sending its message to a receiver:

from 2018-06-11 07:55:24 to 2018-06-11 09:43:16

difference: 1:47:51





In the beginning, one should pick up a sensor unit id from a dropdown on the right part of the application form.

Then one should move a mouse onto the vertex in the left chart, which corresponds to the **receiverid**, to have the date (time) range, the difference in time whereas the sensor was sending its message to a chosen receiver and the plot of the change of the **rssi** value over time in seconds.

The decision to initially choose a graph with vertices was made owing to the fact that many receivers could be scattered around and the sensor could move from one receiver to another over time. In this case it would be a directed graph with corresponding edges. However, the available data is small, consequently, no movements of a sensor are present there.

3.2 Suggestions. Visualization

As was mentioned above, the visualization of a directed graph with vertices and edges that will correspond to the movements of a sensor among receivers during time.

Additionally, one can visualize the **request_responseTime** in order to show whether we have a problem at a place. For example, from a descriptive statistics it was noticeable that once a sensor had really long request response time. Continuing the topic of monitoring problems or collisions, one can visualize the battery level and the temperature of a certain sensor. For instance, it can be implemented by the color change from green to red. In other words, when the battery is running out or the temperature of a sensor is rising significantly over a small period of time I would suggest to point out these occasions with a blinking red spot with attached description of

a problem.

A similar approach of monitoring collisions can be implemented to a sensor's configuration or a request status