Challenge 1 - Smartphone based bike path evaluation

Group 6 - Maria-Simona Chirita, Sören Coremans, Alexandru-Andrei Craciun, Robert Ignat

Introduction and requirement analysis

For this challenge, we were required to employ the sensor capabilities of an Android smartphone in order to detect anomalies on bike paths (such as potholes or bumps) and to mark these on a visual map.

In order to achieve this, we needed to implement the following main requirements:

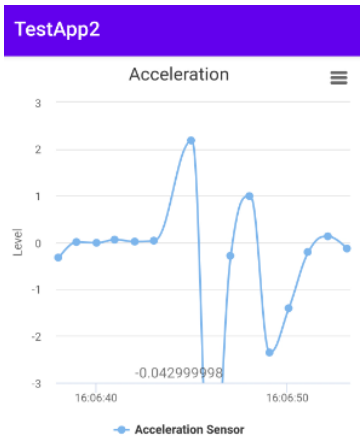
- The application should aid users to evaluate the quality of bike paths by detecting and marking anomalies while attached to a bike;
- The application should process and filter sensor data in real time;
- The application should detect anomalies based on filtered data in real time;
- The application should mark anomalies by storing their location;
- The application should display marked anomalies on a live map;

The goal was to create a system that could accurately detect anomalies in real time. A first evaluation metric is therefore execution time. A system that executes the data processing, the anomaly detection and the marking on a map in real time will get more points than a system that simply records the data and executes anomaly detection at a later date. Reliability is another important metric, as an app that only detects 50% of anomalies or is too sensitive and additionally detects speed bumps and slopes will lead to incorrect evaluation of bike paths. As a consequence, it is necessary that the application detects all obvious anomalies. This will be tested during the demonstration.

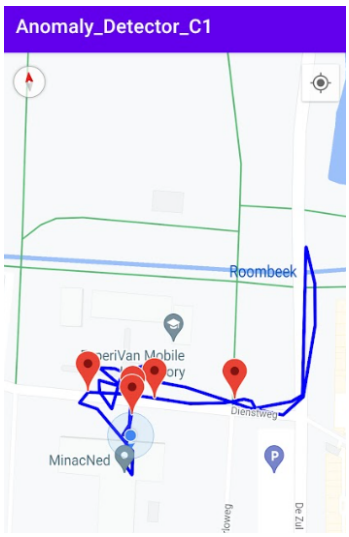
Methods and design

At the start of the challenge, none of us had any experience with creating mobile apps or using smartphone sensors. So, we first spent some time learning how to develop an app in Android Studio.

After that, our next goal was to be able to read a smartphone's acceleration sensor data in real time. Using an IBM light sensor tutorial as the template, we managed to display the acceleration data in real time, and then graph it on a Spline chart. The purpose of the chart was to give us a visual representation of how anomalies affect the sensor, as understanding this was necessary for us to be able to create a good filter and to determine the threshold which distinguishes an anomaly from speed bumps and slopes. For the same reason, we saved the generated data in a file, so that we could study it later.



In the meantime, we also figured out how to get a phone's current coordinates through code. After that, we managed to integrate a map from Google into our app and to put markers on it. These features allowed us to visually represent anomalies in a practical way. In addition, we were later able to visually track the route of the bike on the map, making it easier to see what roads were "scanned" for anomalies.



Following a few tests, we noticed that detecting a big spike in vertical acceleration was enough to reliably spot anomalies. Therefore, we added a tweakable final threshold variable, so that we can easily tune it later if needed. In order to account for noise in the data and possible sensor error, we applied a simple moving average on the data.

The last step was to combine the map and location tracking code with the sensor data reading and processing script. When doing this, we cleaned the code up a bit and replaced the Spline chart with the live map. The chart was not needed anymore, as the purpose of the app is to display anomalies, not acceleration values.

Evaluation

For evaluation, the most important factor is how reliably the app can detect anomalies, and how accurate the detection is. In order to test this, we drove a bike around campus, going over speed bumps and sometimes intentionally driving over a small ledge or into a pothole. At first the detection was poor, but we knew that this was because we set a high threshold. After tweaking it a few times, we found a value that seemed to make the

app detect all obvious anomalies while also not producing false alarms. Some more bike riding made it clear to us that the system produces reliable results.

Another factor was the visual performance of the map. While testing the detection mechanism, we also monitored how well the route was being tracked and whether the anomaly markers were placed correctly. While the refresh rate of the map was limited, the markers worked as intended. The route was also working, with the exception of a bug where it would not show up the first time you run the app since installing it. Even though there were some slight location inaccuracies, those seemed to be caused by the limitations of the phone's location service.

Lessons learned & conclusion

When we first heard of what we needed to accomplish before the deadline, we thought that it was an almost overwhelming task. However, once we started working towards the implementation the goal became more and more reachable.

We are satisfied with the app we created, and by working on it we came to realize just how much progress we can make in a few days.

Of course, we did encounter some issues during development, mainly related to library dependencies and unexpected crashes, but it was nothing unusual for software development. While the app does have some minor bugs, such as the one where the path would not appear the first time after installing, these bugs do not get into the way of the app's main function.

When it comes to possible improvements, there are a few things worth mentioning. First, the anomaly detection could be further refined by taking into account speed and rotation. In order to make it more user friendly, the app might allow for adjusting the threshold at runtime. The app was only tested with bikes without suspension, so a calibrating function that would adjust the threshold would make it better for different types of bikes. Lastly, the application could save the coordinates of all marked anomalies, and then perhaps share those locations with a central server or database.