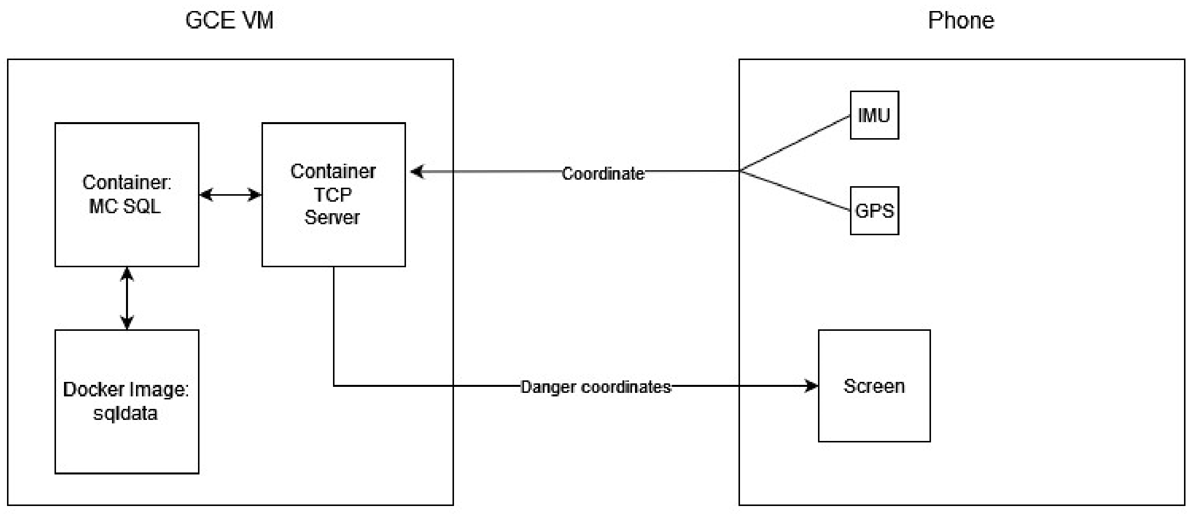
BikeSafely

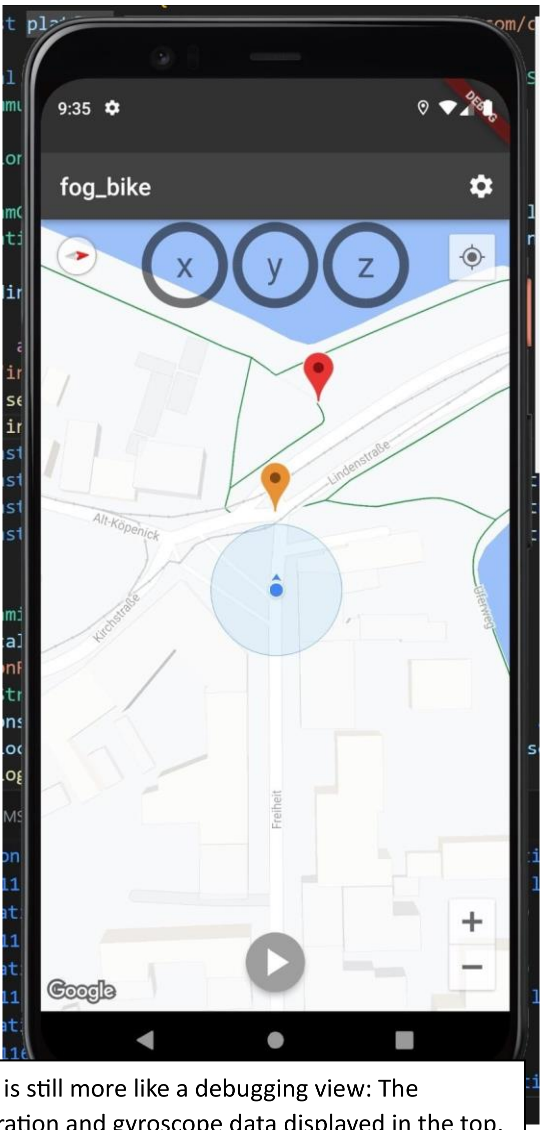
The Concept:

We have developed an app to alert cyclists to dangerous places. Users have to turn on the app and the GPS and see in a 3 color system how safe the roads and paths are. The dangerous spots are entered by the phone through the Inertia Measurement Unit (IMU). If the IMU detects a strong breaking or an evasive maneuver it sends this location to the server. In order to fulfill the requirements of the assignment, the location is also transmitted without deflections from the IMU. This data could then be used for bike lane planning to build them where there is a lot of traffic or dangerous places.



Phone

We used zeromq as a message library. Since this did not work with the request and reply pattern under dart, we wrote the tcp logic in java. The rest of the app is written in dart. The IMU constantly records acceleration and gyroscope data. When the gps position changes, the list is scanned for deflections that are based on an evasive or braking maneuver (test data for this we recorded on the bike). Depending on the strength, the coordinate is classified as low, medium or high. If no such event has occurred in the time, it is classified as traffic. The coordinate is then written to a queue. The queue is processed by the tcp service and sent to the server. If the cell phone does not receive a response from the server, the coordinate remains in the queue. We have set a reply timeout of 200ms. For each new failed attempt to reach the server, the time between attempts is doubled. If it fails 10 times in a row to reach the server, the socket will be closed and a message is shown to the user, that he should restart the app. In response to the successfully transmitted location, the cell phone receives all danger points in a radius of 100m as coordinates which are then displayed in the respective color (yellow, orange, red) on the map.



The UI is still more like a debugging view: The acceleration and gyroscope data displayed in the top, a map and a start button to start the gps recording. Here you can see an orange and red danger point in front of the person.

TCP Server

The TCP server runs in a Docker container. This makes it easy to operate and it can run on a machine in Berlin for data security purposes, for example. The server receives messages from the clients which contain single or multiple coordinates. These are then stored in the corresponding tables of the SQL database. For any coordinate that is received by the server, it queries the database to see if there are any danger spots within a radius of 100m. If there are any, these are sent to the client as an answer. If there are none, the server replies with the last coordinate that was send with the type set to “smooth”.

SQL Server

The SQL Server runs in Docker for the same reasons as the TCP Server. The Microsoft SQL was taken as the image: mcr.microsoft.com/mssql/server:2022-latest. To make the data persistent in case the container crashes, the data is stored in a Docker volume.

The setup guide for both containers is explained in the readme of the server repo.

Future improvements

So that users do not have to look at their cell phones while driving, the danger levels could be signaled by audio signals. To avoid sending too many coordinates, you could show the user a list of the danger points and he decides for himself where he has only braked and where it was really dangerous. On the backend side it would be good to implement mechanisms that restart the TCP or SQL server if they crash. A more efficient way of querying the whole database for each location that is received should be implemented.