2024 Summer Research Assistant Assessment: Android Data Collection App Development

Context

Background

A disproportionally high crash rate can be found on roadway horizontal curves, even though the horizontal curves only make up 5% of roadway mileage, they account for 25% of fatal crashes. Determining appropriate advisory speed is an important aspect of curve safety assessment, and the in-service curve characteristics, including, curve radius, superelevation, and BBI angels, are extremely important in setting up appropriate advisory speed.

With the advancement of sensor technologies and the wide adoption of smart gadgets, low-cost mobile devices (such as a smartphone) are generally integrated with sensors like GPS and interictal measurement units (IMU). An innovative framework has been proposed in the NCHRP IDEA-214 report for using the sensor data from low-cost smartphones to determine curve advisory speeds. And leverage the fleet owned by transportation agencies to enable crowdsourcing data collection which allows more time and money to be saved and more frequent assessments to be performed to proactively address safety concerns and save more lives.

Current Android Application (AllGather)

Previously, a simple Android application named AllGather was developed by the research lab to collect GPS, IMU, and Video data from driving vehicles. The main workflow designed for this application is simple:

1. Driver sets up the smartphone in the vehicle.
2. Driver opens the app and presses start recording.
3. Driver drives the vehicle as normal, while the smartphone records GPS, IMU, and Video data.
4. After completing the drive, the driver stops the recording.
5. Collected data is locally stored on the phone and can be exported for further analysis.

The following section provides a short overview of the application, and how you may use this application for testing during this assessment. Although the application is designed to

be used during driving events (as shown in figure below), for this assessment, the Android Studio's emulator should be sufficient for you to test this application.

Figure 1: Example setup for smartphone data collection

When opening the application, you will see the interface below asking for a Driver ID. Simply input any six-digit number (it's a placeholder for now and functionally insignificant).

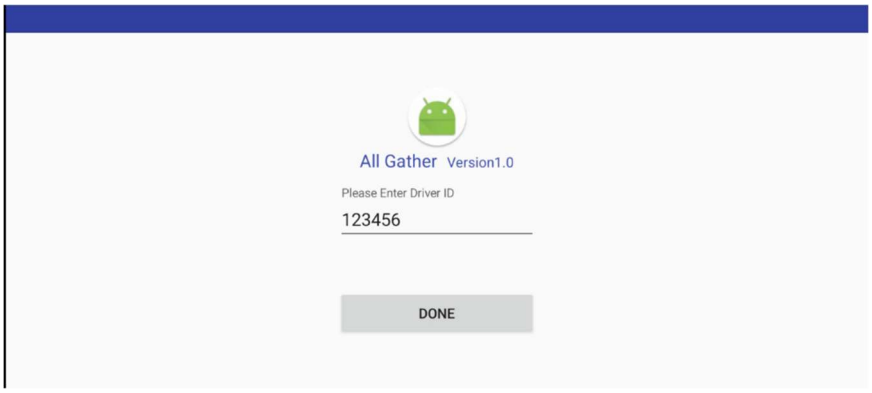

Figure 2: Driver ID View

After inputting the Driver ID, the main view will be visible. Below is an overview of the interface:

1. GPS Connection Indicator: this should indicate GPS Connected before data collection.
2. Video Recording Toggle: if the user wishes to only collect the sensor data (GPS and IMU) and not the video data, they can use this toggle before data collection to disable video recording only to save storage space.
3. Record Button: button to start and stop data collection. After stopping, the collected data can be found in the phone's local storage (will explain further below).
4. Data Cache Button: Irrelevant to this assessment, DO NOT press.

5. Upload Data Button: Irrelevant to this assessment, DO NOT press.
6. Calibration Button: Irrelevant to this assessment, DO NOT press.

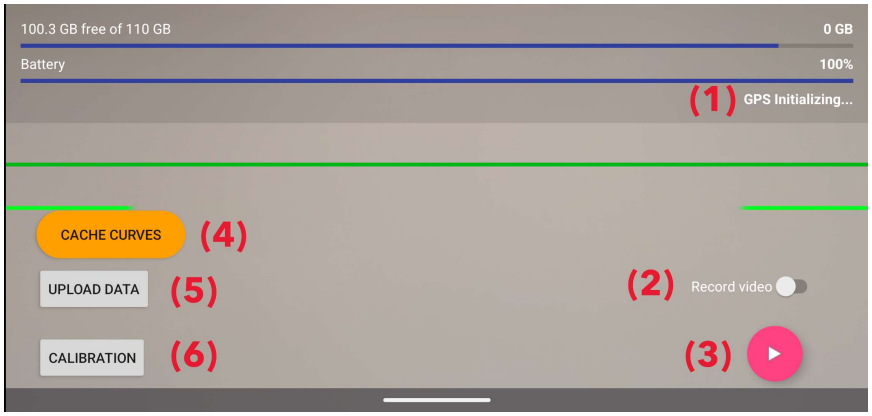

Figure 3: Application Main View

Export and Inspect Collected Data

All collected data can be found in the phone's local storage at this path: *"\Internal shared storage\Android\data\edu.gatech.ce.allgather\files\Documents\data"*

Each time the user presses the record button, a folder name using the data collection timestamp will be created representing the collected data. The sensor data stored in that folder follow the following structure:

```
├── {dataset_folder (timestamp of when data was collected)}
|   ├── {driver_id}
|   |   ├── {device_IMEI}
|   |   |   ├── acceleration
|   |   |   |   ├── *_acc.csv
|   |   |   ├── calculation (optional)
|   |   |   |   ├── *_calibration.csv
|   |   |   ├── calibration (optional)
|   |   |   |   ├── *_calibration.csv
|   |   |   ├── camera (optional)
|   |   |   |   ├── *_cam*.mp4
|   |   |   ├── location
|   |   |   |   ├── *_loc.csv
```

Figure 4: Folder Structure of AllGather Collected Data

This assessment will mainly involve the location and acceleration csv files. You can find an example data collected using the app in "Data/Test Data".

Assessment Questions

This assessment includes two questions which are designed to mimic the primary responsibilities of this role. You should read the instructions of each question carefully to understand the context and requirements of each question.

Question 1: Implement a testing framework that mocks sensor data by loading it from previously collected csv files.

Context

Currently, the app is designed to collect data in a moving vehicle. However, there is a need for a testing framework that controls the sensor data used in testing and tests the edge computing functions without needing to drive the vehicle. This may be achieved by creating mock sensor objects in the AllGather app, instead of providing actual sensor readings. These mock objects can provide sensor data defined in csv files (these files could be the previously recorded actual data). During testing, the app will run using these mock objects instead of the actual sensor objects, thus giving us control as to what sensor data to be used and move the need to drive for testing the application.

Requirements

- Review "Code/AllGather-Android" code to understand how sensor data (GPS and IMU) are used and recorded to csv files.
- Implement mock sensor objects (GPS and IMU) that will replace the actual sensor objects in testing mode. The mock object will provide sensor data read from CSV files (recorded from previous actual data collection) to any downstream objects (e.g., objects that record sensor data, or objects that process sensor data).
    - You may specify a location to store the previously collected sensor data to be used by the mock objects. Previously collected sensor data will have the folder structure shown in Figure 4.
    - Mock objects should provide sensor data stored in the acceleration and location csv files, and may provide sensor data at faster rate the then recorded by the timestamps to accelerate testing time.
- Use dependency injection to allow easy switchover between actual sensor objects vs mock objects to any downstream objects that use sensor data.
- Implement a testing mode toggle, when pressing the start button with this enabled, the app will run using the mock objects and predefined CSV files in the local storage.

- Just like normal data collection, this test will also generate data in the local storage.
- Unlike normal data collection, rather than waiting for the user to stop the recording, the recording will stop after the test is completed.

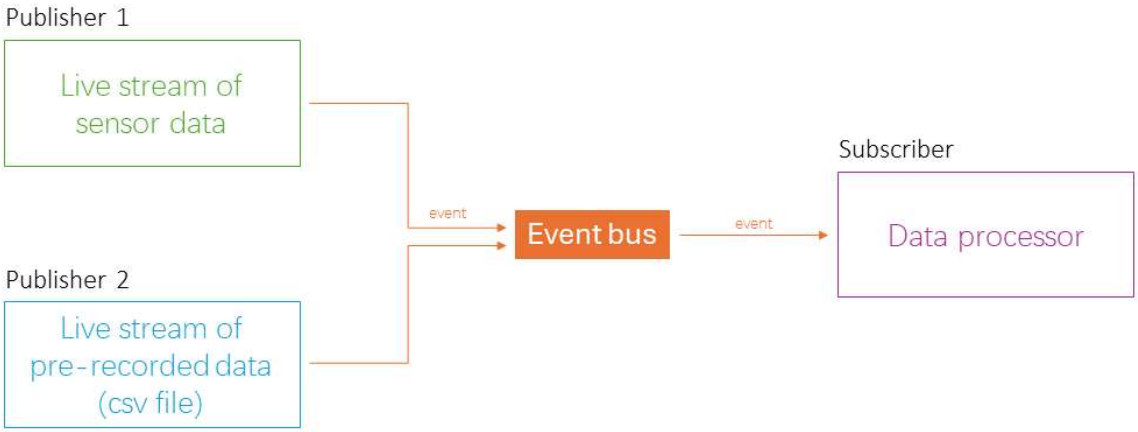

Figure 5: Data retrieval and processing framework following the publisher-subscriber model

Question 2: implement and test location-based event triggering from proof-of-concept

Context:

One of the edge computing functionalities that is planned to be added to the app includes requiring the app to know which curve the vehicle is currently traveling on. As discussed during the first meeting, a "CurveFinder" tool was previously developed to extract curve GIS (Geographic Information Systems) features on roadway centerline GIS data, as shown in figure below.

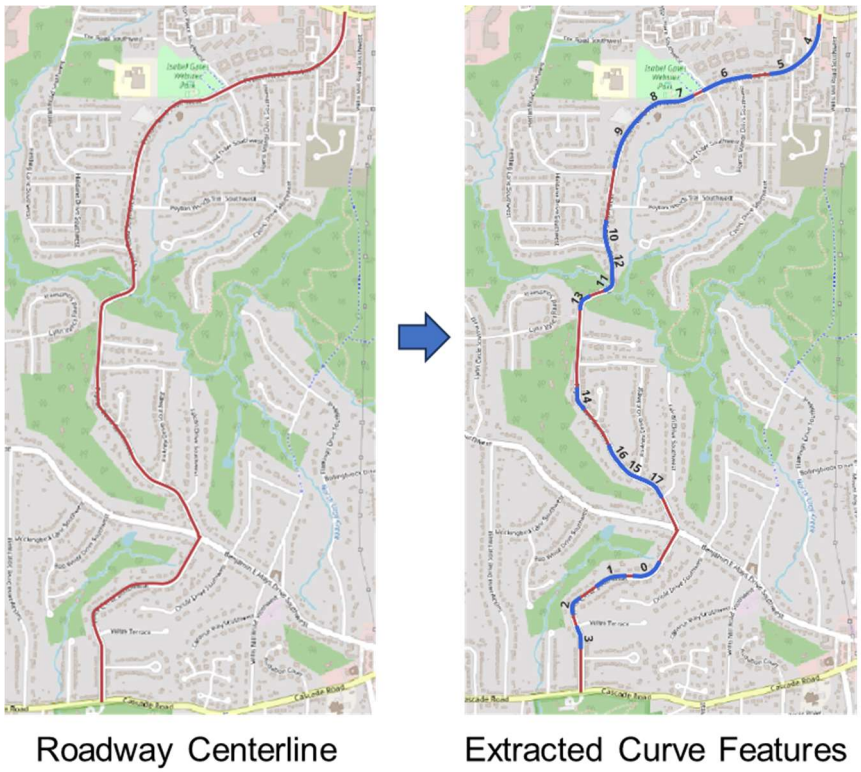

Figure 6: Example Extracted Curve Features Using CurveFinder

These extracted curves can be considered as a curve inventory (list of curves), and stored locally on the phone as a GeoJSON file (a type of GIS file format). For the application to know the curve that the vehicle is currently traveling on, this can be done by performing "spatial query" using the current GPS position against the curve inventory in the GeoJSON file. The spatial query function takes the current GPS position and determines the ID of the curve that the vehicle is currently traveling on. Previously, we had developed a proof-of-concept in Java that performs this spatial query using GPS data read from CSV, and curve inventory locally stored as GeoJSON. In this question, you will be asked to review the proof-of-concept code and implement the spatial query function in the AllGather App to enable

the app to determine the curve ID in real-time during data collection, and record on the spatial query results in location csv file.

Requirements:

- Review the PoC (proof-of-concept) in "Code/android-geospatial" folder and understand how to perform spatial query using GPS data and curve inventory stored in GeoJSON files.
- Implement the spatial query function in AllGather app so a curve ID is determined for each GPS data point in real-time during data collection.
    - The spatial query function should be involved every time there is a new GPS data point available as **shown in the sequency diagram in Figure 6**.
    - You can copy the same GeoJSON files used in the PoC to the AllGather app to be used for the spatial query. **The file can be found in "android-geospatial/app/src/main/assets/curve_inventory.geojson".**
    - The spatial query should return either the curve ID or None if the current GPS is not on a curve.
        - Each curve in the curve inventory has many predefined attributes, among these attributes, the combination of the following three attributes, **"c_segid, c_sectid, c_id",** can be used to uniquely identify a curve in the inventory, and should be treated as the curve's ID.
    - The modified AllGather app should append new columns in the output GPS csv file with the information of the corresponding curve ID for each GPS point.
- Run and test the implemented spatial query function using the test framework implemented in question 1.
    - The test should use the data found in "Data/Test Data" folder to mock the sensor data.
- The output GPS csv file should be compared to the ground truth GPS csv file in "Data/Reference Data", the output GPS csv file should have the same format and curve IDs compared to the ground truth GPS csv.

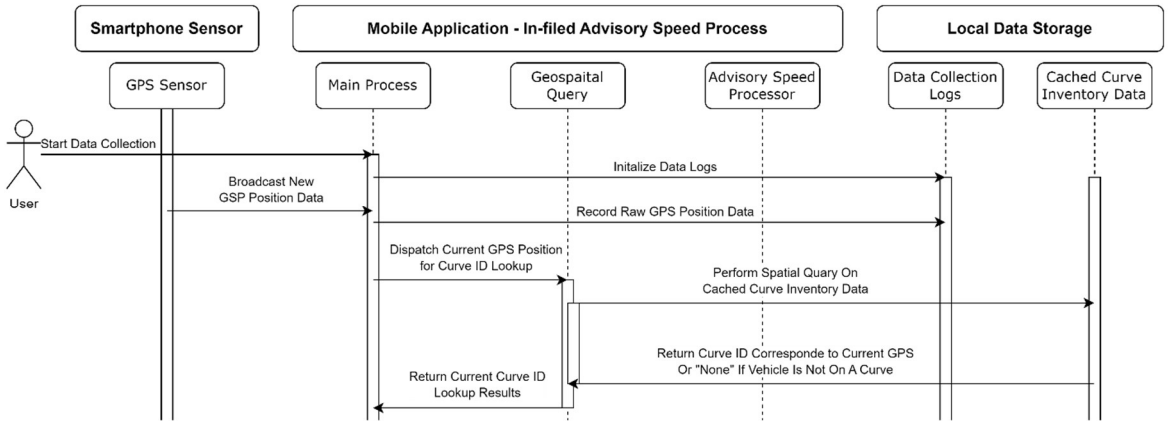

Figure 7: Geospatial Query Sequence Diagram

Deliverables and Timeline

- Source code along with the compiled apk and test results files
- A report in pdf format with the following
    - Explanation of your approach.
    - Demonstration of the results that shows the objectives in the questions are achieved.
    - Documentation of challenges faced, and how you solved them.

Please send the deliverables, in a single zip or 7z file, to Prof. James Tsai (james.tsai@ce.gatech.edu), and CC Mohsen Mohammadi (mohsen@gatech.edu) and Lucas Pingzhou (pyu68@gatech.edu) with the subject line "Evaluation - [Your Name] - Research Opportunity on Android Development".

Please refer to the email for deadline of this assessment.