AR-driving-assistant Project Overview

Overview

The goal of this document is to highlight the key components of the AR-driving-assistant project in order to foster a better understanding of the state the project was left in.

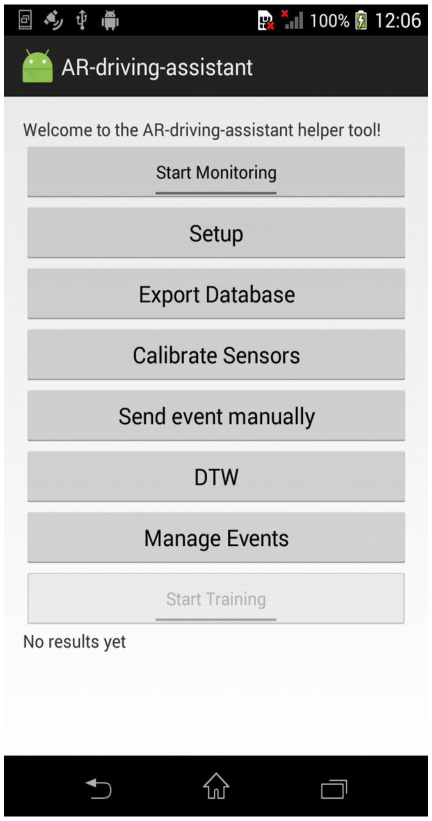
You will find the installation and deployment instructions in the README.md file on the Github page.

Github link : https://github.com/limvi-licef/AR-driving-assistant

Usage

1. Android App

The Android app is composed of a single Activity with a scrollview containing a bunch of buttons and Dialogs to setup and start the sensor monitoring. The Android phone used must possess the necessary sensors in order to work. Also, the phone must be setup correctly (see Locations Services in phone Settings and Location settings in google account settings) for the Location sensor to work.



Welcome to the AR-driving-assistant helper tool!

No results yet

- The first button is pretty self-explanatory. It is a toggle button that starts or stops the sensor monitoring.
- The Setup button opens a dialog that shows two fields. The (Current User) ID field, which will be used when each sensor data is saved to show who was driving when the sensor registered the data. The second is the HoloLens IP address field, which will be used to communicate with the HoloLens (or the other Android). You will find the correct IP address in the Android hotspot settings once the HoloLens is connected to the Android device. By default, it should be 192.168.43.103, while the Android device hosting the hotspot should be 192.168.43.1. The inputs are saved as Shared Preferences.
- The Export Database exports the content of the database as a json file to the device, found in the AR-driving-assistant subfolder once the device has been rebooted.
- The Calibrate Sensors button is used to calibrate the linear acceleration sensor. Put the device on a flat surface and wait for a few seconds.
- The Send event manually sends an event to the HoloLens. Choose the type of event and the message that will be displayed and send it. This also has the side effect of overriding the default IP address that the Unity app uses to communicate with the Android app, if for some reason the IP address to use is not the default 192.168.43.1.

- The DTW opens a Dialog allowing to select which sensor are to be used during the DTW process (see below in the algorithms section). Choosing no sensors will disable DTW. This choice is saved as a Shared Preference. The sensor distance cutoff (the value used to detect if a match was found) can also be modified here.
- The Manage Events button will list all Training Events in the database and allow you to delete all events or a specific one.
- Finally, the Start Training button will create a new Training Event and is only available if Start Monitoring is toggled on. When you first press it, it will open a Dialog where you can input the necessary info for it. The label must be unique and is used to identify the event, while the event type and message are used to send the event to the HoloLens in the case where a match is found during DTW. When you press "Start" on the dialog, it will save the timestamp. Press the button again, which will now be "Stop Training", and a second timestamp is saved and used to create the Training Event. However, it will fail and send a Toast to the screen if a sensor finds no data or if no sensors was specified in the DTW dialog.

2. Unity App

The Unity app is designed to work alongside the Android app and can only be used while the Android app is ready to received and return data through a TCP connection. It is comprised of 5 different panels. The first is the login screen, where the user chooses his ID in a dropdown list. The refresh button will actualize the list with the users in the Android app database (it can take up to 10 seconds for the list to refresh). The new user button will open the new user panel.

When creating a new user, the app will send the user info to the Android app, who will be responsible to verify it is valid. If valid, it will set the dropdown option to the new user, while a non valid user input (mostly the ID can be invalid since it is unique) or network error will display a red error message.

The next panel, after taping the validate button, will open the menu, which will allow you to go to either the Application screen or the Settings screen. The Application screen is to be used while driving and will display the events when the Android app sends them. The Settings screen shows multiple options. The Retroaction button, will open a window, then fetch and display a list of dates corresponding to the list of unique days where sensor data (using location data) is associated with the currently logged in user. The size slider (HoloLens only) controls the size of the hologram. The sound slider will impact the sound an event makes when it is displayed. The speed checkbox will display (or not) a small text field on the Application screen to update the car speed in km/h. The Contact button will display a short text with information about the app.

Furthermore, the HoloLens app also has an invisible button underneath the hologram (the cursor will show when gazing at it). Tapping it will allow the user to move the hologram where the user wants it until another tap gesture is made. While the hologram will be locked to the gaze's X axis, the Y axis will keep its position.

Code Structure

1. Android App

The android app code is divided in 11 different packages. The app's core packages are the Activities, the Algorithms, the Receivers and the Runnables. Basically, the MainActivity registers the receivers, who in turn launch the runnables, who will finally send the data to the algorithms.

Activities

Contains the only activity in the app. The MainActivity registers the sensors listeners, creates the UI buttons, manages the different DialogFragments and listens for broadcast messages.

Algorithms

Contains the algorithms used in the app. The Monotone Segmentation Algorithm is used to reduce the noise in the data. The Dynamic Time Warping algorithm is used to calculate the Euclidian distance between different segments of data in order to match similar patterns. TimeSeriesExtended is a modification on the TimeSeriesBase from the FastDTW library that allows a single data point to be removed/added to the TimeSeries instead of recalculating the whole thing.

Config

Contains different classes with constants used throughout the app, extracted here together for easier modification. AwareSettings regroups the settings for the different Aware sensors. Communication contains the constants used to communicate with the Unity app. DynamicTimeWarping has all the settings for the DTW. SensorDataCollection determines how the sensors data is collected.

Database

Contains the database schema and the database helper to manage the database.

Fragments

Contains the different DialogFragments linked to each button in the MainActivity.

Models

Contains the data classes used throughout the app. The sensors subfolder holds the different SensorTypes used for the DTW algorithms. The TimestampedDouble, SegmentationAlgorithmReturnData and ExtremaStats are used to send data between the different algorithms and runnables. The Event and User classes are used to communicate with the Unity app.

Network

Contains the TCPListenerThread class that constantly listens to requests from the Unity app and sends data back to it.

Receivers

Contains the different sensor listeners. When a receives a new data point, it either saves it directly to the database (Temperature and Location Receivers) or sends it to the appropriate Runnable (Linear Accelerometer, Rotation and Location Receivers) so the data will be analyzed before being saved.

Runnables

Contains the different runnables used by the app. The MatchEventRunnable will try to find a match between the different TrainingEvents and the data accumulated since it last ran. It will do so every time the ComputeAlgorithmRunnables are done running. The ComputeAlgorithmRunnables will accumulate the data sent to them and will periodically (currently 1 min) send them to the MonotoneSegmentationAlgorithm and then the DTW. The RewriteAlgorithmRunnables will periodically fetch the last 10 minutes of data before passing it through the MonotoneSegmentationAlgorithm.

Tasks

Contains the different AsyncTasks used by the app. Each task is linked to one button in MainActivity, except for the MatchEventTask, which is unused since it has been replaced by the similar named runnable. It was kept in order to compare the performance of AsyncTask vs Runnable.

Utils

Contains some static utility classes used throughout the app. Specifically, the Statistics is used to keep statistics about the data after is has been through the Monotone Segmentation algorithm.

2. Unity App

For the Unity app, most of the work was done though the editor. The canvas contains an object named “UI” that contains the five different screens present in the app. Each screen has its own components associated with their own scripts. The Managers gameobject holds different scripts, custom or from the HoloToolkit, that are used throughout the app. The HoloToolkit ones manage the HoloLens gesture and gaze controls, while the custom ones manage the network, the users and the events to display. Those scripts are located in the Assets/Resources/Scripts subfolder. The EventManager and ShowEventScript both manage the events to be displayed on the Application screen. The TCPListeners, TCPSender and JsonClasses are used to for the networking. The Config file groups the constants used in the app for easy access and modification. The TapToPlaceUI script allows the user to move the hologram. The PopulateDropdownScript, NewUserForm and Retroaction scripts manage the requests to send to the Android app through the TCPSender. The UserManager is responsible the hold the users list and to set the current user. Finally, the other scripts are small scripts, mostly for managing the UI. Switching screens is handled by using the SetAsLastChildren method to set a panel on top of the others.

Concerns

Below you will find concerns about the behavior of certain parts of the project.

- The linear accelerometer sensor and the Dynamic Time Warping (DTW) algorithm. While the other sensors have shown good initial results in their ability to distinguish patterns using DTW, the linear accelerometer sensor during initial experimentation has not shown the same promise. This could be due to a lack of precision from the sensor, or simply a lack of proper testing. Nevertheless, more experimentation is required.

- The Dynamic Time Warping (DTW) algorithm can take a long time to process the data (see the DTW tests results in the appendix) depending on the amount of data and the quantity of Training Events to compare the data to. As such, running the DTW algorithm with all three sensors should be avoided unless there are very few events in the database. The major issue is that the DTW is launched after the runnables are done, which is every minute. Thus, the processing time of the DTW should be kept under a minute if at all possible, else the next DTW will start lagging behind and it could take a long time to process all subsequent DTW.

  Still, there could be a few ways to speed up the process, but they come with a precision cost. Currently, the distance between each TimeSeries is calculated for every different segment of data that can be created within the last minute of data, with a segment being the same length as the event itself. To do so, between each distance calculation, the segment TimeSeries has its first data point removed and a new one added, until every data point is used. As such, it is possible to skip one data point (or more) between each segment. It would reduce the precision since you would have half the data, but it should be nearly twice as fast to process. Another way would be to find a way to discriminate segment that cannot possibly create a match and not process those. Since each distance calculation takes about 25-30 milliseconds, that way to discriminate should take less time than that for there to be a gain.

- There is currently no way to manage the app’s data aside from wiping the whole thing. That said, after one 45-60 minutes car ride with the app active the AR-driving-assistant database was about 2 mbs, so it will take a while until it is a problem.

Tools

During the project, I wrote some quick command line tools to help with a few tasks.

Segmentation.jar takes an old data file for one car experiment from before this project and parses it throught the Monotone Segmentation Algorithm then divides it into 1 minute segments – same as the one in AndroidApp, in order to compare results. The file itself must follow a specific format and must contain a single car ride (check the that the file has the data from only one car ride). If present, the -dtw argument takes a csv file containing two columns (first being timestamp, second being the values) and compares each minute-long segment with the sample provided and calculates the distance for each.

```
usage: java -jar Segmentation.jar
  --csv <PATH>                    [mandatory] the path to the csv file with the values to process
  --dtw <PATH>                    [optional] the path to the csv file with dtw segment
  --segment-length <SECONDS>   [optional] divide data into SECONDS-long segments : default is 60 seconds
  --tolerance <VALUE>             [mandatory] tolerance for the Monotone Segmentation algorithm
```

DTW_analysis.jar extracts the exported json database into multiple csv files in order to facilitate the analysis of the data in regards to the DTW algorithm. It creates a csv for the TrainingEvents data, another for the DTW results for the particular sensor and a last one for the sensor data. You can also specify the user ID to only extract the data associated with the user, instead of all the data if not specified.

```
usage: java -jar DTW_analysis.jar
  --json <PATH>      [mandatory] the path to json database file
  --sensor <TYPE>   [mandatory] the sensor type to extract : Speed, Acceleration, Rotation
  --user <ID>          [optional] extract only data associated with user, if specified
```

TimeSeriesDistance.jar simple calculates the Euclidian distance between two lists of comma separated values.

```
usage: java -jar TimeSeriesDistance.jar
  --ts1 <TIMESERIES>   the first series, each value separated by a comma
  --ts2 <TIMESERIES>   the second series, each value separated by a comma
```

Notes

Below you will find miscellaneous notes that may or may not be useful.

- From their Github page[1] : “The HoloToolkit is a collection of scripts and components intended to accelerate development of holographic applications targeting Windows Holographic”. This project does not use the HoloToolkit much, it is mostly used for their HoloLens Input Module that allows app to recognize gestures such as air-taps and translate them to Unity events. Nonetheless, it might be useful to keep the HoloToolkit up-to-date. Currently, the HoloToolkit bundled in this project was updated to Unity Editor version 5.5.0f3. To update it, first download or clone their Github project source code and open the folder within in Unity. In the project view, right-click the Assets and choose Export as a package. Next, open the AR-driving-assistant, remove the old HoloToolkit assets folder and finally right-click the Assets and choose Import → Custom package and select the newly created .unitypackage file. See their Getting Started page on their Github for more information.

- Aware Framework[2] : The Aware Framework was used to easily collect data from the linear acceleration, location and weather (currently not used since android hotspot does not allow internet access without a SIM card) sensors. However, should the need arise, it should be a relatively trivial task to collect the data ourselves instead, like it was done in the RotationReceiver.

    - Also, Aware saves its own databases on the device in the /AWARE folder. You should periodically delete those databases, especially the linear accelerometer one, as they can grow to several hundred mbs over time.

    - Finally, a crash may occur during the Aware setup on certain devices when there is no bluetooth or the bluetooth capabilities are not correctly detected.

1 https://github.com/Microsoft/HoloToolkit-Unity
2 Web page : http://www.awareframework.com/
Github : https://github.com/denzilferreira/aware-client

To-Do

Below you will find a list of incomplete tasks.

- The branch UDPCommunication replaces the TCP implementation with a UDP one. For the android scene in the Unity app in that branch, however, there is a bug where the Retroaction request crashes the UDP listener thread and all further UDP communication does not work. For now, the RequestRides method causing the issue was disabled. For the HoloLens scene on that branch, it doesn’t crash the thread, but the panel is not updated either.
- Fill License section on the Github README.md.
- Fill the Contact Info Panel text (Settings Display → Contact Panel → Contact Info Panel → Text) on the UnityApp for both scenes.

Appendix

1. DTW Test Results (in seconds)*

Length of event in seconds	Time to process (Acceleration Only)	Time to process (Rotation + Acceleration)
Test 1 : 8.447	13.036	22.350
Test 2 : 18.934	15.581	27.944
Test 3 : 18.934 + 2.934 (21.868)	18.254 + 8.727	29.012 + 17.033
Test 4 : 2.268 + 3.030 + 2.983 + 2.787 + 3.664 (14.732)	8.470 + 8.325 + 9.607 + 8.807 + 8.829 (44.038)	14.986 + 15.930 + 16.424 + 16.491 + 17.344 (81.175)

Length of event in seconds	Time to process (Rotation Only)	Time to process (Rotation + Acceleration)
Test 5 : 3.682	10.587	19.766
Test 6 : 8.787	12.590	23.553

* For the Speed sensor, there is one fifth of the data to process (1 every second vs 1 every 200 milliseconds for the acceleration and rotation sensors) so it should more or less take a fifth of the time.