Terrain mapping from mobile sensors for estimating wheelchair accessibility

Siddhartha Thota
Department of Computer Science
University of Toronto
Email: thota@cs.toronto.edu

***Abstract*—Using sensors embedded in mobile devices, and by taking observations from massive number of users, it's possible to estimate terrain quality of sidewalks in urban areas. By making use of location information from GPS and accelerometer readings alone, a system is designed to map sidewalks of a city. A machine learning model is built that's able to classify accelerometer data into terrain information. Then, the GPS information is used to correlate multiple accelerometer readings, localize the terrain tags and store them. Road segments are eventually assigned a binary label, with a confidence that increases with the amount of data at hand. The system is verified with walks around Toronto, by using an interface on Android phone and a web interface, powered by backend servers.**

***Keywords*—terrain estimation, mobile sensors, wheelchair accessibility**

I. Introduction

While there are detailed maps of cities for driving and walking directions, there's a lack of information about navigation for wheelchair users. Few or no services provide a navigation service that can tell a wheelchair user what sidewalk to take, the quality of sidewalk and difference between sidewalk quality of two different routes a user can take. This is mainly because of the complexity of estimating such information from central sensing system, like satellite images or radars. A more localized sensing system has to be used to estimate the quality of sidewalks for wheelchair users.

On the other hand, a large amount of data is collected and processed on modern day phones. Accelerometer data is almost always being processed by the apps on the phone, even when the phone is sitting idle in the user's pocket. Activity recognition for fitness information is an example application of this. GPS position of a phone is also almost always available in the phone. By combining these two sensors, an app on the phone can infer the quality of terrain and then send it to a central data server, thereby creating labels for road segment that the user is currently walking on. This is the design that's proposed in this paper.

II. Background

To date, multiple ways of generating this data have been published. Processing Google Streetview images[1] is a promising way of doing this. The method involves hard image processing algorithms, and there is a possibility of the data from Streetview being outdated and not sensitive to day-to-day changes on the sidewalks (roadworks and such). The difference between this approach and ours will be the fact that

Fig. 1. Sidewalks tagged with estimated road quality using accelerometer data

the system design allows for updates from daily contributors to show up dynamically in the database of sidewalks, allowing for a immediate update of day-to-day impact on sidewalk usability to show up on our maps.

In an approach very similar to ours, Frackelton et. al.[2] propose a solution that uses GPS, accelerometer and gyroscope data from a dummy wheelchair to measure sidewalk quality. The paper does not talk about the type of data collected, or the algorithms used to perform classification, or the interface in which the results are presented. The difference between that work and ours will be that we can use data from normal mobile phone users, and the sensors do not have to be mounted on a rolling device like it is in the former's. We can also simply scale up the number of contributors by including a low-power android module in other apps that already use accelerometer and GPS on the phone.

For individual components of our design, we use ideas that were presented by various papers. In the first step of identifying the kind of terrain from accelerometer data only, we take cues from the work of Hernandez et al [3]. Feature engineering steps

from here were useful in feeding the accelerometer data into a classical machine learning model, although while using a neural net based model, feature engineering was not necessary altogether.

We also use the excellent map snapping algorithm provided by Newson and Krumm[4]. The map matching algorithm worked flawlessly, and an implementation of the algorithm by two parties were used in the end, both of which are cited with the implementation details to follow.

The following sections will be: System Desgin, where we elaborate the phases of our system, Implementation, where we show the components and how they were built, and a Results section to elaborate our results. A conclusion and a future work section will follow.

III. System Design

The application works in three different stages, a training phase, a data collection phase and a user interaction phase. The last two stages can run concurrently, as they feed off the same database. The first stage needs to be run separately, and is human-intensive. Here, we elaborate each phase of this process.

A. Training phase

First, we were not sure if accelerometer data is enough to estimate terrain. Various experiments were conducted indoors as well as on the streets, and it was observed that there is a significant visible difference between normal walking and going over an obstacle/walking down the stairs. It was observed that the integral of the accelerometer signal, and the double integral, both showed significant features that could be identified as one of the former activities. Following this, a feature engineering and model selection process was done.

A 32-sample window (with sampling frequency average of 16Hz) was selected so as to take a 2s sample in a window. STFT, power spectrums, minimum, maximum, mean, std, and zero crossovers were considered for features. A feature vector upto 140 features was then created, for each window. The windows overlapped 50% with each other. The features were then passed to two different models.

KNNs gave good results, but with the small amount of data, a problem of overfitting was observed. Improvements on the KNN-based approach is left as future work. We then considered a multi-layer dense neural network based model. A 64 node layer, followed by three dense layers of 32, 16 and 8 followed by an output layer of 2 states were used to learn the data. With less than 80 epochs, we obtained an accuracy of 80%. It was noted that with the small amount of data at hand, again, an overfitting problem could happen. An experiment with more tagged data and one with synthetic datasets is a possibility. Synthetic data can be generated by repeating tagged data and introducing noise in it. This approach is also left for future work.

A model that's generated is then stored and made available to the webserver that serves the UI. This model can be updated offline, as the UI server is stateless and will fetch the latest model every time it gives out a classification.

B. Data collection phase

For data collection, authors walked on the streets around University of Toronto buildings. GPS data was collected on the phone, and accelerometer data was also written into a file. At the end of the day, all files were transferred to the server, where it was processed. The server would take the GPS data, simplified it with the Ramer-Douglas-Peucker algorithm[5]. It would then score the accelerometer data as positive or negative using the classification model stored in the Training Phase. The scores were per-window for the entire length of accelerometer data. A correlation mechanism was created that would take the relative segment length of each segment in the GPS track, fetch the window scores for that relative length from the entire accelerometer data, and extract a single score value from those scores. It is then assigend to the GPS segment from the track.

A server was built to talk to the android app, collect the data and process it. It ran on the author's laptop, and runs really quickly with modest specifications. GraphHopper[6], an excellent open source implementation of several map tools required for this project, was used to create a user interface to look at collected results. The map screenshots in this paper are from this UI. The javascript from the frontend was modified to also talk to our server. When a new road segment was submitted to the server, the data collected and scores assigned are returned by the server. The UI then displayed it, which the authors used to verify the collected data and scores. A screengrab from this UI is shown in fig 1.

Various road segments were walked by the authors, but a limitation of this work is the sparsity of data that is present, even after a few hours of data was collected. A better result can be obtained, according to us, by collecting massive amounts of data. The data processing pipeline is kept robust and simple to enable this. This shall remain the chief direction of future work of the authors.

C. User deployment

A user interface can ask for a user's source and destination, and create a path for the journey using the GraphHopper engine. The path consists of road segments. Segments can be then passed to our server, which can fetch the saved scores for each segment. This segment can be augmented with the route itself, displayed on the UI. This way, the user can then reject a path and request for another path that is better, in case she finds it hard to access on their wheelchair.

IV. Implementation

A. Backend Server

A flask-based backend server runs on the author's laptop. The server has two roles. First, talk to the android app and collect, process and save data. Second, to serve an API for the UI with data verification, and possibly for the end user's UI. This uses numpy, gpxpy, pandas, tensorflow python, mysql and python simpleHTTP server, all open source python libraries. A representation of the components is drawn in fig 2.

B. GraphHopper Server

An instance of the GraphHopper server was running in a remote Ubuntu machine. This instance is instantiated with the

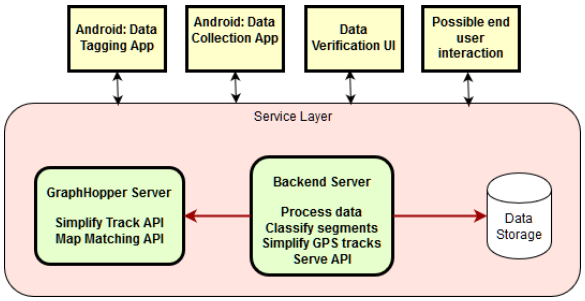

Fig. 2. Components of the system

roadmap of Toronto downloaded from the Open Street Map project. The instance is then capable of matching any GPS track with a possible series of road segments on the map. Internally, it uses the same map matching algorithm proposed by [4].

C. Android App

An android app was designed, to collect, tag and send the data to the server. The screenshot of this app is shown in fig 3. This app collects GPS positions whenever the phone makes available. A consumer version of the app can use passive GPS data provided by Android. This means Android will only give the app the GPS data when the phone actually gets an updated location because some other app on the phone asked for it. This ensures that our app doesn't use any additional battery, but also makes sure data collection is effective when the user is doing a task that's navigation-based.

There are several shortcomings of this app. The tagging interface needs the user to tap a button, which means the screen should be on. This was found as the biggest hindrance in collecting data. A better technique could result in a large amount of training data. The app also was found to be killed by the Android OS when it was not in the foreground for a set amount of time. This calls for a better design of an app that runs in the background.

D. UI

GraphHopper UI was modified to create a map UI to interact, visualize and verify data collected by the taggers. The UI is a simple map, with a javascript backend that accepts a GPS exchange format (GPX) file. The javascript talks to the GraphHopper Server for map matching, and plots the matched route. It then contacts the Backend API and asks for the scores attached to each of the road segments. Then, it marks the segments with negative scores in red.

V. Results

A. Tagging accelerometer data

The accelerometer data was tagged as "No Tag" vs "Tag", where "Tag"s are any unevenness on the surface that taggers walked on. An example data from this dataset looked like fig 4.

The model learned the data at reasonable rate, with confusion matrix showing 80% recall per class. The learning is not perfect, but for the scope of this project we settle for a bad

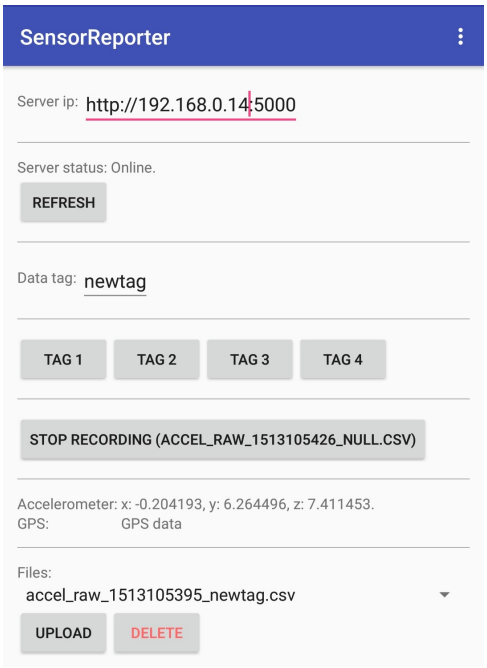

Fig. 3. Android app for data collection and tagging

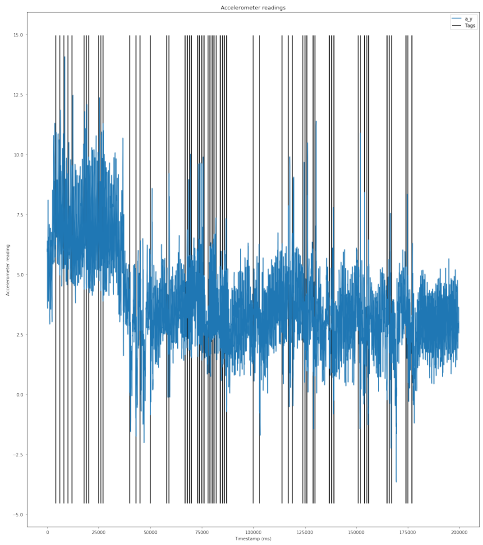

Fig. 4. Accelerometer data example

learning model. In fact, the model and the learning method are not worthy of being mentioned in this report, embarrassingly so.

B. Tagging road segments to labels

Training data is obtained as rows of accelerometer data, GPS data, tags all with a timestamp. The first step is to send the GPS data through the simplify algorithm and then to GraphHopper server to get the road segment sequence of the entire GPS track. This forms the basis for tagging. Further, we label the accelerometer using our learned model. The model returns a tag for every window in the accelerometer data. Now, we split these tags and assign them to each road segments, with the number of windows per segment proportional to the relative

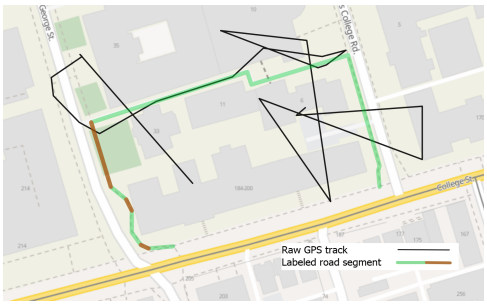

Fig. 5. Going from raw GPS track to tagged road segments

length of the segment to the whole track. This information is stored in a database indexed by the start and end of the road segment, for future lookups. The entire workflow is demoed in 5.

C. Evaluation Plan

The evaluation of this project can be done in a live deployment. We envision two ways of evaluating it: by continuously improving the collected data, and from end-user evaluations. In the continuous learning front, when there is more and more data available for each road segment, a confidence value can be measured, which gives the confidence of the system across data from multiple users. The end-user evaluations can depend on how an end-user interface is designed, but we can imagine end-users reporting accuracy and errors back to the system, thereby increasing or decreasing the confidence of thes system.

VI. Conclusion

With the data that's available in today's sensor-filled phones, and the amount of use already happening in these sensors, it's obvious that the data can also be used to collect auxillary, humanitarian uses like this one. The most important takeaway from this project for the authors was the fact that sensors are running everywhere, data is being generated everywhere. Finding efficient, safe, privacy-aware ways of processing and using information from these sensors, creating meaningful applications like this one can be beneficial for the society.

The algorithms and steps used in this project are not the best accuracy by any standards today. In fact, most of the methods of data processing and learning in the project have been readily available, heavily tested and easy to build. With this report, the authors would like to reiterate that it's possible to make use of the methods we already know and build meaningful applications for the people in need.

The authors plan to work on this project further. The accelerometer-based tagging algorithm can take a lot of improvements, which can decrease the processing load on the part of contributing users. It's possible to process the accelerometer readings in-situ, and store only the tags for the corresponding GPS locations. It's then possible to improve the GPS location data drastically, by using filtering methods, velocity-based estimations and other location sensing methods than GPS itself. Further, it's also possible to improve the way tags about road segments are accumulated. With increasing amount of contributors installing our app, it's possible to actually perform intelligent inferences about road segments depending on how fast the user was moving, what's the label learned for the segment and where the user went before and after that particular road segment. All these areas remain to be full of possibilities.

Acknowledgment

The authors would like to thank the authors of following open-source github repositories: [7], [8]. The authors would also like to thank their project supervisor, Dr. Khai Truong, Associate Professor, Department of Computer Science, University of Toronto.

References

[1] K. Hara and J. E. Froehlich, "Characterizing and visualizing physical world accessibility at scale using crowdsourcing, computer vision, and machine learning," *ACM SIGACCESS Accessibility and Computing*, no. 113, pp. 13–21, 2015.

[2] A. Frackelton, A. Grossman, E. Palinginis, F. Castrillon, V. Elango, and R. Guensler, "Measuring walkability: Development of an automated sidewalk quality assessment tool," *Suburban Sustainability*, vol. 1, no. 1, p. 4, 2013.

[3] J. Hernandez, D. J. McDuff, and R. W. Picard, "Biophone: Physiology monitoring from peripheral smartphone motions," in *Engineering in Medicine and Biology Society (EMBC), 2015 37th Annual International Conference of the IEEE*. IEEE, 2015, pp. 7180–7183.

[4] P. Newson and J. Krumm, "Hidden markov map matching through noise and sparseness," in *Proceedings of the 17th ACM SIGSPATIAL international conference on advances in geographic information systems*. ACM, 2009, pp. 336–343.

[5] D. H. Douglas and T. K. Peucker, "Algorithms for the reduction of the number of points required to represent a digitized line or its caricature," *Classics in Cartography: Reflections on Influential Articles from Cartographica*, pp. 15–28.

[6] P. Karich and S. Schröder, "Graphhopper," *http://www. graphhopper. com, last accessed*, vol. 4, no. 2, p. 15, 2014.

[7] G. Project, "Hmm-lib for map matching by graphhopper," https://github.com/graphhopper/map-matching/tree/master/hmm-lib, 2016.

[8] @bmwcrait, "hmm-lib: Implementation of hmm based map matching algorithm," https://github.com/bmwcarit/hmm-lib/releases/tag/v1.0.0, 2016.