Group 5: Context awareness final report

Marc Skodborg 201206073 **Stefan Madsen** 201206043 **Simon Fischer** 201206049

ABSTRACT
Find My Bike is an app that will passively track the user's bicycle position and serve as a reminder when the parking location has been unanticipatedly forgotten. It seeks to determine the user's activity based on accelerometer data, and uses *OpenStreetMaps* map data to compensate for the potential inaccuracies connected to the GPS system and provide more accurate estimates. The classifier performance is promising, although in practice, the process needs some refining as too many false positions are reported to the user, likely because of limited data as input to the training of the activity classifier.

INTRODUCTION
One of the many problems a distracted young student encounters, is that after a full day of trudging around campus one might find it difficult to remember were one left ones bike. The goal of our app is to alleviate this problem by passively monitoring where the user leaves his or her bike to avoid the need for the user to actively take actions to remember the position of his bike, as this might just as easily be forgotten.

We attempt to do this by identifying user activity based on accelerometer data through machine learning techniques. However, as we have a limited time frame, we have chosen to limit our scope to three activities, namely walking, cycling, and walking with bike. We hope to discover that this will allow us to mark the position where the user switches from an activity involving his or her bike to an activity that does not, and then, upon the user's request, present this position to the user as a possible location of said bike, all of which quite nicely follows Schilit and Theimer's definition of context:

> location, identities of nearby people and objects, and changes to objects [6]

as we are attempting to track the changes in position of an object identified by its user.

SYSTEM REQUIREMENTS
Based on our initial thoughts, we have defined the following requirements.

Functional
- Allow the user to quickly locate their bike with a reasonable accuracy
- Automatically update bike position based on user's activity
- Determine user's relevant activity from accelerometer and positional data
- Adapt to user-specific movement- and positional patterns over time to improve accuracy

Non-functional
- Manually mark the position of the bike before leaving it
- Movement calibration, by user stating current activity explicitly

DESIGN AND IMPLEMENTATION
When designing the app there are two primary focus points. The first is collecting and using training data to train a classifier to allow for differentiation between activities with a satisfying accuracy, and the second is making the app useful for the user through putting the activity classification to use to solve the problem the app seeks to solve. In our current design, the focus is primarily on the first part, with only a smaller demo of the final app shown.

Data collection
For determining the user's higher level context, i.e. the user's activity, we rely on low level device internal data, which we want to passively track at all times. Due to battery consumption considerations, and the fact that GPS will not always be available, we did not want to rely on data from the GPS, despite this causing a 10.4% drop in accuracy as found by [3], hence we have chosen to only use the phone's accelerometer as data source for this task.

When collecting the data, we need to track accelerometer events during the three different activities our app should distinguish between, namely walking, cycling and walking with a bike. We do this by associating each of the three activities with their own button, starting the tracking of the accelerometer events, to let the app know which activity we are currently doing, and labelling the data accordingly. When the button is then pressed again, the tracking is considered done, and the data will be saved to an `.arff` file in the phone's `Download` folder, named according to the activity undergone.

When collecting our data, we placed our phone specifically in our right pocket with the screen against the leg. We agreed to do it this way to ensure consistent data to reach the most accurate model. This phone positioning constraint is primarily due to the limited amounts of data we were able to gather during the course, but could presumably be eliminated in a real scenario with more data input to the training of the classifier with the phone in different positions. At the moment we have no data when holding the phone, or when the phone is in the jacket and other scenarios, which is an issue left for future investigation.

App for the user

Our app can be, but is not yet, divided into two separate parts. One part dealing with monitoring the user's activity, and one which the user uses to access the stored bike position. The first part would need to be a background service, as our app's functionality requires this part to always be running and evaluating should the need arise to determine at what position in the past the bicycle was parked at, and should therefore not stop running because a user terminates the app. The second part is the presentation of the information gathered by the background service, probably in the shape of a marker on a map. This part would also house any kind of calibration options and other settings.

In order to improve our bike position estimate, we utilize external data in the form of *OpenStreetMaps* which is a community-driven database housing map details such as the location of benches, trees, buildings and, particularly interesting to our app, bicycle parking positions and areas around the world. This information is accessed via a read-only mirror, `overpass-api.de`. We use this information together with the accuracy of the position estimate as provided by the GPS to more accurately pinpoint the parking position of the bike. We follow the logic that if there is a bicycle parking area within the accuracy range, as illustrated in figure 1, then odds are the user utilised the bike parking. We relate the GPS accuracy and the need for external data by basing the geographic area in which we download the corresponding data on the reported accuracy of the GPS.

As an example, the GPS could report an accuracy of 20m, illustrated in Figure 1 by a blue circle around the position estimate indicated by the red star in the middle. We now define a window by the positions 20m north, south, east and west of the estimated position by the GPS. The map details within the window are downloaded and scanned for bicycle parking areas, and, if one is found, this is assumed to be where the bicycle was parked. The more accurate the GPS estimate is, the smaller window we will define, and the less likely we will be to find a bicycle parking area within it. Thus, the external data will play a role proportional to the uncertainty of the GPS estimate, becoming more and more relevant the less certain we are of the initial estimate. If we do find a bicycle parking area, as the one defined by it's corner points as seen marked with green stars, we will then assume the point defining the bicycle parking area closest to the GPS estimate, as indicated by the yellow line, is the best fit for a prediction on the actual position of the bicycle and report this.

We have implemented this in a limited fashion, as during the implementation of this, we discovered that in the `xml`-data provided by *OpenStreetMaps*, bicycle parking can be marked in more than one way, i.e. being specified as a single point, or an area defined by multiple points. In our implementation we have only taken the two mentioned formattings into account, there might exist more unknown to us as of this writing.

This estimate-correction functionality could be placed in either the background service or the app itself. One could argue that unless the user wants to see the position, there is no need to contact a web service in order to update the position, and therefore we should do this in the app when the user wants to see the bicycle position. Performing this adjustment in the app, however, could result in longer load times for the user. As the downloaded data is small enough, and assuming the user usually can wait a few seconds while the server is contacted and the map data is downloaded, and assuming he will remember where his bike was parked more times than not, we have chosen the correction to only happen when the user requests the position estimate, and not as part of the background routine.

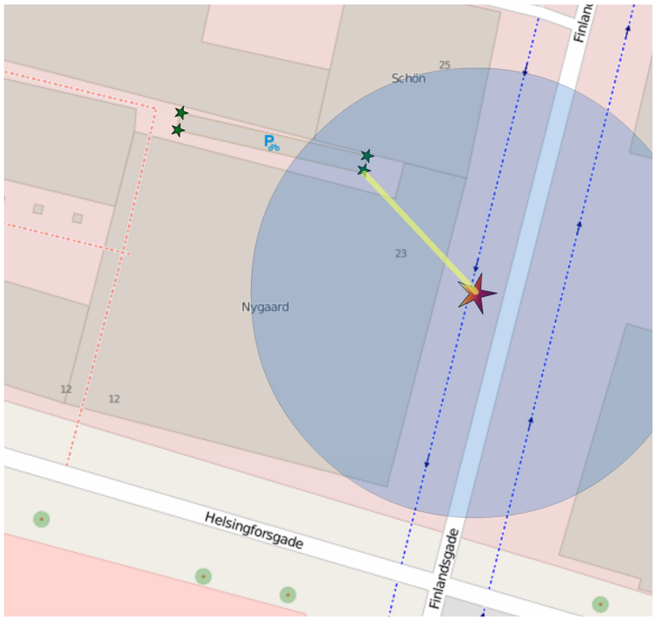


Figure 1: Updating bike position using OpenStreetMaps

We also envisioned the possibility that the user could calibrate the app to be more closely attuned to his or her specific walking/biking style. We have not implemented this, but it would be implemented much like our data collection procedure, where the user would have to actively indicate their current activity. We would then send this data to a server, which would retrain a classifier based on the data for the general classifier, but be more specifically trained for each user by somehow weighting their own data more. This is a task for a server as retraining a classifier is a fairly computationally heavy task.

EVALUATION AND RESULTS

When collecting data, we merge it into windows of 32 samples as we collect them. We chose windows of 32 samples for multiple reasons. We tried with multiple window sizes and all resulted in a model that had a satisfying accuracy of around 90%, using the leave-one-out strategy for evaluating classifier performance on our limited data sets. As the performance was close to identical on the tested window sizes, we chose the smallest one tested as to more quickly gather time windows to classify upon, resulting in a quicker activity recognition. This is crucial, as it allows for a GPS fix before the user has moved a meaningful distance from the bicycle when parking it.

In [5] they further mention that experiments have been conducted, which showed that the accuracy of activity recognition "decreases as the window size increases". If we used window sizes corresponding to 25 sec time slots, using the sensors and reading frequency default to our phone at hand, as we tried initially, the user would have moved a significant distance from the parking position as we gathered the handful of windows needed to determine activity with higher certainty. If the window sizes, on the other hand, correspond to 5 sec time slots, the distance is much smaller before we notice the activity change and record the user's position.

When collecting the data, we create overlapping windows as this helps to alleviate transitional noise as mentioned in [5]. Next, we calculate the features max-, min-value and standard deviation for each window, however, we have not tested whether these features are the optimal ones to use in our case, or if some of the other features mentioned in [5] would have given better results, but in future work this is an aspect that could be investigated.

To generate a classifier, we chose the C4.5 decision tree algorithm, as represented by WEKA's J48. We chose this for several reasons. Firstly a decision tree is quite easily visualised, thus allowing for an easier understanding of what it is actually doing, and secondly because the C4.5 algorithm consistently achieves a high score in [1], and lastly, it outperformed the `Naive Bayes` algorithm with 9% on our test set. With the data we managed to collect split into two separate sets, one for training and test respectively, we managed to get 87% of our test set correctly classified, using the classifier depicted in figure 2a.

This decision tree seems surprisingly simple, which could indicate that our training data is fairly uniform, and could therefore do with being extended. This simplicity did not carry over the classifier trained on our full data set, as depicted in figure 2b. On the contrary, this full classifier is fairly complex, and branches multiple times on the same feature. This apparent complexity is, however, not necessarily a sign of a good classifier, but could mean that we overfit the data, and that the decision tree should be pruned to better generalize to unknown data. A good example of possible overfitting can be seen in the bottom left corner of figure 2b, where the classifier branches three times in a row on the max feature, and the three branches classify a total of 14 out of roughly 850 data points, so possibly not worth the effort. The simple decision tree resulting from training on the train set could be explained with our choice of using the strategy of leave-one-out while only having two people gathering data. The training data might indicate a very distinct feature separating one person's style of walking and bicycling, but when mixing it with another person, whose walking an bicycling styles might differ a lot from the first person's, the complexity is much bigger. More data would, as usual, help us generalize and improve.

As the classification of activities is more difficult and less accurate than first anticipated, false activity transitions will occur. As we tested the app in practice, parking by a marked bicycle parking area by the Veri Center shopping mall in Risskov, Aarhus[1] and walking around inside the center to limit the GPS accuracy, we achieved a lot of falsely reported activity changes on the way and while walking around inside the mall. One idea might be to record and present a history of several possible estimates and let the user decide on the most probable one.

DISCUSSION

Walking with bike

When tracking the position of the bicycle, we need to know when the bicycle is changing position. This happens when we are cycling, or walking with the bike. We have collected one hour of data where we were walking with a bike and tried to use this with the rest of our training data at the time, and tested our new model. With the new data we went from 98% correctly classfied to 77%. The model had a hard time recognising whether a data point was `walking` or `walking with bike`.

There could be several reasons why the model cannot recognize the difference between the two classes.
When testing `walking with bike`, we did not have that much data, and all the data we had from walking with a bike came from the same user. This does not give us representative data, and combined with walking data from multiple users this might have been a huge flaw. Before we can make any conclusions to whether we can distinguish those two activities, we need more data, and we need more data from multiple users.

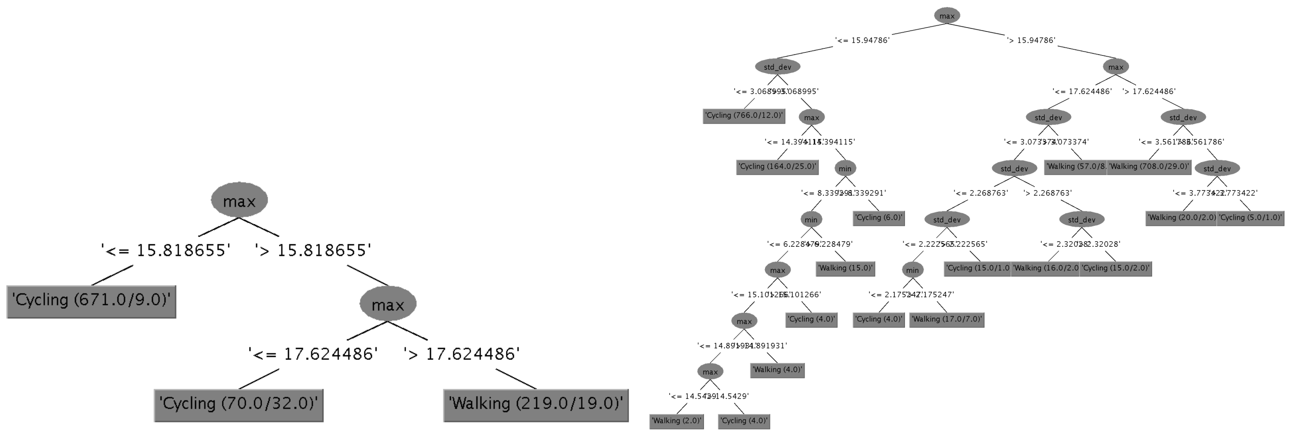
Another reason why we are not able to distinguish those two activities might be that there are no sensable difference to the phone. After all, the phone is located close to the user's legs. Our intuition tells us that there probably are no difference, but we chose to test it anyway, in the hopes that the data would convincingly tell us otherwise.

Sensor fusion

When we are tracking the user's activity, we have so far only used the accelerometer in the user's phone. We have found it to be hard to distinguish between walking and walking with a bike. This is a significant flaw in our design, and should be solved for the app to be useful. A way to potentially improve our results is to include data from the accelerometer in a smart watch. When holding a bike we expect the user to move his arms differently than when walking without holding a bike by his side.

We imagine the accelerometer data of both devices being used in conjunction to define an observation, and their respective behaviours over time define a window. If we only used the smart watch, we imagine the same problems appearing when trying to differentiate between cycling and walking with the bicycle. When combining the data, we do see the potential of being able to differentiate, based on intuition. This, of course, needs to be backed up by experiments in future work. The

[1]latitude 56.1887033 and longitude 10.2151839 in the OpenStreetMaps data



(a) Decision tree based on training set (b) Decision tree based on full data set

Figure 2: Decision tree classifiers based on WEKA's j48 algorithm

possible solution is not without problems, as our app would then be dependent on external devices, which most user probably would not have at the time of writing.

Another way to ensure the app's functionality is to use a beacon on the bicycle. The app would then record when it approached the beacon, and log the position of the user when leaving the proximity of this beacon. This solution too has its disadvantages. The app would require external devices to be useful once again, but most likely perform much better too, as the uncertainty tied to machine learning being part of the solution would no longer be.

Different amount of data points

When collecting the accelerometer data we used two different phones, and realised that the amount of data points gathered each second was not the same. On one phone we would get four times as many data points compared to the other phone, a fact of reality also discussed in [4]. This presents some challenges. If we do not get the same amount of data, it takes different amount of time to collect a window size. On one of the phones, we saw a window be collected every few seconds, while on the other it took 20-30 seconds before we had collected a window. When recognizing whether a person is changing activity from cycling to walking, this can cause problems as previously discussed.

One way we could fix this is changing the window size definition. Instead of having a fixed size of 32 samples, we can define a window to be the data collected in a certain time slot. This gives us a very precise way to know when a window is collected, and we can identify the user's activity when he is changing his activity. Further, this allows for comparing and combining data from the previously mentioned smart watch. We have not tested this approach, and we do not know if will work well enough in practice, or if we would need a more complicated model like the ones mentioned in [4]. It will be a consideration and an experiment left for future work.

Future work

Due to the nature and time frame of this project, there are a lot of areas that has left room for improvements. A quick comparison with [2] makes this abundantly evident, some examples could be their 54 features versus our 3, despite us both solely relying on accelerometer data.

Another aspect that might be worth considering is the multilevel classification mentioned in [2]. In our app we could start with a coarse grained classifier to determine if the user is moving in a motorised vehicle or not This would allow our app to ignore all motion determined to be caused by motorised transportation means.

CONCLUSION

The goal of the application was to offer an app that passively monitors where the user parks his or her bicycle, in case this location will unanticipatedly be forgotten.

Much of the challenge proved to be the activity recognition and the accuracy hereof. While we feel the project is doable, our results suffered primarily due to lack of data. We achieved a promising successful classification rate, but the practical use of the application still suffers from too many false observations of a change of the user's activity. We managed to use external data from *OpenStreetMaps* to compensate for the varying availability and accuracy of the GPS position estimates, but the classification of activities needs more work all in all.

REFERENCES

1. Reza Entezari-Maleki, Arash Rezaei, and Behrouz Minaei-Bidgoli. 2009. Comparison of classification methods based on the type of attributes and sample size. *Journal of Convergence Information Technology* 4, 3 (2009), 94–102.

2. Samuli Hemminki, Petteri Nurmi, and Sasu Tarkoma. 2013. Accelerometer-based Transportation Mode Detection on Smartphones. In *Proceedings of the 11th*

ACM Conference on Embedded Networked Sensor Systems (SenSys '13). ACM, New York, NY, USA, Article 13, 14 pages. DOI: http://dx.doi.org/10.1145/2517351.2517367

3. Sasank Reddy, Min Mun, Jeff Burke, Deborah Estrin, Mark Hansen, and Mani Srivastava. 2010. Using Mobile Phones to Determine Transportation Modes. *ACM Trans. Sen. Netw.* 6, 2, Article 13 (March 2010), 27 pages. DOI:http://dx.doi.org/10.1145/1689239.1689243

4. Allan Stisen, Henrik Blunck, Sourav Bhattacharya, Thor Siiger Prentow, Mikkel Baun Kjærgaard, Anind Dey, Tobias Sonne, and Mads Møller Jensen. 2015. Smart Devices Are Different: Assessing and MitigatingMobile Sensing Heterogeneities for Activity Recognition. In *Proceedings of the 13th ACM Conference on Embedded Networked Sensor Systems (SenSys '15)*. ACM, New York, NY, USA, 127–140. DOI:http://dx.doi.org/10.1145/2809695.2809718

5. Xing Su, Hanghang Tong, and Ping Ji. 2014. Activity recognition with smartphone sensors. *Tsinghua Science and Technology* 19, 3 (June 2014), 235–249.

6. Nervo Verdezoto. 2015. Introduction to Context-Awareness - Lesson 1. PowerPoint slides. (2 November 2015). Lecture notes from the course Context Awareness at Aarhus University.