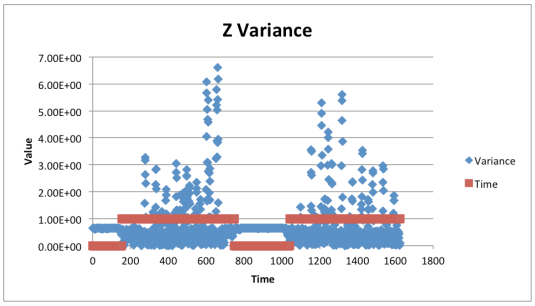
IN4254 Final Report Option 1
Mayank Jain (Student ID: 4437209)
Jerun Trajko (Student ID: 4435508)
Device: Samsung Galaxy S2 GT-I9100
Operating System: Android 4.4.4

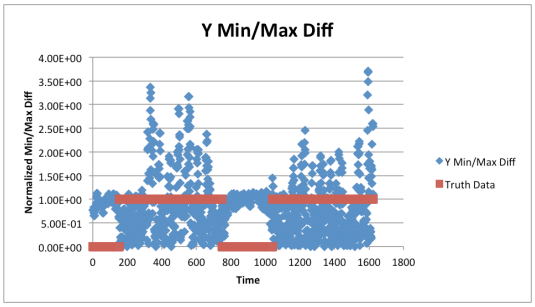
Queuing

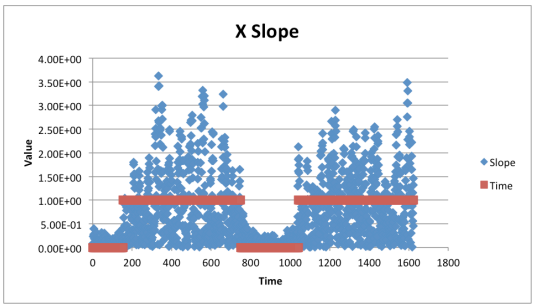
We started by collecting the X, Y, and Z accelerometer values for walking and standing data, and extracted as many features from it as possible. We calculate the slope, variance, and difference of the max/min values on each axis. We gathered this information from groups of nine data points, as this accentuated the variance and max-min differences. Smaller group sizes led to less accurate results during real world testing. A larger value would increase accuracy, however at the expense of speed, especially during transitions from starting to stopping and vice versa. Not very much training data was required, simply 20 seconds of walking around stopping and starting was sufficient.

Data Processing

We processed the raw accelerometer data files in Python to extract different features. We extracted the slopes, variance and max/min differences on each axis. We initially tried to keep group sizes small, but when we generated graphs in Excel and realized that we weren't able to tell the difference between walking and standing by sight, we discovered that bigger groups worked better. We then normalized the data so that we could input it into Weka. We tested multiple K-sizes and distance weighting settings with the dataset to see what gave us the greatest accuracy. We were surprised to find that neither of the factors significantly altered the accuracy of the algorithm except for extremely large values of K (not even K=1). Our testing data was around 97% accuracy regardless of the K-size. The least helpful feature by far was the max/min difference, which on it's own had only 85% accuracy (compared to ~95% accuracy for the other two features), however it increased our total accuracy by 2%. This surprised us as we actually expected the slope values to be the least helpful feature.







Prediction Method

We used a double pass approach when comparing, choosing the most common of the last 5 predictions. As described above, we extract our desired features from readings in groups of 9, creating 5 new data points from the extracted features. hen we used a KNN approach where we looked at the 15 data points and saw if in that data set the majority of points were moving or standing still. The majority queue status is assigned to the group of 3 data points given to us by the phone. We stored the data on the device in a KD tree. We initially started by recalculating the distance from each trained point for every new reading, but we quickly realized how inefficient that was in terms of processing power, as the app

couldn't handle so many calculations so quickly. Another concern is that KD trees become more inefficient as you increase the K-size, however we didn't notice any app performance decrease by increasing K from 6 to 9, and thus decided to use all the features we had calculated.

Localization

We took approximately 100 readings in each cell broken down into 25 readings each while standing in 4 different directions that were 90 degrees rotated. We moved in a grid fashion, although this was occasionally not possible due to items in the cells, and also accounts for some deviation away from the 100 readings. We made sure to distribute all the readings across the full area of each cell in order to have consistent results no matter where the application is being run. A large issue we came across during our testing data retrieval was that the data varied with time but we tried to gather all the data later in the afternoon in order to keep it consistent. There were many local hotspots from students in the building but we felt that they didn't alter our performance results as much as we initially thought.

Data Processing

We initially processed the data by created a 18x255 array for every MAC address of the probability distribution for each cell and RSSI value. We initially tried to convert these into a Kernel density estimation to smooth the data while retaining a truer representation of the values of the data in each cell. We thought a normal distribution might be too naïve to handle the size of the cell and the changes in signal strength from one corner to another. This method increased accuracy in offline data processing by at most 5%, however after running multiple tests on the phone on different days we came to the conclusion that due the lack of consistency in the signal strength over time this didn't work as well as we'd hoped.

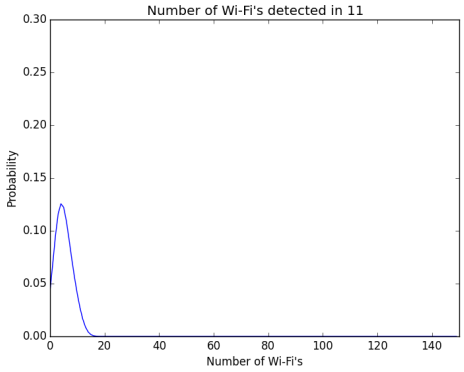
To clean the data we eliminated MAC addresses from the data that weren't present in at least 5 scans, as we assumed they weren't consistent enough to provide an accurate probability distribution. At first we worried that this would not be enough due to the number of local hotspots on the floor. These hotspots were present on almost every scan in the cell. While processing the data offline, we generally received a 65%-75% accuracy rate, and over 90% accuracy including neighboring cells. Below is our generated confusion matrix.

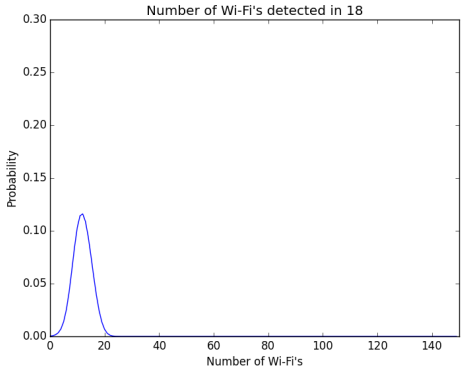
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
7	-	2	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-
-	5	-	2	-	-	-	-	-	-	-	-	-	-	-	-	-	-
-	-	3	6	-	-	-	-	-	-	-	-	-	-	-	-	-	-
-	1	-	7	-	1	-	1	-	-	-	-	-	-	-	-	-	-
-	-	-	2	8	-	-	-	-	-	-	-	-	-	-	-	-	-
-	-	-	1	3	2	3	-	-	-	-	-	-	-	-	-	-	-
-	-	-	-	-	-	6	2	-	-	-	-	-	-	-	-	-	-
-	-	-	-	1	-	1	5	3	-	-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	2	8	-	-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-	1	8	-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-	-	-	10	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-	-	1	-	6	1	-	-	-	-	-
-	-	-	-	-	-	-	-	-	-	-	2	5	1	1	-	-	-
-	-	-	-	-	-	-	-	-	4	-	-	2	2	1	-	-	-
-	-	-	-	-	-	-	-	-	-	-	-	-	3	5	2	-	-
-	-	-	-	-	-	-	-	-	-	-	-	-	-	2	8	-	-
-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	3	7	-
-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	1	1	8

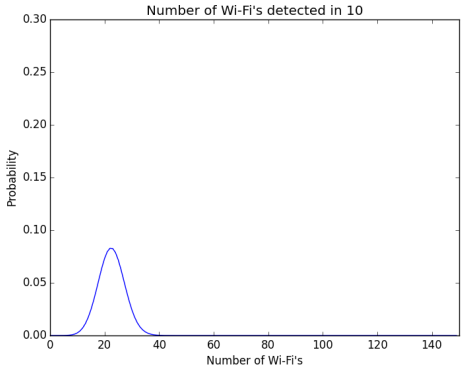
To process the data we used Python, and the NumPy, SciPy, and MatPlotLib libraries.

Prediction Method

When making cell predictions on the device, we take the 10 strongest Wi-Fi signals and use it to compare the probabilities. We start from the weakest of the 10 signals and work towards the strongest. Thus the weakest signal affects the probability $1/2^{10}$ as much as the strongest signal. If there are less than 10 signals available, we also incorporate the probability distribution of the number of MAC addresses sensed. This mostly applies to cell 11 and 18, which are in corners of the building, and are otherwise the hardest to predict because of the variability of the Wi-Fi data. Keeping track of the number of distinct MAC addresses made these cells very easy to distinguish and improved our performance greatly.







Most of the signal quantity distributions look like cell 10, with a mean above 20. Cell 11 and 18 are the only ones with a mean well below 20.

Each Wi-Fi's signal strength is also averaged over time, thus the longer you stand in a cell the more likely you are to receive the correct localization. We used an array which was indexed by cell number. At each cell index, we stored an integer which told us how many times a prediction was made that we were in that cell. Once at least 2 Wi-Fi scan predictions have completed on a certain cell, we report the most commonly predicted cell to the user. With time, the array will keep getting the prediction frequencies at each cell and recalculate what cell you are in. We found out that with time this was the best approach because the correct cell number would usually end up being the majority giving us the right prediction.

Challenges & Innovation

Queue

One of the initial obstacles we came across was that our data processing process used a lot of calculations. As we mentioned, storing the trained data in a KD-tree was the most efficient solution. We also cut the speed of the accelerometer sensor events significantly (about 10x), as the increased speed didn't significantly affect accuracy but doubled the processing power and battery drain. We also noticed early on that the app initially worked best only with the person it was trained on, however adding variability to the training set was sufficient to solve this problem.

Wi-Fi

By taking into account the number of Wi-Fi's present in each cell we dramatically increased our accuracy for cells 11 and 18. Rather than reporting the strength after reaching a threshold value, we simply report the cell with the greatest probability over the course of the calculation. This allows us to use our method of working from the weakest Wi-Fi's towards the strongest, and weight them accordingly, without the risk of returning a prediction prematurely based on less accurate data. By waiting for multiple scans to take place and returning the most common value, we also increase the accuracy of our predictions (and data) over time. Thus the user can make the tradeoff between time and accuracy without us forcing them one way or another.

Individual Workload

Queue -

Jerun Trajko

In the Queue part of the project I began working a lot with the files. I initially made a temporary layout in which we can see

accelerometer data and start scanning after a button is hit. I then created a file which take this data and store it in .txt format on the sd card. The values were stored in the format: time,x,y,z. I then created a second file which stored slopes, maximums, minimums, and max/min differences of the data in the first file. I calculated all of these elements after grouping the data in the first file in groups of 3 accelerometer recordings. I also worked on the normalization of the testing data. I created a function which took the mean and standard deviation of slopes, max/min differences, and a data point to normalize the data. It returned an array of the normalized data. All of this was also linked to the final application layout that I created.

Mayank Jain
I worked on all the offline data processing in Python, Excel, and Weka. I also decided how best to optimize the algorithm (eg experimenting with the app to determine to best values for K in KNN and the number of accelerometer readings). I also integrated the KD-tree and wrote the code that actually uses all the calculations to make the prediction, such as the double pass method and the grouping of data points.

Wi-Fi -
Jerun Trajko
Like the queue part I worked on getting the Wi-Fi data and writing it to a file. I made a temporary layout where the user puts in what cell they are in and they can scan, displaying the results in a text field. Initially we stored everything from the time, cell number, bSSID, rSSID, and SSID (although we only really needed the MAC address and not the name of the Wi-Fi hotspot). I was also in charge of getting the data. I then created a dictionary/map to store the probabilities. It took a MAC address and returned a double array where the axes were the number of cells and all possible rSSID values (0...255). I did this by parsing a file consisting of the MAC address, cell number, and all rSSID probability values from 0 to 255. I took a few more testing data recordings to improve the accuracy since we had some more time. Finally, I created the final layout for the application.

Mayank Jain
I worked on all the offline data processing in Python, which included calculating the probabilities, generating the confusion matrix, understanding the data well enough to make the tweaks in our algorithm described in the Innovation section. I wrote the code in the app that calculates the probability of each cell, and returns a single prediction. I also wrote the code that determines when there are enough predictions to present it to the user.

Possible Future Directions

Queue
While our app is pretty accurate, it could be possible to make the algorithm more efficient, so that it can run in the background of apps with a more complex purpose. This could include optimizing our training data set by removing redundant features or extremely similar data-points, and removing extreme data-points to save space (although the accuracy of KNN isn't affected by outliers). Also because KNN is an algorithm with simple training but complex testing, it probably isn't the best algorithm for a mobile device with processor and battery life limitations.

Wi-Fi
One obvious improvement would be to gather more data. Initially we took 100 Wi-Fi scans at each cell with each 25 in a different direction but obviously, having more would improve our accuracy. Furthermore, we recorded most of the data at the same time. If we took more data at different times it would create more accurate data to compare to. Having a large quantity of data is a limitation of the Bayesian method however. Using particle filters would allow us to increase our accuracy to be more precise than a 4x4m cell, and also require much less testing data. If the location of the routers is known, it should also be possible to triangulate the user's position

based on the signal strengths of multiple MAC addresses. A potentially easy way to calibrate the data for different phones/users would be to set up a beacon of some sort (Bluetooth?) at the door/elevator, as you will always know their location at that time.

References

Our main references were the lecture slides. We used other standard sources such as StackOverflow and Android Documentation. We used the Java KD-tree implementation by Julian Kent which can be found here: http://robowiki.net/wiki/User:Skilgannon/KDTree.