Magic Wand: Custom Motion Gestures to Control a PC

Tanuj Mittal
UCSB
tanujmittal@umail.ucsb.edu

Alick Xu
UCSB
alickrxu@umail.ucsb.edu

Harshitha Chidananda
UCSB
harshitha@umail.ucsb.edu

ABSTRACT
Motivated by the scope of leveraging motion sensors in smartphones, we developed an application that allows a user to make gestures to control a PC. We utilize accelerometer and gyroscope readings to classify gestures using machine learning techniques. We provide an extremely flexible way to define custom gestures with little training and very high accuracy in classifying the gestures with minimal turnaround time. Our main contribution is that a user can define their own gesture by training it five separate times. Then, they can assign that gesture to a particular action to control the laptop. In order to classify features from the time series of accelerometer and gyroscope readings, we use an algorithm called FastDTW, a faster version of Dynamic Time Warping, which normalized the time data and then fed the data into a 1-nn classifier. We conducted experiments and found that a 1nn classifier with FastDTW is very hard to beat. The main focus of the paper is on learning and classifying custom gestures. But, it also encompasses laptop smartphone communication, converting keystrokes to actions and attaining good level of accuracy.

KEYWORDS
Time series data classification; Dynamic Time Warping; DTW; knn classifier; Fast DTW; parallel classification; gesture recognition

INTRODUCTION
Smart phones today are equipped with number of sensors. Almost all smartphones today come with accelerometer and gyroscope sensors. These can be used to detect 3-axis changes in acceleration and rotation of the phone itself respectively. Each movement in the smartphone gives different values of x, y, z in both the accelerometer and the gyroscope. This myriad set of values can be used to perform various actions. We are using this from a pool of sensors to associate each movement with an action. Gestures have been a basic form of expression for humans for years. They are a natural and intuitive way that people communicate in daily life. However, it is only recently that gestures have been explored as a new way of user interaction with technology. We wanted to achieve this goal. We are coupling sets of accelerometer and gyroscope values with different gestures in the application. Each gesture is a set of distinct values and hence this can be used very effectively in gesture controls. Our goal was to create an application that could accurately detect gestures, and interact with a computer using these gestures. We hypothesized that having a system like this would offer a whole new paradigm of user interaction. We were able to implement such a system, dubbed Magic Wand. Magic Wand has two components

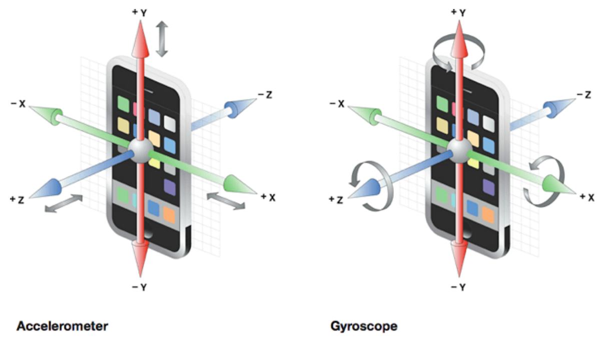

Figure 1. Accelerometer and Gyroscope sensors

- an android application, and a Java server. The android application records accelerometer and gyroscope readings when a user performs a gesture, holding the smartphone in their hand. The application then sends the raw recorded readings over to the Java server, which classifies the data as a certain gesture, then performs the action corresponding to that gesture. The feature that was core to our application was the ability to train new gestures. Every user is different, and prefers using different gestures for tasks. We wanted to be able for users to be able to use their own gestures and map those gestures to their own preferences for computer actions. We did not find any research that tried to do this, so we implemented it ourselves with success, making a framework on which custom gestures can be learned by a user. The contributions that we made are:

- Users can train the model to use their own custom gestures.
- Creation of a parallel classifier.
- Provide ability to control a laptop with a smartphone.
- User study evaluating the accuracy of this system.

BACKGROUND
We wanted to create this system because we observed the current space of gesture detection and felt that it did not fit our needs. Most gesture implementations that we have observed have many limitations. Some gestures are hardware specific - for example, on the Moto X, there is a specific chip that detects if you are shaking your phone or flicking your phone to activate certain hardware features, such as turning on/off the flashlight or activating [5]. This is very useful, but since it is hardware specific, there is no way to have these gestures available to everyone. In order to use these gestures, a user must specifically possess that particular phone. Other gesture implementations are simply inaccurate. The communication between laptop and mobile smart phone happens using wifi.

Figure 2. Communication between laptop and mobile smartphone

Wifi has wider range compared to that of Bluetooth and also large amount of data can be sent easily using wifi.

There are implementations out there which use accelerometer and gyroscope readings to classify gestures, but do not use machine learning to increase the accuracy. What results is that these implementations are very specific - the gesture can only be measured by specifically holding the phone in a certain orientation and moving it in a specific way to register the gesture as detected. Because of this, these implementations are not very user-friendly, and are hard to learn to use. Realistically, when performing gestures, users should be able to make the general shape of the gesture and have it register properly. With human gestures, this is the case, so how come it is not the case with smartphone gestures?

We came across research detailing how this has been attempted before, first using only accelerometer data and performing machine learning on that raw data [8]. This proved to be useful - however, there was still the problem that the phone had to be held in a certain orientation in order for gestures to register. This was due to the fact that they did not use the gyroscope. There were other papers that detailed how they used gyroscope and accelerometer data time series to classify their data [1]. We found that one of the algorithms, Dynamic Time Warping (DTW), was the most accurate at normalizing the raw data values, as well as easy to implement. Our research also showed that using DTW filtered data and passing it into a k-nn classifier was the most accurate course of action. However, DTW is a dynamic algorithm, meaning that it has a quadratic runtime complexity $O(n^2)$. We had to figure out how to make this faster in order for such an application to be feasible.

There is some research in the field of time series classification which details on how it can be scaled and used in production level applications [6]. Some techniques like indexing the data for narrowing down the probable classes in the model also provide a huge performance boost. However these techniques are only applicable if the amount of training data is huge, i.e. billions of time series. For our use case, we cannot rely on such methods because the domain on which the classifier is working is very small and will hardly provide any improvement by using iSAX [4]. Some research groups have published tools [7] for performing fast time series classification on billions of training data. They provide a great reference on different techniques for optimizing the system.

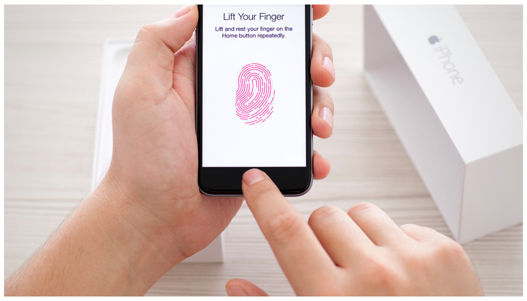

Figure 3. Registering a finger in TouchID

One thing we were interested in was allowing users to implement their own gestures. Most smartphone gestures have preprogrammed gestures that might not be intuitive or preferred for certain actions. We wanted to push the limits and create a framework that allowed users to make their own gestures and map it to whatever they wanted. This was both challenging and exciting - we had no idea if this was going to work. We imagined this training process to work similar to how Apple iPhones train their TouchID functions by scanning users fingerprints multiple times to register them. We envisioned our gesture training process to work in a similar fashion - having the user train their gesture by performing it a handful of times to register it. However, it was important to strike a balance between enough training data and too much training data that a user is not compelled to create their own gestures. If a user had to do their gesture hundreds, or even thousands of times to register their gesture accurately, such a feature would never be used. Even the research we perused did not specify how training their gestures in this fashion would be feasible.

Therefore, we sought to create such a system - a gesture recognition smartphone application that allows users to create their own custom gestures easily. We were able to succeed in this front, and will talk about our implementation and results in the rest of this paper. In addition, we wanted our system to be accurate, easy to use, and ubiquitous - any smartphone should be able to download our application and use it out of the box. On that front, we mostly succeeded - the application is completely working, and only needs a couple of UI enhancements and tweaks to make it more intuitive for beginner usage. Our Java server can be installed on any PC using the jar package. However, given more time, the whole system could use a bit more polish and user-friendly refinements.

CHALLENGES

Interacting directly with a keyboard or touch based device requires user attention and focus. We wanted to explore if it is possible to identify arbitrary user movements using the motion gestures in a smartphone to perform actions on a PC. Our goal is to provide user with option to define any kind of gesture and associate that with an action on a PC. This required us to have a very fast way to train user gestures on the fly using a smartphone itself. It also required us to enable fast and accurate motion gesture detection so that the user doesnt have to concentrate too much on performing the gesture itself. We did not want to put any kind of restriction on the kind of motion gesture therefore we needed a mechanism to detect

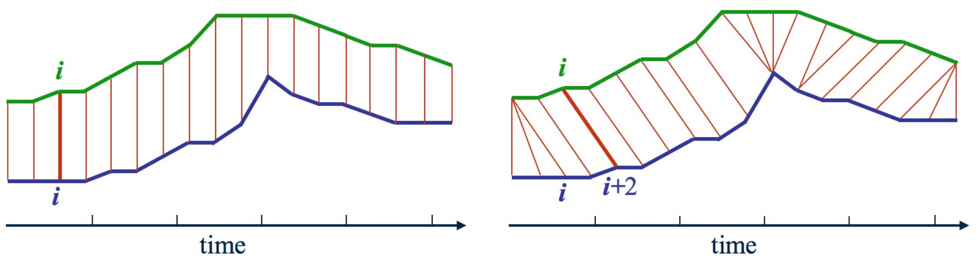

Figure 4. Euclidean distance Vs Dynamic time warping

the gestures irrespective of scale and time taken to perform it. We also wanted to offload any kind of expensive computation from smartphone to the PC for energy and performance reasons, although we recognize that it would be interesting to see how putting the classification computation on the smartphone would affect energy.

There were some challenges in order to achieve the above requirements, namely:

- Fast way to record and send the motion sensor data from a smartphone to the PC.
- Accurate way to measure similarity between different kind motion gesture data, which considers the gestures with different time and scale similar.
- A way to classify gesture data and provide flexibility in adding new kind of gestures.
- Extremely fast turnaround time from when the user performs a gesture till the action is performed on a PC for intuitive reasons.

SOLUTION

Record and send motion sensor data

We use WiFi based communication between the smartphone app and the PC application. The raw motion sensor data is sent as soon as the user records a gesture. The PC app is responsible for processing and detecting a gesture. Currently the application requires both the smartphone and the PC to be on same local network. There is a button in the Android app which the user needs to hold in order to start and stop recording the motion gesture data. Holding the button starts storing the motion sensor data, and releasing the button stops the stream of data. All the data recorded is then sent to the PC app over WiFi. Since the amount of data being recorded is very minimal (almost 100 time points in a gesture) it is pretty straightforward to use alternative channels like Bluetooth, although we did not test this.

Training

In order to train a new gesture, the user creates a new gesture with a name and the command that they would like it to be mapped to - although both the name and command can be changed later. Then, the user performs the gesture 5 times in order to train the system. When training, the gestures should be done as accurately as possible. We chose 5 times because it struck a balance between having enough data points to provide accurate classification results, and not having too many training input that the user gets frustrated.

Measure Similarity between Time Series Data

Techniques like Euclidean distance (or Manhattan) compute distance between every i-th point in one time series to every i-th point in the other time series. This puts a lot of constraints when it comes to our use case. We want to consider time series with different time scales similar. We also want to minimize any kind of artifacts caused due to subtle changes in how the motion is performed like moving the hand in a small circle and moving the hand in a big circle. We are primarily focused on just the shape of the time series curve irrespective of magnitude and time scale of the feature. Therefore we cannot use such distance functions.

Dynamic time warping is an algorithm for measuring similarity between two temporal sequences which may vary with time and speed (Fig 4). For instance, faster movement of phone along x-axis and slower movement of phone along x-axis is both treated as same in dynamic time warping. This is a much better distance measurement for our case. However it is quadratic in time complexity. This makes it really slow in actual implementation. We used original DTW algorithm in our initial prototypes of the application and found that it was barely usable because of huge delays due to DTW computation.

FastDTW [2] is a more time optimised version of DTW which works on the same principle but uses a limited warping window at every point in time series for distance calculation. At a high level, it reduces the number of data points that must be calculated. This gives a huge boost in terms of speed because it is almost linear in time complexity while providing similar results as that of DTW. We used euclidean distance to measure the actual drift between different features of two time series at a particular point once we determine the optimal warped points using FastDTW.

Classify Time Series Data

Once we had our optimal warped points using FastDTW, we passed these points into a classifier to classify the gesture. To that end, we have developed a k-nn classifier in the PC application which performs the classification of time series data

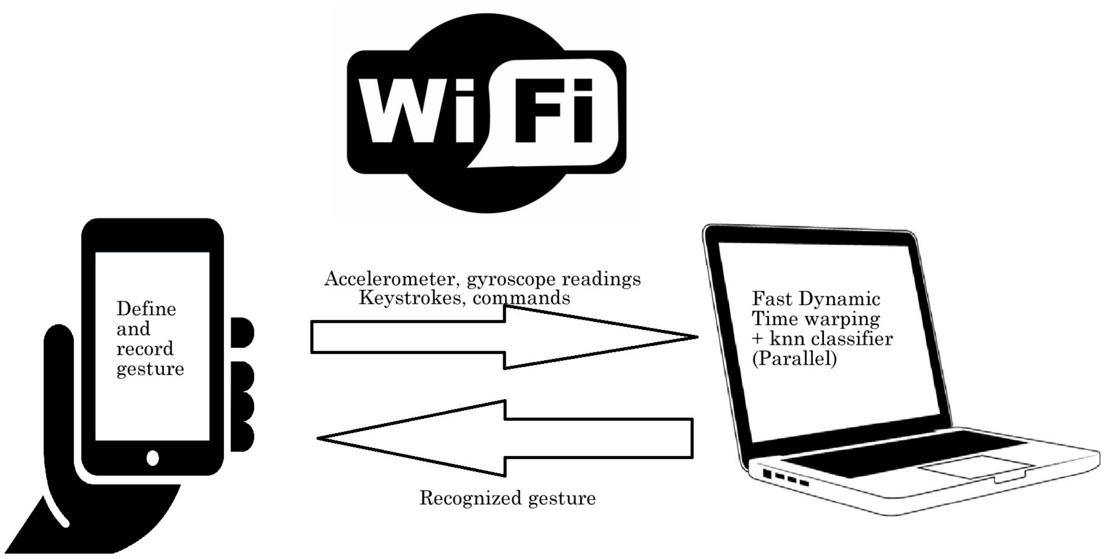
Define and record gesture
Accelerometer, gyroscope readings
Keystrokes, commands
Fast Dynamic Time warping + knn classifier (Parallel)
Recognized gesture

Figure 5. Communication through wifi

of gestures. It uses FastDTW algorithm for calculating distance between any two time series. We also tested different values for k and found that 1-nn performed the best. We also made sure that all the training data required to perform calculations for classification is preloaded in memory so that there is absolute minimum delay in computation.

Parallelizng the Classifier

We required extremely fast turnaround time because we wanted to perform action on PC almost realtime. k-nn performed well, but was rather slow - a gesture would be correctly classified seconds after the gesture was performed, which was very disconnected. We decided to parallelise the classifier so that the results can be computed faster. Each FastDTW distance calculation between any two time series is done independently using a threadpool in Java. So, the time series that we are trying to classify is compared to each training time series at the same time. The parallel k-nn classifier performed almost 2x better than the serial classifier in our test environment. Therefore, it was essential to have this feature in order to make the whole system interactive.

IMPLEMENTATION

We developed an Android application which used WiFi to communicate with the PC app. The Android app provides way to record and send motion gesture data. It records data from accelerometer and gyroscope every 2ms while a button is held to record the data. Once the whole gesture is recorded, it then sends the data to the PC. It also provides means to train new gestures and specify the action which should be performed on PC using them.

The PC application was first written in Python and used FastDTW with 1-nn classifier. For performance reasons, the classifier was parallelized but the CPython interpreter doesn't actually allow concurrent execution of threads. We tried using Jython interpreter instead which had compatibility issues with our automation code which was responsible for performing actions on PC. We then had to port all of our existing code over to to Java to continue developing the PC application. It uses an open source implementation of FastDTW [3] along with a k-nn classifier which runs in parallel using a threadpool.

In the high level view, the application works as follows (Fig 5). The application allows the user to enter the IP address and it requires that laptop and mobile smartphone should be connected to the same wifi for connectivity between these two devices. Wifi ensures large amount of data can be sent and wider range for connection. The user then holds the button to record a gesture. The user trains the gesture for five times and associate the trained gesture to some action. When a user performs a gesture, the accelerometer and gyroscope reading from the phone is sent to to the laptop using wifi. The laptop has machine learning algorithm that classifies the gesture as a particular gesture. When gesture is recognized, the associated gesture comes into action. When the user types some keystrokes on mobile, the same is simulated in the laptop. This way the user has complete control over the laptop using mobile through gestures and keystrokes.

EVALUATION

We conducted a brief evaluation for recording the accuracy of the whole system. We used a OnePlus One Android smartphone installed with Android 5.1 to run the Android app, and a Macbook Pro 13” Retina 2015 to run the PC application. Person 1 trained 7 gestures mentioned in Fig 6. Each gesture was performed 5 times for training. We then had the trainer, i.e. Person 1, as well as two other persons, Person 2 and Person 3, perform the gestures 10 times each. We performed this using 1nn, 2nn and 5nn classifiers. The results are presented in the Table 1, Table 2 and Table 3 respectively.

As seen from the results, all the variants of k-nn have 100%

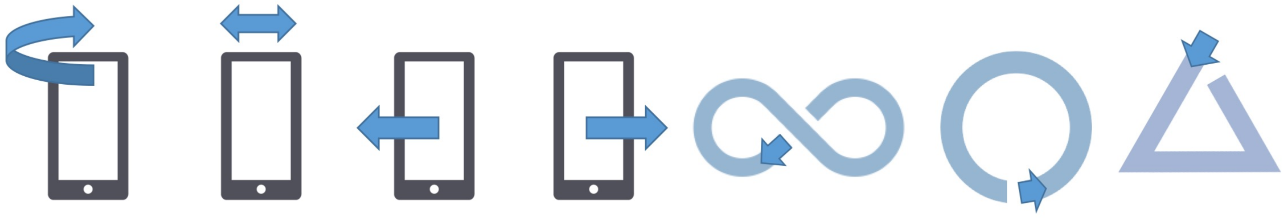
Figure 6. Motion gestures used for evaluation. Flick, shake, left, right, infinity, circle and triangle.

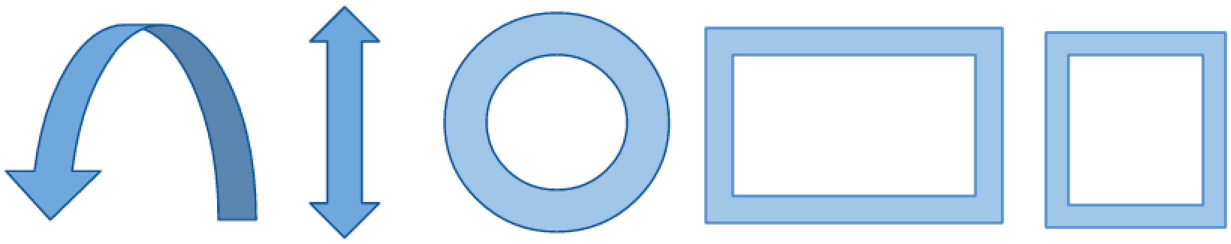
Figure 7. Motion gestures used for evaluating the accuracy of similar gestures. Class 1: whip and Top-down. Class 2: Circle, rectangle and square

accuracy for classification when Person 1 used the application. This shows that the system performs the best when the user is the trainer itself. For most users, this will probably be the case - smartphones are very personal objects, and the user who possesses the smartphone will be the person using the gestures 99% of the time. Therefore, this is an extremely desirable result. For the other users, Person 2 and Person 3, both 2-nn and 5-nn performed worse than 1-nn, with 5-nn performing the worst. This could be due to the fact that the system relies on a very small amount of training data - a gesture is only trained 5 times.

We also performed evaluation for similar shapes and computed the efficiency of the system. We computed performance for similar looking shapes. This was done to find out how well the system acts for similar looking gesture as this was on of a major challenge.Class 1 included shapes such as whip and up-down which are very similar to each other. Class 2 included shapes such as circle, rectangle and square (Fig 7). The application proved that it performs very well for whip and up-down because the readings are a combination of gyroscope and accelerometer. From Table 4, it can be observed that circle has 100% accuracy. While, square and rectangle are slightly inaccurate because these two shapes are very similar to each other and requires more concentration from the user. Also, the gesture works best when the same individual acts as both user and trainer.

Gesture	Person1	Person2	Person3
flick	10	10	10
shake	10	10	6
left	10	10	10
right	10	9	10
infinity	10	10	10
circle	10	10	9
triangle	10	10	9

Table 1. 1nn results

Gesture	Person1	Person2	Person3
flick	10	9	9
shake	10	6	6
left	10	3	10
right	10	10	10
infinity	10	10	10
circle	10	10	9
triangle	10	8	9

Table 2. 2nn results

Gesture	Person1	Person2	Person3
flick	10	9	9
shake	10	8	6
left	10	9	7
right	10	10	9
infinity	10	10	10
circle	10	9	9
triangle	10	8	9

Table 3. 5nn results

LIMITATIONS

Although the developed system works end-to-end and provides a way to train custom motion gestures and perform actions on PC, there are some things which can be improved and worked upon.

- Currently user needs to hold a button on the smartphone app to start and stop recording the motion gesture. The app can be extended to constantly monitor the movements of the smartphone and intelligently detect the start and end of the motion gesture by analyzing rapid movements. However this might reduce the accuracy of classifier due to noise.

- The system varies in performance when the user is not the trainer for the gestures. However this is not a big problem. The application requires only small amount of train-

Gesture	Person1	Person2	Person3
whip	10	10	10
top-down	10	10	10
circle	10	10	10
rectangle	3	7	9
square	10	10	6

Table 4. 5nn results

ing data (5 set of readings) for a particular gesture. So new users can quickly train and use the system with very high accuracy.

- The PC app can only perform actions using simulated keystrokes. However this can be extended to include deep integrations with various services.

CONCLUSION

We developed an end-to-end system for training and detecting custom motion gestures using an Android smartphone with complete flexibility to define custom actions to be performed on a PC. The evaluations prove the fact that 1nn with DTW is exceptionally accurate for time series data classification. Parallel 1nn with FastDTW gives a huge performance boost and makes the whole system very interactive with minimal latencies. The system proved to be highly accurate if the trainer is the user itself and gave really good results for other people too. With some UI touches and some pre-loaded gestures, we feel that we could make this available on the Google Play store.

INDIVIDUAL CONTRIBUTIONS

All the team members worked on researching and studying existing mechanisms for classifying time series data. Alick took the responsibility of prototyping the classification algorithms in Python, Harshitha worked on providing a way to associate gestures with an action and simulate the keystrokes on a PC. Tanuj worked on the Java application for the PC with added performance enhancements to the classifier. Alick and Tanuj worked on the Android application for providing option to record/send gestures to the PC app and edit the existing ones. Harshitha worked on providing a mechanism to type remotely on the PC using keyboard on the Android app. All of them participated in the user study evaluation.

REFERENCES

1. Combining Acceleration and Gyroscope Data for Motion Gesture Recognition using Classifiers with Dimensionality Constraints. `https://www.fxpal.com/publications/combining-acceleration-and-gyroscope-data-for-motion-gesture-recognition-using-classifiers-with-dimensionality-constraints.pdf`.
2. Fast Time Series Classification Using Numerosity Reduction, Proceedings of the 23rd international conference on Machine learning. `http://alumni.cs.ucr.edu/~xxi/495.pdf`.
3. FastDTW implementation modified for use as a library in production applications. `https://github.com/cscotta/fastdtw`.
4. iSAX: Indexing and Mining Terabyte Sized Time Series. `http://www.cs.ucr.edu/~eamonn/iSAX/iSAX.html`.
5. Motion Gestures in Moto X. `http://www.androidcentral.com/looking-gestures-moto-x-pure-edition`.
6. Scaling up dynamic time warping for datamining applications. `http://www-ai.cs.uni-dortmund.de/PublicPublicationFiles/keogh_pazzani_2000a.pdf`.
7. The UCR Suite. `http://www.cs.ucr.edu/~eamonn/UCRsuite.html`.
8. uWave: Accelerometer-based personalized gesture recognition and its applications. `http://www.sciencedirect.com/science/article/pii/S1574119209000674`.