Sparky - Gesture Based Home Automation Application

Gary Graham, Jia Ming (Anson) Liang, and Jyotheeswar Arvind M

***Abstract*—Traditional home automation applications mainly use buttons on the mobile phone touch display to toggle the state of home devices. These applications usually require users to navigate through clusters of buttons to control the specific device. Most users would rather flip the switches on their home wall to toggle the specific devices instead of going into their applications to find the correct button. We built an Android application to utilize motion gestures to control the home devices. This will eliminate the process of users trying to find the correct button on the screen. With our application, users can toggle a specific device by simply performing a unique gesture with their mobile phone. Our application connects to a Spark Core, which we use for our home automation system. Spark Core is an Arduino-compatible Wi-Fi enabled device that can be used for developing internet-connected hardware. This device enables our application to let users control their home devices anywhere with internet access.**

***Index Terms*—Android, Arduino, Gesture, Home Automation, Spark Core.**

I. Introduction

With the development of our application, we wanted to address what we believe to be an up and coming problem in the world of Internet of Things (IoT). In the home automation space, there are many manufacturers. Each of these manufacturers typically develop applications to support their proprietary devices. However, a quick search on the mobile applications stores show that the UI and usability of these applications leave much to be desired[2]. Cluttered interfaces with no regard to user experience abound. With this in mind, our goal was to create a home automation application that is not only more aesthetically pleasing, but also more functional.

Traditional home automation applications have stuck to the tried and tested method of utilizing buttons inside an application, typically on a mobile device, to toggle the state of devices at home. However, we have many more ways to interact with our mobile devices than simple taps. The overarching goal for home automation is to make the experience seamless for users. If it is not faster or more convenient to use your mobile phone to toggle a device, then why bother? If users want to turn off a light, and it takes them two seconds to take out their phone, 5 seconds to scroll over to the application and open it, 1 second for it to load, and 1 second for users to tap on the correct button; they have taken 9-10 seconds to do something they could have done in the same time by simply standing up and going to flip the switch manually. We believe that using gesture-based triggers, combined with widgets, users can achieve a more satisfactory experience, while reducing the time it takes to actually perform the task at hand.

For the purpose of demonstration, we will only use the 3-Axis accelerometer data available in our Android device to perform gesture recognition. The algorithm we use is a simple discrete time series data matching algorithm called Dynamic Time Warping (DTW). It should be noted that gesture recognition is non-trivial and high degrees of accuracy will require using advanced machine learning techniques.

This paper gives an overview of our application, explaining how we setup and connect to the Spark Core home automation system. This paper also explains how we implement our gesture based cognition to control a Spark Core and show results of user experience tests.

II. Overview of the Application

The goal of this application is quite straightforward; improve the current state of home automation user interaction. To perform this, we go about it in three ways. Firstly, we improve the communications between mobile and the physical devices at home. Secondly, we implement gesture recognition as a device-toggling mechanism. Third, we build our prototype using publically available devices, which will allow users to extend our work.

The prototype of the device involves the following:

1) Spark Core
2) Relay Board
3) 2x Electrical Outlet
4) Android Device

Users can plug any of their devices into either one outlet or the other. Then, within the android application, they can toggle the power to each of the outlets individually, and become informed if the status of one of the outlets changes.

For user interaction, we have decided on a multi-faceted approach. The novelty of the application comes from our gesture recognition system which allows the mobile device to register gestures even while the phone is asleep. This in turn toggles power to the devices. Beyond this, we have also implemented widgets, which allows users to define their own interface on their home screens. For example, you can split half your home screen between two widgets for easy access to the functionality, even when not inside the app. Furthermore, we allow users to toggle devices in the traditional sense, using buttons inside the application itself.

To begin with, we discuss the communications method used. To build our prototype, we used a Spark Core[1]. The device exposes a REpresentational State Transfer(REST) API to anybody who has the credentials, and operates over a Wi-Fi network. The advantage of this is that it is extremely light-weight, and requires no proprietary technology. The Spark

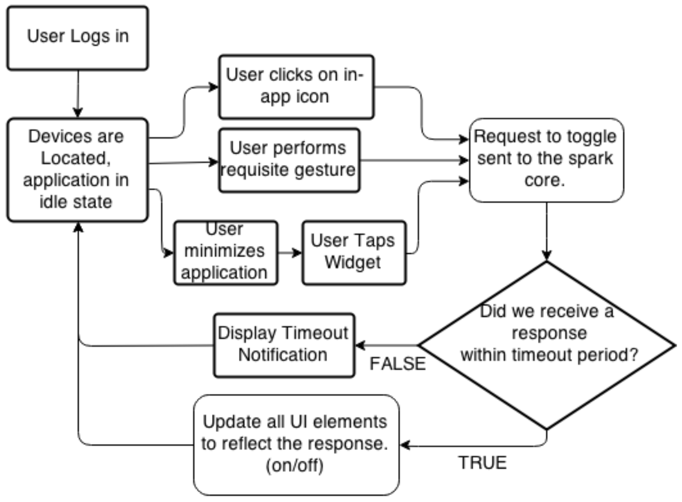
User Logs in
User clicks on in-app icon
Devices are Located, application in idle state
User performs requisite gesture
Request to toggle sent to the spark core.
User minimizes application
User Taps Widget
Did we receive a response within timeout period?
Display Timeout Notification
FALSE
Update all UI elements to reflect the response. (on/off)
TRUE

Fig. 1. Application workflow diagram

Core, being an Arduino-based microcontroller, also allows us to quickly and remotely modify the code running on it.

In this section we give an overview of our Android application prototype, Sparky. We omit low-level implementation details in favour of providing a high-level overview of the interactions. In order to allow the mobile phone to communicate, we have written a communications library that allows us to interact with the Spark Core. It is upon this interaction which the backbone of the application rests. Whenever an intent to toggle is registered, whether by gesture recognition or otherwise, a call is made to the library, which handles request and response. Since we have many UI elements indicating the status of the device(in-app, multiple widgets), we use the observer pattern to handle notifying all listeners as to the status of the device. For the in-app and widget UI, we use minimalistic icons that represent the devices they are toggling. This declutters the interface and provides an intuitive idea of how to use the appplication. Fig. 1 shows the overall basic usage of the application.

There are equivalent workflows for other tasks, for example changing the current gesture, but we omit them here due to brevity requirements.

III. Gesture Recognition

Our application uses the DTW algorithm to do gesture recognition. DTW was originally developed for speech recognition by measuring the similarity between two discrete time series data. Instead of speech, we compare accelerometer data for gesture recognition. DTW is relatively simple to implement and it works for simple gestures. Below is a pseudo-code algorithm for DTW:

```
double DTW(double 01[1...T1], double O2[1...T2])
  declare double DTW[0...T1, 0....T2]
  declare integer i, j
  for i:=1 to T1
    DTW[i,0] := positive infinity
  for j:=1 to T2
  DTW[0,j] := positive infinity
  DTW[0,0] := 0
  for i:=1 to T1
    for j:= 1 to T2
      DTW[i,j] := distance(O1[i], O2[j]) +
        minimum(DTW[i-1,j]), DTW[i,j-1], DTW[i-1,j-1])
  return DTW[T1,T2]
```

Since the data reading is in 3-Dimensional space, we treat each time interval reading as a 3-Dimensional point. The distance function we use in the pseudo code takes the norm between the two points.

Before gesture recognition, we train a gesture by obtaining and storing 32 discrete accelerometer data. This accelerometer data combined with a gesture label form a gesture object. Our application, by default, contains two predefined gestures: Bump Front and Bump Left. We choose these two gestures as default because it is very simple for another person to repeat the movement and it is unique enough that when someone puts their phone in their pocket for example, they will not accidentally trigger the gesture.

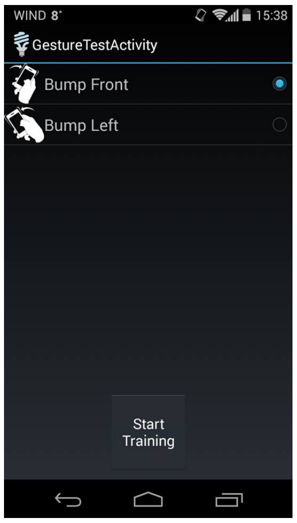

Fig. 2. Default gestures selection page for Sparky

Bump Front in fig. 10 is defined by holding the device straight, tilting it forward by 90 degrees and back to the original position within 2 seconds. Bump Left is defined similarly, tilting the phone to the left by 90 degrees and back to the original position within 2 seconds. Users have the option of creating their own custom gestures.

Once users select the gesture for a particular device, and make their preference about listening to gestures being performed in the background, the application listens to the accelerometer data. This data is compared with the stored gestures using the DTW algorithm. DTW will return the distance for every stored gestures. This distance indicates how similar the two gestures are, with a smaller distance indicating they are very similar and vice versa. We take the geture with the smallest distance as the recognized gesture.

DTW allows some variance in gesture motion and speed when a person tries to repeat the gesture they trained. The variance is controlled by a threshold variable in our program. We find that small variance will allow us to get high percentage of true positive recognition but we realize that it takes multiple tries for users to re-do the same gesture they trained. On the other hand, large variance will allow users to easily get their gesture recognized but will also have high chance of getting false positive recognition.

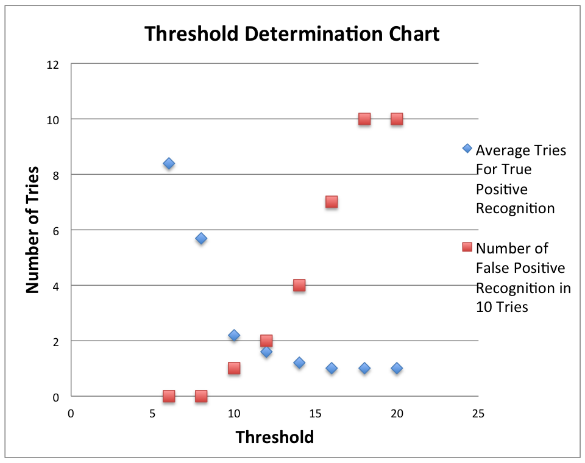

Fig. 3. Chart used to determining distance threshold

We set the distance threshold by experiementation. We try to set the threshold so that we minimize both the number of tries to get true positive recognition and the number of accidental false positive recognition. In fig. 3, the diamond scatter plot is the average number of tries to get a proper gesture recognized. We have a person train their own gesture and try to reproduce the same gesture at different thresholds multiple times. We then take the average of this set to the threshold. The square scatter plot is having a person doing 10 random gestures and see how many gestures will get recognized (these are false positive results). We can see in fig. 3 that the ideal threshold value is around 12.

IV. User Experience Tests

The application was tested by 17 users aged between 21-27 and we obtained their feedback. The participants of the testing consisted of mainly graduate students with a strong technical background.

A. Methodology

The participants tested the application individually and not more than two participants were in the testing room at a particular point of time. The participants were initially briefed about the overall idea of the application. The team demonstrated the functioning of the application first and explained how to perform the default gestures. Then the participants were given access to the mobile device on which the application is running. The participants tested the entire functionality of the application including changing the assigned gestures and creating their own custom gestures. There was no set time limit for the participants to use the application. Upon their stating that they have completed testing the application, they were directed to fill in an online feedback form. The questions on the feedback form were structured to get the users' rating on factors such as the intuitiveness of the application, their comfort using the default gestures, their willingness/interest in adapting such technology to control home appliances using our mobile application.

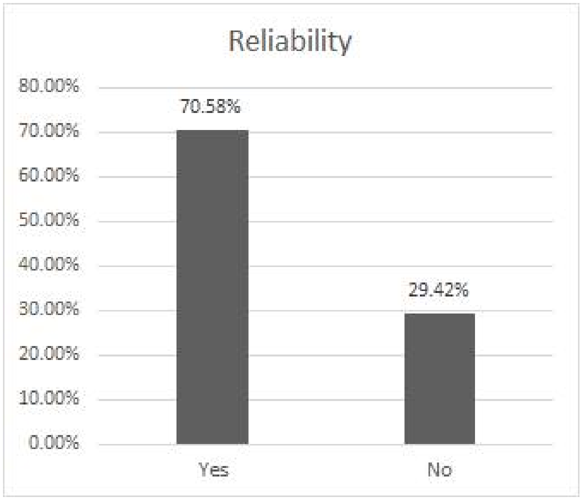

Fig. 4. C1

B. User Feedback

Users submitted their feedback and comments online. Users also provided their consent on the form before submitting their feedback.

Some comments from the users are: "Takes a bit of trial and error to figure out the default gestures. They seem pretty sensitive. ", "The default gestures are good, but , take some time to adjust to the user's personal way of doing these gestures. So, even before using the default gestures, learning these from the user might make this a bit easier","why does every single thing have to be internet connected nowadays. Dont track any usage data!" etc.

C. Inference

The application experience rating on a scale of 10 was found to have a mean of 8.58, and median of 9. Although users rated the default gesture experience at a mean of 7.88 and median of 8, about 53% of the users needed multiple attempts to get accustomed to the way the default gestures have been defined. This is attested by the rating on the ease of figuring out the default gesture, with a mean of 7.5 and median of 8. Since the data obtained when a gesture is performed is reliant on orientation and the starting position at the time of doing the gesture, users needed practice before being comfortable. Once they learnt the default gestures, the usability of the application became smooth. The application's intuitiveness received a mean rating of 8.11 with a median of 8. On being comfortable to use the application in a real life setting, users gave a positive 70.5% yes. It is interesting to note that 76.47% of the users were comfortable using our application to control household devices, albeit this can be due to the users being in the 21-27 age group and most users being from a technical background.

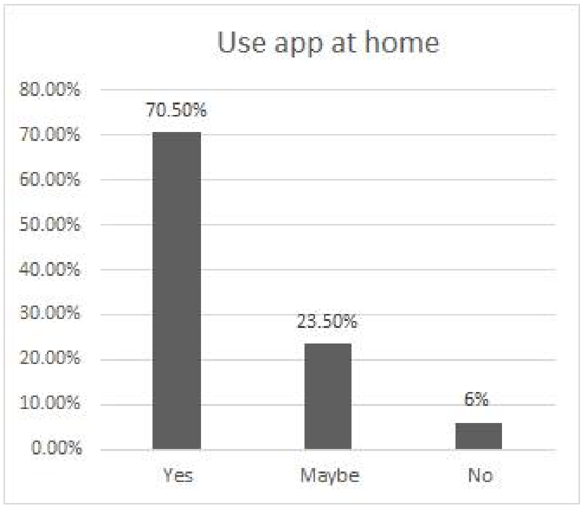

Fig. 5. C2

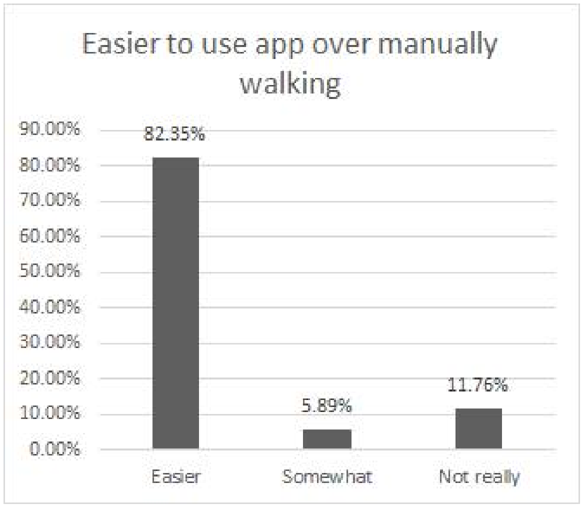

Fig. 6. C3

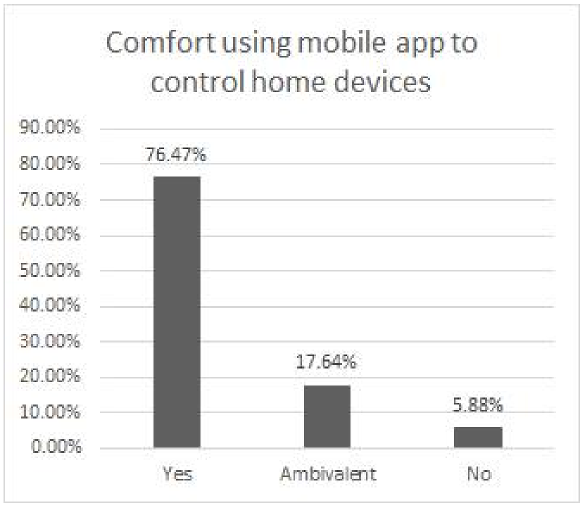

Fig. 7. C4

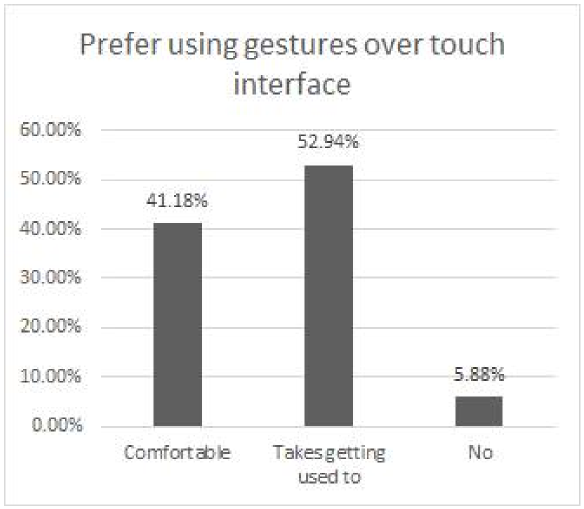

Fig. 8. C5

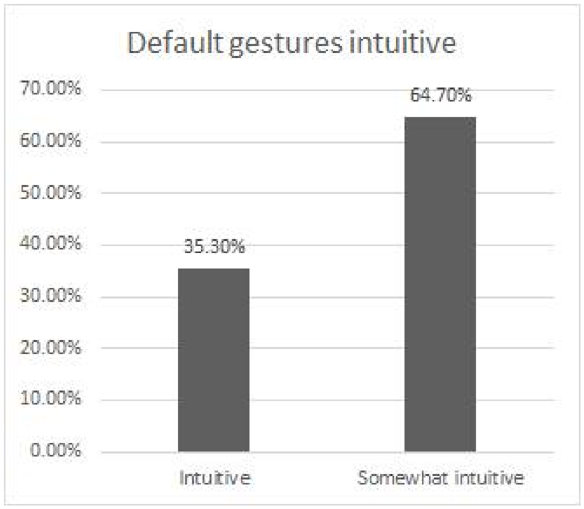

Fig. 9. C6

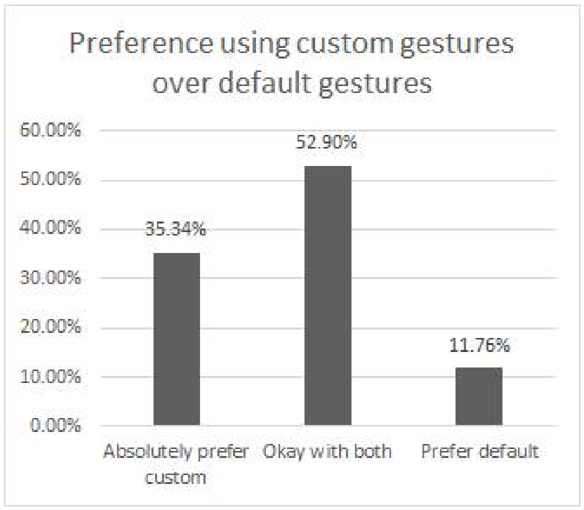

Fig. 10. C7

V. Future Work

We aim to improve our application in the future by providing an interface for multiple Spark Cores to be connected to our application. Functionality to add more devices, with corresponding images, are also targeted.

We aim to improve the accuracy of the gesture recognition by incorporating the 3-Axis gyrometer. We would also want to have an option for users to utilize the magnetometer to control multiple devices with the same gesture but perform the gesture at a particular direction.

DTW is an effective algorithm when the gesture is a simple motion. It allows small variation in speed and motion. However, our user feedback results show that the default gestures are minimally comfortable because the algorithm generally performs poorly when the moving rate and motion have a large variation; as is common among different users. Also, with more complicated gestures DTW will not work effectively. One alternative is to do gesture recognition is Hidden Markov Models (a machine learning technique). This technique is better because it uses knowledge in probability and stastics to account for larger variation of speed and motion, but still retain the accuracy of recognizing gestures.

We also aim to test the application among a wider audience to observe how much this technology/application is acceptable to the greater populace.

VI. Conclusion

References

[1] H. Kopka and P. W. Daly, *A Guide to LaTeX*, 3rd ed. Harlow, England: Addison-Wesley, 1999. https://www.spark.io/ https://play.google.com/store/apps/details?id=com.webmtn.android.app