On the usage of smart devices to augment the user interaction with multimedia applications

Alan Ferrari, Vanni Galli, Daniele Puccinelli, and Silvia Giordano
Institute for Information Systems and Networking
University of Applied Sciences of Southern Switzerland (SUPSI)
Manno, Switzerland
firstName.lastName@supsi.ch

Abstract—Wearable devices have recently gained a foothold in the market with the uptake of smartwatches. The strong tie between a smartwatch and its owner, the highly predictable position of a smartwatch on the body, and its internal sensors are enabling a wide array of applications that leverage the user context.

In this paper we focus on a gesture recognition system to augment the user interaction with multimedia applications. We define a set of seven gestures that are relevant across several applications and we collect an extensive dataset with two smartwatches (the Motorola Moto360 and Apple’s Watch).

We use Long Short Term Memory neural networks for gesture recognition based on sensor data from both smartwatches. We provide an extensive evaluation of the classification accuracy of the system and provide a sensitivity analysis to find the Long Short Term Memory configuration that maximizes the classification accuracy. We also show the extent to which Long Short Term Memory neural networks outperform traditional machine learning approaches.

We also illustrate an application we built for the Android and iOS platforms that allows developers to easily integrate the gesture recognition in their own systems. We conclude the paper with a description of use cases to underscore the potential impact of our contribution.

I. INTRODUCTION

Recent advances in smart devices can provide new opportunities to improve the user experience by making device-to-user interaction as natural and fluid as possible. Moreover, the uptake of wearable devices is paving the way to novel solutions that leverage the kinematics of the human body to augment the user experience. In this paper, we focus on the interaction between smartwatches and applications that rely on multimedia content. Today’s gold standard is represented by commercial systems such as Microsoft Kinect and Nintendo’s Wii Remote.

Microsoft Kinect\textsuperscript{1} is a set of motion sensing input devices developed by Microsoft for Xbox and Windows PCs. It is based on webcam and infrared sensors data analysis and enables users to interact with their devices without the need for a physical controller through a natural user interface based on the detection of body movements. The biggest limitation of the Kinect is in terms of portability, because it constrains the user to maintain a fixed position and a fixed angle of view. Another limitation is the difficulty of multiuser simultaneous recognition. Even if the Kinect identifies the kinematics of multiple bodies, it is unable to tell multiple users apart even with its built-in voice-recognition support.

Nintendo’s Wii Remote\textsuperscript{2} offers sensing capabilities for the Nintendo’s Wii platform. It allows the user to interact and manipulate items on screen via gesture recognition. The key device is a handheld radio command that uses an accelerometer as well as optical sensors to detect hand movements. The handheld radio command simplifies multiple user recognition compared to Microsoft Kinect (it is inherently single user), but strongly constrains the body kinematics of the user and also limits the user to a fixed position.

Both the Kinect and the Wii Remote share platform-dependency and lack of portability. To move past these limitations, we propose the use of smartwatches, which are inherently portable, relatively cheap, and reasonably nonintrusive. In this paper, we show how smartwatches can be used to build a user interaction system based on gesture recognition. Specifically, we investigate the adoption of smartwatches to leverage user gestures to improve the user interaction with multimedia applications. The goal is for the user to be able to interact through common gestures to be detected through the smartwatch in a way that is totally transparent to the user, who can focus on the gestures and simply forget about the smartwatch and its interaction with the system to be controlled. This way, the smartwatch all but disappears, as the most powerful technologies ought to do in the words of Mark Weiser\textsuperscript{3}.

The key contributions of this paper are:

- the definition of a set of common gestures that may be used to manage multimedia devices with smartwatches;
- the design, implementation, and thorough evaluation of a smartwatch-based gesture recognition engine;
- the definition of a scalable software architecture for the two most common smartwatch operating systems (Android Wear and Apple watchOs) that can be employed for a wide array of applications;
- the illustration of the implementation of a set of use cases that exploit the functionalities of the proposed architecture.

\textsuperscript{1}\url{https://dev.windows.com/en-us/kinect}
\textsuperscript{2}\url{http://www.nintendo.com/wiiu}
The paper is organized as follows. We begin with a review of the state of the art and continue by illustrating the devices we employ as a hardware foundation of our gesture recognition engine. We then present the design and implementation of the gesture recognition engine and its evaluation, and provide the architectural details of a complete system based on our gesture recognition engine. Before our closing remarks, we illustrate the details and the implementation of two use cases.

II. RELATED WORK

Activity recognition is a widely studied research area. In the last decade, the release of novel mobile and wearable devices has allowed developers to create a set of new applications that exploit context-awareness in order to gain a more dynamic and less intrusive interaction with their users. Dey and Abowd [1] define a system as context-aware if it uses context to provide relevant information and/or services to the user, where relevancy depends on the user’s task. Activity recognition is a subset of context information and one of the first notable contributions in this space was provided by Ravi et al. [2], who used a homemade board and an iPAQ Palm device to collect information. Authors recorded a dataset that contains seven different activities (ranging from walking to brushing teeth) and on two subjects on multiple days. They use a set of different classifiers (from Naïve Bayes to Hidden Markov Model) to estimate the corresponding activity and they achieved a reasonably high accuracy using a single triaxial accelerometer; however, they noted that it is hard to recognize activities involving hand movements (i.e. brushing teeth).

Nowadays, personal smart devices offer a set of internal sensors (such as accelerometers, gyroscopes, and even pressure sensors) that have been utilized in a large body of research and applications. One of the first efforts to retrieve the activity with personal smart devices was proposed by Kwapisz et al. [2]; the authors use a gyroscope on an Android-based smartphone and recorded a set of six activities (walking, jogging, ascending stairs, descending stairs, sitting, and standing); they used a population of 29 participants. They used decision trees (J48), logistic regression, and multilayer neural networks to classify human activities with over 90% accuracy.

Recent efforts use Hidden Markov Models as their main classification method [3]. Others are experimenting with different kind of sensors; for instance, Vanini et al. [4] present a technique based on barometric pressures mixed with Long Short Term Memory neural networks [5] and studied a dataset of six different activities, achieving over 95% accuracy.

Gesture recognition is a subset of activity recognition. It is described by Mitra et al. [5] as: “Gesture recognition pertains to recognizing meaningful expressions of motion by a human, involving the hands, arms, face, head, and/or body” and has been studied for years in the field of Human Computer Interaction. A widely adopted approach is vision-based motion recognition, where multiple camera record motion and machine learning techniques are used to recognize gestures. Notable early contributions in this space include the work of Campbel et al. [6] on the recognition of ballet steps, and the work of Brant et al. [7] on the classification of T’ai Chi movements. Recently, the CamShift algorithm ([8]) has augmented the performance of those systems to a significant extent.

The emergence of wearable devices has paved the way to novel gesture recognition strategies. Chambers et al.[9] use a hierarchy of gestures detected with the usage of an accelerometer placed in a wrist band and the Hidden Markov Model as machine learning technique. The authors trained the model to recognize Kong-Fu moves. Kern et al. [10] use a wrist bracelet to detect activities such as keyboard typing and handshaking. They use on Naïve Bayes classifier executed in an IPad carried around in the user pocket. Schörer, et al. [11] uses a WII controller as input device in conjunction with Hidden Markov Model classifier.

The previously mentioned approaches can achieve significant accuracy levels but they always rely on the main classifier having huge impact on the energy consumption. By triggering the classifier only after a key event has been recognized, lower-cost systems can be provided. One notable example is provided by Junker et al. [12] where authors use a two stage approach where the identification of a specific motion event with a simple similarity search triggers the use of a Hidden Markov Model classifier.

The recent uptake of smartwatches has opened new opportunities in gesture recognition. Porzi et al. [13] present a smartwatch-based system to detect gestures and help visually impaired individuals communicate using a computer in environments where voice communication is impractical (e.g. owing to excess noise). Kernel methods are used to recognize a set of simple gestures and user alerts are managed through vibration feedback. Chao et al. [14] propose a system to predict simple finger, wrist, and arm movement detected with a custom-made smartwatch.

Xu et al. [14], [15] propose a system based on a custom-made smartwatch mixed with additional sensors placed over a finger and on the arm to recognize hand gestures; they compare different classifiers and achieve an average accuracy of about 95%. In [16], Wang et al. propose MOLE, a novel framework to use motion sensor in smartwatches to steal information. The authors use a set of data filtering techniques along with Bayesan networks to use smartwatch data to recognize the set of the 100 most frequently used English words with a reasonably high accuracy.

In our recent work [17], we present a novel authentication strategy for Bluetooth-equipped smartwatches. We use the built-in smartwatch sensors to detect whether two users have shaken hands. If so, the devices exchange a soft authentication privilege that is suitable for applications with relaxed security needs (for instance, an application to exchange business cards). We evaluate the system using different machine learning techniques and we investigate their performance in the dimensions of accuracy and energy consumption.

III. A FIRST LOOK ON SMARTWATCH SENSOR DATA

To achieve a clear understanding of the problem we start collecting data using two popular smartwatches:
Motorola’s Moto 360⁴, based on the Android Wear⁵ operating system. The Moto 260 is equipped with an all day battery that can be charged wirelessly, has dual microphones for voice recognition and noise-rejection, and a vibration motor that allows tactile feedback. It uses Bluetooth 4.0 (Bluetooth Low Energy) and WiFi. It contains a heart-rate sensor and 9-axis accelerometer.

- The Apple Watch ⁶ is the first smartwatch equipped with Apple’s watchOS. It uses a linear actuator to provide haptic feedback and is equipped with a built-in heart rate sensor and 9-axis accelerometer.

We collect an extensive dataset that includes the gestures of 15 volunteers, chosen from the student body and faculty of our institution. The gestures of interest are summarized in Table I. Each participant was asked to perform each of the seven gestures four different times while wearing the smartwatch; hence, the dataset includes 420 gestures. We also record traces of a set of common activities performed by the participants such as keyboard typing and drinking from a glass in order to investigate whether our system detects false positives adding an extra 120 gestures to the dataset reaching an overall number of 540 entries. We employ a sampling rate of 10 Hz, similarly to other studies [13], [17]

Figure 1 shows a sample trace with two simultaneous readings; the small difference between the two traces can be ascribed to the slightly different position of the two devices on the wrist and the slightly different sampling time.

### IV. Gesture Recognition In Practice.

We start with raw sensor data in the form of three-axis acceleration and consider a sliding time window of one second [17]. Following the work of Knapsiz et. al. we retrieve a set of 5 features:

- **Arithmetic Mean** for all the three axis over the time window denoted as \( \mu \)
- **Standard deviation** for all the three axis over the time window denoted as \( \sigma \)
- **Average Absolute Difference** for all the three axis over the time window denoted as \( \delta \)
- **Max Value** for all the three axis over the time window denoted as \( M \)
- **Min Value** for all the three axis over the time window denoted as \( m \)

The next step will be the recognition of the gestures given the previous information. We are using machine learning...
techniques and the input vector $V$ will be:

$$V = [\mu_x, \mu_y, \sigma_x, \sigma_y, \mu_z, \sigma_z, \delta_x, \delta_y, \delta_z, M_x, M_y, M_z, m_x, m_y, m_z]$$

(1)

The inference process is shown in Figure 2. There are two main steps; the first one is a linear classifier that detects if there is some sort of activity that might be a gesture. If the linear classifier determines that a possible gesture might be in progress, the input data is passed on to a Long Short Term Memory (LSTM) recurrent neural network [18]. This two-step inference process saves energy by avoiding the use of the more computationally intensive LSTM neural network whenever its use is not warranted.

A. Linear Classifier

A linear classifier is a traditional machine learning technique that takes a classification decision looking at the linear separability of the input data. We use the $k$-means cluster algorithm [19] and $k$-nearest neighbor [20] to find the cluster centroid. We therefore compute the Hamiltonian distance between each cluster point and the cluster centroid and find the maximum distance. When a vector is passed as input to the Linear Classifier, we compute its Hamiltonian distance with each cluster centroids; if it is below the maximum distance, we infer that the input vector may correspond to a gesture and pass it on to the deep learning classifier.

B. Deep Learning

The key idea behind deep learning[21] is to employ multiple processing layers with multiple non-linear transformations. Recently, deep learning approaches have been applied in many fields ranging from artificial vision and speech recognition and have been shown to outperform traditional approaches to a significant extent ([22], [23], [24]).

In this paper, we focus on Long Short Term Memory (LSTM) neural networks [5], a deep learning technique originally published in 1997 by Hochreiter and Schmidhuber. LSTM is a recurrent neural network suited for the classification and the prediction of complex time series assembled with blocks (called memory cells); each block can be viewed as a “smart” network unit that can remember a value for an arbitrary length of time. An LSTM block contains gates that decide if the input has to be remembered or not; if applicable, the input is passed on to the next iteration. LSTM is trained through backpropagation over time, a modified approach to the standard backpropagation applied to traditional neural networks. Deep Learning techniques have been recently applied to mobile sensing by Lane et. al. [25].

We begin with a sensitivity analysis of LSTM to determine the correct network depth (number of layers). We then measure the classification accuracy for each gesture using cross validation. Finally, we compare LSTM with traditional machine learning techniques in order to show the superior performance of deep learning.

C. LSTM sensitivity analysis

The key question we are trying to answer in this section is the following: what is the best and least energy-costly LSTM configuration? Similarly to Zurada et. al. [26], we answer the question by performing a sensitivity analysis of the LSTM structure. The parameter we are considering is the depth of the LSTM network (i.e., the number of layers); we start from 3 layers and we stop at 100 to keep the computational complexity reasonable. We then train and evaluate the network with a single gesture and we repeat the experiment 100 times for a single configuration in changing the network seed. Figure 3a shows how the accuracy varies as the number of layers grows. We observe that an excellent accuracy can be achieved with as few as nine layers. Higher numbers of layers such as the ones showed in Figure 3b are conducive to a reduction of the classification accuracy because of overfitting.

D. Classification Results

To thoroughly evaluate our solution, we employ four-fold cross-validation. The key idea is to partition the dataset into a set of complementary subsets; one of the subsets is used as training set and the others are used as valuation set. To reduce the variability multiple rounds of the validation are performed using different partitions. Following the results in the previous chapter we decide to use the LSTM with 9 layers and we run each single experiment for 100 rounds for each single gesture, dynamically changing the random seed used to initialize the random generator and, consequently, the initial LSTM state (random restart).

In Figure 4 we show detailed results for all the 7 gestures we have selected tested in the dataset. The highest accuracy is obtained with the push forward gesture where the classification accuracy is close to 98% accuracy. In general, a high classification accuracy is achieved with all gestures that mostly revolve around a single axis (i.e. L, R, F, U, and D). Based on the kinematics of the body movement, we conjecture that this high accuracy is probably due in the strong similarities of this specific movement across different people.

Gestures that involve multiple axes (in our case, C and A) can be classified with a lower accuracy compared to the single-axis gestures and lie slightly below the 90% accuracy mark. We noticed that participants had a harder time mimicking these gestures, thus contributing to the poorer classification accuracy performance. The classification accuracy could be improved by asking individual users to do their own training.
Fig. 3: Sensitivity analysis: LSTM classification accuracy as a function of the number of LSTM layers.

Fig. 4: LSTM performance results for each single gesture; we see that LSTM achieves excellent results for each gesture other than C and A, where the performance is still reasonably good but significantly poorer than for the other gestures.

E. Energy Efficiency of the classifiers

As Figure 5 shows, on average, LSTM is roughly twice as energy-costly as a linear classifier. Each time the linear classifier avoids forwarding the input values to the LSTM, the energy savings are therefore considerable, which is significant given the energy-constrained nature of smartwatches. We now delve into the details of Long Short Term Memory Neural Networks, which represent the core of our approach.

F. Comparison to traditional machine learning techniques

While the use of deep learning and, specifically, LSTM is becoming increasingly widespread, traditional machine learning techniques continue to be more popular. In this section, our goal is to show how deep learning stacks up against traditional techniques, specifically backpropagation neural networks and Naïve Bayes.

Backpropagation neural network (BPROP) [27] were originally introduced in the 1970s and remain to date a widely adopted machine learning approach. Backpropagation neural networks have been applied with success in a set of domains ranging from activity recognition to face recognition. Naïve Bayes is a machine learning algorithm that applies the Bayes theorem with the naive assumption of independence between variables. Naïve Bayes has been adopted in a long list of domains offering good results.

In this section we compare a nine layer LSTM to these other two solutions. BPROP is configured to have the same layer count as LSTM. The test is performed with a single
Our end goal is to offer a solution where developers can easily connect for having access to the gestures. To this end, we implement a plug-in architecture that allows other applications to be easily integrated. Figure 9 shows the high level architecture. The smartwatch is the entry point where we collect and preprocess the data and run the classification engine illustrated in the previous section. The smartwatch forwards to the smartphone the detected movement immediately after the inference. The training of the network is performed offline in a resource-rich cloud environment. The cloud trains the LSTM classifier with the dataset we provide and stores the LSTM coefficient in a JSON file, which is later to the smartphone. The JSON format serves to streamline the integration into both iOS and Android platforms. We enable the effortless integration of third-party plugins into the system. Working with two distinct smartwatch requires some architectural effort to maintain a common solution.

A. Plug-in Architecture on Android Platform

To enable the easy integration of plugins into a Java Environment (e.g. OSGi[7]), the main tool is the Dynamic Class Loading, offered in the JVM that dynamically loads Java Bytecode inside the JVM; the code could be stored inside an external class or a Jar file. Unfortunately, all these strategies cannot be directly applied in Android because the Android Virtual Machine (both Dalvik and ART[8]) uses its own version of the Bytecode, named Dalvik Bytecode[9], that is optimized for environments with scarce resources such as mobile devices. We implement our solution using the class loader offered in Android platforms, namely the DexClassLoader, which loads the classes saved in a jar or an APK file containing the classes.dex entry[10]. The Android’s SDK[11] offers a standard tool named dx to pass from Java to dex.

We define a standard interface that has to be implemented inside the plugin class; this interface returns a set of parameters such as the name and the version. Whenever the class is loaded or unloaded, the system calls his dedicated APIs; whenever a gesture is detected, the system calls the appropriate method.

All the plugin-classes must be converted to dex and stored inside a classes.dex file. After the conversion the .dex file is placed inside a jar file jointly with another file in XML format called pluginfo.xml that contains some basic information about the plugin that the application must know, like the entry point (the class that implements the interface and the description of the plugin showed to the user).

B. Plug-in Architecture on iOS Platform

To enable a plug-in architecture, the software is designed to load a dynamic library at runtime. OSX accomplishes this by loading dylib files; however, iOS apps that download code from the Internet are automatically rejected from the App Store for the sake of security[12], thus making it impossible to download the dynamic library and, therefore, precluding the use of a plug-in architecture. As a workaround, developers who wish to publish their own plugin within our architecture may implement the Swift protocol[13] we provide and create a static library.

When a developer includes their own plugin, the static library is included in the application and a new version of the app is published in the App Store. This workaround ensures full compliance with the Apple developer guidelines.
augment the user experience with presentation support.

A. Home Entertainment System Integration

There exist various home entertainment products such as Apple TV 14, Google’s Chromecast 15, or the Roku Player 16. In our solution, we use the Open Source Platform Raspberry Pi 2 17 where we install the Open Source Media Center (OSMC)18.

Because OSMC communicates with external applications through REST calls, we implement such calls inside our plugin. Table II shows the connection between gesture and action; the key idea here is to offer a similar interaction to the one provided in the Microsoft’s Kinect.

A usage example is presented in Figure 10 where we see a left to right hand swipe that instructs OSCM to move to the next available video.

B. iWork Keynote Integration

Apple Keynote 19 is a presentation software that belongs to the iWork office suite. We develop a plugin that enables the control of Keynote on a computer through a smartwatch leveraging our gesture recognition engine. The communication flow goes through a smartphone using Bluetooth RFCOMM sockets so the presenter can solely rely on her personal devices and not on the infrastructure in the presentation room. We employ a Mac Book Air and develop a system daemon that accepts Bluetooth connections whenever applicable. Whenever a gesture is recognized, Apple Script 20 is used to generate a

<table>
<thead>
<tr>
<th>Key</th>
<th>Movement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Next Clip</td>
<td>Hand swipe to the left</td>
</tr>
<tr>
<td>Previous Clip</td>
<td>Hand swipe to the right</td>
</tr>
<tr>
<td>Start/Pause</td>
<td>Push forward gesture</td>
</tr>
<tr>
<td>Volume UP</td>
<td>Arm and hand raised</td>
</tr>
<tr>
<td>Volume Down</td>
<td>Arm and hand lowered</td>
</tr>
<tr>
<td>Fast Forward</td>
<td>Arm and hand rotated clockwise</td>
</tr>
<tr>
<td>Fast Backward</td>
<td>Arm and hand rotated anticlockwise</td>
</tr>
</tbody>
</table>

TABLE II: Players actions connected to Gestures

14http://www.apple.com/it/tv/
15http://www.chromecast.com
17https://osmc.tv/
18https://developer.apple.com/

VII. CONCLUSIONS AND FUTURE WORKS

In this paper, we propose a smartwatch-based gesture recognition engine to augment the interaction between users and multimedia applications. Our system leverages deep learning, specifically in the form of Long Short Term Memory neural networks. We use sensitivity analysis to find the simplest network configuration that offers the highest accuracy and show that LSTM can recognize a wide array of gestures with high accuracy. Moreover, we compare LSTM with traditional approaches and show that LSTM significantly outperforms them. We illustrate the details of our Android OS and iOS implementations and present two specific use cases.

Our future plans include the release of the system as an open source platform for gesture recognition. In parallel, we plan to extend the gesture recognition engine and offer a unified solution that can work with sensory input from smartwatches as well as other devices (including smartphones).

ACKNOWLEDGEMENTS

This work was supported by the EU FP7 ERANET program under grant CHIST-ERA-2016 UPRIS-E-IOT.

REFERENCES


[9] Graeme S Chambers, Svetla Venkatesh, Geoff AW West, and Hung Hai
Fig. 10: Example of activation gesture in the "Home Entertainment System Integration" scenario. In Figure 10a, the user is watching a video. With the left-to-right hand swipe shown in Figures 10b and 10c, the user switches to a different video, which the user watches in Figure 10d.


