REACH: Enabling Single-Handed Operation on Large Screen Mobile Devices

Varun Perumal
Dept. of Computer Science
University of Toronto
varun@cs.toronto.edu

Ahmadul Hassan
Elec. & Computer Eng.
University of Toronto
ahmadul.hassan@gmail.com

Zahid Abul-Basher
Mech. & Industrial Eng.
University of Toronto
zahid@cs.toronto.edu

ABSTRACT

We introduce REACH, a system that leverages force sensors built into a case for a mobile device to detect a user's interaction with the device. Using the force sensors, REACH can detect when the user is trying to reach the far corners of the phone or is simply squeezing it. We implemented a hardware prototype for REACH and the software required to analyze the grip patterns and predict them We demonstrate that it is possible to use the hardware case to accurately determine the type of grip that the user is employing. This is achieved through the use of machine learning algorithms which prove to be quite robust and adept at performing this type of classifications. . REACH was able to distinguish between three grip patterns on the device with a cross-validation accuracy of 99%.

Categories and Subject Descriptors

H.5.2 [**Information interfaces and presentation**]: User Interfaces—graphical user interfaces

General Terms

Design, Experimentation, Human Factors

Keywords

Data analytics, Gesture recognition

1. INTRODUCTION

While the rising popularity and adoption of large screen mobile phones presents users with a larger canvas for content consumption, it also poses several significant usability issues. The most important one being the ergonomic mismatch that these larger devices pose for single-handed use. Most of today's mobile interactions are centered on quick, short and intermittent usage patterns. Hence, single-handed interaction with mobile devices is of importance, since it supports these kinds of usage patterns. All device manufacturers who make large screen mobile devices have implemented ways to enable single-handed use. Apple's larger phones allow the user to double tap the home button to quickly pull the entire interface closer to the bottom edge of the screen, Samsung, allows users to snap windows to different corners of the screen to allow for one handed access. However, these methods add an extra step that the user must go through in order to interact with the device, thus breaking the continuity of the interaction.

With REACH we propose to solve the above mentioned problem by appending sensor systems to the phone which can detect when the user is trying to reach the far corners of the phone and affect changes in the UI accordingly. As a bonus, project reach also explores the use of this appended sensor framework to enable additional interaction on mobile devices. Although prior work has been done on sensing grip, our project aims to use this information for a very specific purpose.

CSC2525 '14 UofT, CA

2. RELATED WORK

Many researchers have suggested that the devices should be intelligent enough to detect user's situation for better support as in [9] and [13]. For instance, *ability based design* aims to find the best match between the ability of the users and the interfaces [19]. There are also researches to recognize the activity of users on devices (also known as *activity recognition*). Choudhuri *et al.* [2] built a wearable device with sensors to detect the activity of the users. In [16], Laerhoven used an accelerometer in a phone to recognize different motions of walking, climbing stairs, *etc.* Schmidt *et al.* [13] also used accelerometer but to detect both the user movement and the place of the device itself whether it is in the hand or on a table or in a suitcase. GripSense [4] used gyroscope and vibration motor to classify the user's touches based on the pressure on the screen. There is also many studies in the context of detecting hand postures. Harrison *et al.* [6] and Kim *et al.* [12] used touch sensors to detect the pattern of user's grips on mobiles. Furthermore, Taylor and Bove [15] used accelerometers to improve the detection of the changes in the grip dynamically.

Many researchers also studied hand posture on devices to make them more intelligent and interactive to the situations caused by posture. For instance, Wobbrock *et al.* [20] studied different hand postures and measured the finger performance with mobile devices. Holz *et al.* [8] have evaluated systematic error in selecting the target with finger touch. Researchers [7, 17, 11] also found that mobile interfaces are designed for double-handed operation although users may prefer to use one single hand. Karlson *et al.* [10]

studied those interfaces and evaluated the performance of thumb mobility on those interfaces. Azenkot and Zhai [1] showed that different hand postures lead to different touch patterns, thus, effect the performance of typing on mobile devices. AppLens and LaunchTiles [11] designed interfaces based on different thumb gestures for one handed interactions.

Fitzmaurice *et al.* [3] introduced the idea of "graspable user interfaces" where you can control the interface by interacting with a physical object. SqueezeBlock [5] is an implementation of this idea in which it provides haptic feedback according to the level of "squashiness" on a physical object. Wimmer *et al.* [18] deployed optical fibers into a surface of device to detect grasping pressure. Harrison *et al.* [6] used FSRs for squeezing pressure detection. Strachan and Murray-Smith [14], used muscle tremor as a form of input to detect pressure on devices by leveraging accelerometer logs.

REACH differs from previous works in that it uses the hardware implementation of force sensors to study users grip patterns on mobile phones. This opens the possibility to detect the user's activity from the pressure he applies on the device in realtime manner.

3. OVERALL DESIGN OF REACH

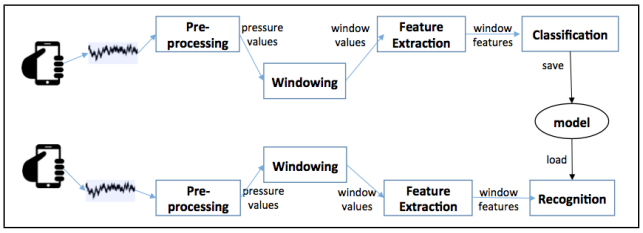

Figure 1: Design of REACH

The design of REACH involves three distinct phases - building the hardware to collect and transmit force sensor data, performing off-line analysis to train a classification model and finally performing real-time evaluations to detect grip patterns and perform a corresponding action. The overall design of REACH is illustrated in Figure 1.

The built-in force sensors continuously send force values to the classification phase of the system. In classification phase, the force values are windowed and suitable features from the window values are extracted. These features are then used to find a suitable classification model and the parameters of the model are saved. In recognition phase, the saved classification model is used to detect the corresponding actions on the phone. We describe each phase of REACH in details in the following sections.

4. HARDWARE DESIGN OF REACH

In order to achieve our aforementioned vision we have augmented the sides of the mobile device with a total of 16 Force Sensitive Resistors (FSRs). These sensors change their resistance in response to the amount of force applied on them - the higher the force the lower the resistance. As shown in Figure 2, the sensors are split into two groups of 8, one grouping on each side of the device. The placement of these sensors has been optimized to ensure that a right-handed user has most contact with them while using the device. All the sensors are mounted onto a specially 3D printed case that house the acquisition electronics. A cable connects this setup to an Arduino Mega that handles the job of transferring the data to a Java application application running on a laptop. The hardware reads all sensor values at a rate of 1000 Hz. These values are then logged by the application for further processing.

Figure 2: a) The sensors on the 3D printed case b) Connection to the Arduino c) A visualization of the force values

5. SOFTWARE DESIGN OF REACH

In addition to the hardware implementation by selecting appropriate force sensors and locating them in the appropriate places around the mobile device, we also build the classifier for the hand grip. In this section, we discuss how we collect force sensor values from mobile device for training and how we implement the classifier for grip pattern detection. We then used the model to predict the pattern in realtime manner.

For training the model, we used the Weka (Witten & Frank 2005) machine learning library for Java. We used Bayesian network, Support vector machine and Classification tree algorithms for training the model on the collected data. We performed off-line training on a laptop and extracted the parameters of the best model to implement the realtime version of REACH.

5.1 Tap Detection as Proxy

While waiting for the hardware to be completed, we decided to perform some experiments to provide insight into how to classify the grip patterns. Tap detection was chosen as a proxy for grip pattern detection. We used the accelerometer data available from an Android phone to try and classify instances when a user performs a single tap on the back of the device.

The accelerometer data from 3 axis (x, y, z) is analogous to the force data available from the 16 sensors for grip detection. Similarly, both accelerometer and force data will change when an action is performed by the user. The only difference is that the accelerometer data changes depending on the orientation of the phone, while the force data is unaffected. To mitigate this, all data collected and analyzed for tap detection focused on keeping the phone in portrait orientation, the y-axis facing away from the floor and the z-axis facing towards the user.

5.1.1 Determine Window Size

The Android sensor collects accelerometer data on a periodic basis. This sampling interval is neither guaranteed nor specified by the Android SDK, but using the default mode, it was found to be 20ms on *average*. After analyzing the data, it was noticed that the characteristic pattern of a single tap lasted for a duration of 400ms on average. This duration is defined as the *window duration*.

Dividing the window duration by the sampling interval gives us a *window size* of 20. The accelerometer data is partitioned into windows with the specifications above, and implemented as a *sliding window*. This means that the starting time for each subsequent window differs by the sampling interval. Contrast this against a *jumping window*, where the start time for subsequent windows would be the window duration. Using the *sliding window* protocol ensures that a possible tap is not missed during the evaluation process.

Picking the right window size is crucial since it has a direct impact on the accuracy of the classification model. If the *window size* is too small, then the complete characteristic pattern of a tap will not be captured, leading to incorrect classifications. However, using a window size that is too large will capture extraneous noisy data that will also reduce the classification accuracy. This topic will again be explored for grip pattern detection.

5.1.2 Determine Features

Upon observing the data, it was noticed that acceleration values of the x, y and z axis all changed when the tap action was performed. The *inter*-window change was captured by calculating the *mean* for a window, and this was used as one of the features for training the model. There was a visible change in the acceleration values within the window duration, and this *intra*-window change was represented by calculating the *variance* of a window.

5.1.3 Classify a Tap

Even though a windowing principle was used, windows clustered near a tap duration all show a similar characteristic pattern, albeit time-shifted. Therefore while performing the manual classification of the training data, we decided to label, on average, 20 windows as a single tap. This means that when a tap is being evaluated with real-time data, we would expect 20 back-to-back windows all to be predicted as a tap. Once such a scenario is detected, we would report a successful tap as being detected.

5.1.4 Tap Model Performance

We collected acceleration values of the x, y, and z dimension of three different activities on the mobile: *None*, *Motion*, and *Tap*. The mean and variance of each window for each dimension is then calculated and are used as instance features for training a model. The Motion activity is added to represent the boundary between Tap and None activities. The corresponding activity of each window is labeled manually (*i.e.*, None, Motion, and Tap) and used as class feature for training a model.

We used Bayesian network, Naive bayes and Support vector machine algorithms with Weka's default configuration for training. We also used different number of instances with different ratios between class features. In all cases, Bayesian network was the winner among other algorithms. Table 1 shows the confusion matrix of using Bayesian network algorithm with 10-fold cross validation of 2250 instances (among them are 1000 None, 1000 Motion, and 250 None instances). The matrix shows that 76 of Tap activities are classified as Motion which is reasonable due to the similarity between these two activities. While 142 of Motion activities are classified as None, we believe that these are the instances which exist in the boundaries between Motion and None activities.

5.2 Grip Classifier

Table 1: Confusion Matrix using Bayesian network with 10-fold cross validation

	None	Motion	Tap
None	970	29	1
Motion	142	825	33
Tap	1	76	173

We used the insights gained from tap detection to guide our grip classification process, namely a sliding window evaluation protocol and similar features to describe a grip pattern. We begin by attempting to classify three grip patterns in our prototype: None, Squeeze, and Reach. In None, the subject holds the device without performing any activity. In Squeeze, the subject is applying a squeeze-force on the device and in Reach, the subject is moving his thumb finger to reach the top of the device while holding the device.

The 12 force sensors around the device continuously reported data every 1 ms on average. Based on our experience with tap detection, we felt that these values could be safely aggregated into a 20ms sampling interval for the purpose of grip detection. Figure 3 shows the change in a sensors value for a *window size* of 60. We can clearly see that choosing a value of 20 would result in the loss of essential information, while a value of 60 would admit noisy data. It was determined that a *window size* of 50 represents the average case.

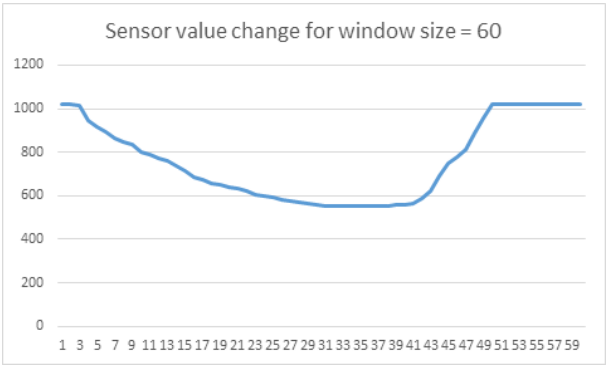

Figure 3: Sensor value changes for window size 60

Similar to tap detection, we calculated the mean and variance for each *sliding window*. It was observed that different people perform the same gesture in slightly different ways. To increase the robustness of the trained model, we decided to use the mean values of all the sensors located on one side of the device. The variance for each sensor was calculated with respect to these averaged means. For each window the following features were calculated -*mean left*, *mean right* and *variance* for each sensor.

5.2.1 Variability of Grip Patterns

We wanted to establish a good baseline performance for grip pattern detection, which led us to collect all training and evaluation data from 1 subject. The individual was asked to perform Hold, Squeeze and Reach every 15 seconds for a duration of 1 minute. Each dataset collected involved repeating this process 3 times. We noticed that even for the

Table 2: Grip model cross validation accuracy with multiple datasets

	initial dataset			expanded dataset		
	count	SVM	Bayes	count	SVM	Bayes
None	13751	99.4%	98.8%	20773	99.6%	99.1%
Squeeze	871	97.0%	99.7%	871	95.5%	97.5%
Reach	679	97.9%	99.3%	1361	94.0%	95.6%

Table 3: J48 Grip model cross-validation Confusion Matrix

	total	None	Squeeze	Reach	Correct
None	20773	20729	9	35	99.8%
Squeeze	871	9	855	7	98.2%
Reach	1361	16	4	1341	98.5%

same individual, subsequent datasets resulted in a variability of the grip patterns. The model that was trained for the first dataset performed poorly for the second dataset. However, when the second dataset was used to train the model, cross-validation performance was very similar to the original dataset. We will explore the performance further in the Evaluation section.

6. EVALUATION

Our initial goal of having the grip classification being performed in real-time on the device was abandoned due to time constraints. Therefore we focused our evaluation criteria on how accurately the grip patterns could be identified. While training the model, we performed a 10-fold cross-validation that provides insight into the model's performance.

6.1 Model Evaluation

Table 2 demonstrates shows the performance of the model when trained with two datasets, and two algorithms - SMO and BayesNet. As expected, the prediction accuracy decreases when a larger dataset is used. This is due to the variability of the manner in which the gestures are performed. Conversely, this should also result in a more robust model that can accurately predict a gesture with small variations.

It was determined that the best performance achieved by a model for the expanded dataset was using the J48 tree classifier, as shown in Table 3. This makes intuitive sense since all the features are thresholded, which can be easily expressed as a *classification tree*.

6.2 Off-line Evaluation

We decided to use the J48 tree grip detection model since it produced the best cross-validation results. Since we are not performing real-time evaluation, we first record our evaluation dataset in a manner similar to the training dataset - generating a log of the sensor values as the user performs the squeeze and reach gestures. This dataset is then parsed to generate the sliding windows and extract the features.

We wrote a simple Java program running on a laptop that would load the J48 grip classification model, read the features of each window and ask the model to classify it as None, Squeeze or Reach gesture. Without looking at the predictions for each window, we classified each window manually. Figure 4 and Figure 5 displays a comparison of the predicted and actual window classifications for Reach and Squeeze grips, respectively. The average difference between the classifications was 2% on average, with a maximum of 6%, for both types of gestures.

Similar to tap detection, a gesture is detected when 50 consecutive windows are classified as the same. It was observed that the misclassified windows all tended to be at the beginning or the end of each gesture duration. For each of the Squeeze or Reach gestures, there was always a set 50 consecutive windows that were correctly classified. This means that the gestures were detected with 100% accuracy.

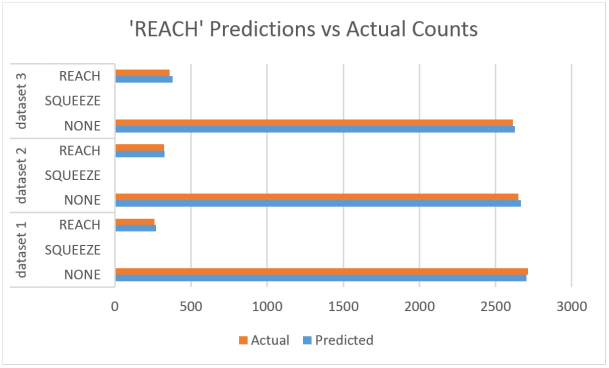

Figure 4: 'Reach' grip evaluations for 3 distinct datasets

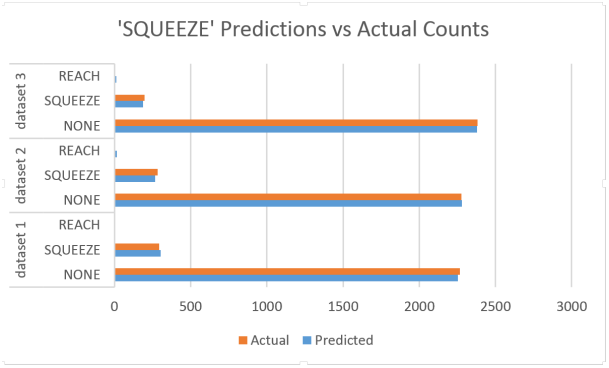

Figure 5: 'Squeeze' grip evaluations for 3 distinct datasets

7. CONCLUSIONS AND FUTURE WORKS

The initial goal of REACH was to detect a specific grip type and perform an action as result. Although we were not able to implement it in an end-to-end manner, we believe that we have demonstrated the feasibility of such a system. It is impressive to see that the classification tree machine learning algorithm was able to accurately distinguish between 3 types of grip patterns.

For future work, the following key areas were identified -

- In future prototypes, we intend to work on miniaturizing this setup and giving it the ability to wirelessly connect to phone applications (see Figure 6).
- Interface with the hardware wirelessly to collect the force sensor data. This data would then dynamically be windowed, the features extracted and the model would make a prediction, all in real-time.

- Collect a larger training dataset with the grip gestures being performed by multiple users. Explore other features that could lead to better predictions.
- Train a model that can accurately detect gestures for a large population size. This *average* model could be supplemented with specific user data to increase prediction accuracy.
- Create a user interaction scheme that dynamically responds to the grip gestures and evaluate its effectiveness.

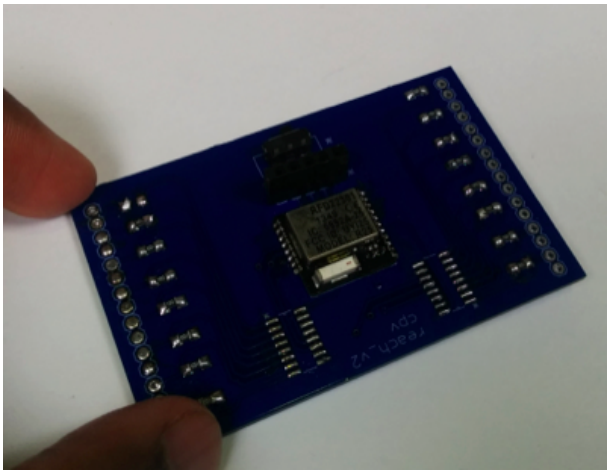

Figure 6: Hardware miniaturized proof of concept

8. REFERENCES

[1] S. Azenkot and S. Zhai. Touch behavior with different postures on soft smartphone keyboards. MobileHCI '12, pages 251–260, 2012.

[2] T. Choudhury, S. Consolvo, B. Harrison, J. Hightower, A. LaMarca, L. LeGrand, A. Rahimi, A. Rea, G. Bordello, B. Hemingway, et al. The mobile sensing platform: An embedded activity recognition system. *Pervasive Computing, IEEE*, 7(2):32–41, 2008.

[3] G. W. Fitzmaurice, H. Ishii, and W. A. S. Buxton. Bricks: Laying the foundations for graspable user interfaces. CHI '95, pages 442–449, 1995.

[4] M. Goel, J. Wobbrock, and S. Patel. Gripsense: using built-in sensors to detect hand posture and pressure on commodity mobile phones. In *Proceedings of the 25th annual ACM symposium on User interface software and technology*, pages 545–554. ACM, 2012.

[5] S. Gupta, T. Campbell, J. R. Hightower, and S. N. Patel. Squeezeblock: Using virtual springs in mobile devices for eyes-free interaction. UIST '10, pages 101–104, 2010.

[6] B. L. Harrison, K. P. Fishkin, A. Gujar, C. Mochon, and R. Want. Squeeze me, hold me, tilt me! an exploration of manipulative user interfaces. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 17–24, 1998.

[7] K. Hinckley and H. Song. Sensor synaesthesia: Touch in motion, and motion in touch. CHI '11, pages 801–810, 2011.

[8] C. Holz and P. Baudisch. Understanding touch. CHI '11, pages 2501–2510, 2011.

[9] P. Johnson. Usability and mobility; interactions on the move. In *First Workshop on Human Computer Interaction with Mobile Devices*, 1998.

[10] A. K. Karlson and B. B. Bederson. Understanding single-handed mobile device interaction. Technical report, 2006.

[11] A. K. Karlson, B. B. Bederson, and J. SanGiovanni. Applens and launchtile: Two designs for one-handed thumb use on small devices. CHI '05, pages 201–210, 2005.

[12] K.-E. Kim, W. Chang, S.-J. Cho, J. Shim, H. Lee, J. Park, Y. Lee, and S. Kim. Hand grip pattern recognition for mobile user interfaces. In *Proceedings of the National Conference on Artificial Intelligence*, volume 21, page 1789, 2006.

[13] A. Schmidt, K. A. Aidoo, A. Takaluoma, U. Tuomela, K. V. Laerhoven, and W. V. d. Velde. Advanced interaction in context. HUC '99, pages 89–101, 1999.

[14] S. Strachan and R. Murray-Smith. Muscle tremor as an input mechanism. UIST '04, 2004.

[15] B. T. Taylor and V. M. Bove Jr. Graspables: grasp-recognition as a user interface. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 917–926. ACM, 2009.

[16] K. Van Laerhoven and O. Cakmakci. What shall we teach our pants? In *Wearable Computers, The Fourth International Symposium on*, pages 77–83. IEEE, 2000.

[17] L. Weberg, T. Brange, and A. W. Hansson. A piece of butter on the pda display. CHI EA '01, pages 435–436, 2001.

[18] R. Wimmer. Flyeye: Grasp-sensitive surfaces using optical fiber. TEI '10, pages 245–248, 2010.

[19] J. O. Wobbrock, S. K. Kane, K. Z. Gajos, S. Harada, and J. Froehlich. Ability-based design: Concept, principles and examples. *ACM Transactions on Accessible Computing (TACCESS)*, 3(3):9, 2011.

[20] J. O. Wobbrock, B. A. Myers, and H. H. Aung. The performance of hand postures in front- and back-of-device interaction for mobile computing. *Int. J. Hum.-Comput. Stud.*, 66(12):857–875, 2008.