Inaccuracy in Touch Screen Tapping

Author

Mamoona Javaid

2011-NUST-MS-CS-01

Supervisor

Dr. Muhammad Muddassir Malik

DEPARTMENT OF COMPUTING

SCHOOL OF ELECTRICAL ENGINEERING & COMPUTER SCIENCE

NATIONAL UNIVERSITY OF SCIENCES AND TECHNOLOGY

ISLAMABAD

JULY, 2015

Inaccuracy in Touch Screen Tapping

Author

Mamoona Javaid

2011-NUST-MS-CS-01

A thesis submitted in partial fulfillment of the requirements for the degree of

MS Computer Science

Thesis Supervisor:

Dr. Muhammad Muddassir Malik

Thesis Supervisor's Signature:_____

DEPARTMENT OF COMPUTING

SCHOOL OF ELECTRICAL ENGINEERING & COMPUTER SCIENCE

NATIONAL UNIVERSITY OF SCIENCES AND TECHNOLOGY

ISLAMABAD

JULY, 2015

NUST School of Electrical Engineering and Computer Science

A center of excellence for quality education and research

Certificate

Certified that the contents of thesis document titled “Inaccuracy in Touch Screen Tapping” submitted by Miss Mamoona Javaid have been found satisfactory for the requirement of degree.

Advisor: _____

Dr. Muhammad Muddassir Malik

Committee Member1: _____

Dr. Aimal Rextin

Committee Member2: _____

Dr. Hamid Mukhtar

Committee Member3: _____

Mr. Shamyl Bin Mansoor

Declaration

I certify that this research work titled "*Inaccuracy in Touch Screen Tapping"* is my own work. The work has not been presented elsewhere for assessment. The material that has been used from other sources it has been properly acknowledged / referred.

Signature of Student

Mamoona Javaid

2011-NUST-MS-CS-01

Language Correctness Certificate

This thesis has been read by an English expert and is free of typing, syntax, semantic, grammatical and spelling mistakes. Thesis is also according to the format given by the university.

Signature of Student

Mamoona Javaid

2011-NUST-MS-CS-01

Signature of Supervisor

Acknowledgements

All praises are due to ALLAH, the most Merciful, and most Beneficent, Who has guided me throughout this work. I am thankful the Almighty ALLAH for the guidance and endless help, he bestowed on me during my thesis.

I am indebted to my family and friends, who supported and helped me along this journey by providing the moral and emotional support I needed to complete my thesis. To them, I am eternally grateful.

I take this honor to express my gratitude to my supervisor Dr. Muhammad Muddassir Malik for his guidance and patience throughout my whole thesis.

It's worth mentioning to appreciate Dr. Aimal Rextin for being on my thesis guidance committee and for his extraordinary support and assistance. He helped me find a way out for every issue I faced.

I would also like to thank Dr. Hamid Mukhtar, and Mr. Shamyl Bin Mansoor for being on my thesis guidance committee.

Dedicated to my exceptional parents, adored siblings and benevolent husband whose tremendous support and cooperation led me to this wonderful accomplishment

Abstract

Smart phones have become an essential handheld computing device in recent years. People use their phones while performing various routine activities such as standing, walking, traveling, shopping, eating, working etc. Text entry is one of the most common feature on smart phones. Performing routine activities and typing accurately on phones simultaneously has become difficult for users especially with touch screen keyboards. Touch screen keyboards also known as virtual keyboards lack tactile feedback and their typing mistakes are higher than physical keyboards. Although there has been extensive research in area of virtual keyboards for improving text entry on smart phones, still we lack a keyboard which can work better in all activities. Most of the research incorporated text entry in static position which can tackle stationary activities such as sitting, standing or lying down. While today's demand is more than just static activities. In recent years walking activity has also been considered. As a result of this ongoing research many keyboards have been developed. This work aims to find out specific keyboards for specific activities. Instead of developing one keyboard which can work better in all activities, we analyze the performance of 5 existing virtual keyboards in three activities. We developed an application for text entry and 30 participants were asked to perform testing of our application. Participants were asked to fill a survey at the end of each activity. ANOVA analysis and pairwise comparisons showed that Android Landscape performed better in travelling and walking activities while sitting activity doesn't show any significant performance difference among keyboards.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Overview

These days, *smart phones* is an essential personal hand-held computing device, it is estimated that one third of all world population will be using smart phones by the year 2017 [6]. Initially, mobile phones were used only to send text messages or making phone calls, however these days a wide variety of applications are available for smart phones [5]. Still, even now, text entry is the most common activity on smart phones [13, 15]. Hence, it is reasonable to assume that if the usability of text entry is improved in a smart phone, then a significant impact on the overall usability of smart phone usage can be achieved. This is probably the reason why software developers as well as research community is continuously trying to develop improved virtual keyboard for smart phones.

1.2 Smart Phone Usage and Virtual Keyboards

A common user uses her smart phone in a variety of different locations and while doing many different activities [13]. It is reasonable to assume that a majority of activities a user performs while using a mobile phone can be divided into the following *activity classes*: user is *stationary*, user is *walking*, and the user is *travelling* in a vehicle. These three types of activity classes require different types of interfaces due to the accelerations experienced by the body and differences in attention that a user can give to an interface [27, 28, 7]. The current research in optimizing virtual keyboard focuses on improving their efficiency and effectiveness when the user is performing a single activity class. In terms of keyboards, increasing *efficiency* means reducing the time in which a user inputs a particular text; while improving the *effectiveness* of a keyboard means reducing the errors a user is likely to make while she inputs a particular text. There has been extensive research on improving text entry of virtual keyboards when the user is stationary [20, 33, 36, 19] and some research when the user is walking [34, 26]. As a result of this ongoing research many keyboards have been developed [3, 2, 1]. But, there has been no research on how one can optimize typing performance across the three activity classes.

The three activity classes differ in terms of acceleration and body movement. Thus typing across the three activity classes can be improved significantly if a smart phone is able to activate a virtual keyboard that is most suitable for the current activity class of the user. Note that *activity recognition* is a well studied problem and several well established algorithms exist for it [10, 39, 22, 16]. Activity recognition is used in many applications, such as health care [8, 21] and context aware systems for elderly people [12]. It is now available on Android SDK, allowing programmers to develop ap-

plications that can behave differently when the user is performing different activities [4]. However, keyboard performance across different activity classes with the help of activity recognition algorithm can only be improved, if different keyboards have different performance in unequal activity classes. The main contribution of this work, is to investigate the performance of 5 different keyboards available on Android Play and study if their performance is significantly different across unequal activity classes.

Chapter 2

Literature Review

Many different approaches have been used to improve typing experience of users of virtual keyboards in recent years. This research has many aspects like, virtual keyboard typing accuracy, typing while walking, comparative analysis of various keyboards. We will briefly look into each of the aspect and what work has already been done in that area.

2.1 Virtual keyboards

Virtual keyboards have been under consideration of researchers for many years now. Lack of tactile feedback in virtual keyboards make them prone to mistakes. Different techniques have been utilized for improving typing experience of virtual keyboards. These techniques can be divided into sub-categories as follows.

2.1.1 Predictive Language Model

Many of the available virtual keyboards use language model to predict the next letter or complete word. This technique has been used drastically to

facilitate users in speeding up their typing. A *Language Model* [18] predicts the word based on the currently typed letters. It uses probability distribution of strings to predict the correct word. Many virtual keyboards such as Swype, Swiftkey, Thickbuttons etc. use language models either for word prediction or next letter prediction. Google Android and Apple's soft keyboards also include language models. It is optional for the user to turn prediction on or off in these built in keyboards.

2.1.2 Shifting touch position

Users hold mobile devices in a tilted position. Research showed that this tilt in position results in touching below the target key [35]. The effect of typing below the target can be alleviated by a *shift* in touched position to accurately identify target key. Android keyboard already provides 10dp upward shift in touched position. *Dp* refers to density independent pixels, which is an abstract unit where 160dp equals to one inch. Henze et. al. [20] presented an adaptive shift algorithm which improves performance of typing accurately compared to the fixed 10dp shift. *Adaptive shift* is shifting the center of touched distributions to the center of keys.

2.1.3 Key-Target resizing

Key-target is referred to a key's hit area which is the square box bounding and separating it from other keys' areas. Many researchers proposed resizing of key-target according to the predicted key based on a language model and the actual touch position. The resized area is not squared area but can take any shape according to the touched position shown in Figure 2.1. Gunawardana et. al. [19] proposed an anchor key-target resizing algorithm. *Anchor* of a key corresponds to the center area of the key which shouldn't be assigned

to other keys in process of key-target resizing. Anchor key-target resizing adjusts the key sizes such that the key's anchor still returns the actual key instead of predicted key to avoid users facing a situation of not being able to type an intended letter. Rudchenko et. al. [33] proposed personalized key-target resizing which takes data of single user and resize the target key according to the input of a specific user. This further improved performance from general key-target resizing.

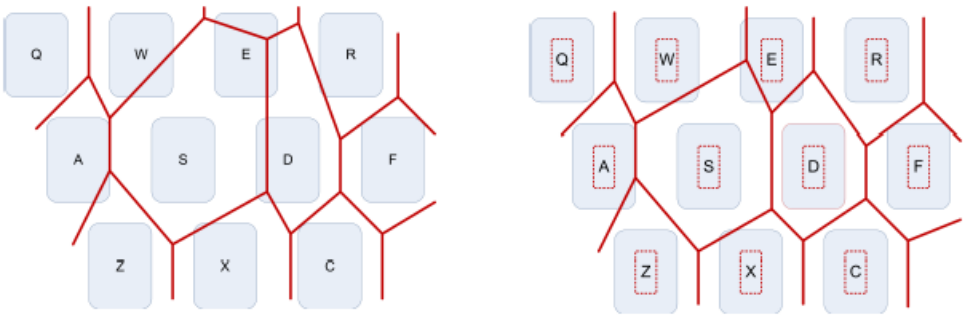

Figure 2.1: Key Target Resizing

2.1.4 Custom layouts for virtual keyboards

Most common layout for virtual keyboards is Qwerty. Although qwerty performs well in physical keyboards, it fails to do so in virtual environment. Researchers have explored many custom layouts [14, 30, 11, 38] for virtual keyboards to improve typing speed. Yanshan Wang [38] proposed a novel soft keyboard QWERT which is inspired by the Qwerty layout. *QWERT* is designed with large buttons for easy access of letters however the large size of buttons only allow a small numbers of direct access letters on screen. To access other letters which are not directly available users need to touch and

slide up the finger which can be annoying and time consuming. Qwert layout is shown in Figure 2.2. Another similar layout QWERTH[14] was designed by Dunlop et. al. with 5 keys per row. It also has the similar problem like QWERT layout of two letters per key for some keys. It is shown in Figure 2.3. Oulasvirta et. al.[30] presented KALQ layout for easing two thumb entry. They proposed a keyboard with keys on two partitions shown in Figure 2.4. Many other layouts have also been designed such as Metropolis [40], EasyWrite[11], and OPTI[25] to name a few but none of them gained popularity as the Qwerty layout did. Qwerty still remains the most common layout and most research has been done to improve the typing experience of Qwerty users.

Figure 2.2: QWERT Layout

Figure 2.3: Qwerth Layout

2.2 Typing while walking

Walking activity has also been under consideration by many researchers. Schildbach et. al. [34] proposed that increase of button size by 20% and 40% reduce the error rate of typing while walking. Their results show that increasing the size by 20% give significant improvements and increasing by

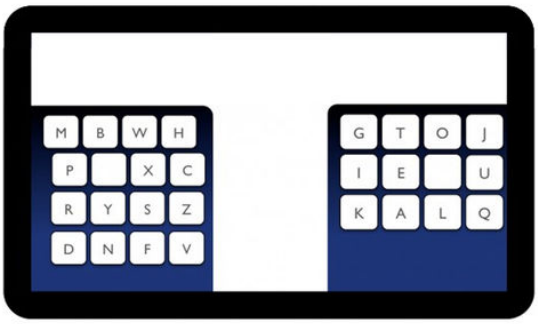

Figure 2.4: KALQ layout

40% completely compensate the effect of walking. Goel et. al. [17] presented a technique to improve typing accuracy while walking by incorporating mobile phone's accelerometer data. They used tap location, tap travelling, and accelerometer values as features for their models. They proposed two models distance and acceleration model and walking pattern model. Combining the result of both models improves accuracy significantly. Ahmad et. al. [32] developed NoShake, which dynamically compensates for screen shaking by shifting the contents of screen to opposite direction of shaking. They used accelerometer data to deduct shaking using an empirical threshold value and applied a simple physics model mass-spring-damper to compensate for shaking. The model is parametric and can be fine-tuned by the user. Hugo et al. [29] related visually impaired user to motor impaired users and presented a study where participants were given motor impaired interfaces for text entry to compensate the visual demands of typing while walking. They analysed the data from this study and developed new solutions for text entry improvements. Although not much improvements they could achieve, they did found out that Qwerty keyboard is still preferred over voice input.

2.3 Comparative studies of virtual keyboards

Sachi et al. [26] did a comparative analysis of various soft button sizes to determine which size is most suitable for walking activity. They developed a testing program with four different keyboard sizes and tested over a number of users with two activities i.e. standing and walking. They found 3mm key size is suitable for text entry on soft keyboards. They further discovered that bigger size doesn't decrease error significantly. In this study customized qwerty keyboards were used. Tom Page[31] presented an analysis of five virtual keyboards available in market to show which keyboard works better. This study focused on keyboard but not the activity, and the dataset was very small. Only 6 participants were chosen for testing.

All these works tried to improve text entry on virtual keyboards. But existing works either have focused on one activity for text entry problem or if multiple activities have been under consideration then the purpose was to develop a unique keyboard which can work better in all the activities. Instead of developing new methods for typing accurately on virtual keyboards this study explores the problems in typing using existing keyboards in activities such as sitting, walking, and travelling.

Chapter 3

Methodology

3.1 Virtual Keyboards Selection

Five keyboards were selected to test under three different activities. The selected keyboards are Android Portrait (Figure 3.1), Android Landscape(Figure 3.3), Android 3x4 T9 (Figure 3.2), Swiftkey (Figure 3.5)and Thickbuttons (Figure 3.4).

3.1.1 Android Portrait, Android Landscape and Android 3x4 T9

Android Portrait, Android Landscape and 3x4 T9 were selected because they are the most commonly used keyboards by Android users due to the built in availability. Another factor in choosing different variations of Android keyboard is that Landscape and 3x4 T9 provides larger buttons than the Portrait layout.

3.1.2 Swiftkey

Swiftkey keyboard is one of the best keyboard among all custom keyboards [9, 37]. It utilizes the language model for word prediction. It not only predicts the current word based on typed letters but also predicts next words. Swiftkey learns users writing style and add the new words to its dictionary for later prediction. It also provides gestures feature to facilitate typing experience. Swipe up is used to switch to upper case writing. Swipe right to left deletes the previous word. It also provides auto complete which can be accomplished by pressing space-bar while typing a word.

3.1.3 Thickbuttons

Thickbuttons uses the technique of enlarging predicted keys for ease of selection. In Virtual keyboards it is a very common issue that keys are too small to select with fingers and mostly fingers occlude multiple key areas. Thickbuttons target this problem by enlarging the actual key size visually. As the user types the letters prediction becomes narrower and fewer keys enlarge to maximum, making it easy to select a desired key. However, due to constant changing size of keys, it is hard to remember key location. The performance of these keyboards was analysed in sitting, travelling and walking activity classes.

3.2 Android Application

An Android application was developed for text entry on each keyboard. Motivation behind the application was to track the information about incorrect typing. The question was, how incorrect typing can be measured. A person

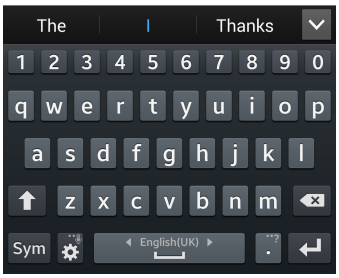

Figure 3.1: Android Portrait

Figure 3.2: Android 3x4 T9

Figure 3.3: Android Landscape

Figure 3.4: Thickbuttons

typing too slow can make few mistakes in typing while a person typing fast can make more mistakes. Counting the number of incorrect letters alone was not sufficient to measure inaccuracy. So an application was developed to log information about mistakes and the time taken to enter a word. A *Mistake* is an incorrect letter typed by the user. *Time* is the total time taken to enter

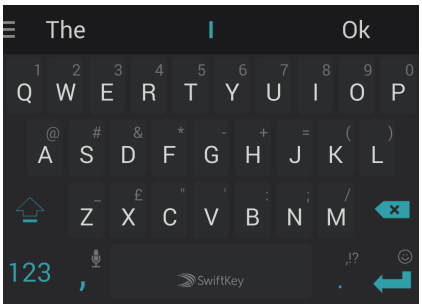

Figure 3.5: Swiftkey

a word. Mistake count and Time are accumulated for one keyboard to find out total mistakes and total time.

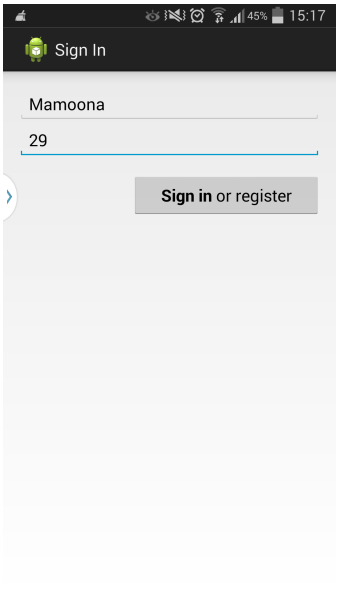

Figure 3.6: Login or Register

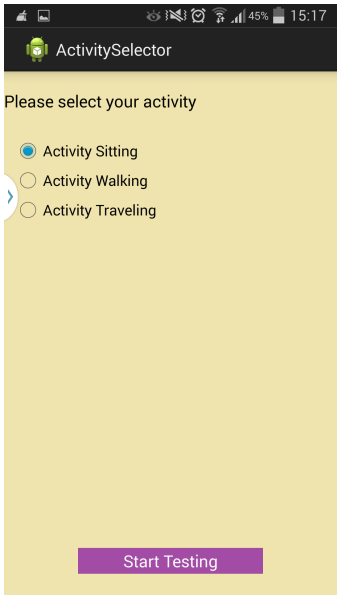

Figure 3.7: Activity Selection

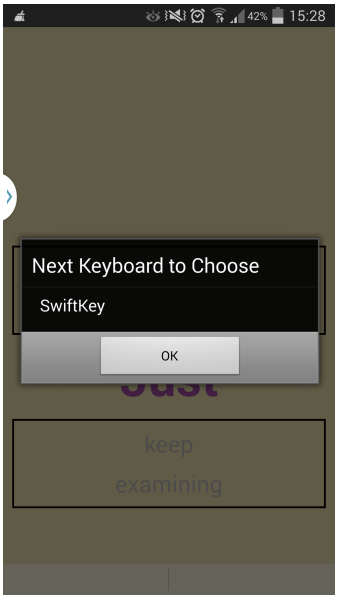

Figure 3.8: Keyboard to Select

In the start of application, user is asked to enter demographic information (Figure 3.6). After login, activity selection screen appears with three options

Figure 3.9: Keyboard Selection

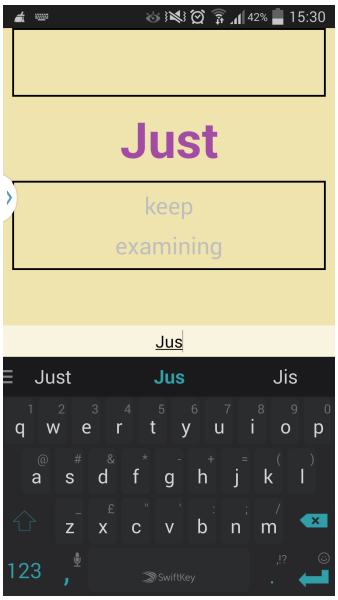

Figure 3.10: Text Entry

containing the three activity classes (Figure 3.7). On the next screen, a message box appears showing which keyboard to select. It is shown in Figure 3.8. When user press Ok button, application invokes input method selection setting which is shown in Figure 3.9. User is presented with a single word at a time from the phrase to avoid problems arising from using space-bar to separate words because the focus here was to analyse individual character being typed (Figure 3.10. The application also displays upcoming words for assistance considering the fact, when we type on our mobile phones we usually know what we are going to type next. Once the user has completed typing on one keyboard, the application tells user to select the next keyboard which is displayed on the screen. Next keyboard is chosen at random by the application.

3.3 Phrase set

Phrases were taken from MacKenzie and Soukoreffs phrase set [24]. This phrase set is published specifically for evaluating text entry techniques by focusing on including those phrases which are short, easy to remember and are representative of language. 45 phrases were selected for each activity from the set. To ensure typing of all alphabets, five pangrams were also added. These 50 phrases were divided among five keyboards equally. One pangram was added to each keyboard phrase set.

3.4 Participants

30 users ageing from 15 years to 34 years were selected to use this application for text entry. All of the users had touch screen experience of 1 year to 6 years. The group included students, engineers, and teachers. 60% of all users used Qwerty Portrait keyboard for typing as their default keyboard. Approximately 3% users used 3x4 T9. 30% users used Qwerty Portrait and Qwerty Landscape alternatively while only 3% used Qwerty Landscape only.

3.5 Procedure

Users were given an introduction to the whole purpose of study and a practice session to familiarize themselves with the application before starting testing. Experiment was designed such that the user typed 50 phrases in one activity and took a break. The whole experiment took around 1 hour per user. The sequence of activities was different for each user to ensure randomization.

For walking and travelling activities special conditions applied which will be discussed in next chapter.

3.6 Survey Questionnaire

Users were given a survey questionnaire at the end of each activity. The survey contained questions regarding each keyboard. The purpose of the survey was to obtain feedback of users about which keyboard they liked for particular activity. The questions of Survey were taken from Lund's Usability Questionnaire[23]. The original questionnaire from Lund contained a large number of questions. We took a small subset of questions from original questionnaire. To choose what questions to include in our survey from the usability questionnaire we conducted a small survey on 5 participants. All the participants were University lecturers. They were given a brief introduction of what we wanted to achieve from the survey. They were then given a questionnaire containing all the questions from the original questionnaire and they were asked to choose what questions they think should be included in Survey. Based on their answers we formed a set of 8 questions for our Usability survey.

Chapter 4

Data Analysis

4.1 Experimental Design

An experiment is to perform a series of actions on a group of objects to carefully observe the response. The data generated from an experiment need to be carefully analysed. The Design of Experiments (DOE) or Experimental Design defines the steps or plan to perform experiment, acquire data, and analyse it to answer question of interest for which experiment is carried out. Experimental design first determines the objectives of an experiments. In this study, the objective was to determine which keyboard perform better than others for specific activity.

4.1.1 Control

It is essential to perform the experiment in identical conditions to avoid external variables effecting the outcome of experiment. This is referred to as *control*. To ensure control treatment, users were asked to walk at a constant pace. For travelling activity, users were taken on a drive on a smooth road. The driving speed was kept around 50km to 60km per hour. These conditions

were taken into account to ensure that each user has the same environment for testing and testing is not effected by other variables such as driving speed and walking pace etc.

4.1.2 Randomization

Randomization refers to randomly assigning objects or individuals to an experimental group. Randomization prevents from involving potential biases or judgements. To ensure randomization users were given different sequence of activities. Sequence of keyboards to appear in one activity was also randomized. Moreover the phrases appear randomly for each keyboard.

4.2 ANOVA

For data analysis ANOVA is used. *ANOVA* refers to Analysis of Variance which is a method to compare multiple means to find out significant differences among them. To analyse which keyboard works better ANOVA analysis is applied on mistakes and time data logged by the application.

4.2.1 Mistakes Data

ANOVA is applied on each activity mistakes data separately to determine the performance of each keyboard in a given activity. Table 4.1 contains data for multiple comparisons in sitting activity. Results showed that Sitting activity data didn't show any significant difference across keyboards while walking and travelling data showed significant differences. We further applied multiple comparisons on travelling and walking data to see which keyboard's performance differ significantly. Table 4.2 shows the results of

multiple comparisons of mistakes data in walking activity. With 5% level of significance we can see Landscape has small percentage values compare to other keyboards such as 3x4 T9, Swiftkey and Thickbuttons. While all other keyboards have higher values specifying no performance difference among them. Similarly Table 4.3 shows data of multiple comparisons of mistakes data in travelling activity and Landscape keyboard has significantly smaller values than other keyboards. Among all keyboards Qwerty Landscape keyboard has significant performance difference from other keyboards for both activities i.e. travelling and walking. Multiple comparisons only show that the difference exists but to ensure that this different is positive we analysed box-plots of data and the results showed that Landscape performed better than other keyboards for these two activities.

	3x4 T9	Landscape	Portrait	Swiftkey
Landscape	1.00000	-	-	-
Portrait	1.00000	1.00000	-	-
Swiftkey	1.00000	1.00000	1.00000	-
Thickbuttons	1.00000	1.00000	1.00000	1.00000

Table 4.1: Multiple comparisons of mistakes in sitting activity

4.2.2 Time Data

Multiple comparisons of total time taken by users in each activity is given in tables 4.4, 4.5 and 4.6. In all the activities, 3x4 T9 keyboard has significant time difference with other keyboards. From the box-plots of time data given in Figures 4.4, 4.5, and 4.6, it is observed that 3x4 T9 took more time. Thus, it can be concluded that 3x4 T9 performed worst in time among all keyboards

	3x4 T9	Landscape	Portrait	Swiftkey
Landscape	0.012	-	-	-
Portrait	0.258	0.704	-	-
Swiftkey	1.000	0.099	0.773	-
Thickbuttons	0.887	0.016	0.887	1.000

Table 4.2: Multiple comparisons of mistakes in walking activity

	3x4 T9	Landscape	Portrait	Swiftkey
Landscape	0.00052	-	-	-
Portrait	0.43735	0.43735	-	-
Swiftkey	0.83842	0.01361	0.83842	-
Thickbuttons	1.00000	0.00204	0.43735	1.00000

Table 4.3: Multiple comparisons of mistakes in travelling activity

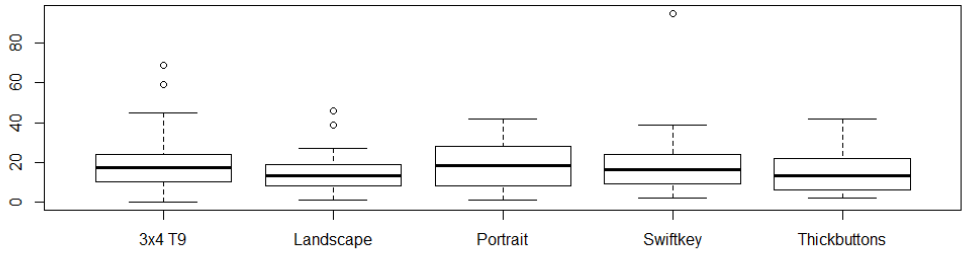

Figure 4.1: Boxplot of mistakes data for sitting activity

for each activity.

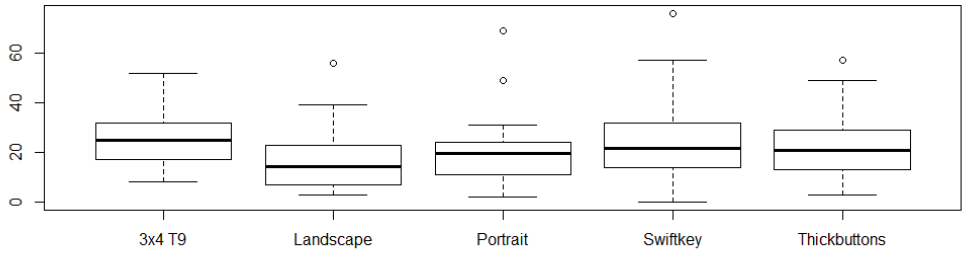

Figure 4.2: Boxplot of mistakes data for walking activity

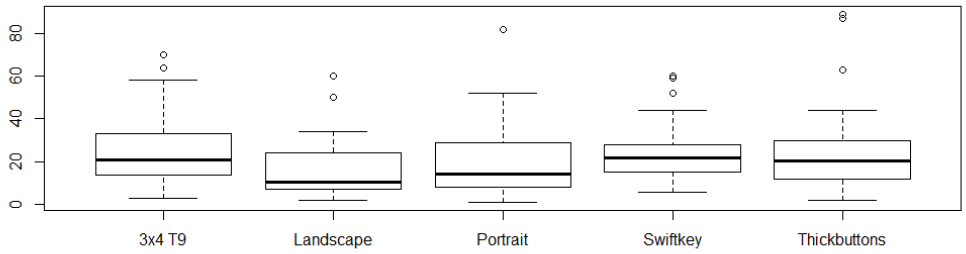

Figure 4.3: Boxplot of mistakes data for travelling activity

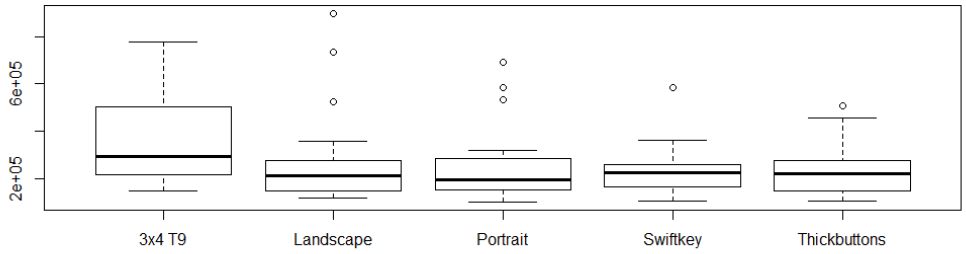

Figure 4.4: Boxplot of time data for sitting activity

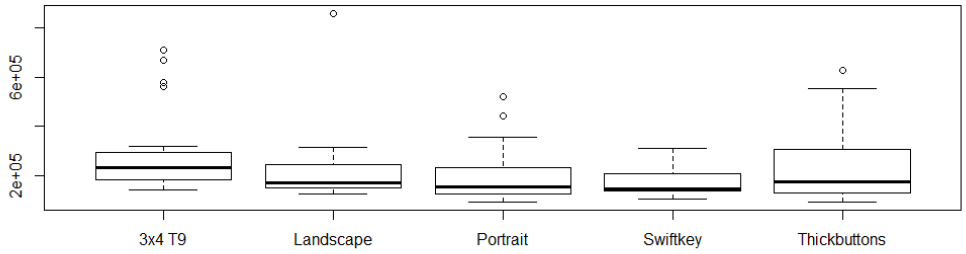

Figure 4.5: Boxplot of time data for walking activity

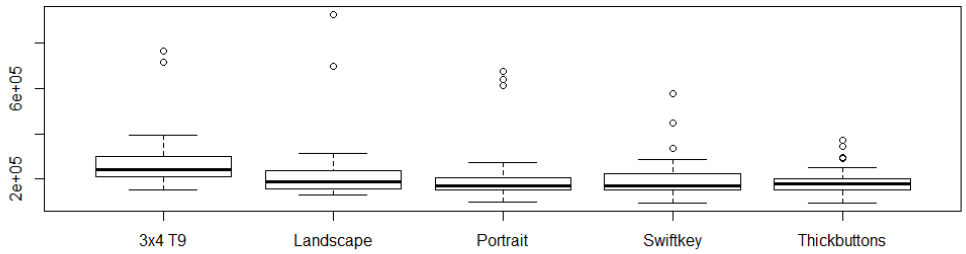

Figure 4.6: Boxplot of time data for travelling activity

	3x4 T9	Landscape	Portrait	Swiftkey
Landscape	0.05351	-	-	-
Portrait	0.00150	1.00000	-	-
Swiftkey	0.00025	1.00000	1.00000	-
Thickbuttons	0.00015	1.00000	1.00000	1.00000

Table 4.4: Multiple comparisons of time data in sitting activity

4.3 Usability Survey

Survey questionnaire was taken from Lund's Usability Questionnaire [23]. This questionnaire is divided into four categories of questions usefulness,

	3x4 T9	Landscape	Portrait	Swiftkey
Landscape	0.11060	-	-	-
Portrait	0.04429	0.94666	-	-
Swiftkey	0.00023	0.79412	0.94666	-
Thickbuttons	0.83816	0.94666	0.43914	0.11060

Table 4.5: Multiple comparisons of time data in walking activity

	3x4 T9	Landscape	Portrait	Swiftkey
Landscape	0.483	-	-	-
Portrait	0.018	1.000	-	-
Swiftkey	0.011	1.000	1.000	-
Thickbuttons	0.011	0.989	1.000	1.000

Table 4.6: Multiple comparisons of time data in travelling activity

ease of use, ease of learning, satisfaction. Overall scores of survey data is shown in Figures 4.7, 4.8, and 4.9 for sitting, walking and travelling activity classes respectively. Portrait keyboard was liked most by participants for sitting activity while for walking and travelling activities Portrait and Swiftkey keyboards both obtained good scores. To analyse it further we looked into the scores for each category type. Table 4.7 contains score of each category type for each keyboard in all activities. For sitting activity, Portrait keyboard outperforms other keyboards. For travelling activity Swiftkey received higher score for usefulness and ease of use categories while Portrait received higher scores in other two categories. For walking activity Swiftkey outperforms other keyboards except for ease of learning category in which

Portrait received slightly better scores.

From survey data we can see that participants favoured Portrait and Swiftkey keyboards while the mistakes and time data showed different results.

Category type	Activity	Portrait	Landscape	3x4 T9	Swiftkey	Thickbuttons
Usefulness	Travelling	316	274	202	336	242
Ease of use	Travelling	496	446	328	510	450
Ease of learning	Travelling	186	178	136	176	168
Satisfaction	Travelling	332	260	198	294	246
Usefulness	Walking	318	310	124	348	208
Ease of use	Walking	500	498	404	508	382
Ease of learning	Walking	192	192	148	188	150
Satisfaction	Walking	332	280	196	350	200
Usefulness	Sitting	348	298	246	310	224
Ease of use	Sitting	544	494	358	488	368
Ease of learning	Sitting	210	184	130	184	138
Satisfaction	Sitting	372	292	186	282	216

Table 4.7: Survey data

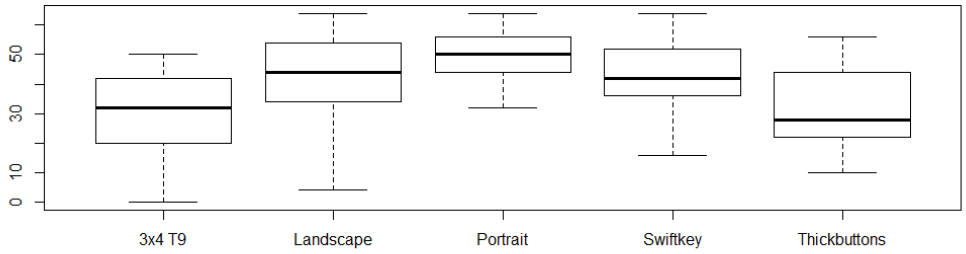

Figure 4.7: Boxplot of Survey data for sitting activity

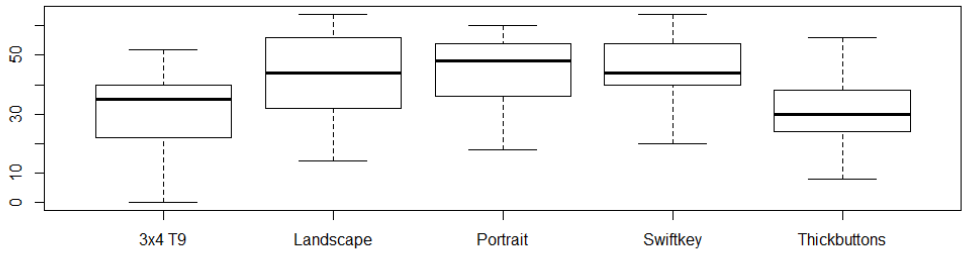

Figure 4.8: Boxplot of Survey data for walking activity

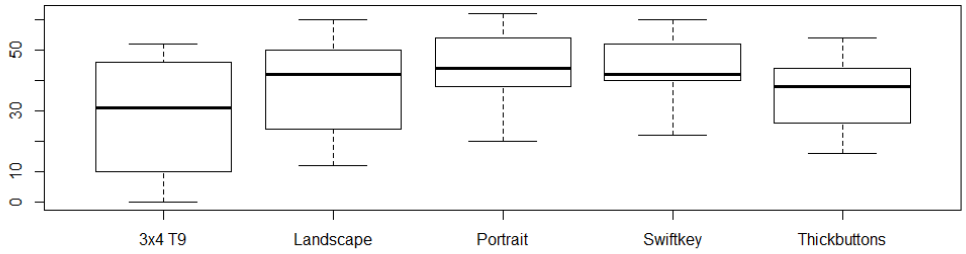

Figure 4.9: Boxplot of Survey data for travelling activity

Chapter 5

Conclusion and Future Work

5.1 Overview

Virtual keyboards have gained attention of researchers for many years. The result of this ongoing research is improved better performing virtual keyboards available in market. We aimed to find out performance of various virtual keyboards in different activities. For this purpose we chose 5 keyboards and performed a survey with 30 participants.

5.2 Conclusion

Android Portrait keyboard outperforms other keyboards in sitting activity. This result was consistent among survey data and the data logged by our application. As Android Portrait is used most commonly by users and sitting is the native position in which typing is the primary task with no other distraction on mind, this can be the reason why participants favoured Portrait keyboard. While in other activities secondary tasks occlude human mind such as in walking activity primary task can be typing but walking without

getting bump into something is a secondary task. Thus a simple Portrait keyboard doesn't perform similar in those activities. Our application logged data showed that Android Landscape keyboard performed well with lesser mistakes made in these activities. Android Landscape utilize full length of Mobile phones for keyboard keys hence keys are bigger in size. Feedback from participants showed that Swiftkey Keyboard was liked by the users in usefulness and ease of use categories. We performed testing on Swiftkey keyboard in Portrait layout. We believe if Swiftkey is used in landscape layout, it can outperform Android Landscape keyboard. 3x4 T9 performed worst in all cases. Thickbuttons keyboard was on average.

5.3 Future Work

Swiftkey keyboard can be tested in landscape layout in walking and travelling activities to see if performance improved. During our testing, participants face most problem in typing while travelling. Cumulative result for Mistakes was also highest for travelling activity. To the best of our knowledge there isn't any significant research been done for typing while travelling. We think this is a new challenge for researchers of virtual keyboards. Furthermore we only tested 5 keyboards. There are a number of other keyboards in market and can be tested for walking and travelling activity classes.

Appendices

Appendix A

Usability Survey on Virtual Keyboards

Usability Survey On Virtual Keyboards Of Mobile Devices

Name:_____

Age:_____ **Profession:**_____

Gender: ☐ Male ☐ Female

Experience of touch screen typing: ☐ Yes ☐ No

Which touch screen keyboard are you using (you can select multiple)?

☐ Qwerty Portrait ☐ 3x4 T9 ☐ Swiftkey

☐ Swype ☐ Qwerty landscape ☐ Thickbuttons

☐ Other, please specify _____

How long have you been using touch screen keyboard? _____

Please select your activity

☐ Sitting ☐ Walking ☐ Traveling

Kindly provide feedback for the keyboards you have used in testing application.

Android Portrait

		Strongly Agree	Agree	Neutral	Disagree	Strongly Disagree
1.	It helps me be more productive.	o	o	o	o	o
2.	It saves me time when I use it.	o	o	o	o	o
3.	It requires the fewest steps possible to accomplish what I want to do with it.	o	o	o	o	o
4.	I can use it without written instructions.	o	o	o	o	o
5.	I can recover from mistakes quickly and easily.	o	o	o	o	o
6.	I learned to use it quickly.	o	o	o	o	o
7.	I am satisfied with it.	o	o	o	o	o
8.	I feel I need to have it.	o	o	o	o	o

Android Landscape

		Strongly Agree	Agree	Neutral	Disagree	Strongly Disagree
1.	It helps me be more productive.	o	o	o	o	o
2.	It saves me time when I use it.	o	o	o	o	o
3.	It requires the fewest steps possible to accomplish what I want to do with it.	o	o	o	o	o
4.	I can use it without written instructions.	o	o	o	o	o
5.	I can recover from mistakes quickly and easily.	o	o	o	o	o
6.	I learned to use it quickly.	o	o	o	o	o
7.	I am satisfied with it.	o	o	o	o	o
8.	I feel I need to have it.	o	o	o	o	o

Android 3x4 T9

		Strongly Agree	Agree	Neutral	Disagree	Strongly Disagree
1.	It helps me be more productive.	o	o	o	o	o
2.	It saves me time when I use it.	o	o	o	o	o
3.	It requires the fewest steps possible to accomplish what I want to do with it.	o	o	o	o	o
4.	I can use it without written instructions.	o	o	o	o	o
5.	I can recover from mistakes quickly and easily.	o	o	o	o	o
6.	I learned to use it quickly.	o	o	o	o	o
7.	I am satisfied with it.	o	o	o	o	o
8.	I feel I need to have it.	o	o	o	o	o

Swiftkey Keyboard

		Strongly Agree	Agree	Neutral	Disagree	Strongly Disagree
1.	It helps me be more productive.	o	o	o	o	o
2.	It saves me time when I use it.	o	o	o	o	o
3.	It requires the fewest steps possible to accomplish what I want to do with it.	o	o	o	o	o
4.	I can use it without written instructions.	o	o	o	o	o
5.	I can recover from mistakes quickly and easily.	o	o	o	o	o
6.	I learned to use it quickly.	o	o	o	o	o
7.	I am satisfied with it.	o	o	o	o	o
8.	I feel I need to have it.	o	o	o	o	o

Thickbuttons Keyboard

		Strongly Agree	Agree	Neutral	Disagree	Strongly Disagree
1.	It helps me be more productive.	o	o	o	o	o
2.	It saves me time when I use it.	o	o	o	o	o
3.	It requires the fewest steps possible to accomplish what I want to do with it.	o	o	o	o	o
4.	I can use it without written instructions.	o	o	o	o	o
5.	I can recover from mistakes quickly and easily.	o	o	o	o	o
6.	I learned to use it quickly.	o	o	o	o	o
7.	I am satisfied with it.	o	o	o	o	o
8.	I feel I need to have it.	o	o	o	o	o

Appendix B

Measuring Usability with the USE Questionnaire

Measuring Usability with the USE Questionnaire

Lund, Arnold (Arnie) M.

2008-09-05

USEFULNESS		Include	Neutral	Remove
1.	It helps me be more effective.			
2.	It helps me be more productive.			
3.	It is useful.			
4.	It gives me more control over the activities in my life.			
5.	It makes the things I want to accomplish easier to get done.			
6.	It saves me time when I use it.			
7.	It meets my needs.			
8.	It does everything I would expect it to do.			
EASE OF USE				
9.	It is easy to use.			
10.	It is simple to use.			
11.	It is user friendly.			
12.	It requires the fewest steps possible to accomplish what I want to do with it.			
13.	It is flexible.			
14.	Using it is effortless.			
15.	I can use it without written instructions.			
16.	I don't notice any inconsistencies as I use it.			
17.	Both occasional and regular users would like it.			
18.	I can recover from mistakes quickly and easily.			
19.	I can use it successfully every time.			
EASE OF LEARNING				
20.	I learned to use it quickly.			
21.	I easily remember how to use it.			
22.	It is easy to learn to use it.			
23.	I quickly became skillful with it.			
SATISFACTION				
24.	I am satisfied with it.			
25.	I would recommend it to a friend.			

26.	It is fun to use.			
27.	It works the way I want it to work.			
28.	It is wonderful.			
29.	I feel I need to have it.			
30.	It is pleasant to use.			

Appendix C

Android Application for Text Entry Code

C.1 LoginActivity.java

```
package com.example.typingwalk;

import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.text.TextUtils;
import android.view.KeyEvent;
import android.view.View;
import android.view.inputmethod.EditorInfo;
import android.widget.EditText;
import android.widget.TextView;

/**
 * Activity which displays a login screen to the user
 */
public class LoginActivity extends Activity {

      // Values for email and password at the time of the login attempt.
      private String mName;
      private String mAge;

      // UI references.
      private EditText mNameView;
      private EditText mAgeView;

      @Override
      protected void onCreate(Bundle savedInstanceState) {
            super.onCreate(savedInstanceState);

            setContentView(R.layout.activity_login);

            // Set up the login form.
            mNameView = (EditText) findViewById(R.id.name);

            mAgeView = (EditText) findViewById(R.id.age);
            mAgeView
                        .setOnEditorActionListener(new
TextView.OnEditorActionListener() {
                              @Override
                              public boolean onEditorAction(TextView
textView, int id,
                                          KeyEvent keyEvent) {
                                    if (id == R.id.login || id ==
EditorInfo.IME_NULL) {
                                          attemptLogin();
                                          return true;
                                    }
                                    return false;
                              }
                        });


            findViewById(R.id.sign_in_button).setOnClickListener(
                        new View.OnClickListener() {
```

```
                              @Override
                              public void onClick(View view) {
                                    attemptLogin();
                              }
                        });
      }

      /**
       * Attempts to sign in or register the account specified by the
login form.
       * If there are form errors (invalid email, missing fields, etc.),
the
       * errors are presented and no actual login attempt is made.
       */
      public void attemptLogin() {

            // Reset errors.
            mNameView.setError(null);
            mAgeView.setError(null);

            // Store values at the time of the login attempt.
            mName = mNameView.getText().toString();
            mAge = mAgeView.getText().toString();

            boolean cancel = false;
            View focusView = null;

            // Check for a valid age
            if (TextUtils.isEmpty(mAge)) {

      mAgeView.setError(getString(R.string.error_field_required));
                  focusView = mAgeView;
                  cancel = true;
            }

            // Check for a valid name
            if (TextUtils.isEmpty(mName)) {

      mNameView.setError(getString(R.string.error_field_required));
                  focusView = mNameView;
                  cancel = true;
            }

            if (cancel) {
                  // There was an error; don't attempt login and focus the
first
                  // form field with an error.
                  focusView.requestFocus();
            } else {
                  Intent intent = new Intent(this,
ActivitySelector.class);
                  intent.putExtra("name", mName + mAge);
                  startActivity(intent);
                  finish();
```

```
                }
            }

}
```

C.2 ActivitySelector.java

```
package com.example.typingwalk;

import android.support.v7.app.ActionBarActivity;
import android.support.v7.app.ActionBar;
import android.support.v4.app.Fragment;
import android.content.Intent;
import android.os.Bundle;
import android.view.LayoutInflater;
import android.view.Menu;
import android.view.MenuItem;
import android.view.View;
import android.view.ViewGroup;
import android.widget.RadioButton;
import android.os.Build;

public class ActivitySelector extends ActionBarActivity {

	private int ActivityType;
	private String userName;
	@Override
	protected void onCreate(Bundle savedInstanceState) {
		super.onCreate(savedInstanceState);
		setContentView(R.layout.activity_selector);
		Intent intent = getIntent();
		userName = intent.getStringExtra("name");

	}

	public void onRadioButtonClicked(View view) {
	    // Is the button now checked?
	    boolean checked = ((RadioButton) view).isChecked();

	    // Check which radio button was clicked
	    switch(view.getId()) {
	        case R.id.radSitting:
	            if (checked)
	                ActivityType = 0;
	            break;
	        case R.id.radWalking:
	            if (checked)
	                ActivityType = 1;
	            break;
	        case R.id.radtraveling:
	            if (checked)
	                ActivityType = 2;
	            break;
	    }
	}

	public void onStartTesting(View view){
		Intent intent = new Intent(this, MainActivity.class);
		intent.putExtra("activitytype", ActivityType);
		intent.putExtra("name", userName);
		startActivity(intent);
```

```
        }

    }
```

C.3 MainActivity.java

```
package com.example.typingwalk;

import java.io.Console;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.PrintStream;
import java.util.ArrayList;
import java.util.Calendar;
import java.util.List;
import java.util.Random;

import android.text.Editable;
import android.text.SpannableString;
import android.text.TextWatcher;
import android.util.Log;
import android.app.Activity;
import android.app.AlertDialog;
import android.content.Context;
import android.content.DialogInterface;
import android.content.Intent;
import android.content.res.Configuration;
import android.hardware.Sensor;
import android.hardware.SensorEvent;
import android.hardware.SensorEventListener;
import android.hardware.SensorManager;
import android.os.Bundle;
import android.os.Environment;
import android.os.Message;
import android.view.Menu;
import android.view.MenuItem;
import android.view.View;
import android.view.inputmethod.InputMethodInfo;
import android.view.inputmethod.InputMethodManager;
import android.widget.EditText;
import android.widget.LinearLayout;
import android.widget.TextView;

public class MainActivity extends Activity /*implements
SensorEventListener*/
{
      private String[] mWords;
      private TextView mTop, mSecTop, mLast, mSecLast;
      private TextView mTypingText;
      private EditText mInputText;
      private int mStartPoint = 0;
      private int[] mTexts;
      private String currWord;
      private String typingWord;
      private SensorManager manager = null;
      private Sensor accelerometer = null;
      private boolean bTyping = false;
      private String letter;
```

```
	private String intendedletter;
	private String strSensorData;
	private String[] arrPhraseSet;
	private ArrayList<Integer> keyboardsInd = new ArrayList<Integer>();
	private Random rand = new Random();
	private AlertDialog chooseKeyboard;
	private int ActivityType = 0;
	private String activityFile = "";
	private ArrayList<Integer> phraseInd = new ArrayList<Integer>();
	private ArrayList<Integer> pangramInd = new ArrayList<Integer>();
	private long wordStartTime = 0;
	private Calendar calendar = Calendar.getInstance();
	private int currKeyboard;
	private String Keyboardlogging = "";
	private int numberOfMistakes = 0;
	private String userName;
	private Activity _that = this;
	InputMethodManager imeManager;

	@Override
	protected void onCreate(Bundle savedInstanceState) {
		super.onCreate(savedInstanceState);
		Intent intent = getIntent();
		ActivityType = intent.getIntExtra("activitytype", 0);
		userName = intent.getStringExtra("name");

		imeManager = (InputMethodManager)
getApplicationContext().getSystemService(INPUT_METHOD_SERVICE);
		String keyboardtype = ChooseNextKeyboard();
		chooseKeyboard = new AlertDialog.Builder(this).create();
		chooseKeyboard.setTitle("Next Keyboard to Choose");
		chooseKeyboard.setMessage(keyboardtype);
		chooseKeyboard.setButton(DialogInterface.BUTTON_POSITIVE,
"OK", new DialogInterface.OnClickListener() {
			public void onClick(DialogInterface dialog,int id) {
				imeManager.showInputMethodPicker();
				imeManager.showInputMethodPicker();
			}});
		chooseKeyboard.show();


		setContentView(R.layout.fragment_main);



		LoadPhrases();



		mTop = (TextView) findViewById(R.id.topTextview);
		mSecTop = (TextView) findViewById(R.id.SecondTopTextView);
		mTypingText = (TextView) findViewById(R.id.TypingTextView);
		mLast = (TextView) findViewById(R.id.LastTestView);
		mSecLast = (TextView) findViewById(R.id.secondLastTextView);
```

```
            mInputText = (EditText) findViewById(R.id.InputEditText);

            mTexts = new int[]{
                        R.id.topTextview,
                        R.id.SecondTopTextView,
                        R.id.TypingTextView,
                        R.id.secondLastTextView,
                        R.id.LastTestView,
            };

            mTop.setText(mWords[0]);
            mSecTop.setText(mWords[1]);
            mTypingText.setText(mWords[2]);
            mSecLast.setText(mWords[3]);
            mLast.setText(mWords[4]);
            currWord = mWords[2];
            typingWord = "";
            intendedletter = "";
            mStartPoint = 0;

            mInputText.addTextChangedListener(new TextWatcher() {

                  @Override
                  public void onTextChanged(CharSequence s, int start, int
before, int count) {
                        if(wordStartTime == 0){
                              calendar = Calendar.getInstance();
                              wordStartTime = calendar.getTimeInMillis();
                              Log.e("time start: ",
Long.toString(wordStartTime));
                        }
                        if(bTyping == false){
                              bTyping = true;
                        }
                        String str = s.subSequence(start,
start+count).toString();
                        letter = str.toLowerCase();
                        if(start+count <= currWord.length() ){
                              intendedletter = currWord.substring(start,
start+count).toLowerCase();
                        }

                        typingWord += str;
                        if(s.toString().contains("\n"))
                        {
                              calendar = Calendar.getInstance();
                              Long temp = calendar.getTimeInMillis();
                              Log.e("end time: ", Long.toString(temp));
                              Long timetaken = calendar.getTimeInMillis()
- wordStartTime;
                              Keyboardlogging += Long.toString(timetaken)
+ "," +
                                          GetKeyboardName(currKeyboard) +
"," + currWord + "," + s.toString();
```

```
                              numberOfMistakes = 0;
                              wordStartTime = 0;
                              TextView tv;
                              mStartPoint++;
                              if(mStartPoint + 1 > arrPhraseSet.length){
                                    writeStringToTextFile(Keyboardlogging ,
activityFile);
                                    if(keyboardsInd.size() == 5){
                                          _that.finish();
                                          return;
                                    }
                                    String keyboardtype =
ChooseNextKeyboard();

      chooseKeyboard.setMessage(keyboardtype);
                                    chooseKeyboard.show();
                                    mStartPoint = 0;
                                    LoadPhrases();
                              }
                              int index = mStartPoint;
                              mInputText.setText("");
                              for(int i=0; i<5; i++)
                              {
                                    tv=(TextView) findViewById(mTexts[i]);
                                    if(index <= mWords.length-1){
                                          tv.setText(mWords[index]);
                                          if(i == 2)
                                          {
                                                currWord = mWords[index];
                                          }
                                          //break;
                                    }
                                    else
                                          tv.setText("");
                                    index++;
                              }

                        }

                  }
                  @Override
                  public void beforeTextChanged(CharSequence s, int start,
int count,
                              int after) {
                        // TODO Auto-generated method stub
                        Log.w("testing app
beforetextchange:",s.toString());

                  }

                  @Override
                  public void afterTextChanged(Editable s) {

                  }
```

```
            });


      }

      private void writeStringToTextFile(String s, String f)
      {
            File sdCard = Environment.getExternalStorageDirectory();
            File dir = new File (sdCard.getAbsolutePath());
            dir.mkdirs();
            File file = new File(dir, f);
            if(!(file.exists())){
             try{
                 file.createNewFile();}
             catch(Exception e){
                return;}}
            try
            {
                FileOutputStream f1 = new FileOutputStream(file,true);
//True = Append to file, false = Overwrite
                PrintStream p = new PrintStream(f1);
                p.print(s);
                p.close();
                f1.flush();
                f1.close();
            }
            catch (FileNotFoundException e)
            {
            }
            catch (IOException e)
            {
            }
      }

      @Override
      public boolean onCreateOptionsMenu(Menu menu) {

            // Inflate the menu; this adds items to the action bar if it
is present.
            getMenuInflater().inflate(R.menu.main, menu);
            return true;
      }

      @Override
      public boolean onOptionsItemSelected(MenuItem item) {
            int id = item.getItemId();
            switch(id){
            case R.id.action_settings:
                  imeManager.showInputMethodPicker();
                  return true;
            case R.id.action_keyboard:
                  chooseKeyboard.show();
                  return true;
            }
```

```
            return super.onOptionsItemSelected(item);
      }

      private void LoadPhrases(){

            String[] phrases =
getResources().getStringArray(R.array.phrases_sitting);
            String[] pangrams =
getResources().getStringArray(R.array.pangrams_sitting);

            switch(ActivityType){
            case 0:
                  activityFile = userName + "_sitting.txt";
                  break;
            case 1:
                  activityFile = userName + "_walking.txt";
                  phrases =
getResources().getStringArray(R.array.phrases_walking);
                  pangrams =
getResources().getStringArray(R.array.pangrams_walking);
                  break;
            case 2:
                  activityFile = userName + "_traveling.txt";
                  phrases =
getResources().getStringArray(R.array.phrases_traveling);
                  pangrams =
getResources().getStringArray(R.array.pangrams_traveling);
                  break;
            }

            int nCount = 1;
            int nIndex = GetNextIndex(pangramInd, 5);
            pangramInd.add(nIndex);
            String words = pangrams[nIndex];
            while(nCount < 10){
                  nIndex = GetNextIndex(phraseInd, 45);
                  phraseInd.add(nIndex);
                  words += " " + phrases[nIndex];
                  nCount++;
            }
            arrPhraseSet = words.split(" ");

            int size = arrPhraseSet.length + 2;
            mWords = new String[size];
            for(int i = 0; i < size; i++)
            {
                  if(i<=1)
                        mWords[i] = "";
                  else
                        mWords[i] = arrPhraseSet[i-2].trim();
            }
      }
```

```
	private String ChooseNextKeyboard(){
		int currInd = GetNextIndex(keyboardsInd, 5);
		keyboardsInd.add(currInd);
		currKeyboard = currInd;
		return GetKeyboardName(currInd);
	}

	private int GetNextIndex(ArrayList<Integer> list, int limit){
		int currInd = rand.nextInt(limit);
		Object obj = new Object();
		obj = currInd;
		while(list.contains(obj)){
			currInd = rand.nextInt(limit);
			obj = currInd;
		}
		return currInd;
	}

	private String GetKeyboardName(int index){
		switch(index){
		case 0:
			return "Portrait";
		case 1:
			return "Landscape";
		case 2:
			return "3x4 T9";
		case 3:
			return "SwiftKey";
		case 4:
			return "Thickbuttons";
		}
		return "Portrait";
	}

	@Override
	public void onConfigurationChanged(Configuration newConfig) {
		super.onConfigurationChanged(newConfig);
		LinearLayout bottomcontainer = (LinearLayout)
findViewById(R.id.bottomContainer);
		if(newConfig.orientation ==
Configuration.ORIENTATION_LANDSCAPE)
		{

			bottomcontainer.setVisibility(View.GONE);
		}
		else
		{
			bottomcontainer.setVisibility(View.VISIBLE);
		}
	}

}
```

REFERENCES

[1] Autocorrect that actually works. [Online; accessed 17-06-2015].

[2] Keyboard that enlarges buttons. [Online; accessed 17-06-2015].

[3] Type fast, swype faster. [Online; accessed 17-06-2015].

[4] Google apis for android. 2015. [Online; accessed 17-06-2015].

[5] Number of apps available in leading app stores as of may 2015. 2015. [Online; accessed 17-06-2015].

[6] Number of smartphones sold to end users worldwide from 2007 to 2014 (in million units). 2015. [Online; accessed 17-06-2015].

[7] Joanna Bergstrom-Lehtovirta, Antti Oulasvirta, and Stephen Brewster. The effects of walking speed on target acquisition on a touchscreen interface. In *Proceedings of the 13th International Conference on Human Computer Interaction with Mobile Devices and Services*, pages 143–146. ACM, 2011.

[8] Gerald Bieber, Jörg Voskamp, and Bodo Urban. Activity recognition for everyday life on mobile phones. In *Universal Access in Human-Computer Interaction. Intelligent and Ubiquitous Interaction Environments*, pages 289–296. Springer, 2009.

[9] Brad Bourque. 5 android keyboards that will have you texting faster than a 13-year-old. 2015. [Online; accessed 21-06-2015].

[10] Tomas Brezmes, Juan-Luis Gorricho, and Josep Cotrina. Activity recognition from accelerometer data on a mobile phone. In *Distributed computing, artificial intelligence, bioinformatics, soft computing, and ambient assisted living*, pages 796–799. Springer, 2009.

[11] PA Condado, R Godinho, M Zacarias, and FG Lobo. Easywrite: A touch-based entry method for mobile devices. In *Proceedings of the 13th IFIP TC13 International Conference on Human-Computer Interaction (INTERACT 2011), Workshop on Mobile Accessibility (MOBACC 2011), Lisbon, Portugal*, pages 1–8, 2011.

[12] Barnan Das, Adriana M Seelye, Brian L Thomas, Diane J Cook, Larry B Holder, and Maureen Schmitter-Edgecombe. Using smart phones for context-aware prompting in smart environments. In *Consumer Communications and Networking Conference (CCNC), 2012 IEEE*, pages 399–403. IEEE, 2012.

[13] Trinh Minh Tri Do, Jan Blom, and Daniel Gatica-Perez. Smartphone usage in the wild: a large-scale analysis of applications and context. In *Proceedings of the 13th international conference on multimodal interfaces*, pages 353–360. ACM, 2011.

[14] Mark D Dunlop, Naveen Durga, Sunil Motaparti, Prima Dona, and Varun Medapuram. Qwerth: an optimized semi-ambiguous keyboard design. In *Proceedings of the 14th international conference on Human-computer interaction with mobile devices and services companion*, pages 23–28. ACM, 2012.

[15] Hossein Falaki, Ratul Mahajan, Srikanth Kandula, Dimitrios Lymberopoulos, Ramesh Govindan, and Deborah Estrin. Diversity in smartphone usage. In *Proceedings of the 8th international conference on Mobile systems, applications, and services*, pages 179–194. ACM, 2010.

[16] Jordan Frank, Shie Mannor, and Doina Precup. Activity recognition with mobile phones. In *Machine Learning and Knowledge Discovery in Databases*, pages 630–633. Springer, 2011.

[17] Mayank Goel, Leah Findlater, and Jacob Wobbrock. Walktype: using accelerometer data to accomodate situational impairments in mobile touch screen text entry. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 2687–2696. ACM, 2012.

[18] Joshua Goodman, Gina Venolia, Keith Steury, and Chauncey Parker. Language modeling for soft keyboards. In *Proceedings of the 7th international conference on Intelligent user interfaces*, pages 194–195. ACM, 2002.

[19] Asela Gunawardana, Tim Paek, and Christopher Meek. Usability guided key-target resizing for soft keyboards. In *Proceedings of the 15th international conference on Intelligent user interfaces*, pages 111–118. ACM, 2010.

[20] Niels Henze, Enrico Rukzio, and Susanne Boll. Observational and experimental investigation of typing behaviour using virtual keyboards for mobile devices. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 2659–2668. ACM, 2012.

[21] Yu-Jin Hong, Ig-Jae Kim, Sang Chul Ahn, and Hyoung-Gon Kim. Mobile health monitoring system based on activity recognition using ac-

celerometer. *Simulation Modelling Practice and Theory*, 18(4):446–455, 2010.

[22] Jennifer R Kwapisz, Gary M Weiss, and Samuel A Moore. Activity recognition using cell phone accelerometers. *ACM SigKDD Explorations Newsletter*, 12(2):74–82, 2011.

[23] Arnold M Lund. Measuring usability with the use questionnaire. *Usability interface*, 8(2):3–6, 2001.

[24] I Scott MacKenzie and R William Soukoreff. Phrase sets for evaluating text entry techniques. In *CHI'03 extended abstracts on Human factors in computing systems*, pages 754–755. ACM, 2003.

[25] I Scott MacKenzie and Shawn X Zhang. The design and evaluation of a high-performance soft keyboard. In *Proceedings of the SIGCHI conference on Human Factors in Computing Systems*, pages 25–31. ACM, 1999.

[26] Sachi Mizobuchi, Mark Chignell, and David Newton. Mobile text entry: relationship between walking speed and text input task difficulty. In *Proceedings of the 7th international conference on Human computer interaction with mobile devices & services*, pages 122–128. ACM, 2005.

[27] Alexander Ng and Stephen Brewster. The relationship between encumbrance and walking speed on mobile interactions. In *CHI'13 Extended Abstracts on Human Factors in Computing Systems*, pages 1359–1364. ACM, 2013.

[28] Alexander Ng, Stephen A Brewster, and John Williamson. The impact of encumbrance on mobile interactions. In *Human-Computer Interaction–INTERACT 2013*, pages 92–109. Springer, 2013.

[29] Hugo Nicolau, Tiago Guerreiro, Joaquim Jorge, and Daniel Gonçalves. Mobile touchscreen user interfaces: bridging the gap between motor-impaired and able-bodied users. *Universal Access in the Information Society*, 13(3):303–313, 2014.

[30] Antti Oulasvirta, Anna Reichel, Wenbin Li, Yan Zhang, Myroslav Bachynskyi, Keith Vertanen, and Per Ola Kristensson. Improving two-thumb text entry on touchscreen devices. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 2765–2774. ACM, 2013.

[31] Tom Page. Usability of text input interfaces in smartphones. *Journal of Design Research*, 11(1):39–56, 2013.

[32] Ahmad Rahmati, Clayton Shepard, and Lin Zhong. Noshake: Content stabilization for shaking screens of mobile devices. In *Pervasive Computing and Communications, 2009. PerCom 2009. IEEE International Conference on*, pages 1–6. IEEE, 2009.

[33] Dmitry Rudchenko, Tim Paek, and Eric Badger. Text text revolution: a game that improves text entry on mobile touchscreen keyboards. In *Pervasive Computing*, pages 206–213. Springer, 2011.

[34] Bastian Schildbach and Enrico Rukzio. Investigating selection and reading performance on a mobile phone while walking. In *Proceedings of the 12th international conference on Human computer interaction with mobile devices and services*, pages 93–102. ACM, 2010.

[35] Andrew Sears. Improving touchscreen keyboards: design issues and a comparison with other devices. *Interacting with computers*, 3(3):253–269, 1991.

[36] Andrew Sears, Doreen Revis, Janet Swatski, Rob Crittenden, and Ben Shneiderman. Investigating touchscreen typing: the effect of keyboard size on typing speed. *Behaviour & Information Technology*, 12(1):17–22, 1993.

[37] Ara Wagoner. The best android keyboard apps. 2015. [Online; accessed 21-06-2015].

[38] Yanshan Wang. A novel soft keyboard for touchscreen phones: Qwert. *International Journal of Human Factors and Ergonomics*, 2(4):246–261, 2013.

[39] Zhixian Yan, Vigneshwaran Subbaraju, Dipanjan Chakraborty, Archan Misra, and Karl Aberer. Energy-efficient continuous activity recognition on mobile phones: An activity-adaptive approach. In *Wearable Computers (ISWC), 2012 16th International Symposium on*, pages 17–24. Ieee, 2012.

[40] Shumin Zhai, Michael Hunter, and Barton A Smith. The metropolis keyboard-an exploration of quantitative techniques for virtual keyboard design. In *Proceedings of the 13th annual ACM symposium on User interface software and technology*, pages 119–128. ACM, 2000.