Dynamic Object Manipulation for Advanced Touch Interactions

Student: Jad Osseiran
Student Number: 20507033
Supervisor: Professor Amitava Datta

This report is submitted as partial fulfilment
of the requirements for the Honours Programme of the
School of Computer Science and Software Engineering,
The University of Western Australia,
2014

Abstract

Current standard touch interactions on familiar multitouch devices such as smartphones and tablets are limited. Using motion sensors and touch data simultaneously, new interactions can be detected to facilitate innovative ways of controlling software applications.

This Final Year Project aims to produce a user calibrated algorithm that calculates the strength of a touch on a capacitive display device. This is achieved by the use of its movements in 3D space as well as the properties of the touch. Two algorithms are designed, implemented and compared. The algorithms, named "passive" and "active", are used to record the device's movements. The optimal algorithm is then used on a population study to determine perceptions of soft, normal and hard touches. Using data from the study, the algorithms are calibrated to return a scalar touch strength value. According to the study perceptions of touch strength are universal thus allowing for a "one size fits all" algorithm.

This algorithm has the potential to improve software for scientific research, open up new user interfaces, and enrich entertainment applications.

Keywords: Honours, Software Engineering, Computer Science, Mobile, Smartphone, Tablet, Touch Strength, Algorithm
CR Categories:

Acknowledgements

The idea for this project was sparked by a discussion with University of Western Australia's Professor Amitava Datta. It has since been scoped to be a suitable final year project.

I would also like to thank Nathan Andrew Wood for the technical conversations and brainstorms throughout the implementation of this project's algorithm.

Contents

List of Figures

CHAPTER 1

Introduction

Touch is one of several ways through which users can interact with computers. However, the advent of the Apple iPhone in 2007 ushered in a major change in the way users interact with touch displays [6]. The change was innovative in bringing intuitive actions such as pinch to zoom and two finger rotate through the use of multi-touch [6]. This led to a tremendous increase in the use of touch screen devices [7]. As of 2012 the industry shipped well above one billion touchscreen units per year [7].

Most of these units are built into smartphones and tablets which use the touch screens to allow users to interact with software applications. The majority of such devices also contain motion sensors allowing them to determine their relative acceleration through the use of an accelerometer and rotation rates through the use of a gyroscope. It is common for software applications to make independent use of touch and motion features.

An example of this is a flying simulator where the device's accelerometer and gyroscope are used to control the direction of a plane. Users tap on the multitouch display to control other aspects of the flight such as speed. The motions and the touches, whilst working simultaneously, are independent of one another. It is possible to control the direction of the flight without changing the speed and vice-versa.

By combining motion and touch detection and processing them to form a single output, software can be developed that introduces new interaction interfaces between users and devices. This paper describes such an approach of creating an algorithm which uses motion and touch data to calculate the strength at which users interact with a capacitive display.

1.1 Problem Statement

Modern smartphones and tablets use capacitive displays to provide tactile multi-touch feedback [6]. Due to the nature of displays it is not possible to determine the strength of an interaction solely using capacitive touch data [6]. A possible solution is to introduce a pressure sensitive sensor under the display [15]. This is a hardware solution and does not support current capacitive devices. A software solution which makes use of the current hardware to calculate the strength of a touch is therefore an attractive prospect.

In order to create a software solution, this project aims to produce a user calibrated algorithm that calculates the strength of a touch on a capacitive display device using its movements as well as the properties of the touch. The algorithm can be implemented in software and installed on already existing hardware thus reducing the need to majorly restructure established solutions. For this reason, an algorithm based solution has the potential greatly benefit industry and academic research.

1.1.1 Scientific Research

Touch strength recognition can be extremely useful in scientific research. For example, where a material under observation is rare or cannot be physically altered; it would be possible to measure such material and model it into a software application. Once rendered users can experiment with various interactions such as 'hammering' the material with strong touches. This added interactive dimension would allow scientists to hypothesise viable physical experiments based on the simulation results.

1.1.2 Entertainment Purposes

Mobile games can greatly benefit from increased interaction with modelled objects and the numerous possibilities of touches. For instance, a door only opens if it pushed with enough strength, or a window only shatters by tapping with the right intensity at the correct location.

1.1.3 Tangible Interfaces

Responding to the strength of a touch allows for another medium of interaction with software content. A hard touch could mean a jump of two levels in a

navigation stack whereas a normal touch only moves up one level. New tangible interfaces are currently being explored in the smartphone and tablet industry such as Samsung's finger hovering implementation [17].

1.2 Overview of the Study

Chapter 2 discusses the background of the study and provides the research context. The chapter explores the workings of capacitive touch screens as well as reviewing related research about the effects of touches on device movements.

Chapter 3 describes the two algorithms implemented in this project: the passive algorithm and the active algorithm.

In Chapter 4 the passive algorithm is used to gather calibration data from a sample population for touch strength calibration. The data gathering process requires participants to interact with a mobile application using their perceptions of a soft, normal and hard touch.

The relevant results from the calibration exercise and the comparison of the two algorithm designs are presented in chapter 5. These results are then discussed and analysed in chapter 6 along with recommendations of future work. Lastly, chapter 7 concludes the paper exposing the findings of the research.

CHAPTER 2

Research Background

This chapter provides the context to better understand the limitations and advantages of capacitive displays. Capacitive displays are of particular importance to this research project as the algorithm produced is designed to run on devices which use them as their primary tool for touch interactions. In this chapter it is shown why touch information from interactions with capacitive displays alone is not enough for strength measurements.

Other parts of a smartphone or tablet aside from the display can be used to help determine the strength of a touch. Motion sensors such as accelerometers and gyroscopes are of particular use. Motion events are created from the device's accelerometer and gyroscope when the device is moved in 3D space. Two papers reviewed as part of this chapter providing insights to possible techniques to determine further information on the strength of a touch from the respective motion sensor readings.

2.1 Existing Solutions And Research

Using touch as a means of interaction with a user interface is very common in devices such as smartphones and tablets. However, registering different types of touch is still an emerging market, with large corporations such as Samsung and Apple exploring new ways to innovate touch interactions [4][15][17]. Samsung developed a technology named "Air View" to detect a finger hovering above a display [17]. This is marketed as one of the main features in their flagship smartphones [17]. Apple is investigating hardware based solutions and has patented a method to integrate pressure sensors in future displays providing force readings on interactions [15].

Reacting to the strength of a touch on capacitive displays has been implemented in the smartphone and tablet industry. An example is Apple's GarageBand iOS application where virtual percussion instruments output different sounds

depending on the strength with which the user taps the screen [1]. However, the approach which Apple has taken has not been made available to the public and serves a very specific application. This paper develops a new approach for measuring the strength of touches that can be used for a wide variety of applications requiring such functionality.

2.2 Capacitive Displays

Capacitive displays are the most prevalent types of displays in the current touch market [6]. Barnette & Omote suggested that this is largely due to the capacitive display's main advantages listed below [6]:

1. High fidelity;
2. Immediate visual responses to touches; and
3. Ability to handle multi-touch inputs.

The basic principle behind a capacitive display can be understood by looking at the simplest form of a capacitor. A capacitor is made of two conducting plates separated by an insulator [10]. In the case of a capacitive display, capacitors sit under the touchable glass screen [6][10]. These capacitors are made of transparent conductors separated by a thin insulating glass layer [6][10]. Fisher depicts the relationship between a simple capacitor and a capacitive display as shown in figure 2.1 [10].

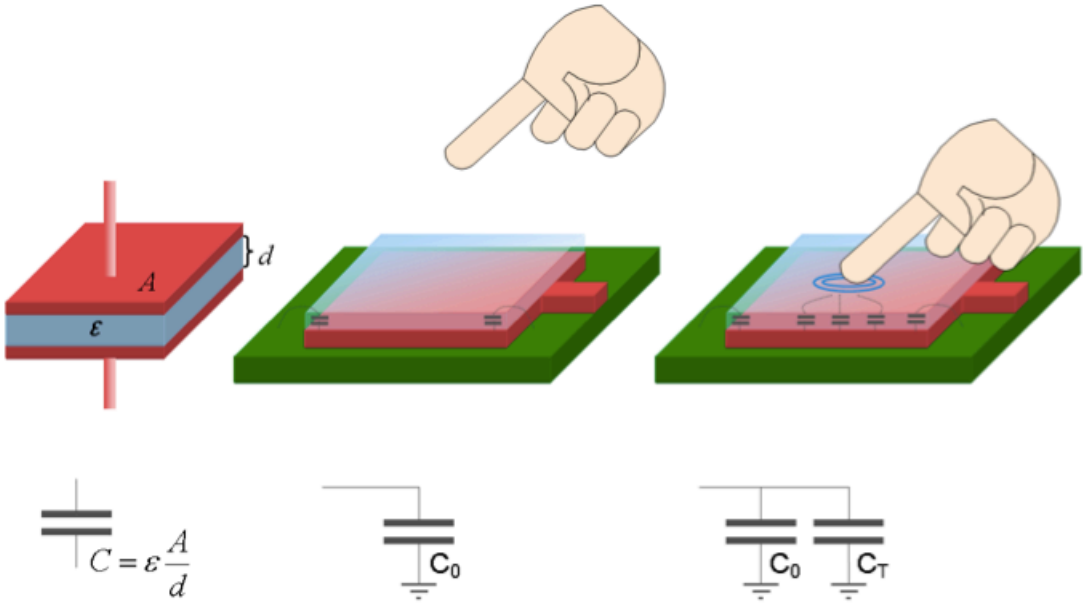

Figure 2.1: Capacitor And Capacitive Displays Link. Source: Fischer, 2010 [10]

Left: A simple capacitor; middle: untouched capacitive display with parasitic capacitance C_0; right: touched capacitive display with additional touch capacitance C_T.

The following formula describes the parameters affecting a capacitor and thus the capacitance of a capacitive display [10].

$$C = \epsilon \frac{A}{d}, \ where \ \epsilon = \epsilon_0.\epsilon_T \tag{1}$$

C is the capacitance;
ϵ_T is the relative permittivity of the insulating material between the plates;
ϵ_0 is the permittivity of free space;
A is the area common to the plates;
d is the distance between the plates.

Formula 1 shows that a change in capacitance can easily be triggered by manipulating A, d or ϵ. This property can be used to determine the capacitance change in a capacitor. In this case, the change of capacitance in a capacitive display allows to derive information on which capacitors had their respective capacities changed.

2.2.1 Mutual Capacitance

The most common implementation approach of capacitive displays used in today's market is mutual capacitance. Mutual capacitance has several benefits including: multi-touch functionality, high touch accuracy, sensor space efficiency, and resistance to electromagnetic interferences (EMI) [6]. The mutual capacitance approach relies on the fact that most conductive objects can hold an electrical charge when they are very close to each other [6]. When another conductive object, in this case a finger, comes close to two other conductive objects, the capacitance between the two objects changes as some of their charge is transferred to the finger [6]. This effect is illustrated in the right image of figure 2.1. It is thus possible to determine the radius of a touch by recording the number of conductive objects whose capacitance have changed. This can be used as a variable to calculate a touch's strength. By design, mutual capacitive display touch detection is computationally intensive. However the benefits discussed justify the performance hit [6].

Mutual capacitive displays have a transparent thin-film conductor consisting of spatially separated electrodes patterned in two layers, usually arranged in rows and columns as shown in figure 2.2 [6][10].

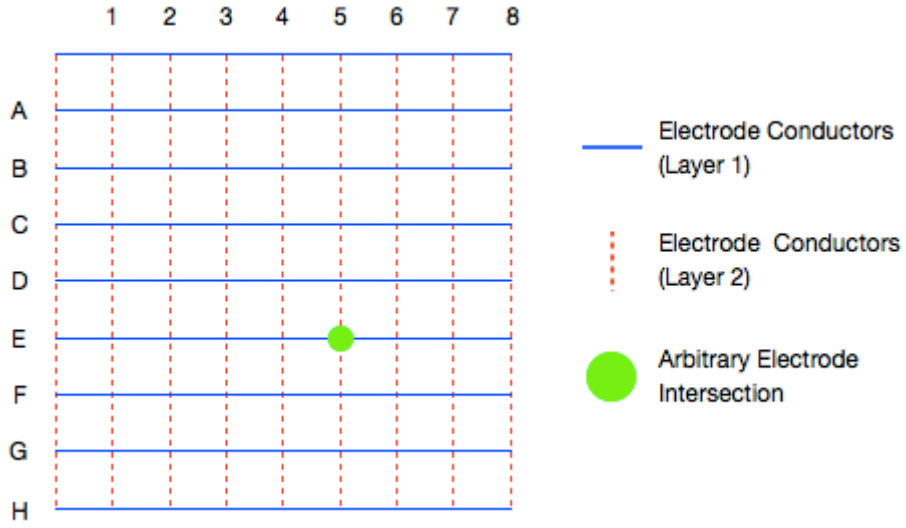

Figure 2.2: Capacitive Display Grid Layout

These two layers are then separated by a thin film insulator to create a capacitance [6]. To determine an exact touch location, the Central Processing Unit

(CPU) scans each row and column's intersections for their capacitance [6]. Using figure 2.2 as an example, the CPU would scan for the intersections between columns 1-8 for row A repeating this process until row H. The capacitance values from multiple adjacent electrodes or electrode intersections are then used to interpolate the exact touch coordinates thereby giving precise results [6]. The operating system (OS) takes these results and filters them back into touch events which can be used at an application level [4][12].

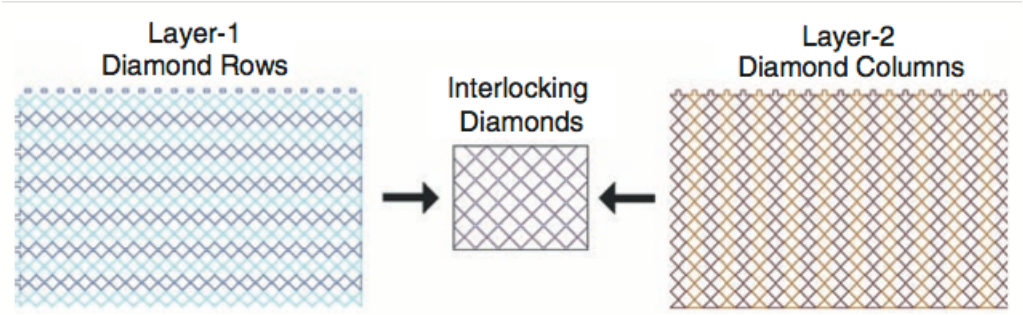

Figure 2.3: Diamond Grid Layout. Source: Barrett and Omote, 2010 [6]

The diamond pro-cap sensor pattern is formed by two interlocking diamond-shaped layers.

The original iPhone used a rows and columns standard grid for its older generation capacitive display (shown in figure 2.2)[6][10]. However newer devices, and devices targeted by this research project, use an interlocking diamond pattern as shown in figure 2.3. This reduces the computational costs and decreases the required physical capacitor sensor space [6]. The reduction of computational costs allows for more resources to be available for other processes.

2.2.2 Shortcomings

Capacitive displays offer multiple advantages as discussed above allowing them to be the dominant display technology in current touch capable devices. There are however shortcomings to this technology, such as noise sensitivity, due to manufacturing imperfections and the inability to register a touch from a non-capacitive object [6]. Another major drawback of special interest to this research project is the inability to record the force of a touch [10]. As explained in section 2.2.1 the only change registered is the change in capacitance. This provides information on how many charged electrons were excited and for how long, which can only be used for the position, radius and duration of a touch.

2.3 Motion Sensors

The movements of current devices equipped with an accelerometer and/or gyroscope produce distinct motion readings which are processed by the OS [4][12]. The inclusion of an accelerometer and gyroscope enables the tracking of a device's position in 3D space with six degrees of freedom (three for acceleration and three for rotation) [4][12]. Motion events are triggered when a device is free to move and an external force, such as a touch, acts on it thus generating accelerometer and gyroscope readings [4][12].

Cai & Chen observed the effects of touch inputs on the device's gyroscope in order to make an educated guess of a user's passcode [9]. Similarly Aviv *et al.* replicated Cai & Chen's passcode logging idea using the device's accelerometer as a side channel to interpret the rough locations of touches [5]. Cai & Chen and Aviv *et al.*'s approaches are reviewed in the following two sections 2.3.1 and 2.3.2 respectively.

2.3.1 Accelerometer Touch Readings

An accelerometer sensor measures the acceleration of a device relative to an inertial observer which is momentarily at rest [4][5]. The accelerometer measures the velocity of the device on the x, y and z axis as shown in figure 2.4. Devices with an accelerometer sensor contain three separate accelerometers (collectively referred to as a single accelerometer) to keep track of the x, y and z axis linear movements [5][4].

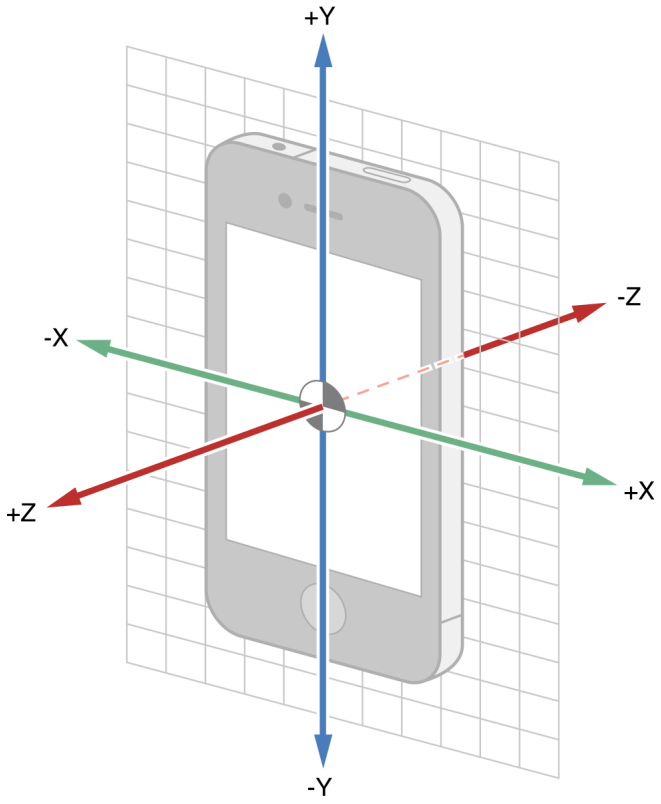

Figure 2.4: The Accelerometer Sensor. Source: Apple 2013 [4]

The accelerometer measures velocity along the x, y, and z axes.

A touch can affect the position of the device in 3D space and thus trigger a change in the accelerometer readings [5]. Aviv *et al.* normalised the results of the accelerometer readings of a touch in order to determine its rough location on the screen [5]. The readings could be analysed to match the different stages of a touch such as: initial contact with the display, the peak of the push against the display, and the finger lifting off display. The location of the touch was used to guess the keys' associated position on the screen [5]. Their research is useful in raising awareness about motion sensors side channels to key-log valuable information such as personal identification numbers (PIN) [5].

This research paper uses the accelerometer to measure the displacement of the device at different times of the touch building on Aviv *et al.*'s approach. However rather than attempting to discern the rough location of a touch, the accelerometer readings are used to assist the calculation of the touch's strength.

2.3.2 Gyroscope Touch Readings

Gyroscopes allow for the measurement of a device's change of orientation relative to the x, y and z plane (depicted in figure 2.5) [9][4]. On current smartphones and tablets this sensor often complements the accelerometer sensor discussed in the previous section.

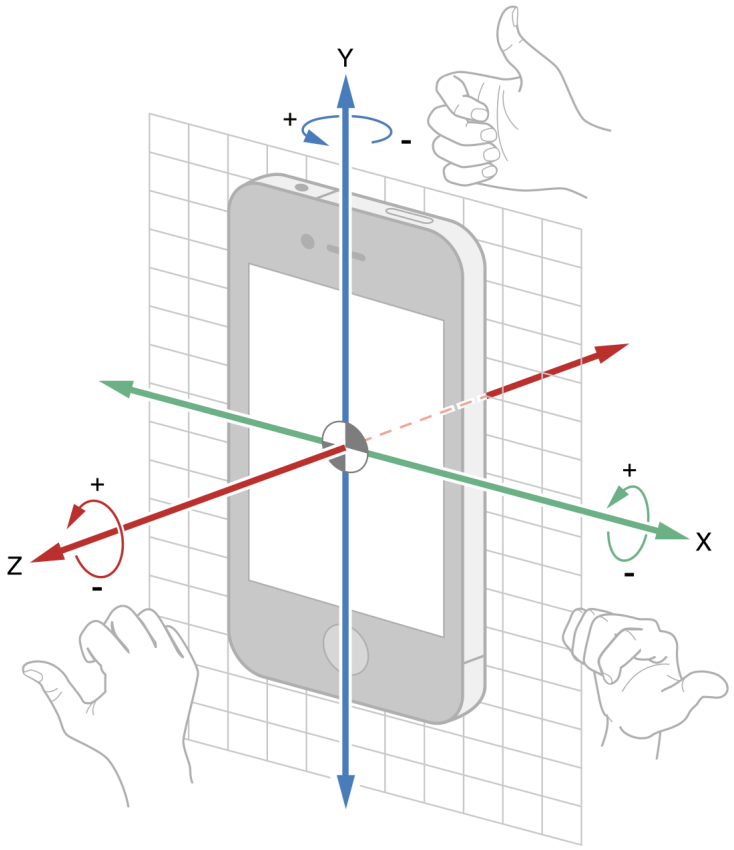

Figure 2.5: The Gyroscope Sensor. Source: Apple 2013 [4]

The gyroscope measures rotation around the x, y, and z axes.

Cai & Chen's paper preceded Aviv *et al.*'s acceleration approach and also used motion data from a side channel to determine the approximate location of a touch to retrieve sensible information [9][5]. However, their research was focused on the use of the gyroscope sensor as a side channel [9]. They established that the motion during typing on a capacitive display depended on several factors [9]:

- The typing finger's striking force;

- The resistance force of the supporting hand;
- The landing location of the typing finger; and
- The location of the supporting hand on the device.

The first two items affect the acceleration of the device and are used by Aviv *et al.* in their accelerometer based approach [9][5]. The last two affect the device rotation and are used by Cai & Chen with the gyroscope based approach [9].

The use of the gyroscope to log touch locations produced different data than Aviv *et al.*'s accelerometer based method [9][5]. Gyroscope readings of touches provide a 3D path which the device followed during the movement [9]. This path is mapped by Cai & Chen to a distinct 3D geometrical pattern in order to determine the approximate location of the touch on the display [9].

The gyroscope sensor is used in this project to help determine the strength of a touch on the screen. The method described in Cai & Chen's paper is used to complement the accelerometer approach used by Aviv *et al.* to eventually provide combined motion readings.

2.4 Background Summary

As Fische and Barrett & Omote specified in their white papers, capacitive displays are by nature incapable of registering enough information to calculate the strength of a touch [10][6]. However, due to mutual capacitance (discussed in section 2.2.1) the displays are able to register fine grained information about a touch [6]. The tradeoff with mutual capacitance is that it is more computationally demanding on devices with limited resources [6]. Despite the computational expense of mutual capacitance, Cai & Chen and Aviv *et al.* have demonstrated that matching motion data from a gyroscope and accelerometer to a touch on a capacitive display is indeed achievable [9][5].

CHAPTER 3

Algorithm Design

Cai & Chen and Aviv *et al.*'s work demonstrated that it was possible to use motion data to complement a touch. However, in both of their approaches, a single motion channel was used [9][5]. This produced a plausible solution as their experiments required finding the location of a touch on a screen regardless of its strength [9][5]. Processing motion and touch data from multiple side channels requires the implementation of a custom algorithm [4][12].

The design of the algorithm presented in this paper builds upon the previous related work done by Cai & Chen and Aviv *et al.* by creating an algorithm which can process and react to motion and touch data immediately after the touch ends. It can be hypothesised that by abiding to specific requirements, a custom algorithm can be created to simultaneously read touch and motion sensors data and process it using concurrent programming to provide immediate tactile feedback. The term "immediate" is used in this project from a human perspective. The time delay between the physical end of a touch and the completion of its processing may not be immediate in computational terms however it would appear instant to a human user.

Two custom algorithms are compared throughout the paper: the active concurrent approach and the passive concurrent approach. The design of both these approaches are described in this chapter.

3.1 Algorithm Requirements

The algorithms created must fulfil several requirements in order to calculate and react to the strength of a touch quickly enough for the interaction to feel natural to the user. The main requirements identified in this projects are:

1. Operations must be processed concurrently;
2. Immediate touch feedback must be provided at the end of a touch;

3. Data structures used are well established and modularly implemented;
4. Operations should be computationally inexpensive; and
5. A maximum quantity of motion data should be recorded during the lifetime of a touch.

The algorithm should make use of concurrent programming to optimise the time taken for the processing of a touch to occur. Concurrent programming allows different operations to run "simultaneously" [13]. Concurrency is a necessary part of this project's algorithm design as the motion and touch data recording must occur simultaneously on different queues [4]. A queue executes tasks either serially or concurrently but always in a first-in, first-out order [3]. This is done to offload the computationally expensive task of continuous recording away from the queue responsible for the user interface [4]. Using this paradigm, the algorithms should offload as many of their computational tasks to concurrent queues to make the touch processing as quick as possible. In particular the processing of a touch should be offloaded to its own queue to allow for a possible new touch to be registered and processed concurrently. Using concurrency, the algorithm can quickly react to touch interactions making the computation and recording of touch data appear seamless from a user's point of view.

Data structures used in the algorithms should be well established in the field and implemented in a modular fashion. The implementation modularity is important as it allows for a data structure to be replaced without affecting the overall working of the algorithms. The use of well known data structures ensures that they are reliable and well understood. This increases the robustness of the end result whilst allowing the algorithm to be comprehensible to a wide audience.

Operations in the algorithm aim to be as computationally inexpensive as possible. Whilst this aim is shared in most programming projects it is particularly important on mobile devices which have limited resources. A computationally expensive task not only uses up more processing power and memory, it typically takes longer than an inexpensive operation. The performance of the algorithms greatly benefits from avoiding these types of time consuming operations.

Lastly, the quantity of the gathered motion data in a touch's lifetime is of particular importance. The lifetime of a touch is considered to be the time period between the start and the end of the touch. The quality of the data is measured by how many motion objects can be recorded within the lifetime of the touch. In this project the data is recorded in multiple small time intervals making every data object recorded valuable. Sacrifices between the operations' performances and the quantity of data recorded are carefully investigated. Making computational performance sacrifices in order to gather more data may contradict the

immediate touch feedback requirement as the operation is likely to take longer. However, excessive focus on performance creates the risk of not recording sufficient information thereby hindering the quantity and thus quality of motion data recorded.

3.2 Active Concurrent Approach

The algorithm is labeled "active" as it activates motion resources only when touches are registered and turns the sensors off as soon as no further touches are recognised. Furthermore, touch and motion data are read and processed on concurrent queues hence the "concurrent" name. This approach is hereby referred to as the "active approach".

The workflow of the active approach is depicted in figure 3.1. The algorithm waits until a touch is registered on the screen at which point it activates the motion sensors. During the lifetime of the touch the motion and touch data are continuously recorded. Once a touch has ended, its recorded data is processed and a strength value is calculated. When there are no more touches on the display, the motion sensors are turned off. The queues and their respective computations are described in this section.

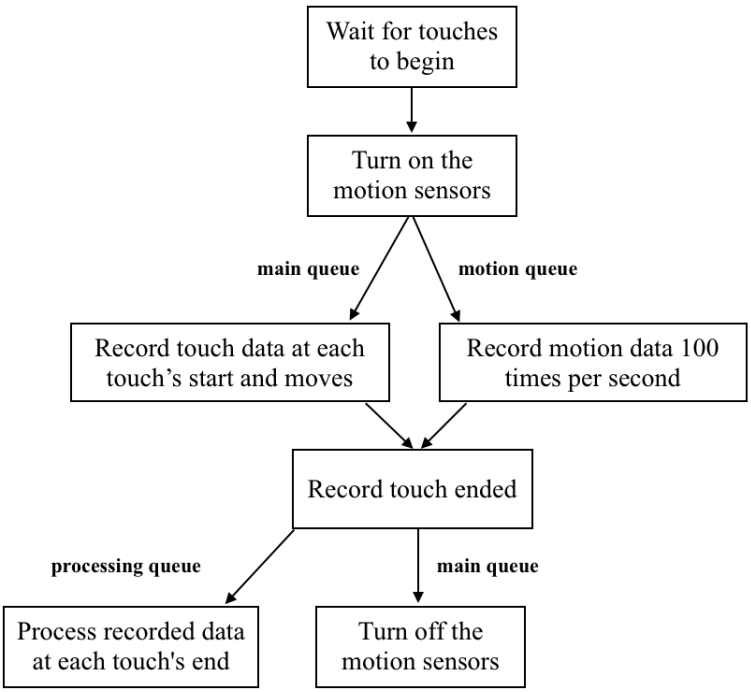

Figure 3.1: Active Concurrent Workflow

3.2.1 Main Queue

The main queue is automatically created by the system and associated with the main thread [3]. This queue handles drawing the user interface and most importantly the touch interactions from the user [3]. It is crucial that this queue remains unblocked throughout the algorithm's lifetime. Blocking this queue causes the user interface to freeze . Users should not have to wait until a touch's strength is calculated in order to interact with the software once again.

It is on the main queue that touch events are handled and that the motion sensors are turned on and off. The motion sensors are turned on each time the main queue registers a new triggering touch. A new triggering touch occurs when the display registers a touch and it currently has no touches already on it. When the sensors turn on, an array which stores the motion data objects, is populated. When the sensors are turned off, the array is emptied to reclaim memory.

The other queues run concurrently from the main queue and whilst the number of concurrent queues may vary, there must always be a single instance of the main queue [3].

3.2.2 Motion Queue

The logic to start and update the motion recording on the motion queue is shown in algorithm 1.

Algorithm 1 Start and Update Motion Readings

```
1:  procedure startDeviceMotion(error)
2:      if active then
3:          return                          ▷ Do nothing, the sensors are already on.
4:      end if
5:      if sensorsUnavailable then
6:          error ← noSensorsError
7:          return
8:      end if
9:      deviceMotionQueue ← newConcurrentQueue
10:     repeat on deviceMotionQueue
11:         updateError ← emptyError
12:         motionData ← motionUpdateCallback(updateError)
13:         if updateError ≠ emptyError then
14:             handle(updateError)
15:         else
16:             handleMotionObjectUpdate(motionData)
17:         end if
18:     until numActiveTouches = 0
19: end procedure
```

Algorithm 1 first checks that it is possible to turn on the sensors and if so it creates the motion queue (*deviceMotionQueue*). A motion data object is populated at each callback from the motion sensors. If the update was unsuccessful the error is handled. However, if the update was successful, the data is handled and stored in the appropriate data structure. In this approach the *motionData* is added to the array created on the main queue. The counterpart algorithm to stop the motion sensors is described in Appendix A.2.

3.2.3 Processing Queue

The processing queue is the most computationally intensive queue. It processes the data read from the motion and touch sensors to ultimately give a strength value for a touch.

In this approach the processing queue retrieves the motion data from the array (which has been created on the main queue) at the end of a touch. This process is shown in algorithm 2.

Algorithm 2 Active Motion Data Retrieving

```
1: procedure RETRIEVEMOTIONDATA
2:     endIdx ← currentMotionArrayIndex    ▷ startIdx was stored at the start of the touch
3:     idx ← startIdx
4:     for idx < endIdx, idx ← idx + 1 do
5:         motionData ← motionArray[idx]
6:         processMotion(motionData)
7:     end for
8: end procedure
```

Line 6 of algorithm 2 processes each of the motion objects to calculate the average acceleration and rotation for the touch. The logic to process the motion is detailed in algorithm 3.

Algorithm 3 Motion Data Processing

```
1: procedure PROCESSMOTION(motionData)
2:     acc ← motionData.acceleration
3:     totalAcc ← totalAcc + sqrt(acc.x * acc.x + acc.y * acc.y + acc.z * acc.z) ▷ Pythagoras is used to store a scalar value in external variable
4:     rot ← motionData.rotation
5:     totalRot ← totalRot + sqrt(rot.x * rot.x + rot.y * rot.y + rot.z * rot.z) ▷ Pythagoras is used to store a scalar value in external variable
6: end procedure
```

Touch data processing occurs once at the beginning and once at the end of a touch as well as each time a noticeable movement in the touch's location is registered. At each stage, the touch's properties are stored by the processing queue for later computation.

Once the touch ends, the stored touch data is processed to produce the maximum touch radius from each phase, the total duration of the touch, and the x and y movements from the start to the end of the touch's locations. The logic to process the touch data is shown in algorithm 4.

Algorithm 4 Touch Data Processing

```
1: procedure processTouch(touchData)
2:     startTimestamp ← endTimestamp ← 0
3:     startX ← startY ← endX ← endY ← 0
4:     maxRadius ← MIN
5:     for all touchData.touchPhaseInfo, touchInfo do
6:         if touchInfo.phase = BEGIN then
7:             startTimestamp ← touchInfo.timestamp
8:             startX ← touchInfo.x
9:             startY ← touchInfo.y
10:        else if touchInfo.phase = END then
11:            endTimestamp ← touchInfo.timestamp
12:            endX ← touchInfo.x
13:            endY ← touchInfo.y
14:        end if
15:        radius ← touchInfo.radius
16:        if radius > maxRadius then
17:            maxRadius ← radius
18:        end if
19:    end for
20:    duration ← endTimestamp − startTimestamp
21:    xDelta ← ABS(endX − startX)
22:    yDelta ← ABS(endY − startY)
23: end procedure
```

3.3 Passive Concurrent Approach

The passive concurrent approach (shortened to "passive approach") differs from the active approach in two major ways.

Firstly, when the algorithm starts, it continuously records motion data and only turns the sensors off when the algorithm is no longer used. This passive continuous listening behaviour is what gives the approach the name "passive".

Secondly the use of concurrent programming is more prevalent in this approach as it maintains four concurrent queues. Whilst concurrent, operations in each queue rely on outputs from other queues to proceed as shown in figure 3.2. The passive approach is described by explaining the queues and their relationships with one another.

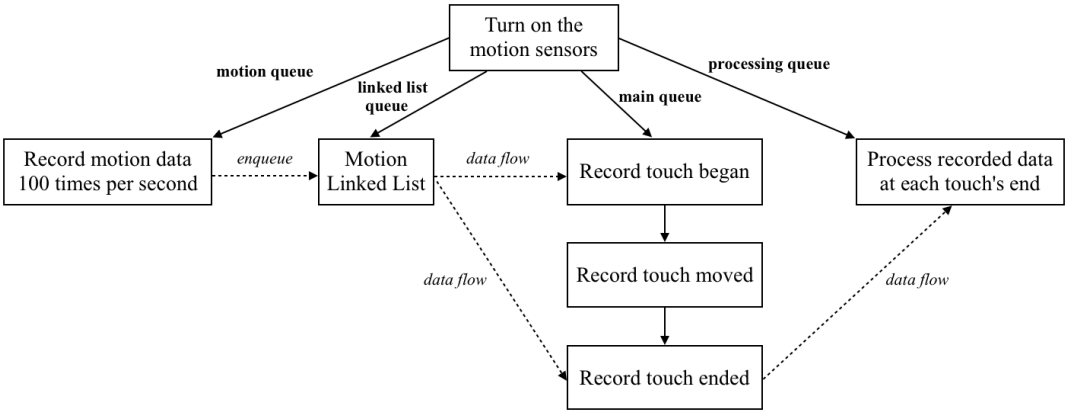

Figure 3.2: Passive Concurrent Workflow

Dotted arrows represent relationships.

3.3.1 Main Queue

The main queue in the passive approach performs much the same tasks as the main queue in the active approach (described in section 3.2.1). The difference in this approach the algorithm only uses the main queue for touch handling. The data structure to store motion objects is separated in its own concurrent queue.

3.3.2 Motion Queue

The passive approach motion queue is slightly different to the active approach's motion queue described in section 3.2.2.

The motion queue is alive for the entire lifetime of the algorithm. It continues recording motion data even if there are no current registered touch interactions. This approach uses the same logic as algorithm 1 to initially start the motion sensors and create the motion queue at the start of the algorithm.

This approach's motion queue is exclusively responsible for recording motion data. The storing of the motion data occurs on the linked list concurrent queue (discussed in section 3.3.3). Thus, line 16 of algorithm 1 enqueues the *motionData* on the linked list concurrent queue.

3.3.3 Linked List Queue

Due to the continuous flow of data from the motion queue a singly linked list data structure is used for storage. The singly linked list brings three major advantages to the algorithm. Firstly, it is ideal for storing data of unknown quantity [19]. Secondly, it is extremely cheap to reset and discard the unwanted data [19]. By simply resetting the reference pointers, the data structure is reseted [19]. And thirdly, appending data to the linked list is done in constant time [19].

A singly linked list (referred to as "linked list" in this project) is a well known data structure made up of nodes which together make up a sequence [19]. Each node contains an object and a pointer to the next object [19]. Special pointers are used to keep track of the head and the tail of the list [19]. In this algorithm the object held within each node contains the data readings of a motion event.

The linked list is reset by setting the head pointer to the tail pointer when there are no more active touches on the screen to free memory.

3.3.4 Processing Queue

The processing queue works similarly to the active approach's processing queue described in section 3.2.3. It differs however in the way that it retrieves and enumerates through the motion data objects. This distinction is due to the fact that the passive approach uses a linked list data structure to store motion data as opposed to the active approach's array data structure.

As soon as a touch is registered the only processing needed is to set a reference to the last motion object recorded in the linked list. At the end of a touch the above process is repeated with the newest motion object. The motion objects between the two reference points are then enumerated and each motion object is processed to create an average acceleration and rotation value for the touch. The logic is shown in algorithm 5.

Algorithm 5 Passive Motion Data Retrieving

```
1: procedure RETRIEVEMOTIONDATA
2:     currentMotionItem ← headMotionItem  ▷ headMotionItem's reference was stored at the start of the touch
3:     tailMotionItem ← lasMotionItemFromLinkedList
4:     while currentMotionItem ≠ tailMotionItem do
5:         processMotion(currentMotionItem.motionData)
6:         currentMotionItem = nextItem(currentMotionItem)
7:     end while
8: end procedure
```

The *processMotion()* call on line 5 in algorithm 5 follows the same implementation described in algorithm 3. Furthermore, the processing of the touch data is identical in both approaches, thus the logic to process the touch data for this approach is described in algorithm 4.

3.3.5 Queue Relationships

There are three relationships between the concurrent queues used in the passive approach. The first is between the motion recording queue and the linked list queue. The second is between the linked list queue and the processing queue. The last relationship is between the main queue and the processing queue.

The motion queue's sole purpose is to continuously record motion data. Once a new motion data object is recorded it passes the object to the linked list queue. This then allows the linked list queue to enqueue the object without blocking the data recording motion queue.

As soon as the main queue registers a touch, a reference to the latest motion object enqueued on the linked list is associated to that touch. This is kept until the process is repeated when the touch has ended. By keeping references to the linked list's motion objects for the touch's start and touch's end the linked list queue can pass on the motion objects stored between these references to the processing queue. The data processing can then occur concurrently without disrupting the other queues.

Lastly the relationship between the main queue and the processing queue is for the main queue to pass on the touch data to the processing queue for processing.

3.4 Platform Selection

In order to create a software algorithm which uses data from motion sensors and a capacitive display, the hardware on which it runs must contain an accelerometer, a gyroscope and a capacitive display. Further the hardware used for this project should be easily attainable and affordable. Smartphones and tablets are ideal as they are, at the time of writing, a commodity and most of them have well established Software Development Kits (SDK). The SDKs integrate well with their motion sensors and multitouch capacitive displays when available.

The algorithm in this project is implemented with the Apple iOS 7 SDK and caters for all its supported devices. This choice is justified as a single binary can be written for all iOS 7 devices which are guaranteed to contain a gyroscope and accelerometer with the same Application Program Interface (API) [2]. The choice to run on both tablets and smartphone allows for wider testing of the algorithms described in this chapter. Whilst other platforms also support universal binaries Apple's SDK is the oldest and most established solution.

3.5 Algorithms Summary

During preliminary touch recording tests, it was found that the passive algorithm constantly recorded more touch motion data (this is further discussed in chapter 6). For this reason, the passive algorithm has been chosen for the next steps of the research. However the algorithms will continue to be compared in the results chapter and the selection of the passive algorithm is discussed in detail and further justified in the discussion chapter.

This chapter described, analysed and compared the active and passive algorithm approaches. By this stage, a portion of the aim has been achieved as an algorithm has been created to use the motions of a device as well as a touch's properties to calculate a strength value, allthough 'strength' has yet to be defined. Further, the algorithm has not yet been user-calibrated.

CHAPTER 4

Strength Perception Sampling

To create a user calibrated algorithm, the passive algorithm is used on a population sample. The participants are instructed to interact with the application running the algorithm (described in section 3.3) by tapping the capacitive display with their perceptions of a soft, normal and hard touch. The hypothesis is that people have different perceptions of the strength of a touch.

This chapter describes the procedure for gathering and processing the information from the population sample. The gathering is undertaken using a mobile application written for tablets and smartphones which contains the strength calculating algorithm. This data is stored on a server and downloaded for processing and analysis. The results are outlined and discussed in the Results and Discussion chapters respectively.

4.1 Selection of Method

The population study aims to have participants of different genders, ages, heights, weights and occupations interact with the algorithm. The sample recruitment method used is a mixture between snowball sampling and quota sampling [8]. In snowball sampling the researcher makes contact with a small group of participants and then uses these to establish contact with others [11][8]. In contrast, the quota sampling method aims to produce a sample that reflects a population in terms of the relative proportion of people of different categories [8]. The categories reflected in this project are gender, age, height, weight and occupation.

The snowball/quota hybrid approach was chosen over convenient sampling. In convenient sampling the researcher makes use of a population that is simply easily accessible [8]. The population which would have been conveniently sampled by this author has considerable experience with smartphones and tablets and thus would not reflect groups with less experience with these devices.

An iOS application named DOMATI is used to allow the participants to

input their touch strength perceptions. The choice of mobile platform used to gather information is described in section 3.4. The application used to gather the data uses a specialised algorithm which extends the passive concurrent approach discussed in section 3.3. In this extension a fifth queue is added to handle the storing of data to the device's local database. The computation to persistently store the data is handled on this queue as soon as the processing queue finishes. The saving queue extension is shown in figure 4.4.

DOMATI is sent to participants through the distribution portal TestFlight [18]. Through this process users can download the application onto their personal devices, input their information and interact with the algorithm with minimal interferences from the author of the report.

4.2 Data Gathering Through the Mobile Application

DOMATI collects anonymous demographic information from the user as shown in figure 4.1. The information can be set as "undisclosed" at the user's discretion. Information collected and the justification for each is listed below:

- Gender: men and women may perceive a touch's strength differently;
- Birth Year: an elderly person may perceive a touch to be softer than a teenager;
- Height And Weight: larger participants may have a natural stronger perception of a touch than smaller people; and
- Occupation: a labourer may have a stronger touch than an office worker.

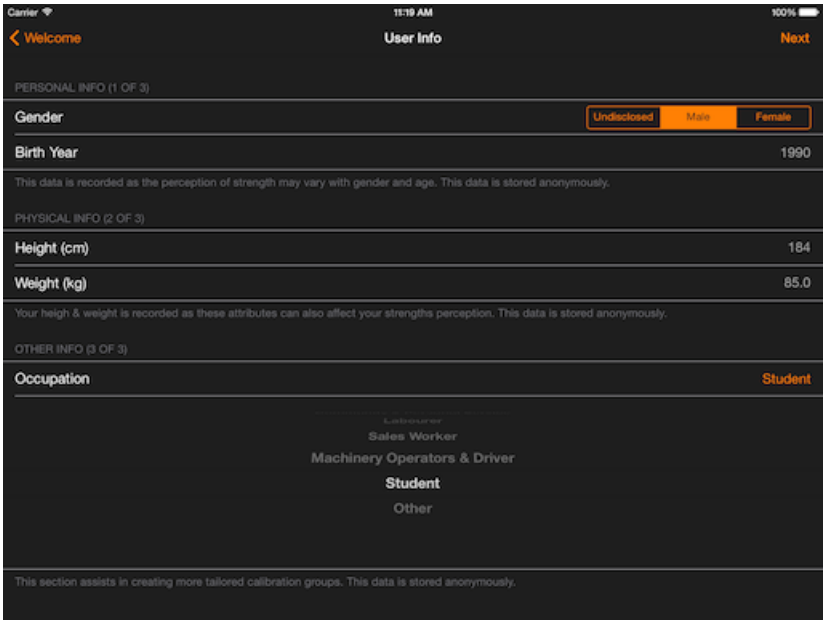

Figure 4.1: DOMATI User Info Screenshot

After entering personal information, the participant is instructed to keep the device in their hand and shake it to proceed. This step is important as the device must be free to move in 3D space for the motion sensors to capture valid data. If the device is on a flat surface and unable to move, the readings are erroneous. By shaking the device this risk is reduced as the shake is only recognised when the device is held.

The participant is then instructed to touch the screen with various strengths. As the display is touched the algorithm records the motion and touch data and stores it internally. The user is instructed to perform three actions on the display:

1. Tap softly, as if you were scared to break the glass;
2. Tap like you would normally interact with your device; and
3. Tap hard, as if you were frustrated with your device (be sensible).

When the strength entries are recorded, the data is uploaded to a server (further discussed in section 4.2.3). The application then informs participants that the exercise is complete and that they may quit it. Figure 4.2 depicts the above workflow.

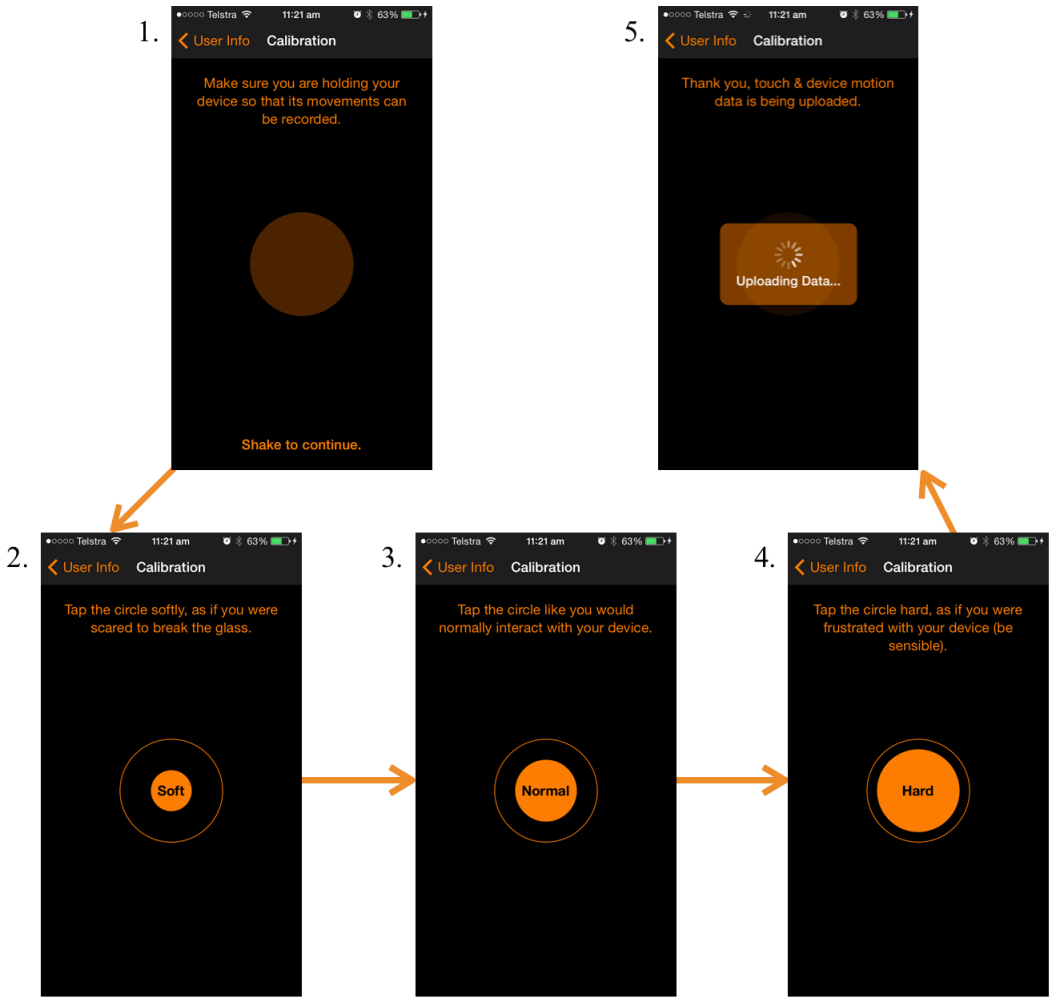

Figure 4.2: DOMATI Strength Gathering Flow Screenshots

4.2.1 Raw Data

Raw data is the unprocessed data recorded by the device. This data is kept for future processing and analysis to help provide the summarised data discussed in section 4.2.2.

Raw motion data is recorded at around a 100Hz, giving 100 readings per second. Each reading is saved to a local database and indexed for later use. The raw data stored is made up of the x, y and z components for the acceleration and rotation of the device and the timestamp of the motion event.

Raw touch data is recorded less frequently and at defined phases of the touch.

Touch readings are stored when the touch begins, moves and ends. There is always one reading for the beginning and ending phases of a touch. The moving readings are saved each time a change in the position of the touch is recorded. The raw data stored at each touch phase consists of the touch phase, the radius, the x and y coordinates and the timestamp of the touch event.

4.2.2 Summarised Touch Data

The number of raw data objects quickly increases and it becomes difficult to discern relevant information. Therefore during the processing of the touch, an object which contains the summarised information from the raw touch data and its associated raw motion data is created. The summary object is stored with references to its raw objects. It contains the following information:

- Acceleration Average (G unit[1]): the average acceleration of the raw motion data for a given touch;
- Rotation Average (radians/seconds): the average rotation of the raw motion data for a given touch;
- Calibration Strength: the strength with which the user was asked to touch the display;
- Duration (seconds): the total duration of the touch;
- Maximum Radius (millimetres): the maximum finger tip radius from the associated raw touch data;
- X and Y Deltas (points[2]): the changes in the x and y coordinates between the start and end of the touch; and
- Device: the device used to record the touch. Used to differntiate between smartphones and tablets. This is used for the descriptive results.

4.2.3 Server Database

Following processing, the data is saved locally on the device. To manually retrieve the data from the devices is not time efficient and individually reaching the participants removes the process' anonymity. For these reasons, a server is used

[1] 1 G Unit = 9.81 ms^{-2}

[2] 1/72th of an inch

to store the data collected from the devices. This allows the data to be collated in one place and downloaded independently of user involvement.

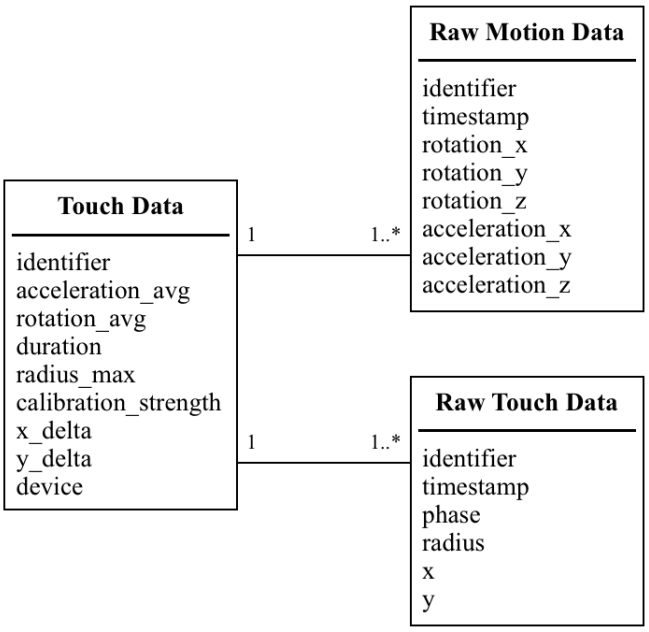

Figure 4.3: Database Schema

The server structure is simple and shares the same schema used for the local database shown in figure 4.3. Upon completion of the touch processing, a sixth concurrent queue called the networking queue is created. This queue handles the network operations of uploading the data as shown in figure 4.4.

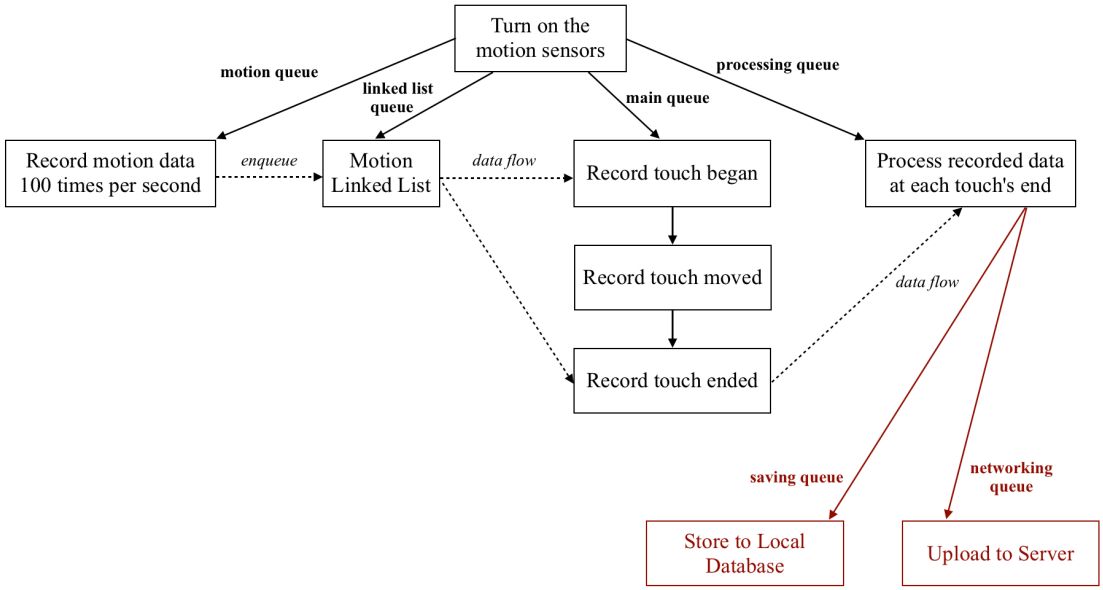

Figure 4.4: Passive Concurrent Workflow With Data Storing and Networking

Arrows represent the queues of the program. Dotted arrows represent relationships.

4.3 Touch and Motion Data Processing

The data from the server is analysed following a month of data collection. Descriptive analysis is used to present demographic data, and statistical analysis is applied to motion and touch data. The non-normality of the motion and touch data distribution is assessed with and without outliers. However the data used for the algorithm is a subset of the data set in which the outliers are removed. The data superset along with the untouched data set can be found in appendices B.1 and B.2 respectively.

Following the assessment, a Kruskal-Wallis test is applied on each variable recorded to determine wether there is statistical difference between the touch strengths as a whole. Kruskal-Wallis tests are non-parametric methods used to compare whether two or more samples are independent from one another [14]. Mann-Whitney tests are then run for each variable to compare the differences between each strengths relative to one another. A Mann-Whitney test is another non-parametric test which tests whether two populations are statistically similar [14]. These tests determine if the touch strengths are statistically different from each other [14].

The data sets are also clustered using the k-means clustering technique. k-means is a major clustering method producing partitions of the entity set into non-overlapping clusters together with within-cluster centroids [16]. These partitions are used to test the hypothesis that people have different perceptions of a touch's strength. Unless this hypothesis is disproved, the partitions should contain distinct clusters matching the different perceptions of strength.

4.4 Normalised Touch Strength

At the end of the touch and once the motion and touch data have been processed, a strength value can be calculated. Using the data from the population study it is possible to create lower and upper strength value boundaries for soft, normal and hard touches. However, the units of measurement and their scales differ for each of the variables. In order to create a single strength value these values must first be normalised according to the strength value boundaries.

Once the variables are normalised a final strength value can be calculated. The final strength unit is a normalised scalar value from 0.0 to 1.0. The algorithm used to normalise the data is described in Appendix A.3. The touch strength value breakdown is as follows:

- 0.0 - 0.33: Soft touch;
- 0.34 - 0.66: Normal touch; and
- 0.67 - 1.0: Hard touch.

4.5 Data Sampling Summary

Using a snowball/quota hybrid sampling technique the raw data and the more informative summarised touch data are gathered. They are then uploaded to the server where the data can be easily accessed and downloaded. Once downloaded, the dataset is descriptively analysed to understand the population demographics. Basic non-parametric statistical analysis is then conducted on a dataset with outliers omitted. Kruskal-Wallis and Mann-Whitney tests are applied to determine statistical differences between soft, normal and hard touches. Finally, k-means clustering is run on the dataset to determine significant groups amongst participants.

After the gathering and processing of data a quantitative value of soft, normal and hard touch strength can be extrapolated from the results. This value is a normalised scalar which can now be fed back into the algorithm providing the user-calibration aspect of the aim which Chapter 3 was unable to address. Furthermore, the duration of the touch strengths is used for comparing the algorithms in the results chapter.

CHAPTER 5

Results

Following the data collection phase described in section 4.2 the server contained a sample population of 110 participants. The participants successfully recorded touches creating a total of 493 summarised touch data (discussed in 4.2.2). The stored raw data objects (discussed in section 4.2.1) were made up of 8698 raw motion objects and 1785 raw touch objects.

The touch summary objects are split into their calibration strengths: soft, normal and hard. The split produced 164 soft touches, 165 normal touches and 164 hard touches. These touch objects are tabulated and graphed as box-plots for visual comparisons between touch strengths. Further the cluster results are presented in a table as well as graphically through bar graphs.

In the last section, the motion recording performances of the algorithms are compared with various time durations. The comparisons are run on hardware structures with and without a dedicated motion co-processor chip.

5.1 Demographic Results

As mentioned in section 4.1, the aim of the ample selection is to reflect the population in terms of the relative proportion of people according to gender, age, weight, height and occupation. The population sample data is visualised in figure 5.1.

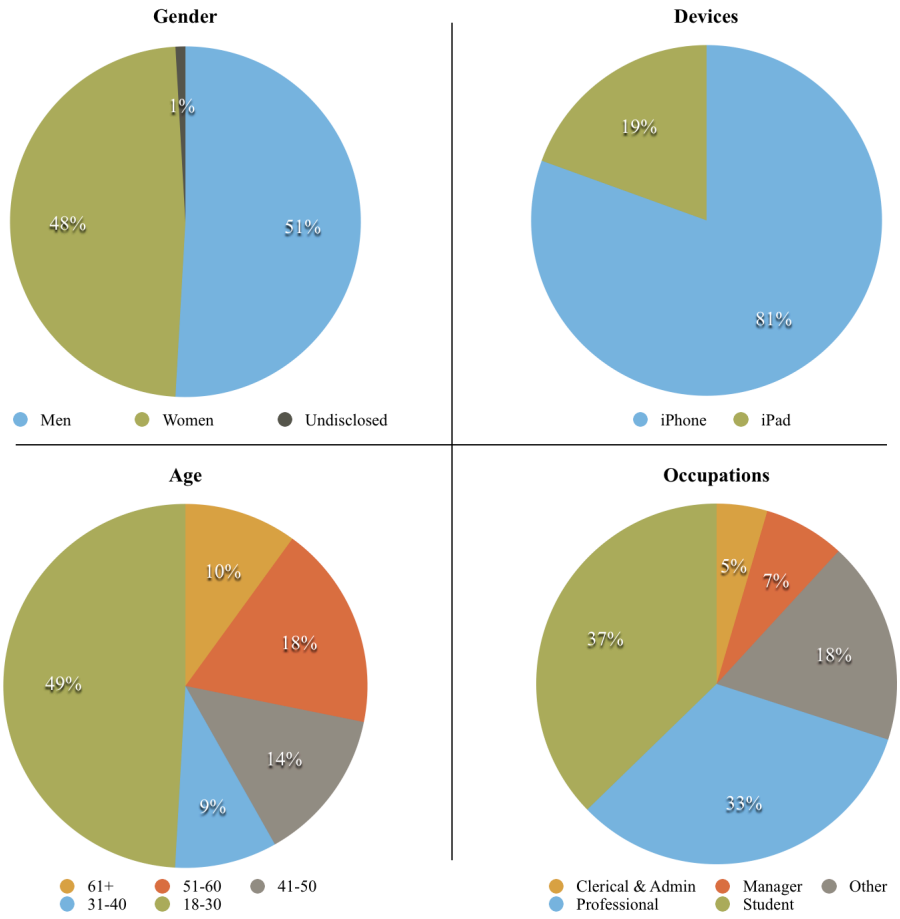

Figure 5.1: Sampling Results

5.2 Summarised Touch Data Results

Table 5.2 presents the values used to run the non-parametric test for the summarised touch data results. A value of less than 0.05 on the Kruskal-Wallis and Mann-Whitney tests is indicative of a significant statistical difference between compared variables [14]. As the value approaches 0.0 the importance of the statistical difference increases [14].

			N (instances)	Median	Maximum	Minimum	*p* Value *Kruskal-Wallis Test*	Post Ad Hoc *Mann-Whitney Test*
Motion Data	**Acceleration Average** (G units, 1G = 9ms^{-2})	**Soft**	158	0.0268	0.1127	S : 0.0053	0.0001	S - N, p = 0.0001
		Normal	162	0.0717	0.5370	0.0099		N - H, p = 0.0001
		Hard	161	0.1949	1.0697	0.0206		H - S, p = 0.0001
	Rotation Average (radian per seconds)	**Soft**	161	0.0648	0.4195	0.0074	0.0001	S - N, p = 0.0001
		Normal	164	0.2254	0.9570	0.0257		N - H, p = 0.0001
		Hard	163	0.6524	3.0971	H:0.0373		H - S, p = 0.0001
Touch Data	**Duration** (seconds)	**Soft**	163	0.0958	0.2999	0.0315	0.0001	S - N, p = 0.0001
		Normal	161	0.1112	0.3674	0.0322		N - H, p = 0.0001
		Hard	159	0.1831	1.1165	0.0163		H - S, p = 0.0001
	Maximum Radius (mm)	**Soft**	164	7.0010	9.9900	5.4100	0.0001	S - N, p = 0.0001
		Normal	164	7.5450	10.6400	5.9100		N - H, p = 0.0001
		Hard	163	8.2700	14.1300	5.6600		H - S, p = 0.0001
	x Delta (points)	**Soft**	161	1.5	7.5	0.0	0.0001	S - N, p = 0.0001
		Normal	164	2.5	14.5	0.0		N - H, p = 0.006
		Hard	157	3.0	18.5	0.0		H - S, p = 0.0001
	y Delta (points)	**Soft**	162	1.5	12.5	0.0	0.0001	S - N, p = 0.015
		Normal	163	2.0	12.0	0.0		N - H, p = 0.0001
		Hard	162	4.0	26.0	0.0		H - S, p = 0.0001

Figure 5.2: Summarised Touch Variables Results

5.2.1 Box-Plots

The data collected is represented in box-plots for each variable which are differentiated further by touch strengths. The data used for the box-plot is the superset of the data used for table 5.2. Dotted lines are used to compare the relative positions of the touch strengths as the vertical axis scales are inconsistent.

Average Device Acceleration

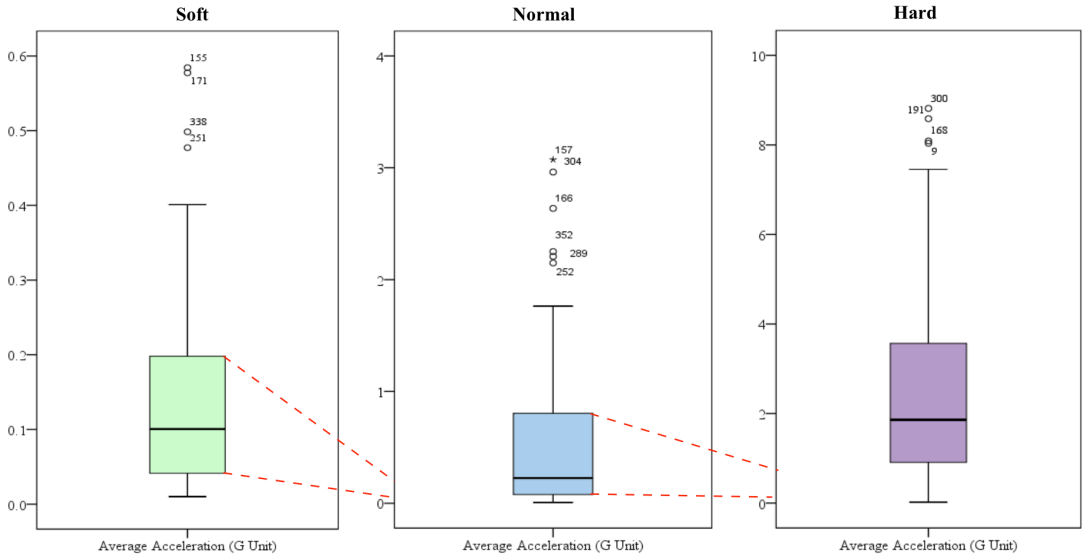

Figure 5.3: Average Acceleration Box-Plots

Average Device Rotation

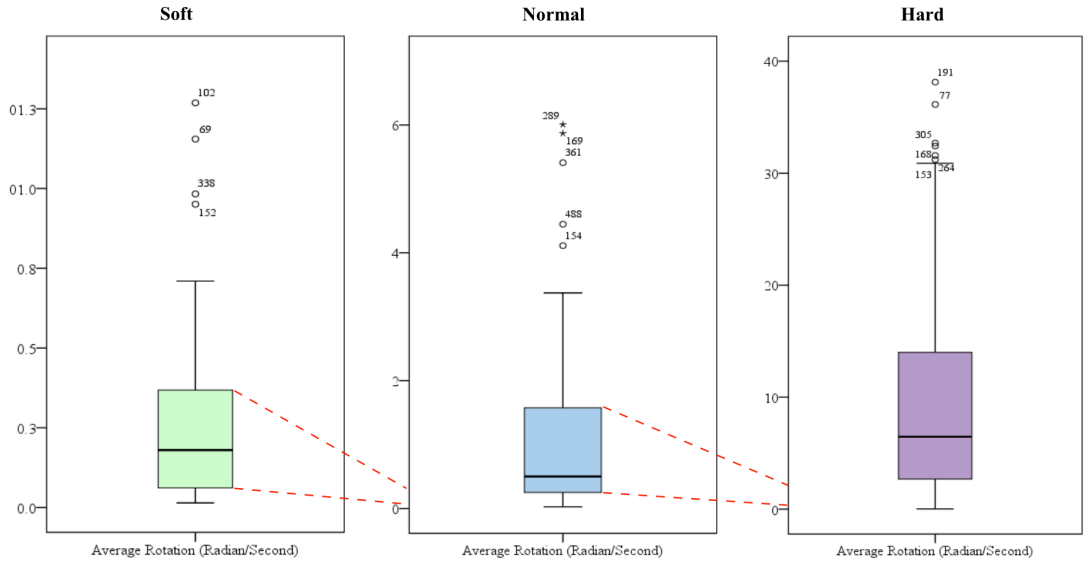

Figure 5.4: Average Rotation Box-Plots

Maximum Touch Radius

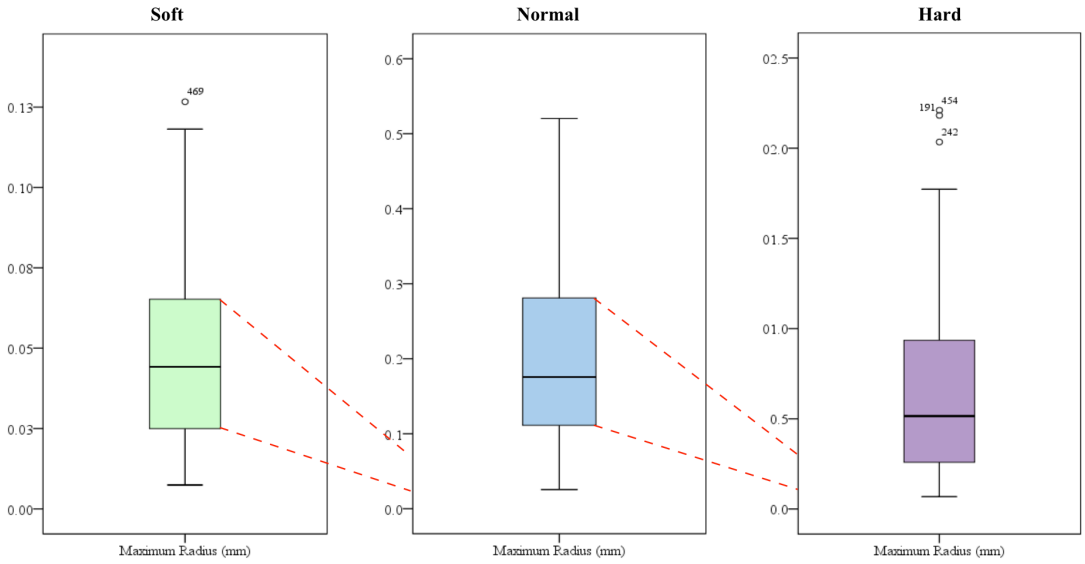

Figure 5.5: Maximum Radius Box-Plots

Touch Duration

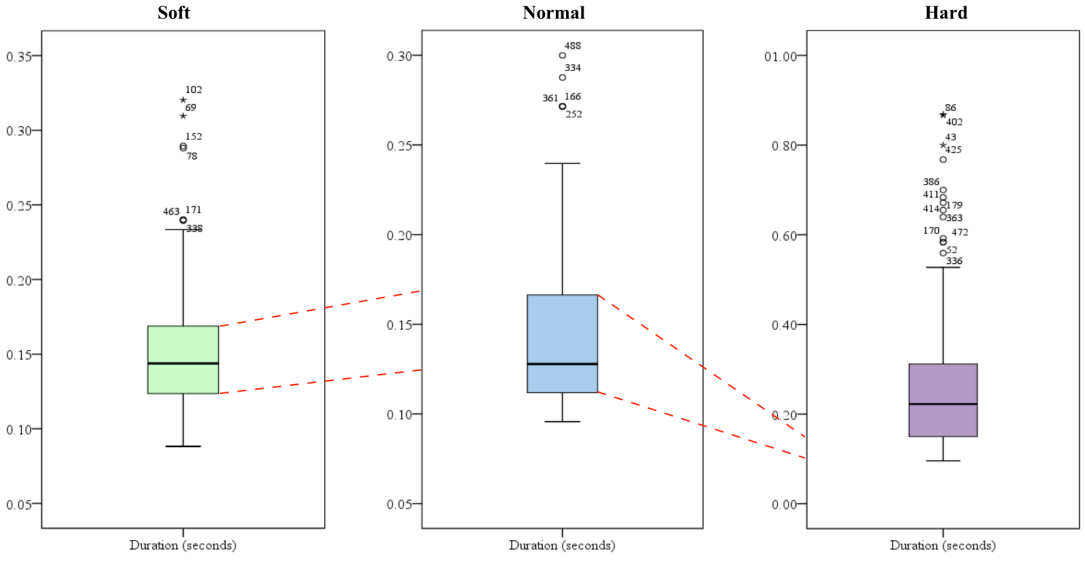

Figure 5.6: Touch Duration Box-Plots

Touch Horizontal Location Change

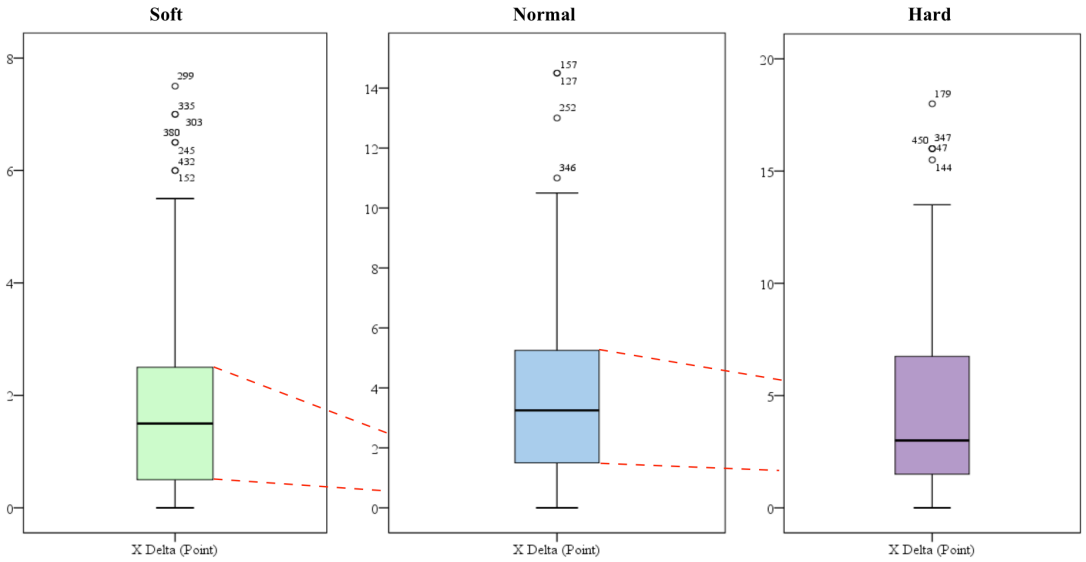

Figure 5.7: X Delta Box-Plots

Touch Vertical Location Change

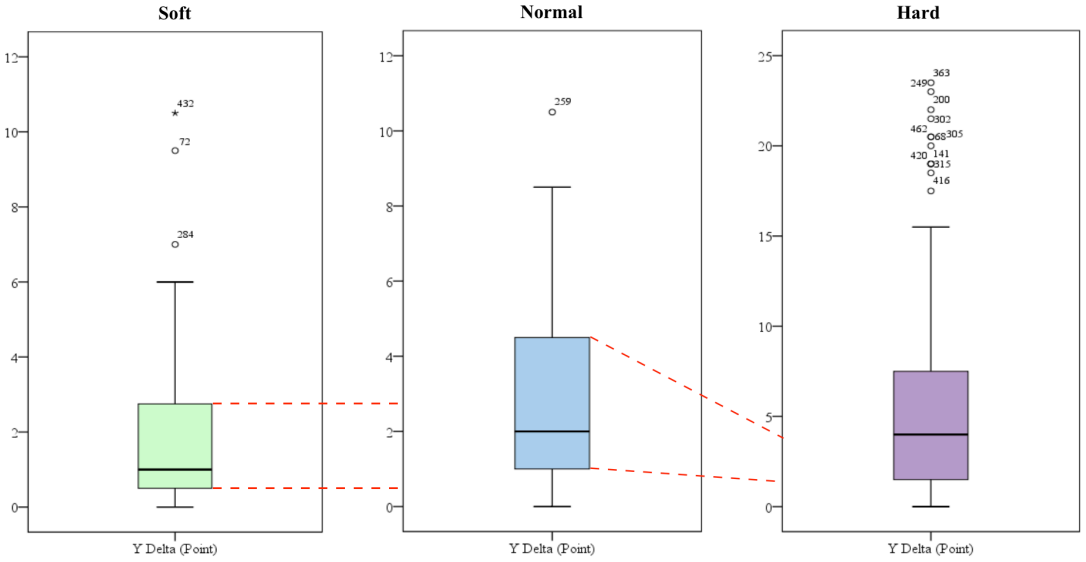

Figure 5.8: Y Delta Box-Plots

5.3 Clusters

The table shows the clusters of each of the variables along with their percentage weights and centroids for each of the touch strengths.

			Clusters				
			All	#1		#2	
			Centroid	Centroid	%	Centroid	%
Motion Data	Acceleration Average (G units, 1G = 9ms^{-2})	Soft	0.0327	0.0234	80	0.0695	20
		Normal	0.1064	0.0755	88	0.3389	12
		Hard	0.2775	0.1671	80	0.7059	20
	Rotation Average (radian per seconds)	Soft	0.0911	0.0601	83	0.2386	17
		Normal	0.265	0.1815	77	0.5517	23
		Hard	0.8188	0.5354	80	1.9354	20
Touch Data	Duration (seconds)	Soft	0.122	0.0948	74	0.2005	26
		Normal	0.1102	0.1012	99	1.5514	1
		Hard	0.2449	0.1727	86	0.6747	14
	Maximum Radius (mm)	Soft	7.0285	6.4657	54	7.6801	46
		Normal	7.6212	6.9852	56	8.4337	44
		Hard	8.6445	7.9541	75	10.7672	25
	x Delta (points)	Soft	1.9472	1.0636	73	4.3721	27
		Normal	3.314	1.7412	70	6.9	30
		Hard	4.7389	2.7292	76	11.2568	24
	y Delta (points)	Soft	2.1142	1.4225	88	7.025	12
		Normal	2.6994	1.4538	73	6.0682	27
		Hard	6.0648	3.7313	83	17.2321	17

Figure 5.9: Summarised Touch Variables Clusters

k-means clustering is used to output the rsults.

5.3.1 Cluster Bar Graphs

The cluster percentile weights from table 5.9 are represented as bar graphs below.

Average Device Acceleration

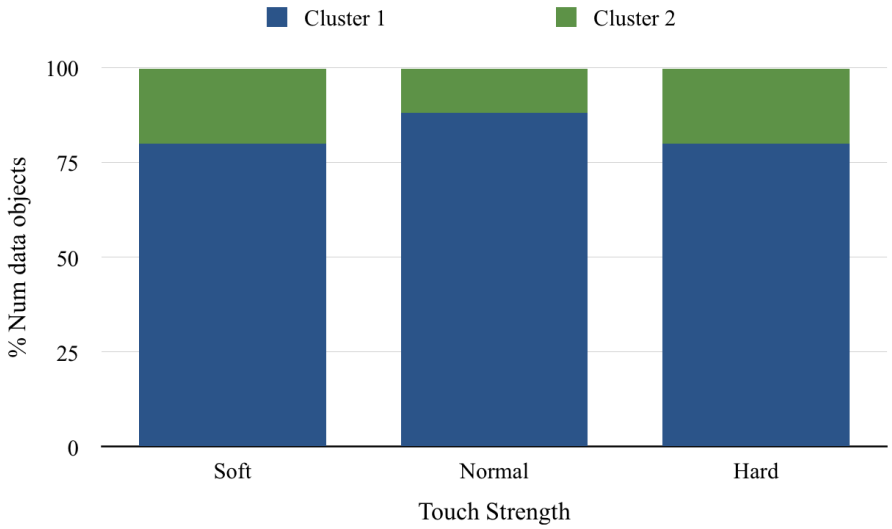

Figure 5.10: Average Acceleration Clusters

Average Device Rotation

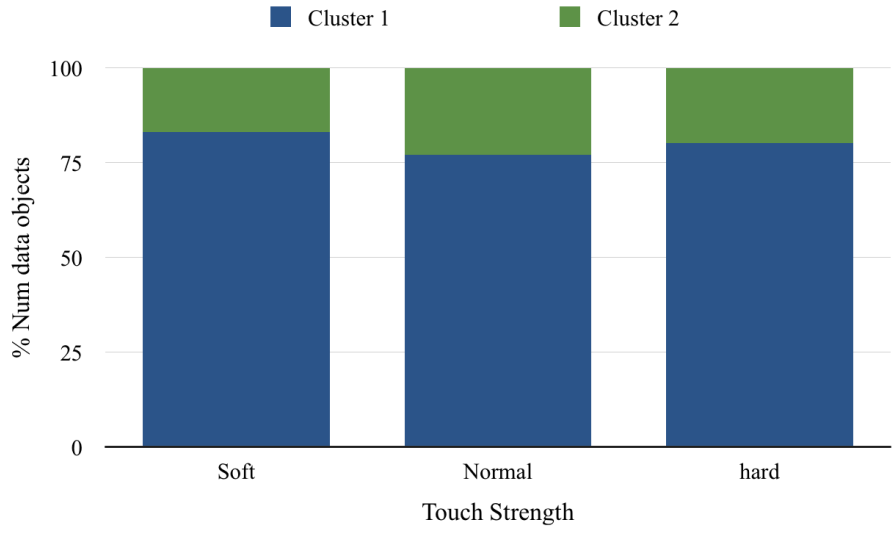

Figure 5.11: Average Rotation Clusters

Maximum Touch Radius

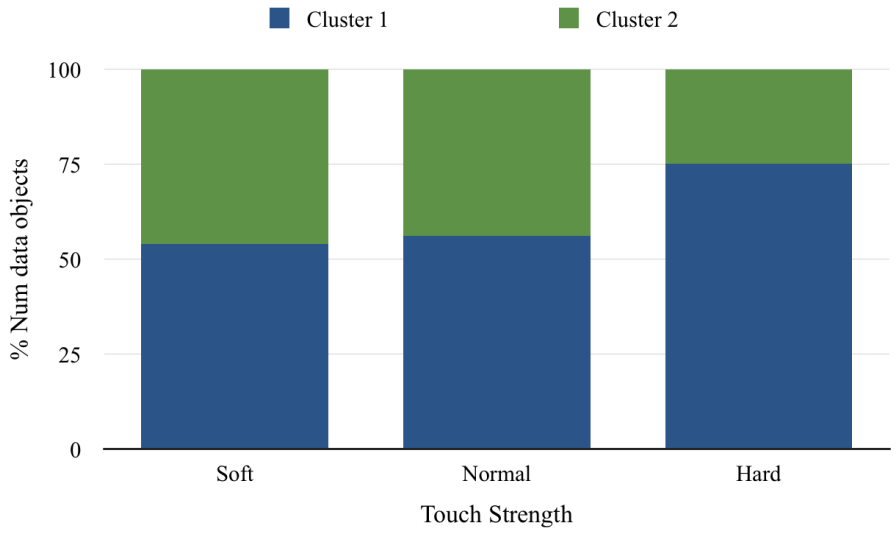

Figure 5.12: Maximum Radius Clusters

Touch Duration

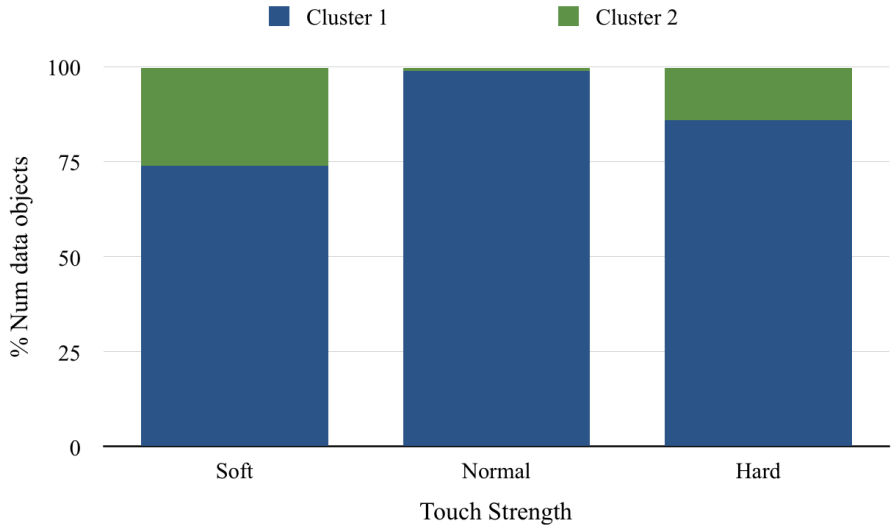

Figure 5.13: Touch Duration Clusters

Touch Horizontal Location Change

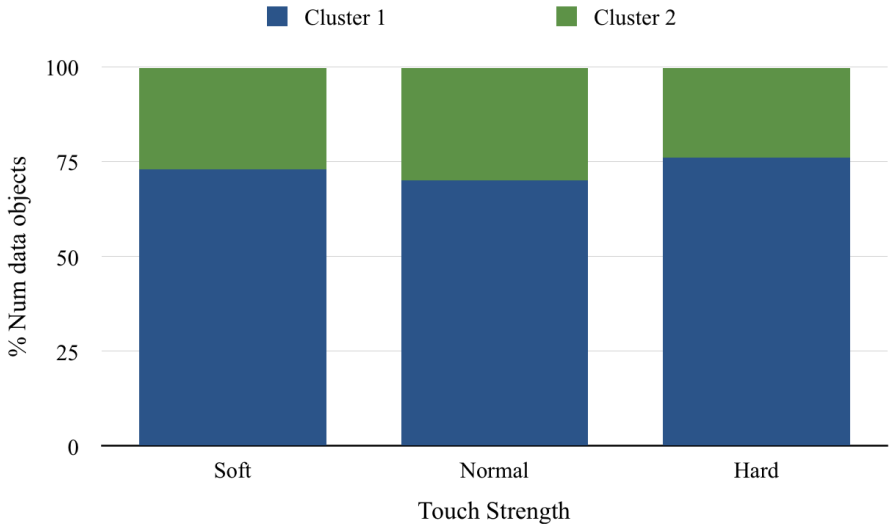

Figure 5.14: X Delta Box Clusters

Touch Vertical Location Change

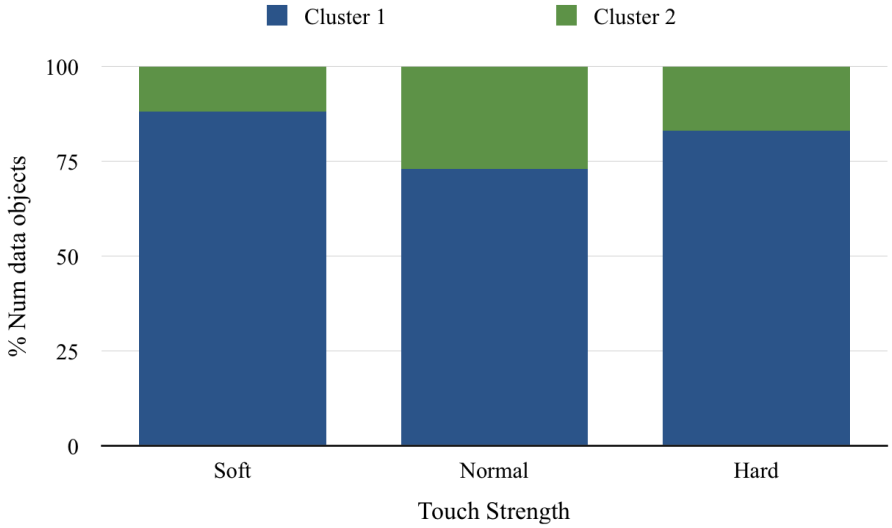

Figure 5.15: Y Delta Clusters

5.4 Algorithm Comparison Results

This section presents the passive and active algorithms comparison results. The results are shown in table and graph forms differentiated by algorithm and hardware architecture.

5.4.1 Arbitrary Touch Durations

Table 5.16 shows the average number of motion objects recorded for both algorithms running on hardware with the motion sensors data being calculated by the CPU and hardware with its own dedicated co-processor motion chip.

		Touch Duration (seconds)					
		0.05	**0.075**	**0.1**	**0.5**	**1**	**2**
CPU	**Passive Approach** (average motion objects recorded)	5	8	10	52	105	215
	Active Approach (average motion objects recorded)	0	1	3	48	100	210
Passive vs. Active (Number of motions passive records more than active)		**5**	**7**	**7**	**4**	**5**	**5**
Motion Co-Processor	**Passive Approach** (average motion objects recorded)	5	8	10	51	101	203
	Active Approach (average motion objects recorded)	0	1	1	43	92	194
Passive vs. Active (Number of motions passive records more than active)		**5**	**7**	**9**	**8**	**8**	**9**

Figure 5.16: Algorithms Motion Recording Table

Figures 5.17 and 5.18 visually represent the data in table 5.16. The figures do not display the two seconds duration as it makes the first 0.1 second difficult to differentiate.

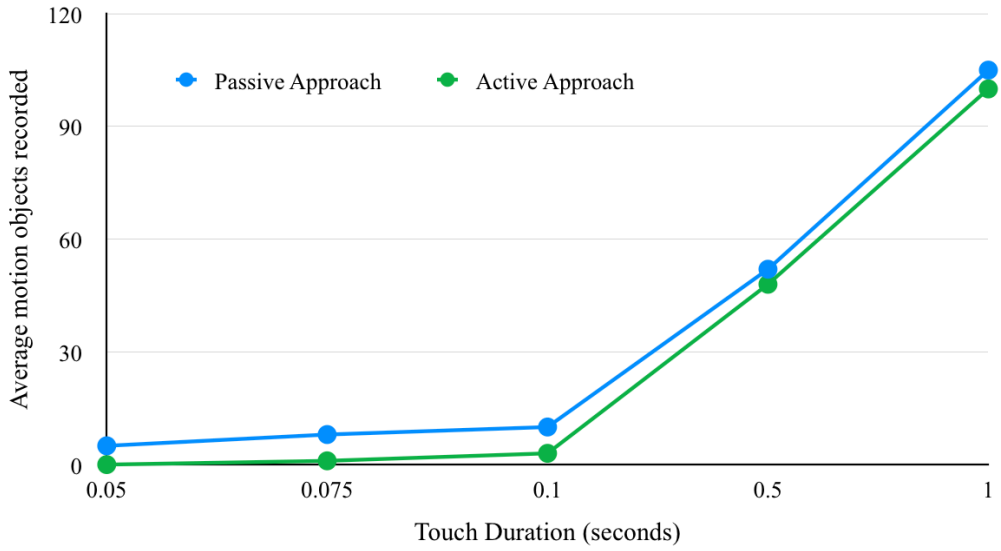

Figure 5.17: Motion Recording Quantity (CPU)

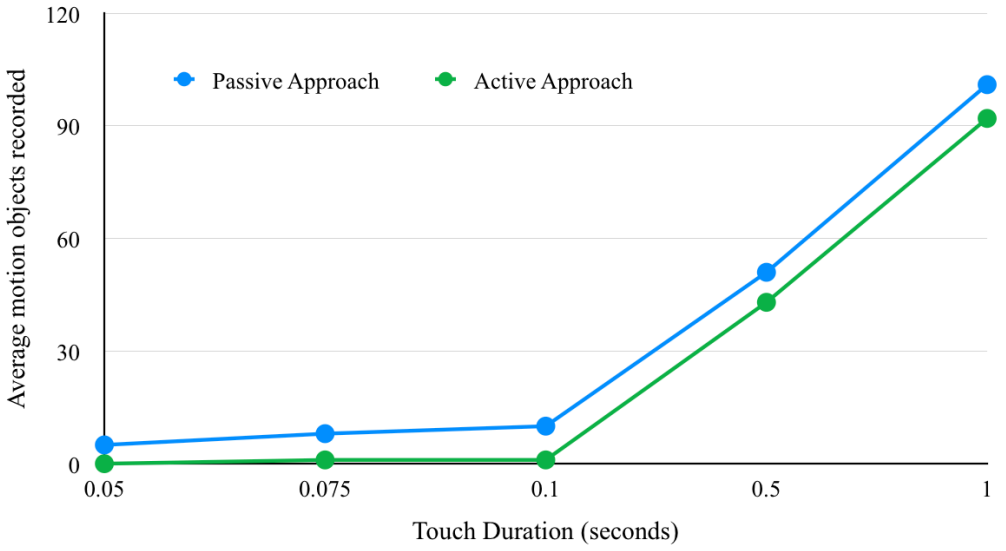

Figure 5.18: Motion Recording Quantity (Motion Co-Processor)

5.4.2 Touch Strengths Durations

Table 5.19 contains the results from running the same test used for table 5.16. However, the durations tested match the mean values of the soft, normal and hard touch strengths.

		Touch Duration (seconds)		
		0.102 Normal Touch Mean	**0.122** Soft Touch Mean	**0.246** Hard Touch Mean
CPU	**Passive Approach** (average motion objects recorded)	12	13	27
	Active Approach (average motion objects recorded)	5	6	20
Passive vs. Active (Number of motions passive records more than active)		**7**	**8**	**8**
Motion Co-Processor	**Passive Approach** (average motion objects recorded)	11	13	26
	Active Approach (average motion objects recorded)	1	4	13
Passive vs. Active (Number of motions passive records more than active)		**10**	**9**	**13**

Figure 5.19: Algorithms Motion Recording Strengths Table

Figures 5.20 and 5.21 visually present the data in table 5.19.

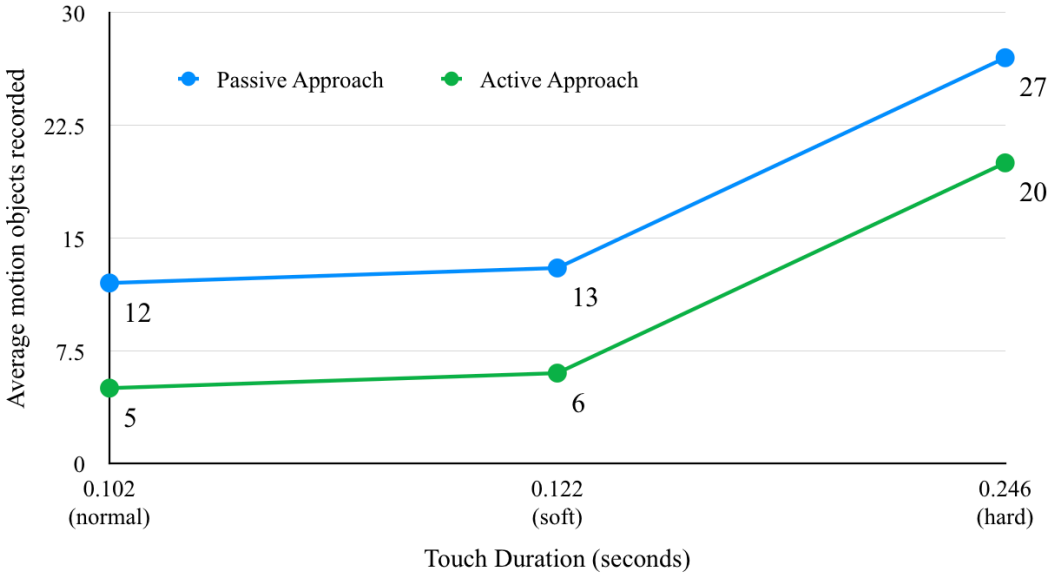

Figure 5.20: Motion Recording Quantity (CPU)

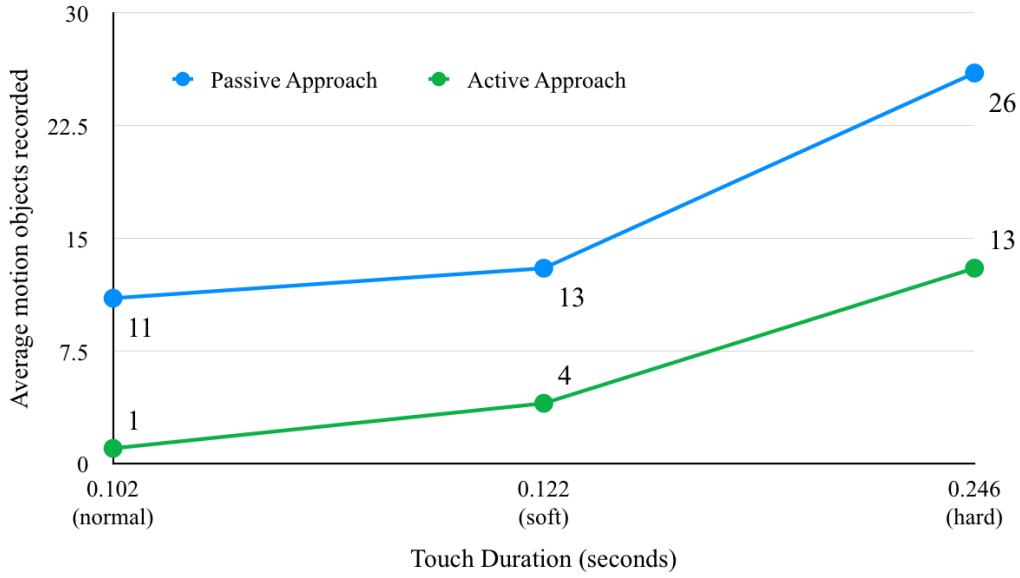

Figure 5.21: Motion Recording Quantity (Motion Co-Processor)

5.5 Results Summary

This chapter presented the demographics of the population sample. The summary objects were processed to produce a table to display the useful non-parametric statistical results. Results from the Kruskal-Wallis and Mann-Whitney difference tests were also shown for each of the variables along with their statistical touch strength differences.

The data was also visually described through box-plots to provide a clear distinction between touch strengths values. Clusters can be seen on the box-plots and thus a k-means clustered set of results was also tabulated and depicted in bar graphs.

Lastly the algorithms' motion recording performances were compared using arbitrary time durations as well as averaged touch durations. The comparisons were conducted on hardware architectures where the CPU processes the motion data as well as on hardware with a dedicated motion co-processor.

These results are analysed, discussed and related back to the aim of this project in the next chapter.

CHAPTER 6

Discussion

This chapter discusses the results of this study with reference to the aim to address the hypotheses made in previous chapters. The passive and active algorithms are compared and their differences discussed to justify an optimal approach. Moreover, the touch data results are evaluated with regards to their usefulness to the algorithm. This chapter builds toward the conclusions of the research and to the answers for the hypotheses; discussed in the next chapter.

6.1 Optimal Algorithm

Both active and passive algorithm approaches attempted to satisfy the requirements described in section 3.1. The concurrency, use of well defined data structures, and immediate touch feedback requirements are met by these two algorithms. They differ in their respective performances in balancing inexpensive computations and recording a sufficient quantity of motion data in a touch's lifetime.

6.1.1 Algorithm Approaches Compromises

The active algorithm is less computationally intensive as it does not require the motion sensors to be constantly active and recording data. However the cost of this optimisation is that the quantity of data recorded suffers with different degrees of magnitudes during the first hundredth of a second on different hardware implementations as shown in table 5.16. This also hinders the quality of the data as motion data matching the start of the touch is not recorded by the active algorithm. The active algorithm approach performs best when recoding durations of half a second and above on a controlled hardware.

In contrast the passive algorithm guarantees a constant quantity and quality of data independent of time and hardware variations. The passive approach is

more complicated by design and requires that the data structure used: a linked list, occupy its own concurrent queue. This approach makes a space complexity sacrifice to resolve the sensor activation time delay. The use of the linked list to store a continuous influx of motion objects allows the motion data to always be current at the cost of storing them in memory until they are no longer valid and purged. The purging logic can be found in Appendix A.1. Table 5.16 shows that neither hardware structures experience a time delay before recording the motion objects. Another drawback of the passive approach is that it is more computationally intensive. Having the sensors on for the duration of the algorithm as well as continually recording the motion data makes this approach more computationally expensive than the active approach.

While both approaches meet the set requirements, they each prioritise different aspects. The resulting effect makes the applications of these algorithms optimal for applications which exploit their respective strengths.

6.1.2 Arbitrary Durations Algorithm Comparison

Table 5.16 demonstrates that the first 0.1 second of motion recording is not handled well by the active approach. This is true for both CPU processed motion data and co-processor processed motion data. However, the CPU structure has a smaller time delay in starting the motion recording than that of a dedicated pre-processor. The time delay of the CPU structure is due to the time taken to turn the sensors on. The instant a touch strikes the display, the active algorithm turns the motion sensors on. However by the time the algorithm begins to store motion data it has missed the initial touch motions. At 100Hz, if the sensors take 0.1 seconds to activate, then approximately 10 raw motion objects are not recorded.

The motion co-processor is a separate chip connected to the CPU by a physical bus. It is thought that the time delay is caused by the time taken for the co-processor chip to turn the sensors on, combined with sending the data back to the CPU on the connecting bus. Having a separate co-processor allows the motion data to be processed without using CPU clock ticks, and could prove to be extremely beneficial in applications which require the motion sensors to be on for extended periods of time.

Figures 5.17 and 5.18 show that the active approach captures motion data at a similar rate as the passive approach after the initial time delay. However, the gap in the number of objects initially recorded is not recovered. It also shows that the number of motion objects recorded by the passive approach is not affected by a change in hardware structure.

6.1.3 Strength Durations Algorithm Comparisons

After processing the data gathered in the population sample, it is possible to compare the algorithms using the average times for each touch strength. Table 5.19 shows that the active algorithm performs poorly for the soft and normal touch durations. This is due to the average duration for these touch strengths coincide with the time taken for the sensors to activate. Therefore the passive approach performs significantly better than the active approach.

The passive approach records approximately the same number of motion objects independent of the hardware structure. Furthermore, it records a predictable number of objects for a given duration. In this case, at close to the 0.1 second mark, the algorithm recorded just above ten objects on both hardware structures. This is due to the fact that this algorithm turned on the motion sensors and began recording data before a touch interacted with the screen. Therefore even in short time bursts such as the soft and normal touches, a reliable quantity and quality of motion data was gathered.

The figures 5.20 and 5.21 both show that with smaller durations, the difference between algorithms is non-negligible. The charts also show that on a smaller duration scale, the CPU structure is more beneficial to the active algorithm approach and could possibly be used to differentiate between a soft and a normal touch. However, the active approach on a motion co-processor structure records an insufficient quantity of motion data to discern between soft and normal touches.

6.2 Universal Touch Perception

The hypothesis from Chapter 4 is that individuals have differing perceptions of a touch's strength. The results indicate that, whilst the data is skewed, there appears to be a universal perception of soft, normal and hard touches. The Kruskal-Wallis and Mann-Whitney test results in table 5.2 suggest that there is indeed a statistical difference between the different touch strengths for each variable. This indicates that there are no overall statistical overlaps between a soft, normal and hard touches.

This difference can be visually demonstrated by analysing the box-plots in section 5.2.1. For the average acceleration, average rotation, and maximum radius variables, the lower and upper quartiles of the data do not overlap. In addition, the values of each of the variables increase as touch strength increases. The correlation between the motion variables and the touch's strength is unsurprising as

the momentum with which a touch strikes the display is transferred to the device itself, affecting its rotation and acceleration accordingly. The maximum radius values for a touch also increase as the touch strength increases with significant differences between strengths.

The other variables are less correlated with the strength of the touch. The duration of a hard touch is longer than a soft and normal touch, although there is no clear distinction between a soft and normal touch. The horizontal and vertical displacement of the touch increases marginally, as the touch strength increases however the difference is not as obvious as with the acceleration, rotation or maximum radius variables.

6.3 Higher Values Clusters

From the box-plots (figures 5.3 to 5.8), clusters can be identified in the higher values of most of the graphed variables. Table 5.9 displays the results of clustering the data using the k-means clustering technique [16].

Variables with uneven clusters are the most reliable to use in the algorithm. Evenly split clusters such as the ones identified in the soft and normal touch strengths in figure 5.12 indicate that there are at least two strong groups gravitating towards their respective centroids. Having such a separation in the data for a particular touch strength makes it difficult to normalise it when calculating the overall scalar touch strength. Therefore, whilst there might be a difference between touch strengths, the majority of the data within each touch strength needs to gravitate towards a single centroid for the variable to be useful to the algorithm's calibration.

It can be observed by examining figures 5.10 to 5.15 that all variables except from the maximum radius have a distinct major centroid capturing a large percentage of data objects. Figure 5.12 shows that the soft and normal strength clusters for the maximum radius variable are divided into two distinct clusters capturing a similar amount of data objects. Therefore, whilst this variable has clear differences between touch strengths it does not have uneven clusters and is not fit to use in the calculation of a touch's strength.

6.4 Future Work

There is the potential for further work on the algorithm and statistical analysis introduced in this project. Future work has the potential to improve the

algorithm, allowing it to register a series of complicated touches and gestures. Other work may focus on a more involved statistical analysis to produce more precise touch strength definitions from the data in order to better calibrate the algorithm.

6.4.1 Varied Touch Interactions

Support for other kinds of touch interactions such as long presses, pinches and swipes could be implemented in future iterations of the algorithm. This would allow for the standard gestures to have a strength aware counterpart adding an innovative way of interfacing with the software whilst remaining intuitive to the users. The problem these implementations face is finding a method for the algorithm to provide strength feedback throughout the touch. This would require constant processing of the touch and continuous strength feedback during the lifetime of the touch.

6.4.2 Fallback Mechanism

As discussed in sections 6.2 and 6.3, the algorithm relies most heavily on the acceleration and rotation caused by a touch to determine its strength. Therefore if the device is placed on a flat surface, the motion sensors would register little to no movement, matching it erroneously to a soft touch. The implementation of a fallback mechanism to handle such cases is a topic worth considering.

6.4.3 Further Statistical Analysis

The statistical analysis conducted in this chapter could be further developed. Future work on the analysis of particular demographic groups and their effects on each separate touch variable could be conducted to answer specific data mining questions. A wider population could be studied using the proposed algorithm allowing for more efforts to go into data gathering and statistical processing and clustering.

CHAPTER 7

Conclusion

This project demonstrates that it is possible to create a user calibrated algorithm to calculate the strength of a touch on a capacitive display device using the motions induced by the touch. However, the properties of the touch are not useful in reliably determining its strength. Following the results discussed in the previous chapter, an optimal algorithm is chosen. The user calibration of the algorithm is achieved through a population study defining the properties of soft, normal and hard touches.

Of the two algorithm implementations described in this project, the passive approach is the optimal selection to calculate the strength of a touch. The compromises of the passive approach discussed in the previous chapter are reasonable for the purposes of this project as the duration of a touch is short, especially in the case of soft and normal strengths. As shown in this paper, turning motion sensors on is an expensive task in relation to the duration of a touch, and time delays vary on different hardware architectures.

An algorithm which prioritises the recording of data and that guarantees an expected constant recording rate is favourable even at the cost of some computational efficiency and increased space complexity. Furthermore, the passive approach's motion object recording is not influenced by the change in hardware architecture. For these reasons the passive approach is the optimal algorithm to record a touch strength on a capacitive display.

The results from the population study defined the properties of soft, normal and hard touches, expecting to find different perceptions of touch strength amongst participants. However, it is found that the participants universally have similar perceptions for each touch strengths. From the results' evaluation it is concluded that a touch's average acceleration and average rotation are the most useful variables for the algorithm.

These findings suggest that the properties of a touch alone are not sufficient to calculate its strength on a capacitive display. The acceleration and rotation motions which it induces on the device are the most reliable source of data which

can be used to calibrate the custom made algorithm to calculate the touch's striking strength on the device's capacitive display.

Bibliography

[1] Apple Inc. "GarageBand" http://www.apple.com/au/apps/garageband/. Accessed: 08 August 2013.

[2] Apple Inc. "iOS 7" http://www.apple.com/ios/ios7/features/. Accessed: 09 August 2013.

[3] Apple Inc. *Concurrency Programming Guide*, 12 2012.

[4] Apple Inc. *Event Handling Guide for iOS*, 01 2013.

[5] Aviv, A. J., Sapp, B., Blaze, M., and Smith, J. M. Practicality of accelerometer side channels on smartphones. In *Proceedings of the 28th Annual Computer Security Applications Conference* (New York, NY, USA, 2012), ACSAC '12, ACM, pp. 41–50.

[6] Barrett, G. L., and Omote, R. Projected capacitive touch screens. *Information Display Magazine 26* (03 2010), 16–21.

[7] Bhowmik, A. K. Natural and intuitive user interfaces with perceptual computing technologies. *Information Display Magazine 29* (07 2013), 6–10.

[8] Bryman, A. *Social Research Methods*, 3rd ed. Oxford University Press, 2008.

[9] Cai, L., and Chen, H. Touchlogger: inferring keystrokes on touch screen from smartphone motion. In *Proceedings of the 6th USENIX conference on Hot topics in security* (Berkeley, CA, USA, 2011), HotSec'11, USENIX Association, pp. 9–9.

[10] Fischer, D. *Capacitive Touch Sensors: Application Fields, technology overview and implementation example.* Fujitsu, 05 2010.

[11] Goodman, L. A. Snowball sampling. *The Annals of Mathematical Statistics 32*, 1 (1961), pp. 148–170.

[12] Google. 2013. input events. (september 2013) retrieved september 17, 2013 from http://developer.android.com/guide/topics/ui/ui-events.html.

[13] Hoare, C. A. R. Communicating sequential processes. *Commun. ACM 21*, 8 (Aug. 1978), 666–677.

[14] Liao, T. *Statistical Group Comparison*, 1st ed. Wiley, 2011.

[15] Lynch, S., Rappoport, B., Rothkopf, F., Drzaic, P., and Myers, S. Embedded force measurement, US20130135244 A1, May 2013.

[16] Mirkin, B. *Clustering A Data Recovery Approach*, 2nd ed. Chapman and Hall, 2012.

[17] Samsung. "Samsung Galaxy S4 - Air View/Air Gesture" http://www.samsung.com/au/galaxy-s4/lifetask.html. Accessed: 09 August 2013.

[18] TestFlight. 2014. testflight. (april 2014) retrieved april 19, 2014 from https://www.testflightapp.com/.

[19] Timnat, S., Braginsky, A., Kogan, A., and Petrank, E. Wait-free linked-lists. In *Proceedings of the 17th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming* (New York, NY, USA, 2012), PPoPP '12, ACM, pp. 309–310.

APPENDIX A

Algorithms Logic

A.1 Data Structure Reseting

Algorithm 6 Reseting Data Structures

```
1: function resetDataStructureIfPossible
2:     boundaries ← false)
3:         ▷ numActivities & numListeners ensure that no active touches are still present
4:     if numActivities = 0 && numListeners = 0 then
5:         emptyDataStructure(currentDataStructure)
6:         didReset ← true
7:     end if
8:     return didReset
9: end function
```

A.2 Motion Sensors Stopping

Algorithm 7 Stopping Motion Sensors

```
1: function stopDeviceMotion
2:         ▷ numActivities & numListeners ensure that no active touches are still present
3:     if isDeviceMotionActive() && numActivities = 0 && numListeners = 0 then
4:         stopDeviceMotionUpdates
5:         resetDataStructureIfPossible
6:     end if
7: end function
```

A.3 Normaliser

Algorithm 8 Values Normaliser

```
1: function NORMALISEDVALUE(value, boundaries)
2:     boundaries ← sortedAscending(boundaries)
3:     numBoundaries ← count(boundaries)
4:     if numBoundaries <= 2 then
5:         return value
6:     end if
7:     minBoundary ← firstItem(boundaries)
8:     maxBoundary ← lastItem(boundaries)
9:     sizeFactor ← maxBoundary/(numBoundaries − 1)
10:    if value < minBoundary then
11:        return minBoundary
12:    end if
13:    if value > maxBoundary then
14:        return maxBoundary
15:    end if
16:    normalisedValue ← 0
17:    i ← 0
18:    for i < numBoundaries, i ← i + 1 do
19:        upperBoundary ← boundaries[i]
20:        lowerBoundary ← boundaries[i − 1]
21:        if value > lowerBoundary&&value <= upperBoundary then
22:            bottomBoundary ← ((i − 1) ∗ sizeFactor)
23:            increment ← ((value/upperBoundary) ∗ sizeFactor)
24:            normalisedValue ← bottomBoundary + increment
25:        end if
26:    end for
27:    return normalisedValue
28: end function
```

APPENDIX B

Sampling Results

B.1 Summarised Touch Data Results Without Outliers

			N (instances)	Mean	Standard Deviation	Variance	Median	Maximum	Minimum
Motion Data	**Acceleration Average** (G units, 1G = 9ms^{-2})	**Soft**	158	0.0327	0.0226	0.001	0.0268	0.1127	S : 0.0053
		Normal	162	0.1064	0.1025	0.10	0.0717	0.5370	0.0099
		Hard	161	0.2775	0.2535	0.064	0.1949	1.0697	0.0206
	Rotation Average (radian per seconds)	**Soft**	161	0.0911	0.0806	0.007	0.0648	0.4195	0.0074
		Normal	164	0.2650	0.1902	0.036	0.2254	0.9570	0.0257
		Hard	163	0.8188	0.6810	0.464	0.6524	3.0971	H:0.0373
Touch Data	**Duration** (seconds)	**Soft**	163	0.1220	0.0599	0.004	0.0958	0.2999	0.0315
		Normal	161	0.1017	0.0497	0.002	0.1112	0.3674	0.0322
		Hard	159	0.2457	0.2083	0.043	0.1831	1.1165	0.0163
	Maximum Radius (mm)	**Soft**	164	7.0285	0.7849	0.616	7.0010	9.9900	5.4100
		Normal	164	7.6212	0.9172	0.841	7.5450	10.6400	5.9100
		Hard	163	8.6445	1.4859	2.208	8.2700	14.1300	5.6600
	x Delta (points)	**Soft**	161	1.947	1.7586	3.093	1.5	7.5	0.0
		Normal	164	3.314	2.9234	8.546	2.5	14.5	0.0
		Hard	157	4.739	4.2756	18.281	3.0	18.5	0.0
	y Delta (points)	**Soft**	162	2.114	2.2230	4.942	1.5	12.5	0.0
		Normal	163	2.699	2.4169	5.841	2.0	12.0	0.0
		Hard	162	5.150	5.8956	34.758	4.0	26.0	0.0

Figure B.1: Non-Parametric Results Without Outliers

B.2 Summarised Touch Data Results

			N (instances)	Mean	Standard Deviation	Variance	Median	Maximum	Minimum
Motion Data	**Acceleration Average** (G units, 1G = 9ms^{-2})	**Soft**	164	0.0427	0.0592	0.004	0.0275	0.4410	0.0053
		Normal	165	0.1195	0.1400	0.020	0.0733	0.900	0.010
		Hard	164	0.3039	0.3212	0.103	0.1200	2.2190	0.0206
	Rotation Average (radian per seconds)	**Soft**	164	0.1005	0.1060	0.011	0.0684	0.6869	0.0075
		Normal	165	0.2710	0.2043	0.042	0.2256	1.2424	0.0257
		Hard	164	0.8380	0.7218	0.521	0.6525	3.9575	0.0373
Touch Data	**Duration** (seconds)	**Soft**	164	0.1320	0.1404	0.02	0.1113	1.7501	0.0321
		Normal	165	0.1303	0.2294	0.053	0.0986	2.5336	0.0316
		Hard	164	0.2855	0.3320	0.11	0.1914	2.3333	0.0163
	Maximum Radius (mm)	**Soft**	164	7.0285	0.7849	0.616	7.0010	9.9900	5.4100
		Normal	165	7.6450	0.9644	0.930	7.5500	11.5600	5.9100
		Hard	164	8.6870	1.5783	2.491	8.2850	15.6200	5.6600
	x Delta (points)	**Soft**	164	2.305	3.4475	11.885	1.5	36.0	0.0
		Normal	165	3.442	3.3488	11.215	2.5	24.5	0.0
		Hard	164	6.040	7.6607	58.687	3.5	48.5	0.0
	y Delta (points)	**Soft**	164	2.567	4.8260	23.290	1.5	51.0	0.0
		Normal	165	2.915	3.1360	9.834	2.0	25.0	0.0
		Hard	164	6.476	6.9806	48.729	4.0	47.0	0.0

Figure B.2: Non-Parametric Results