‘Real Time Magnet Position Detection by Smartphone for a Drawing Application’

A dissertation submitted in partial fulfilment of

The requirement for the degree of

MASTER OF SCIENCE in Software Development

in

The Queen’s University of Belfast

By

Jordan Shaw

13.9.2017

Acknowledgements

I would like to thank Dr Darryl Stewart for all his support in completing this dissertation, Tetsuya Abe et al, Saad Ismail, Mark Eisenberg and Pavel Karpashevich for their previous work in investigating the use of magnets as an input technique to smartphones, and my friends and family for their support and motivation.

Abstract

Input techniques to mobile devices are often dependent on screen-size, sensitivity of touch input or a required power source. These limitations have an impact on use of mobile devices for activities such as digital artwork. This dissertation attempted to test the hypothesis that magnets can be used as an input technique to mobile devices with the required sensor. The software produced tested this technique in the context of an Android drawing application.

Magnetic field readings were taken at each corner of a defined drawing surface. The readings were then used to estimate the magnet's position as it was moved on the surface. Various algorithms were tested to evaluate their accuracy in mapping the magnet's on-page position to an onscreen position. To improve the accuracy of field readings, the earth's field was accounted for and a low pass filter was applied to the results.

It was found that moving a magnetic stylus produces a discernible magnetic field change which can be detected by a mobile device. Using Coulomb's law for Magnetic Fields appeared to be the most accurate method of mapping sensor data to on-screen co-ordinates, however much more research is required in optimising this technique and improving the reliability.

Using a magnet as an input device has drawbacks, such as interference and the dependencies on device sensor sensitivity and orientation. The application developed needs to be used in a carefully controlled environment.

Contents

Introduction

The six chapters of this dissertation will present a clear, concise and fluent review of the development process and experiments undertaken in this project.

- Chapter 1 contains a specification of the problem domain, describing the target audience and existing work in the same area as this project.
- Chapter 2 outlines the proposed solution to the problem domain specified, along with the development model and technologies chosen.
- Chapter 3 deals with the requirements specification for the application.
- Next, Chapter 4 describes the entire application design in depth, including the User Interface and the Software System.
- The implementation of the system, methodologies and testing strategies are then described in chapter 5.
- Chapter 6 contains an honest, detailed evaluation of the software environment used and the experiments undertaken and suggestions for further work.

Decisions made are justified and progress is evaluated throughout this dissertation with reflections made in detail in Chapter 6.

The application developed was given the name *MagnetArt*, and is referenced as such in this dissertation. It is presumed that the user has some knowledge of object-orientated software development and Android Development. The project commenced on 19th June 2017, with a deadline of 13th September 2017. The project was demonstrated at an expo at Queen's University Belfast on 14th September 2017. Development was carried out by one student, Jordan Shaw, under the supervision of Dr Darryl Stewart.

Chapter 1: Problem Specification

1.1 The Problem

We live in a world today in which carrying a smartphone or tablet is for many people, a vital piece of everyday life. Over 77% of American adults [1] and 76% of UK adults own a smartphone as of January 2017 [2]. It is estimated that by 2018 over 2.5 billion people will own a smartphone [3], and over 1.3 billion will own a tablet [4]. These devices are no longer used only for communication, but apps have been produced which digitise tasks normally done by hand, such as note-taking and producing artwork.

This has many advantages, as digital copies are less likely to be misplaced, are easily shared and duplicated and may even free up physical storage space. Digital copies can be easier to edit or revise. Digital art can more readily be produced on the go, without the need to carry a sketchbook, pencils or paintbrushes, only a smartphone or tablet is needed.

Conversely, many studies have shown there are benefits to writing and drawing by hand. Writing by hand requires different cognitive processes than typing, and can aid learning by building a stronger conceptual understanding. Hence many students choose to revise by writing out revision notes by hand. Drawing a sketch requires a different skillset to producing a digital image. A study by researchers in Germany [5] found that being part of a group which learned painting and drawing techniques and producing original art resulted in ‘a significant improvement in psychological resilience’ and increased levels in ‘functional connectivity’ when compared to a group which only evaluated the artwork. Drawing by hand also improves motor skills by use of different tools and developing hand-eye co-ordination.

The problem therefore, is preserving the benefits of manual drawing when producing digital artwork. From the user’s perspective, there can seem to be a disconnect between traditional and digital drawing. When drawing with a pencil, naturally our eyes follow our hand and then our subject, and back to our hand. While drawing digitally using a graphics tablet, our eyes barely leave the screen because there is no image to be seen on the tablet. When using a tablet and stylus, the user will need to adapt to the sensitivity of the pen and the surface of the tablet to ensure hand-eye co-ordination. Drawing onscreen using a fingertip is not as precise as drawing with, for example, a pencil or paintbrush, due to the fingerprint being larger than the point of drawing. While drawing the finger may also cover the drawing point. The drawing process is often hindered by the input technique to the device.

This project aims to bridge the gap between ‘traditional’ and digital drawings - preserving the benefits of both and reducing the limitations. The target audience for the project comprises of individual users who wish to produce drawings on their mobile device display. The application is not only for designers or artists, but is designed to also be suitable for users with a limited artistic background.

1.2 Existing Related Applications and Devices

Various applications and devices have been made to try to bridge the gap between hand drawings or notes and the digital equivalent on tablets and smartphones. In general, these attempt to give the user the feeling of drawing by hand, while producing an image on the device screen. A number of these approaches have been listed below. They have been grouped into three categories.

1. **Drawing applications** which involve onscreen drawing with the fingertip. Many drawing apps exist on all mobile platforms. In these applications, a line is drawn following the motion of the touch input. These apps involve various features to mimic the artwork produced by paintbrushes, pencils, pastels with user defined stroke width and colour.
   These applications have the drawback mentioned previously, that the fingertip is less precise than a pencil or paintbrush, as well as being limited by the screen dimensions.
   One such application is **Sketchbook** [6]. This app, available for Windows, Mac, iOs and Android, gives the user a choice of 170 customisable brushes, designed to blend or texture in a natural way. The user interface is very empty, allowing the majority of the screen to remain blank for use as a drawing canvas.

2. As an addition to applications mentioned above, a **stylus** may be used onscreen. The stylus is held like a brush or pencil, meaning that it feels like a traditional piece of art equipment. The tip of the stylus allows more precision than a fingertip. The stylus does not remove the limitation of the screen size.

3. **Smartpens.** These are digital pens which allow the user to produce handwritten notes or sketches on a notebook while synchronising them with a device to save a digital copy. This means that the user will be able to keep a hard copy of their note or sketch, while a version will be backed up onto their device. Various methods have been used to allow the device to estimate the pen's location on the page. Some examples are:

   - The **Livescribe Smartpen 3** [7] writes like a ballpoint pen and contains an infrared camera. The camera will detect the pen's position when the pen is used on Livescribe's custom paper containing a dot matrix pattern. The position is transferred to the device using Bluetooth along with the companion app.
   - The **Equil Smartpen 2** [8] also writes like a ballpoint pen and uses Bluetooth technology. However, unlike the Livescribe Smartpen, it allows the user to write on any paper. A receiver is clipped at the top of the page used before writing.

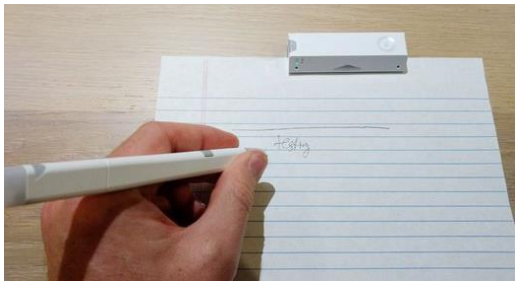


Fig.i- The Equil SmartPen 2 [9]

Chapter 2: Proposed solution and justification of the development model

2.1 Proposed Solution

This project aims to explore using magnetism as an input technique to a mobile device. The magnetometer sensor in a smartphone or tablet uses a Hall Effect Sensor to detect the Earth's magnetic field in the x, y and z axes. This sensor consists of a semiconductor which has a continuous current passing through it. When the device is placed inside a magnetic field, the magnetic flux exerts a force on the material, deflecting the charge carriers in the semiconductor to either side of the semiconductor slab. The movement of positive and negative charge carriers to either side of the slab results in a potential difference, producing a measurable voltage. This is known as the Hall Effect. The voltage is proportional to the magnetic field strength and polarity along the axis each sensor is directed.

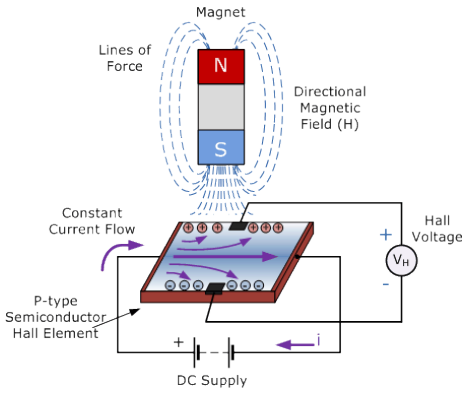


Fig.ii – A Hall Effect Sensor [10]

Generally, the magnetometer is used as a compass, detecting the relative orientation of the device to the Earth's magnetic north. However, use of this sensor need not be limited to use as a compass. If a magnet is placed near the mobile device, the device will detect magnetic field changes in three dimensions. Likewise, if the magnet is moved to a new position, the magnetic field readings will change. By defining limits to the space in which the magnet can move, and by considering the geometry of the system, an algorithm can be applied to transform the magnet field readings into an estimate of the absolute position of the magnet.

In the context of a drawing app, this means that if a stylus with a magnet on the tip is moved within the limits of a defined drawing surface, the device will be able to detect the changes in position of the stylus and by estimating the stylus' position, use this data to produce a drawing onscreen to match the movement, as with the Smartpens mentioned in Chapter 1. Therefore, this project aims to produce a drawing application which uses a specifically designed magnetic stylus as input.

The device will be placed on a flat surface beside the drawing area when using the application. Magnetic field readings will be taken at various locations to calibrate the device with the size of the

drawing area. Once the area is calibrated, the user will be able to draw onscreen as they move the stylus within the defined limits of the drawing area.
Various algorithms to transform the magnetic field readings into positions will be tested, and the results from the experiments will be discussed and evaluated in *Chapter 6*.

By using a magnet as the input, this will remove some limitations from other methods of producing digital art mentioned above. For example, there will be no need for any additional sensors to the device. The magnetometer is housed in the device so the stylus will work without the need for a separate receiver. Unlike the Smartpen devices mentioned, the magnetic stylus will not require a battery. The stylus will be handheld like more 'traditional' art utensils.
The drawing area will not be limited to the dimensions of the device display. It will depend on the size of the surface defined when calibrating.

However, due to the nature of the magnetometer, there are a number of potential limitations with this project. As the app will rely on accurate magnetometer readings to determine the stylus position, it will not be able to be used in an environment where other magnets are present. The app will also have to be calibrated again if the page used is moved slightly. The device and drawing area need to remain in fixed positions relative to each other. The app also relies on the sensitivity of the device magnetometer, which will vary from device to device. Generally increasing the magnetic field strength means increasing the size of the magnet but this could make the stylus bulky and uncomfortable to use.

While *MagnetArt* will be a drawing app, the primary focus of this project is producing an accurate drawing onscreen to match the movement of the stylus, rather than providing many artistic functions such as brushes and layers. It will serve as a concept demonstrator for a magnetic input technique and the required algorithms and considerations. Nevertheless, functions implemented in the application will include the ability to save and view drawings, change the stroke colour and to draw lines or shapes. The requirements for the app will be outlined in detail in *Chapter 3*.

2.2 Justification of the development model

An agile approach was selected. Agile development involves an iterative, flexible approach. According to Mountain Goat Software [11], the projects most suited to the agile process are those with 'aggressive deadlines, a high degree of complexity and a high degree of novelty to them'. The *MagnetArt* project fits within this description. An agile approach ensures working software throughout the project and the ability to adapt depending on the success of a previous section of the project.

Given this is an individual project, it was felt that following one agile model precisely would not be beneficial. Due to the project requiring technical learning and experimenting with various algorithms, the chosen methodology allows sprint lengths to vary if needed and requirements can be reviewed if it is found that a function is not feasible to implement. It was decided to follow a 'Design-to-Schedule' iterative approach. This was chosen due to the fixed deadline for delivery of the project and

prioritisation of the stages in development of the system, meaning the high priority features are implemented first. This fits with the agile principle that working software is the primary measure of progress [12]. To determine high priority features, the requirements were prioritised according to the MoSCoW Technique: grouping initially into 'Must-haves', 'Should-haves', 'Could-haves' and 'Won't-haves'. These priorities were subject to change depending on the feasibility of implementation.

An agile approach was chosen over a plan driven approach for several reasons: A plan-driven methodology requires a clear set of requirements up front, the design and implementation phases are documented and scheduled from the start and the plan is not subject to change. Due to the research nature of this project, the requirements, design and implementation needed to be reviewed regularly to ensure they were feasible, therefore an agile approach was preferred.

The requirements produced tasks according to the *User Stories* outlined in Chapter 3. The tasks were grouped into sprints. These are time-limited periods of development resulting in a working software artefact. The tasks were undertaken in sprints according to priority, ensuring the highest priority tasks are completed first. In this project, a sprint lasted two weeks, and there were 4 sprints.

A Kanban board was used to keep track of sprints. The board allows visualisation of the workflow and reducing the work in progress. Tasks were moved between lists on the board as they were worked on, completed, or if their priority was changed. This approach has the advantage of reduced risk and early feedback since during each sprint only a few requirements were handled.
It was acknowledged that a lack of accuracy of determining the magnet position may lead to a change in requirements. It was also acknowledged that due to the research nature of this project, time should be allocated to allow several algorithms used to estimate the magnet position to be tested and evaluated.

2.3 Justification of the development platform

The development platform chosen was Android for several reasons. The Android platform is designed for a wide range of smartphones and tablets. The Android OS is open source and is the biggest mobile platform in the world. Android provides Application Programming Interfaces (API's), which are clearly defined communication methods between software components. One such API allows access to the device's sensors. This is very useful for this application, which requires access to the mobile device's magnetometer.

The main functionality was written in Java. Java is the official language of Android development, meaning it has the most support. There is a large amount of documentation available. In addition, Java is portable and has Object Oriented Properties. While languages such as C do not require the extra layer of the Java Virtual Machine, making them faster at runtime, the other advantages of Java outweigh this. In this application, the Object-Oriented Principles of Java lead to greater code reuse and efficiency.

The layout of the application was defined using XML. This means the code defining the presentation of the app can be kept separate from the Java code defining its behaviour. The XML code can be modified without the need to adapt the Java code. XML is very readable, separating each element into a list of attributes.

Android Studio was chosen as the IDE. Android Studio is a unified environment which allows development for all Android devices. All related files are encapsulated into a project, including the Java and XML files, directories and Gradle scripts. It also includes tools to debug and monitor a running application. In addition, the Android Studio SDK manager allows downloads of the necessary components to develop for each API level. Therefore, it was decided that the application should be developed on Android Studio, which is the official Android IDE, instead of an IDE such as Eclipse, which does not receive the latest IDE updates.

A Git repository was used to back up and version manage the project. In terms of configuration management, code was appropriately commented and sensible naming conventions were used for classes and documentation.

Chapter 3: Requirements analysis and specification

This chapter describes and justifies the process by which requirements for this project were elicited, specified and prioritised. The requirements are listed along with function definitions, the data model for the application, use cases and a retrospective requirements review.

3.1 Definitions

For sake of completeness and clarity, we here include some definitions. **Requirements engineering** is the overall process used in identifying and communicating the purpose of a software system, and the contexts in which it will be used. A **requirement** is a condition needed to solve a problem or achieve an objective that must be satisfied by the system or system component. **Elicitation** is the process through which the requirements are gathered and understood. The requirements are then documented clearly and completely in the process of **requirements specification.**

The established international standard for Requirements Engineering is ISO/IEC/IEEE29148:2011(E) [13], which describes guidelines for best practice. While the contents of this document are applicable to this project, it was felt that that due to the level of detail, it was unnecessary to use the standard in full given that only one developer was involved and one system was to be delivered. Although the standard was not followed in full, its guidelines were used to ensure the set of requirements produced was complete, consistent, affordable and bounded.

3.2 Requirements Elicitation

Elicitation began at the start of the project with the first meeting with supervisor on 19th June 2017. Generally, requirements engineering involves gathering requirements from clients and end-users of the system. As this system can be considered a concept demonstrator, there was no 'client', so-to-speak, to define the requirements. However, requirements were gathered from the problem description, meetings with the project supervisor and consideration of existing related applications and devices. Most of the requirements were elicited at the start of the project. At each meeting with the project supervisor, some functions were deemed unfeasible for the project, while occasionally new requirements were added during a sprint. This is in keeping with the iterative, agile approach chosen.

The requirements were divided into functional, non-functional and user interface requirements and were documented as user stories.

3.3 User Stories

User stories are a form of agile requirements specification. They are a concise written description of a piece of functionality valuable to a user. Each user story is written in one or two sentences in everyday language, meaning they are understandable to developers and users without being overly

technical. This also means they are easier to modify when requirements change than a more formal specification method. Each user story gives a unit of work that can be delivered in a sprint.

User stories are written in this general format: *As [role], I can [feature] so that [reason].*
Each user story needed to be testable to ensure the requirement was met. This was achieved by recording acceptance criteria in this format: *Given [initial condition], when [an event occurs], then [outcome produced]*

Initially some user stories were large and needed to be broken down into smaller pieces of functionality which each produced a single user story. The user stories form a product backlog. The user stories are divided up into sprints based on their priority and for each sprint the relevant user stories form the sprint backlog. The items in the sprint backlog will be completed during the sprint. To complete each user story, the story was broken down into several tasks that the developer needed to complete to satisfy the user story.

All tasks were tracked using a Kanban board, with each task given a separate card. The board was set up on Trello and included five lists:

- 'Backlog' - a list of tasks to be completed in future sprints
- 'Sprint' - tasks to be completed in the present sprint
- 'In Progress' - tasks currently being worked on
- 'Done'- tasks which have been completed
- 'Extras'- these are tasks which have come from 'Could have' requirements and are not essential but will be added if time allows

Each card contained a short description of the task, which sprint the card belonged to and the associated user stories. A screenshot of the Kanban board used is shown in Appendix D.

3.4 Requirements Prioritisation

In keeping with the design-to-schedule approach, the requirements were prioritised using the MoSCoW method as specified. The top priority requirements were the 'Must-haves' and highly desirable requirements were 'Should-haves'. Requirements which were desirable but less important to the system were 'Could-haves', and were implemented if time allowed. 'Won't-haves' were requirements which were deemed unfeasible. Initially there were no 'won't-haves', but some requirements were reclassified as such during the sprints. Reclassification allowed focus to be kept on higher priority requirements.

3.5 Requirement Attributes

To manage and trace requirements they were given the following attributes:

1. *ID* – a unique identifier
2. *Date Created*- this shows if the requirement was original or a later addition,
3. *Category*

4. *Description*
5. *Priority*-according to MoSCoW method
6. *Story Points*- a relative estimate of the amount of time required, values follow the Fibonacci-like sequence 0, 0.5, 1, 2, 3, 5, 8, 13, 20
7. *Acceptance criteria*- how to test the requirement has been implemented
8. *Sprint*- the sprint that the required will be completed in
9. *Completed*-whether the requirement has been implemented
10. *Tested*- whether the implementation has been tested
11. *Comments*- additional details

3.6 Iterative Requirements Review

The requirements were reviewed at the end of each sprint. There were no meetings with an end user, so the requirements were not changed to reflect their needs. Due to the research nature of this project, requirements sometimes were changed based on their feasibility. The accuracy of the magnet position detection was a limiting factor. During regular meetings with the project co-ordinator the feasibility was considered. The project co-ordinator could evaluate and suggest requirements, and so in that regard may be considered a client.

3.7 List of User Stories

Below are lists of requirements for the project with selected attributes. The requirements are separated into functional and non-functional and are grouped by categories within each table. The tables show a final list of requirements after modifications, additions and removals. The first 5 stories are shown for each table and the remaining functional and non-functional user stories are in the appendix.

User Stories - Functional Requirements

ID	Category	Description	Priority	Acceptance Criteria
1.1	Magnet Detection	As a user, I can move the stylus to cause the application to detect changes in the magnetic field.	Must-have	Given the magnetic stylus is moved near the device, the app will receive callbacks from the device magnetometer due to the changing magnetic field readings.
1.2	Magnet Detection	As a user, I can view magnetic field readings on the device display.	Should-have	Given that the application is receiving magnetic field readings from the magnetometer, these will be displayed to the user.
1.3	Magnet Detection	As a user, the raw readings must be corrected for the terrestrial magnetic field before I can draw.	Must-have	Given the application is receiving magnetic field readings, the application will use these readings to estimate the earth's magnetic field before the user can draw.
1.4	Magnet Detection	As a user, I can place the magnetic stylus in each corner of the drawing surface and store the magnetic field	Must-have	Given the drawing screen is displayed, the user can place the magnetic stylus at each corner of the drawing surface and press a button to save the magnetic field readings at each corner.

		readings taken in order to calibrate the application.		
1.5	Drawing	As a user, I can open a drawing canvas on which to draw.	Must-have	Given that the home screen of the activity is displayed, the user can press a button which will open the new drawing screen which includes a drawing canvas.

User Stories - Non-Functional Requirements

ID	Category	Requirement	Priority	Acceptance Criteria
2.1	Permissions	As a user, I will be able to write to and delete from storage as the application has the required permissions.	Must-have	The application can save images to device storage and delete saved images.
2.2	Permissions	As a user, I can move the stylus which can be detected because the application has the required permission to access the device's magnetic field sensor	Must-have	The application can access the uncalibrated magnetic field sensor of the device to receive magnetic field data.
2.3	Magnet Detection	As a user, I can draw with the onscreen co-ordinates matching the stylus' relative position on the drawing surface to an error of less than 10%.	Must-have	Given that magnetic field readings have been taken in each corner of the drawing surface, an algorithm will be applied to the current magnet field readings to calculate the corresponding screen co-ordinates.
2.4	Screen Orientation	As a user, when I use the application it always be displayed in portrait orientation	Should-have	The application will always be displayed in portrait orientation regardless of the rotation of the device.
2.5	Compatibility	As a user, I can run the app on Android devices which have a magnetometer and run at least API 18.	Must-have	The application will run on devices which run at least API 18. This API is needed to access the uncalibrated magnetic field sensor of the device.

User Requirements - User Interface Requirements

ID	Category	Requirement	Priority	Acceptance Criteria
3.1	User Interface	As a user, I will be presented with dialogs when appropriate when navigating the application.	Should-have	Dialogs are shown to the user when a decision must be made or to show that a function is loading.
3.2	Navigation	As a user, I can navigate between different functions of	Must-have	Given the application is running, the user can access a menu with various buttons depending on the current screen.

		the application using menus		
3.3	User Interface	As a user, the UI behaves as expected	Should-have	The whole interface behaves as the user expects.
3.4	User Interface	As a user, the UI feels familiar.	Should-have	The whole interface is familiar to the user.
3.5	User Interface	As a user, the UI is intuitive.	Should-have	The whole interface is intuitive to the user.

3.8 Function Definitions

Functions are the fundamental actions that must be carried out by the application in response to inputs to produce the appropriate outputs. Functions includes pre-conditions which must be met before the function can execute and post-conditions which are true after the function executes. The following table of function definitions is derived from the list of functional requirements.

Function	Name	Pre-Conditions	Inputs	Processing	Output	Post-conditions	Error Checking
1	Select image from gallery	Home screen displayed, at least one image has been saved	A touch event on the image in scrollView	The path of the image is used to create a new bitmap. The bitmap is scaled and placed in an imageView which is then displayed in a frameLayout	The selected image is displayed in a frameLayout below the scrollView. The delete floating action button is also displayed.	Selected image and delete floating action button are displayed, a new bitmap has been created	If the image cannot be displayed, an error message is displayed in a Toast.
2	Delete image	Image has been selected and the delete floating action button is visible.	A touch event on the delete floating action button and on delete dialog	A dialog is displayed to check that the user wishes to delete the image. The path of the selected image is used to locate and delete the image from memory.	A dialog is displayed to confirm deletion. When confirmed, a toast is displayed to confirm that the image has been deleted. The image is no longer displayed.	The image has been deleted from device memory. No image is selected and the frameLayout is empty. The dialog is dismissed.	If the image cannot be deleted, an error message is displayed in a Toast.
3	Open help screen	Home screen displayed	Touch event on menu item	A fragment transaction is used to display the help screen fragment	The help screen is displayed	The help screen fragment has been added to the view.	If the help screen cannot be displayed, an error message is displayed in a Toast.
4	Open settings	Home screen or new drawing screen displayed	Touch event on menu item	An intent is used to start the settings activity.	The settings screen is displayed	The settings activity has been launched	If the settings screen cannot be displayed, an error message is displayed in a Toast.
5	Choosing an algorithm from settings	Settings screen is displayed	Touch event on radio button	When the user presses the radio button corresponding to an algorithm, the integer corresponding to the chosen algorithm is saved in the application SharedPreferences.	A toast is displayed to confirm that an algorithm has been chosen	The chosen algorithm is saved as an integer in SharedPreferences and retrieved in the new drawing fragment.	If the algorithm cannot be saved to or retrieved from shared preferences, an error message is displayed as a toast.

3.9 Use Case Diagram

The functional requirements were represented by means of a use case diagram. This aided visualisation of the system by grouping functions together and demonstrating flows of actions that the user could take. The model also aided grouping requirements into sprints.

Fig.iii Use Case Diagram

3.10 Data Model

No database was required for this project. However, the application does handle data when storing images to device storage and viewing the saved images. Each image was given a unique file name in the format: **"magnetart-"**+mseconds+**".png",** where **mseconds** is the time at which the image was produced, accurate to the nearest millisecond.

The application also handles data from the magnetic field sensor. This data takes the form of uncalibrated magnetic field readings in microTesla in 3 dimensions. In order to manage this data, an object, MagnetPosition, was created which stores all 3 readings into an array (x,y,z). This meant that operations to be performed on the object could be defined within the MagnetPosition class. Calculations needed to be performed on readings to calibrate the drawing surface were kept in a separate class for the sake of code clarity.

Chapter 4: Design

This chapter will outline the design of the application. It is divided into two main sections: The User Interface Design and the Software System Design.

4.1 User Interface Design

It is important for the interface for the application to look simple and intuitive to use. While calculations were used to handle the magnetic field data and estimate the screen co-ordinates, the logic and the user interface were kept separate.

4.1.1 UI Design Decisions

It was decided to create various menus throughout the application to aid navigation. In the home screen and the help screen, the menu is a dropdown. However, in the new drawing screen, it was decided to place some menu items along the actionbar at the top of the screen. These menu items allowed the user to change the shape being drawn and change the stroke colour without needing to use a dropdown.

The application remains in portrait mode when drawing regardless of the orientation of the device. When drawing the device will be placed on a flat surface before use so it was deemed unnecessary to allow screen rotation, as this will restart the running activity and lead to loss of the image being drawn.

The calculation results were made visible to the user by pressing the relevant menu option when drawing. This was due to the magnet interaction being a concept demonstrator, and aided testing on a device.

The UI and application logic were kept separate where possible. In cases where a UI button would handle magnetic field data, such as when calibrating the drawing surface, the data is sent elsewhere to be handled.

The design of the home screen was inspired by existing note taking apps which allow the user to scroll through and view previously saved notes, with a floating action button used to add new notes.

4.1.2 Justification of Font and Colours

The look and feel of the user interface focused on clarity and simplicity. The base colour is blue, which is used for all toolbars. A red colour was chosen as the accent colour, and therefore is the colour of floating action buttons and dialog text. These colours match the red and blue of the traditional horseshoe magnet, used in the app icon.

Fig.iv- MagnetArt application icon

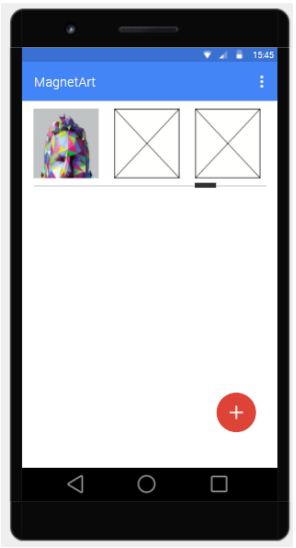
Care was taken to ensure that a red element is not set on a blue background or vice versa, as this can be straining to the eye. The default background colour for drawings is white, and the drawing gallery was given a light grey background to separate the drawing selected from the background. The seekbars used to change the stroke and background colour were given smooth colour gradients to produce a wide range of colours.
The font used throughout the application is Roboto.

Fig.v- Roboto typeface

Roboto was designed to be the system font for the android operating system and therefore was chosen for its compatibility, as well as giving the app a similar feel to other Android applications.

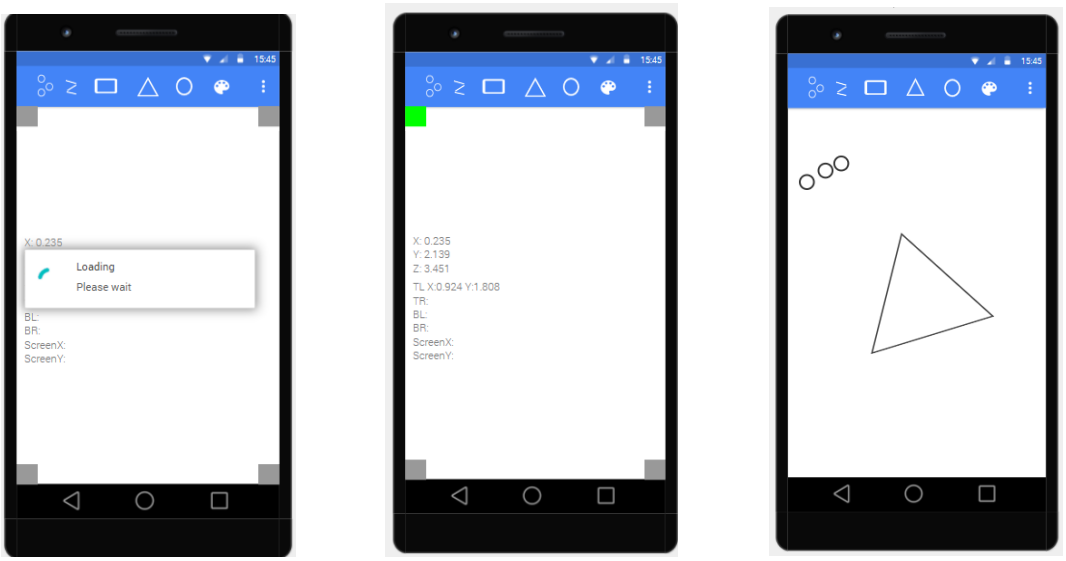
4.1.3 UI Modelling

Wireframes were produced to allow visualisation of how the application would look to a user. Using the wireframes it was possible to imagine the action the user would make and assess if the app was intuitive and simple to use. The wireframes were produced using Justinmind.Prototyper. Selected wireframes are shown below:



Figs.vi and vii

These wireframes show the main screen of the application. This is the screen the user will see when first opening the application. In fig *vi* no image has been selected, and in *vii* an image has been selected



Figs. viii, ix and x

These figures show the new drawing screen, accessed using the floating action button. A dialog is displayed when the buffer is being filled. In *ix* the user is calibrating the application. When a corner is calibrated its corresponding button will turn green. *x* Shows the user drawing after calibrating, all readings and buttons are now hidden.

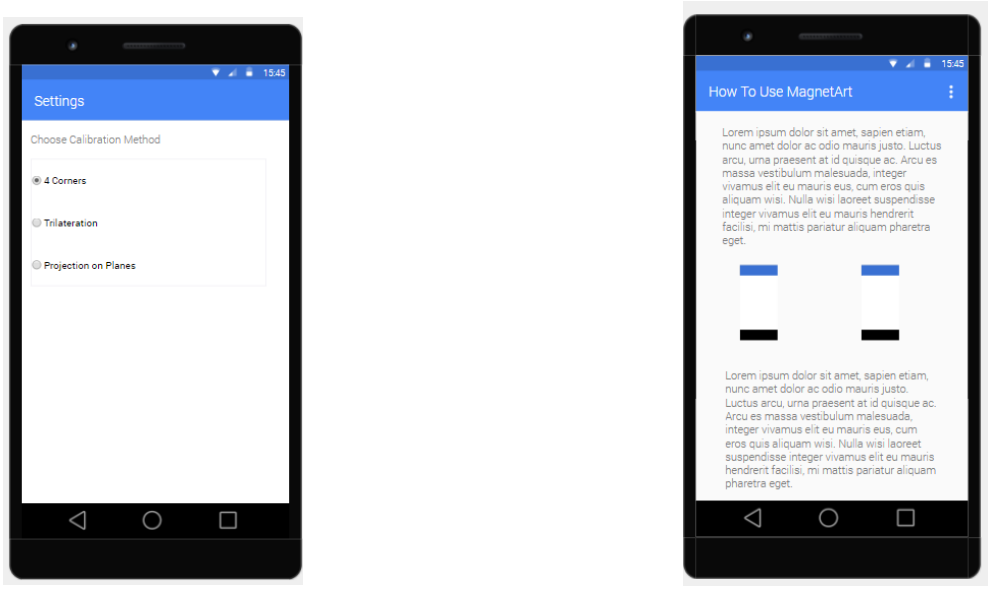


Fig. xi and xii

Fig *xi* shows the settings screen which allows the user to choose a calibration algorithm. Fig *xii* shows the help screen which gives the user information on how to use the application.

4.1.4 Future UI Improvements

While the main focus of this project is the application logic, not the user interface, if this project was to be progressed further, some improvements could be considered.

- When drawing, the chosen shape could be highlighted in the actionbar menu.
- The seekbars used to change stroke and background colour could have more colour options including white and black at each end.
- Help could appear on the new drawing screen to provide the user with a walkthrough when using the app for the first time.

In the event that this project is progressed further, it would be beneficial to review the UI by means of a focus group or questionnaire and invite suggestions on how to improve the UI.

4.3 Software System Design

Where possible, the design followed the SOLID principles of Object-Oriented Design. These are intended to make the design more flexible, understandable and maintainable.

- Single Responsibility Principle- A class should have only one reason to change
- Open/Closed Principle- software entities should be open for extension, not modification
- Liskov Substitution Principle- subtypes must be substitutable for their base types
- Interface Segregation Principle- clients should not be forced to depend on methods that they do not use
- Dependency Inversion Principle- high level modules should not depend on low level modules: both should depend on abstraction. Abstractions should not depend on details, details should depend on abstractions.

4.2.1 Architecture and Project Structure

Where possible, this application follows a Model-View-Controller (MVC) approach. This is achieved by separating out the UI logic and the application data. The View is handled by the XML layout files, the Controllers are the activity and fragments and the Model is composed of the java classes used for calculations. Calculations are made when calculating the magnet position, calculating the colour chosen from the seekbars and when drawing shapes.

In MagnetArt, the project structure is laid out as follows:
The application contains two activities, MainActivity.java, which is the starting point for the application, and UserPreferences.java, which allows the user to select an algorithm. When the user starts the application, the base activity adds a fragment to the view and the home screen of the application is displayed.
To simplify the handling of sensor data, an object, MagnetPosition.java, was defined as an array. Each object contains the magnetometer sensor readings in 3 dimensions.

NewDrawingFragment.java can be described as the Controller, as it handles data from other classes and device state changes. This fragment contains access to the device magnetometer and calls methods defined in MagnetCalculations.java and CalibrationCalculations.java to retrieve the screen co-ordinates. The co-ordinates are then passed to the DrawingView.java class to draw a shape onscreen. This class also handles saving drawings to device storage.
The HelpFragment.java class displays information on how to use the application.
Dialogs for changing background and stroke colour and clearing the drawing are defined in their own classes. The background and stroke colour dialogs instantiate the ColourSeekBar.java custom view. The dialogs have been defined separately to reduce the code in NewDrawingFragment.java and to be reusable. Callbacks from each dialog is handled using an interface.
Each fragment and activity has an associated XML file.

The project was structured in this way to reduce coupling and improve code clarity. Having calculations in a separate class to the calling fragment made it easier to update and test various algorithms. The java classes were divided up into packages to manage the project efficiently.

4.2.2 Design Patterns Used

Builder

The builder pattern was used when creating dialogs using AlertDialog.Builder. This pattern separates the construction of an object from its representation. It is particularly useful in this project as the pattern can be followed to create various dialogs by specifying only the attributes required for each one. For example, the ‘save without quitting’ dialog used in the NewDrawingFragment contains a positive, negative and neutral button, whereas the dialog used to change stroke colour does not have a neutral button. This pattern allows attributes to be declared which are optional.

Observer

The observer pattern is used when receiving callbacks from the dialogFragments. This pattern establishes a pattern between objects so that if the state of an object changes, all dependent objects are notified. Each dialog has an associated interface. The NewDrawingFragment class implements these interfaces and when a button has been pressed on a dialog the specified method is called. For example, the interface BackgroundColourListener contains the methods:

```
void onBackgroundPositiveClick(DialogFragment dialog);
void onBackgroundNegativeClick(DialogFragment dialog);
```

NewDrawingFragment implements this interface and therefore must implement these methods. When the user presses ‘confirm’ on the dialog, the onBackgroundPositiveClick() method is called which will change the colour of the drawing canvas.

The sensor listener defined for the device magnetometer also behaves as an Observer. When a class extends the SensorEventListener interface, it must implement methods to handle sensor events or a change in sensor accuracy.

4.2.3 Future Design Improvements

In the current version of the application, the user is given a setting to change the algorithm. If, by further research and experimentation, the accuracy of the algorithm used to estimate screen co-ordinates was improved then this setting would be unnecessary. This activity allows comparison of algorithms for demonstrative purposes. The system design could be improved as the logic was not fully separated from the UI for some classes. As mentioned, many steps have been taken to separate the code into packages and classes. However, the operations used to save and retrieve a saved image could be moved into a separate class. These methods could then be called in the required fragments. The dependency injection design pattern could be used, such as Butter Knife [15], which uses annotations to instantiate views and handle events such as onClick().

4.2.4 UML Diagrams

Class Diagram

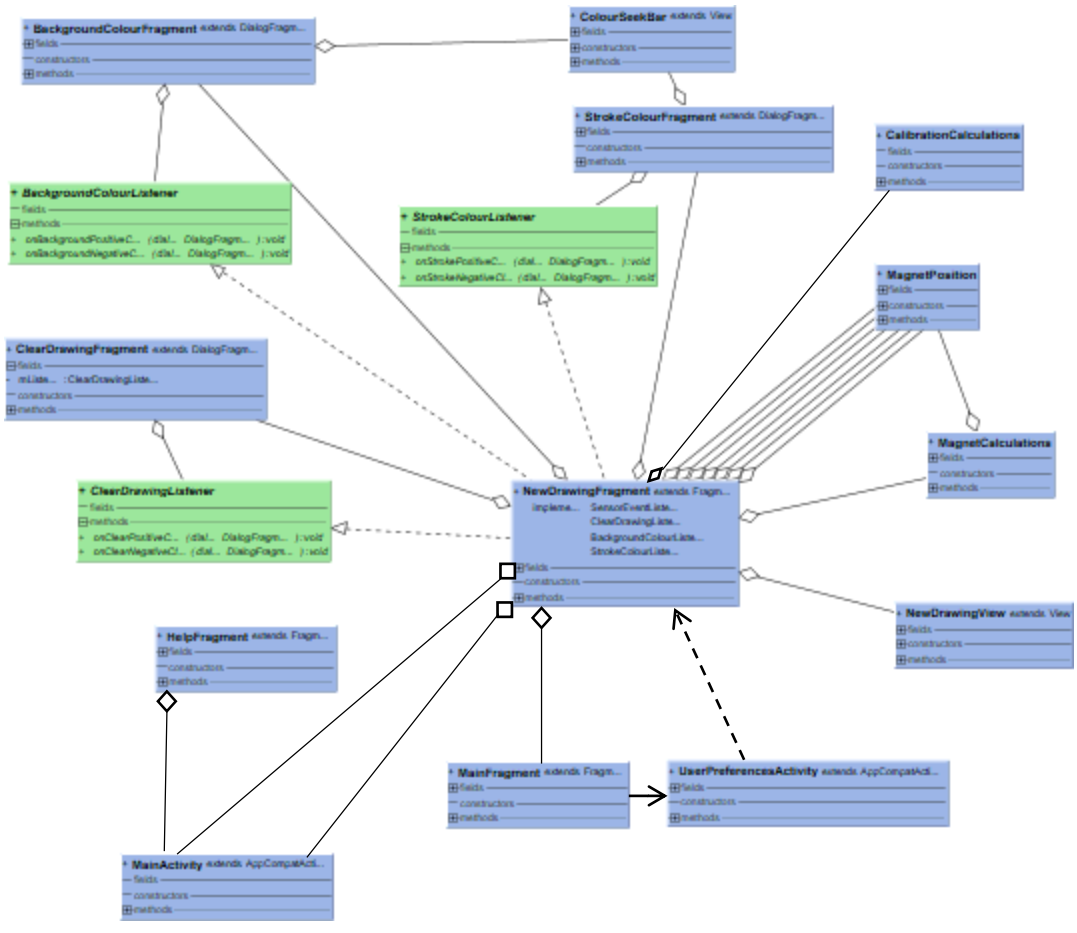


Fig.xiii- Class diagram for the application

Sequence Diagram

Sequence diagram to show sensor readings being handled in the NewDrawingFragment, and the calls from this class to obtain screen co-ordinates.

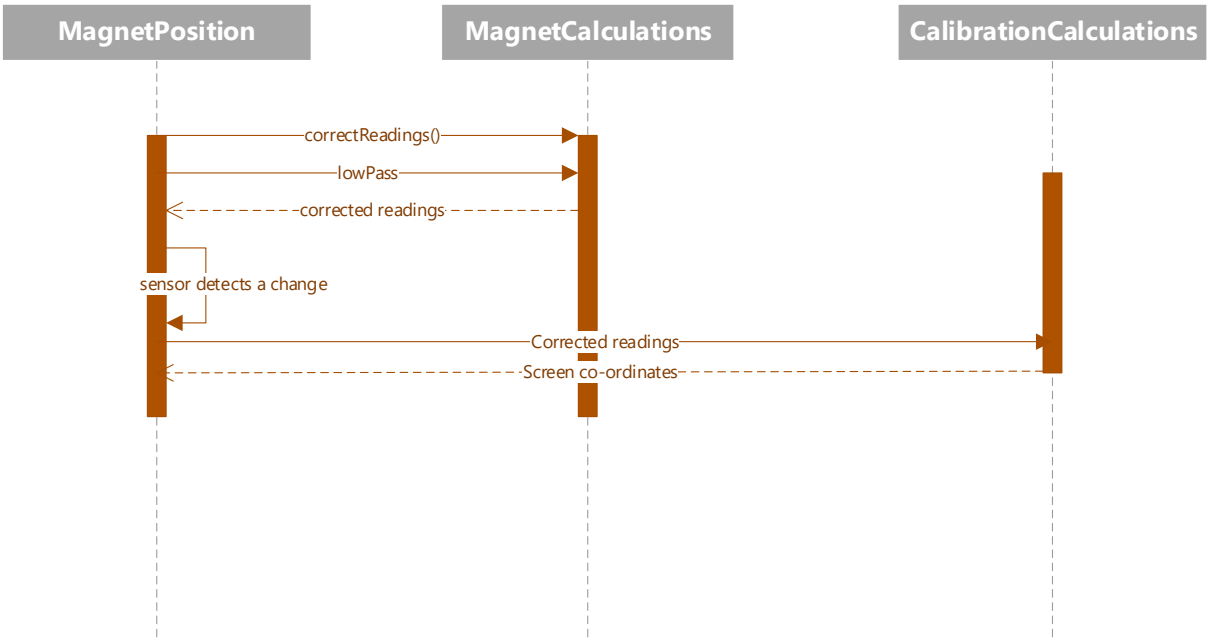


Fig.xiv – A sequence diagram for treatment of magnetometer data

Chapter 5: Implementation

5.1 Coding Standards Followed

The code produced for this project followed the guidelines outlined in the Android Open Source Project Java Code Style for Contributors [16] for variable naming, exception handling, Javadoc comments and imports.

5.2 Sprint Planning and Execution

As discussed in Chapter 2, the methodology planned for this project was a design-to-schedule approach and the work was divided into 4 two-week long sprints. When producing a sprint backlog, the user stories were sorted by priority and those with the highest priority were worked on first. This approach ensured the focus was on implementing the highest priority requirements first. The general approach to developing this application was to start by ensuring that magnetic readings could be taken and processing this data to make it as accurate as possible. After this, the next stage was to use the magnetic field readings to produce screen co-ordinates. This meant a drawing view needed to be implemented. Other drawing features could then be worked on, such as changing the stroke colour or the shape being drawn and adding the ability to save the drawings. Once these main features were implemented, the user interface could be developed further.

When executing the sprints, finding an algorithm to estimate the onscreen co-ordinates proved to be much more difficult and time-consuming than expected. Although this was a vital priority of the application, it was decided that lower priority requirements should be worked on in parallel as algorithms were being trialled. Progress was difficult to measure in the case of the screen co-ordinates, as various factors could influence the sensor readings. This is discussed further in this chapter.

5.3 Testing Strategies

The testing of the application also was based on the priority of the feature. When a feature had been implemented, that feature was tested by running the application on a device or emulator from Android Studio. At the end of a sprint, documented testing was carried out. Nearly all testing was specification based. User stories were used to produce test conditions and then test cases, and these are included in Appendix E. These test cases do not produce complete code coverage, but were used to test the main functions of the application.

Due to the app requiring a magnetic field sensor, the application could not be fully tested on an emulator and was tested by running on a device. The devices used were: A OnePlus X running API 23 and a Samsung Galaxy A3 running API 21.
To test that the co-ordinates produced onscreen matched the expected co-ordinates from the drawing surface, a grid was drawn on square paper. This grid was taken as the drawing surface, and had points marked on it- the corners, the middle of the grid and midpoints between the corners and the

middle of the grid. The width and height of the display of the chosen device were then used to mark expected values at the points on the grid. The device was then placed on the page, 5cm to the left of the grid and the new drawing screen opened. After calibrating, the estimated co-ordinates were made visible and recorded as the magnet was moved to the labelled points on the grid. The expected and actual co-ordinates were then plotted with an associated uncertainty due to magnetic fluctuations.

A controlled environment was used when testing the sensor data, away from any other electronic devices or metallic objects.

In reflection, the testing cannot be said to have been independent because the developer was also the tester. However, this meant that experience-based techniques could be employed to test features after implementation. If more time had been available, methods which included calculations would have been unit tested. Mock values would be passed into these methods and the output from the method was compared with the expected value. The methods unit tested would be those in the MagnetPosition and MagnetCalculations classes.

5.4 Magnets Used

The magnetic stylus was created using a 10mm height by 10mm diameter neodymium magnet. The cylindrical shape was chosen due to the magnetic field lines going radially outwards in the x-y plane, with a much smaller contribution upwards, in the z-axis. This meant that the magnetic field strength would not vary greatly depending on the tilt or rotation of the magnet. The magnet field produced will be radially constant for a fixed distance from the centre. The screen co-ordinates are only dependent on the x and y readings so the minimal z-axis contribution did not affect the results.

Neodymium was chosen as it is a very strongly magnetic material.

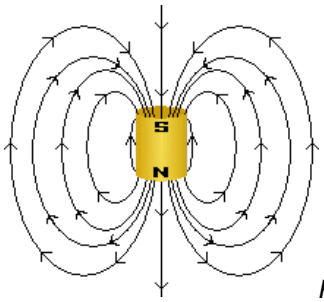


Fig.xv- The magnetic field surrounding a cylinder [17]

It was decided to use a non-electrified magnet as it would have a taken a lot of time and research to implement an electrified magnet. Being able to turn off and on the power supply to a magnet would allow the user to toggle whether drawing is active or not. As this was not possible with a non-electrified magnet, when using the application, it is taken that when the new drawing screen is displayed, the user is drawing unless a dialog is shown.

5.5 Selected Implementations

5.5.1 Processing Magnetometer Sensor Data

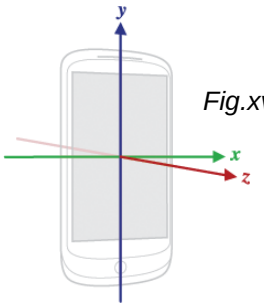


Fig.xvi- Coordinate system (relative to a device) used by the Sensor API. [18]

Sensor data is handled in the NewDrawingFragment. This class extends the SensorEventListener interface and therefore implements the onSensorChanged and onAccuracyChanged methods. The onSensorChanged method is called whenever the device detects a change in the surrounding magnetic field. Due to the terrestrial field, this happens very frequently.

A custom object was created, MagnetPosition, which is an array of 3 doubles. This array is used throughout the project to store sets of magnetic field readings, i.e. the readings in the x, y and z axes. The double type was chosen to remove the need to cast variables when using them in calculations. Before the magnetic sensor data can be used in estimating the screen co-ordinates, it is first processed. It is assumed that when using the application, the only magnetic field changes surrounding the device are due to the movement of the magnetic stylus. Therefore, it is necessary to account for the terrestrial magnetic field. Noise must also be removed from the data. Methods to process the data are defined in the class, MagnetCalculations.java.

The values from each sensor event are then sent to the MagnetCalculations class. To account for the terrestrial field, the first 500 magnetic field readings taken are saved into a buffer. A buffer size of 500 was a trade-off to ensure an accurate estimate without causing a long delay to the user. The average value in each dimension is then taken as the terrestrial field and subtracted from each subsequent reading. The corrected values are returned as a MagnetPosition object and the estimated terrestrial field can also be obtained using its getter. This allows the values to be displayed on screen.

```
/**
 * Method to correct readings for the terrestrial magnetic field
 * @param eventvalues
 * @return
 */
public MagnetPosition correctReadings(float[] eventvalues) {

    double sumex = 0, sumey = 0, sumez = 0; //Buffers are initially empty

    if (!bufferfull) {
        //Add new sensor readings to the buffers to fill it
        if (xarray[buffsize - 1] == 0) {
            xarray[iter] = eventvalues[0];
            yarray[iter] = eventvalues[1];
            zarray[iter] = eventvalues[2];
            iter++;

        } else {
            //Add all buffer values in x, y, and z dimensions
            for (int i = 0; i < buffsize; i++) {
                sumex += xarray[i];
```

```
                sumey += yarray[i];
                sumez += zarray[i];
            }
            //Take an average value for each dimension, this is taken to be the terrestrial magnetic
field
            ex = sumex / buffsize;
            ey = sumey / buffsize;
            ez = sumez / buffsize;

            earthp = new MagnetPosition(ex,ey,ez);//the estimated terrestrial field in 3 dimensions

            bufferfull = true;//x, y and z buffers have been filled

        }

    }

    bx = eventvalues[0];
    by = eventvalues[1];
    bz = eventvalues[2];

    //correct for the terrestrial magnetic field
    dx = bx - ex;
    dy = by - ey;
    dz = bz - ez;

    return new MagnetPosition(dx, dy, dz);
}
```

The uncalibrated magnetic field sensor is chosen as no periodic calibration is performed by the device, ensuring a continuous data stream. The sensor delay is chosen to be SENSOR_DELAY_GAME which ensures readings are taken at intervals of 20,000µs (0.02s) [18]. This ensures that drawing is continuous and prevents a long delay waiting for the buffers to fill.

A low-pass filter is applied to smooth the data. This filter attenuates high frequency readings and therefore reduces noise in the data. This means that the data is more continuous and does not hop around randomly. The algorithm used is shown below [19]:

```
/**
 * Low-Pass filter to attenuate high frequency signals and ensure continuous data
 * @param input
 * @param output
 * @return
 */
public static double[] lowPass(double[] input, double[] output) {

    if (output == null) {
        return input;//first set of data- return input
    }
    for (int i = 0; i < input.length; i++) {
        output[i] += ALPHA * (input[i] - output[i]);//ALPHA controls how much of the input signal is
attenuated
    }
    return output;
}
```

Alpha is the smoothing constant of the filter, and determines how much of the signal is attenuated. ALPHA was currently set to 0.2, which means a large amount of filtering has been applied. ALPHA can have values from 0 to 1. When the sensor readings have been corrected for the terrestrial field and a low-pass filter applied, they can then be used to estimate the screen co-ordinates.

5.5.2 Calibration Algorithms to Estimate Screen Co-ordinates

To draw onscreen accurately, an algorithm must be applied to the corrected magnetic field readings to estimate the relative position of the magnet on the drawing surface. This must then be scaled to the dimensions of the device screen.
Several algorithms were attempted. Their implementations and the accuracy of each are detailed in this section. The algorithms were implemented in the Calibrate() method in the CalibrationCalculations.java class.

A custom View class, NewDrawingView.java on which to draw was created. The details of this class are described in section 5.5.3. This class is instantiated in the NewDrawingFragment, meaning that most of the display is a canvas on which to draw.
Before the user can draw, the limits of the drawing surface need to be defined. Buttons in the corners of the drawing canvas record magnetic field readings when pressed. For example, a user will place the magnet in the top left of the drawing surface and press the top left button onscreen. As they move the magnet to each corner, they will press the corresponding button to save readings.

```
//when each button is pressed, the magnetic field reading is recorded for that corner of
the drawing area and the button is then disabled
btn_tl.setOnClickListener(
        new Button.OnClickListener(){
            public void onClick(View v){
                btn_tl.setEnabled(false);
                btn_tl.setBackgroundColor(0xff00ff00);
                topleftp = new MagnetPosition(currentp.toDoubleArray());
                tlreading.setText(String.format("TL X: %.3f  Y: %.3f Z: %.3f", topleftp.xPosition,
topleftp.yPosition, topleftp.zPosition));
                Log.d(TAG, "Top Left Calibrated- TL Readings are above");
                calibrationcounter++;
                checkCornersCalibrated();
            }
        }
);
```

In the above code snippet, the top left button is pressed, which means the current corrected magnetic field reading is saved as a MagnetPosition object. The button is then disabled and become green to prevent a new set of values being saved. The saved readings are displayed in a TextView, and the calibrationcounter increments. This variable counts the number of calibrated corners.

When readings have been taken in all 4 corners, the MagnetPosition objects saved for each corner and the current readings are sent to a calibration algorithm to estimate the screen co-ordinates. The algorithms attempted will be detailed below.

To test the algorithms fairly, several factors needed to be controlled. They were tested on the same devices in the same environment with the same magnets. The environment needed to be controlled to reduce magnetic field interference.

As mentioned in section 5.3, the dimensions of the device screen needed to be obtained. This was achieved using the following method, where dv is the drawing view, set to fill the display:

```
/**
  * Method to get the screen dimensions using the size of the drawing view
  */
```

```
    public void getScreenDimensions(){
        dwidth = dv.getWidth();
        dheight = dv.getHeight();

    }
```

The screen dimensions can then be used in the calibration algorithm to scale the co-ordinates to the device display. The following method is called when calibration is complete:

```
         //all corners have been calibrated
            if (calibrationcounter == 4) {

//                use selected algorithm
                screenp = CalibrationCalculations.Calibrate(algorithm, topleftp, toprightp,
botleftp, botrightp, currentp, dwidth, dheight);

                calibrationcomplete = true;
```

The calibration method called depends on the integer value *algorithm,* which is the first argument in the Calibrate method call. This can be changed in the UserPreferencesActivity. This is discussed further later in this section.

4-Corners Algorithm

The first algorithm tried was given the name ‘4 corners’. This method involved using the magnetic field readings to find the fractional distance of the magnet along the drawing surface in each axis.

The relative distance along the x-axis in terms of magnetic field readings was calculated using the following code:

```
    //calculate fractional distance along surface in x direction at top
double xloc1 = (topleft.xPosition-current.xPosition)/(topleft.xPosition -
topright.xPosition);
    //calculate fractional distance along surface in x direction at bottom
double xloc2 = (botleft.xPosition-current.xPosition)/(botleft.xPosition -
botright.xPosition);
    //take an average
double avgx = (xloc1+xloc2)/2;
```

The relative distance along the y-axis was estimated in a similar way:

```
        //calculate fractional distance along y axis at left
        double yloc1 = (topleft.yPosition-current.yPosition)/(topleft.yPosition -
botleft.yPosition);
            //calculate fractional distance along y axis on right
        double yloc2 = (topright.yPosition-current.yPosition)/(topright.yPosition -
botright.yPosition);
//            take an average
            double avgy = (yloc1 + yloc2) / 2;
```

The onscreen co-ordinates were then estimated by multiplying the average x and y distances by the screen width and height respectively.

```
xLoc = avgx*dwidth;
yLoc = avgy*dheight;
```

Average Expected and Actual Screen Co-ordinates when using 4 Corners Algorithm					
Expected x / pixels	Actual x / pixels	Difference / pixels	Expected y / pixels	Actual y / pixels	Difference / pixels
0	4181	4181	0	2600	2600
1080	1133	53	1680	1340	340
540	341	199	840	2600	1760
0	533	533	1680	2120	440
1080	993	87	0	1530	1530

When the actual points are compared against expected, the values are seen to be quite far from those expected. Therefore, this was not considered an accurate calibration technique.

Trilateration

Another algorithm trialled was trilateration. Trilateration is the process of determining the relative position of a point using the intersection of circles centred on three reference points.

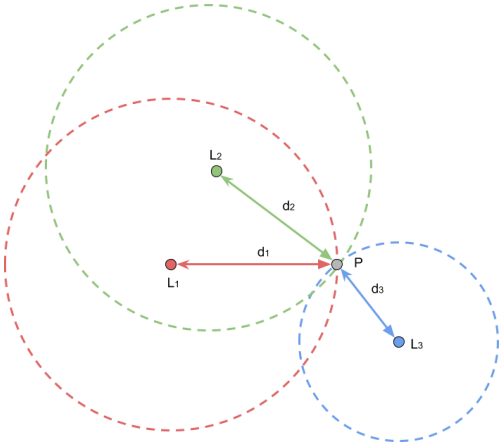


Fig.xvii- Trilateration Method [20]

As shown in the diagram above, P is the unknown point and L_1, L_2 and L_3 are reference points. If these points are at (i_1,j_1), (i_2,j_2) and (i_3,j_3) respectively and P is at (x,y), then the distances from P to each reference point are:

$$d_1 = \sqrt{(x - i_1)^2 + (y - j_1)^2}$$
$$d_2 = \sqrt{(x - i_2)^2 + (y - j_2)^2}$$
$$d_3 = \sqrt{(x - i_3)^2 + (y - j_3)^2}$$

By combining these equations, they can be solved for x and y to get the position of point P in terms of the other points.

$$x = \frac{(d_1^2 - d_2^2 - i_1^2 - i_2^2 - j_1^2 + j_2^2)(-2j_2 + 2j_3) - (d_2^2 - d_3^2 - i_2^2 + i_3^2 - j_2^2 + j_3^2)(-2j_1 + 2j_2)}{(-2j_2 + 2j_3)(-2i_1 + 2i_2) - (-2j_i + 2j_2)(-2i_2 + 2i_3)}$$

$$y = \frac{(d_1^2 - d_2^2 - i_1^2 + i_2^2 - j_1^2 + j_2^2)(-2i_2 + 2i_3) - (-2i_1 + 2i_2)(d_2^2 - d_3^2 - i_2^2 + i_3^2 - j_2^2) + j_3^2)}{(-2j_1 + 2j_2)(-2i_2 + 2i_3) - (-2i_1 + 2i_2)(-2j_2 + 2j_3)}$$

A full derivation can be found at [https://math.stackexchange.com/questions/884807/find-x-location-using-3-known-x-y-location-using-trilateration].

In this case, the reference points used were the top left, top right and bottom left corners of the drawing surface and the current magnet location was the unknown point. The equations were used in terms of magnetic field readings. The readings from the corners of the surface were substituted into the co-ordinates (i_1,j_1), (i_2,j_2) and (i_3,j_3), meaning that the equations above could be used to solve for x and y- the magnet co-ordinates.

```
double i1 = topleft.xPosition;
double i2 = topright.yPosition;
double i3 = botleft.xPosition;
double j1 = topleft.yPosition;
double j2 = topright.xPosition;
double j3 = botleft.yPosition;
double x = current.xPosition;
double y = current.yPosition;
```

The estimated x and y positions were then found as a fraction of the display dimensions and multiplied by the screen width and height.

```
// Find estimated position as a fraction of x and distance and multiply by screen
dimensions
xLoc = Math.abs(((i1 - xPos) / (i1 - botright.xPosition))) * dwidth;
yLoc = Math.abs(((j1 - yPos) / (j1 - botright.yPosition))) * dheight;
```

Evaluation of Trilateration Algorithm

Expected and Actual Screen Co-ordinates when using Trilateration Algorithm					
Expected x / pixels	Actual x / pixels	Difference / pixels	Expected y / pixels	Actual y / pixels	Difference / pixels
0	50	50	0	190	190
1080	1080	0	1680	1797	117
540	907	367	840	963	123
0	962	962	1680	1793	113
1080	1001	799	0	643	643

In general, the points were more accurate than those estimated from the four corners method. The points seem to map reasonably well in the centre reference point and the opposite corner. However, the readings seem to deviate when not near this diagonal.

Nature of Magnetic Field

After attempting these methods, it was considered that an assumption had been made that the magnetic field was changing linearly with distance. This assumption was researched and it was found that it the magnetic field strength should display cubic fall-off with distance. The nature of the magnetic field was tested by moving the magnet along the grid at fixed distances from the phone and taking readings versus distance. The experiment was repeated twice and results are shown in the graph below:

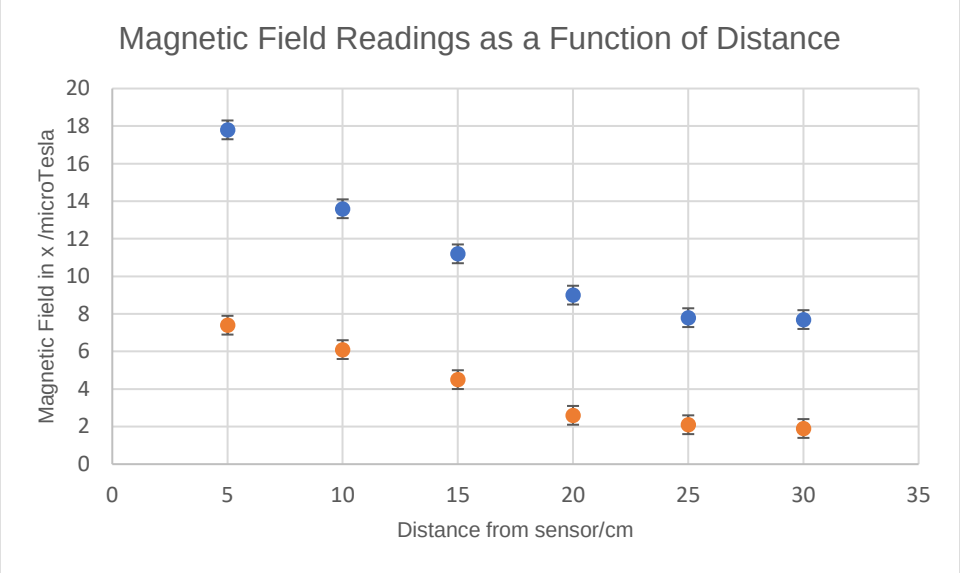


Fig.xviii

The graph confirms the change in magnetic field is not linear, and the difference in consecutive readings is smaller with distance.

Therefore, it was felt that the geometry of the situation needed to be accounted for. To do so, it was decided to use Coulomb's law for magnetic fields to estimate a vector r in the direction of the magnet:

$$\frac{b_r}{b_z} = \frac{r}{l}\left(1 + \left(\frac{l}{r}\right)^2\right)^{\frac{3}{2}}$$

Which leads:

$$\left(\frac{l}{r}\right)^5 + 3\left(\frac{l}{r}\right)^3 + \left(3 - \left(\frac{b_r}{b_z}\right)^2\right)\left(\frac{l}{r}\right) + 2\left(\frac{b_r}{b_z}\right) = 0$$

Where l is the length of the magnet and br and bz are the magnetic field readings in the x and z axes at the magnet location. The equations used for this are taken from a research by Tetsuya Abe et al [21].

Coulomb's Equations

The calculations shown above were implemented in the code. The co-ordinate system used is shown below.

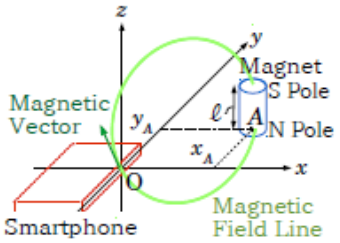


Fig.xix- Co-ordinate system used for non-linear field assumption[21]

When implementing this approach, a method geometryCalc() was called from the onSensorChanged() method after the low-pass filter had been applied. This method takes the corrected magnetic field readings as an argument. In the diagram above, r is a vector from the origin to the magnet. The origin is at the device magnetic field sensor.

```
/**
 * Method to calculate the relative co-ordinates by solving Coulomb's equation
 * @param currentpos
 * @return
 */
public MagnetPosition geometryCalc(MagnetPosition currentpos){

    double dx = currentpos.xPosition;
    double dy = currentpos.yPosition;

    dr = - Math.sqrt(Math.pow(dx,2)+Math.pow(dy,2));//Incremental distance in direction of
magnet
    r=bisectionMethod(currentpos, dr);
    xa = r*Math.cos(Math.atan(dy/dx));
    ya = r*Math.sin(Math.atan(dy/dx));

    return new MagnetPosition(xa,ya,currentpos.zPosition);

}
```

This method calculates r by solving the equation using the bisection method. The estimated r is plugged into the equation and initially tested in an interval of 0.001 to 1. The method is repeated until the solutions from the limits of the interval agree to a tolerance of 0.0001.

```
/**
 * Vector r in direction of magnet can be estimated using the bisection method
 * @param currentpos
 * @param dr
 * @return
 */
private double bisectionMethod(MagnetPosition currentpos, Double dr){

double a = 0.001;//lower boundary of interval
double tolerance = 0.00001;
double b=1;//upper boundary of interval
double p,fa,fp;
int n = 500;//max number of iterations
int iter;//iteration number


    iter = 1;
    fa=equationFunc(currentpos, a, dr);

    while (iter<=n){
        p = a + (b-a)/2;
        fp = equationFunc(currentpos, p, dr);
```

```
        if ((fp == 0) || ((b-a) < tolerance))
            return p;
        iter++;

        if (fp*fa > 0){
            a = p;
            fa = fp;
        }
        else
            b=p;
    }
    return 0.01;
}

/**
 * Plug values from each iteration into the equation
 * @param cp
 * @param esr
 * @param dr
 * @return
 */
private double equationFunc(MagnetPosition cp, double esr, double dr){
    double dz = cp.zPosition;
    return Math.pow(1/esr,5)+3*Math.pow(1/esr,3)+(3-Math.pow(dr/dz,2))*(1/esr)+2*dr/dz;
}
```

After estimating r, the x and y co-ordinates on the drawing surface could then be estimated using:

```
xa = r*Math.cos(Math.atan(dy/dx));
ya = r*Math.sin(Math.atan(dy/dx));
```

These values could then be scaled to the device display using the four corners method detailed previously.

Expected and Actual Screen Co-ordinates when using Non-Linear Algorithm					
Expected x / pixels	Actual x / pixels	Difference / pixels	Expected y / pixels	Actual y / pixels	Difference / pixels
0	50	50	0	140	140
1080	1083	3	1680	1630	50
540	897	357	840	831	9
0	960	960	1680	1670	10
1080	634	1166	0	23	23

This algorithm seemed to give more accurate results in the y co-ordinates, however the x co-ordinates still showed a large difference between expected and actual. The algorithm may need corrected in future.

To allow demonstration of the various calibration techniques, an activity was implemented to allow the user to select an algorithm, UserPreferencesActivity. This activity retrieves a list of algorithms from the layout file, algorithms_list.xml. Each algorithm is assigned a radio button, and when pressed this button will save an integer corresponding to an algorithm using SharedPreferences.

```
/**
 * Saves the algorithm integer in SharedPreferences
```

```
     * @param algo
     */
    private void saveAlgorithmSelected(int algo) {
        try {
            SharedPreferences sp = this.getSharedPreferences(KEY, MODE_PRIVATE);
            SharedPreferences.Editor editor = sp.edit();
            editor.putInt(INT_KEY, algo);
            editor.apply();
        } catch (ClassCastException e) {
            Toast.makeText(this, R.string.variable_wrong_type,
    Toast.LENGTH_SHORT).show();
        }
    }
```

This integer is taken as an argument when calling to the Calibration() method in the CalibrationCalculations, which uses an if statement on this integer to call the correct algorithm.

5.5.3 Drawing

The various drawing methods are contained in the NewDrawingView class. This class contains the float variables screen and screen. These are values for the screen co-ordinates calculated using the methods above, and each time the onSensorChanged() method is called, these values are updated.

The various shapes and lines that can be drawn are detailed below. The methods used are based on the techniques by Gabriele Mariotti [22]. These techniques were changed to allow input from the magnet rather than only touch input.

Spot

An enum activeShape is used to check the current shape selected by the user. The default drawing style is a spot onscreen at the calculated co-ordinates. This spot is 10 pixels in radius.

```
//default is to draw a spot on screen
if(activeShape == DrawingShape.SPOT) {
    drawingCanvas.drawCircle(screenX, screenY, 10, endingPaint);
}
```

Smooth line

A smooth line can be drawn to follow the path of the magnet. This method requires the user to touch the screen while drawing. The method saves the previous co-ordinates and draws a line from the old co-ordinates to the new co-ordinates.

```
// Drawable line
private void onTouchEventSmoothLine(MotionEvent event) {

    switch (event.getAction()) {
        case MotionEvent.ACTION_DOWN:
            drawingActive = true;
            beginX = xPos;
            beginY = yPos;

            pathToDraw.reset();
            pathToDraw.moveTo(xPos, yPos);
```

```
            invalidate();
            break;
        case MotionEvent.ACTION_MOVE:

            float distanceX = Math.abs(xPos - beginX);
            float distanceY = Math.abs(yPos - beginY);
            if (distanceX >= TOUCH_TOLERANCE || distanceY >= TOUCH_TOLERANCE) {
                pathToDraw.quadTo(beginX, beginY, (xPos + beginX) / 2, (yPos + beginY) /
2);
                beginX = xPos;
                beginY = yPos;
            }
            drawingCanvas.drawPath(pathToDraw, initialPaint);
            invalidate();
            break;
        case MotionEvent.ACTION_UP:
            drawingActive = false;
            pathToDraw.lineTo(beginX, beginY);
            drawingCanvas.drawPath(pathToDraw, endingPaint);
            pathToDraw.reset();
            invalidate();
            break;
    }

}
```

Circle

To draw a circle, touch input is also required. When the user places a finger onscreen. The screen co-ordinates at that time will be the midpoint of the circle. When moving the magnet, the diameter of the circle will be changed and will be set when the touch input is removed.

```
// Midpoint drawable circle
private void onDrawCircle(Canvas canvas){
    canvas.drawCircle(beginX, beginY, calculateRadius(beginX, beginY, xPos, yPos),
initialPaint);
}

private void onTouchEventCircle(MotionEvent event) {
    switch (event.getAction()) {
        case MotionEvent.ACTION_DOWN:
            drawingActive = true;
            beginX = xPos;
            beginY = yPos;
            invalidate();
            break;
        case MotionEvent.ACTION_MOVE:
            invalidate();
            break;
        case MotionEvent.ACTION_UP://draw when finger removed from screen
            drawingActive = false;
            drawingCanvas.drawCircle(beginX, beginY,
calculateRadius(beginX,beginY,xPos,yPos), endingPaint);
```

```
            invalidate();
            break;
    }
}
```

The radius of the circle is calculated using the co-ordinates of the midpoint and the co-ordinates when the touch input is removed.

```
/**
 * Method to calculate radius of circle
 * @param x1
 * @param y1
 * @param x2
 * @param y2
 * @return
 */
protected float calculateRadius(float x1, float y1, float x2, float y2) {

    return (float) Math.sqrt(
            Math.pow(x1 - x2, 2) + Math.pow(y1 - y2, 2));
}
```

Rectangle

A rectangle also requires touch input. A corner of the rectangle is drawn at the co-ordinates when the touch input begins and the opposite corner is drawn when it is removed. The length and width of the rectangle are determined by the movement of the magnet when a finger is placed onscreen.

```
private void onDrawRectangle(Canvas canvas) {
    drawRectangle(canvas,initialPaint);
}

private void onTouchEventRectangle(MotionEvent event) {

    switch (event.getAction()) {
        case MotionEvent.ACTION_DOWN:
            drawingActive = true;
            beginX = xPos;
            beginY = yPos;
            invalidate();
            break;
        case MotionEvent.ACTION_MOVE:
            invalidate();
            break;
        case MotionEvent.ACTION_UP:
            drawingActive = false;
            drawRectangle(drawingCanvas,endingPaint);
            invalidate();
            break;
    }
}
```

The four sides of the rectangle are determined using the larger co-ordinate between the starting corner and the finishing corner.

```
private void drawRectangle(Canvas canvas,Paint paint){
    float right = beginX > xPos ? beginX : xPos;
    float left = beginX > xPos ? xPos : beginX;
    float bottom = beginY > yPos ? beginY : yPos;
    float top = beginY > yPos ? yPos : beginY;
```

```
    canvas.drawRect(left, top , right, bottom, paint);
}
```

Triangle

A corner of the triangle will be fixed when touch input is first applied and a side of the triangle will then be determined by the magnet movement. When the touch is removed, another corner will be drawn to complete one side of the triangle. When touch input is then placed onscreen again, the apex of the triangle will be moved with the magnet and will be drawn when the touch is removed once more.

```
private void onDrawTriangle(Canvas canvas){

    if (countTouch<3){
        canvas.drawLine(beginX,beginY,xPos,yPos,initialPaint);
    }else if (countTouch==3){
        canvas.drawLine(xPos,yPos,beginX,beginY,initialPaint);
        canvas.drawLine(xPos,yPos,basexTriangle,baseyTriangle,initialPaint);
    }
}

private void onTouchEventTriangle(MotionEvent event) {

    switch (event.getAction()) {
        case MotionEvent.ACTION_DOWN:
            countTouch++;
            if (countTouch==1){
                drawingActive = true;
                beginX = xPos;
                beginY = yPos;
            } else if (countTouch==3){
                drawingActive = true;
            }
            invalidate();
            break;
        case MotionEvent.ACTION_MOVE:
            invalidate();
            break;
        case MotionEvent.ACTION_UP:
            countTouch++;
            drawingActive = false;
            if (countTouch<3){
                basexTriangle=xPos;
                baseyTriangle=yPos;
                drawingCanvas.drawLine(beginX,beginY,xPos,yPos,endingPaint);
            } else{
                drawingCanvas.drawLine(xPos,yPos,beginX,beginY,endingPaint);
                drawingCanvas.drawLine(xPos,yPos,basexTriangle,baseyTriangle,endingPaint);
                countTouch =0;
            }
            invalidate();
            break;
    }
}
```

5.5.4 Colour Picker

For changing stroke and background colour, a custom View object, ColourSeekBar.java is created, which displays a colour gradient used to pick a colour. The colour bar was divided into 100 and filled by iterating along the bar and adding a new colour to each section. The colour gradient was made

smooth by using an rgb colour with red, green and blue changing in a sine wave. The algorithm is shown below.

```
/**
 * Convert value along bar (0-100) to a color int (0-255)
 * @param val 0-100
 * @return colour 0-255
 */
public static int convertValueToColor(float val){

    float red,green,blue;//Each colour is in the range 0-255 in rgb scale

    float i = 16*(val/100);
    red = (float)Math.sin(0.3*i)*127+128;//colour gradient is made using 3 out of phase
sine waves
    green = (float)Math.sin(0.3*i+2)*127+128;
    blue = (float)Math.sin(0.3*i+4)*127+128;
    //returns a colour int made up of red, green and blue components
    return Color.rgb((int) red, (int) green, (int) blue);

}
```

The chosen colour is shown by a slider which is moved along the seekbar. This slider is made up of four smaller rectangles to form one hollow rectangle.

```
/**Draw the cursor for the currently selected colour (this is a rectangle with a gap in the
middle to allow the user to see the
 * selected colour- made up of four smaller rectangles
 */
protected void drawCursor(Canvas canvas, float start, float end, float top, float bottom) {

    mSliderpaint.setColor(Color.BLACK);
    float progressPercent = progress/100.0f;

    //4 rectangles used to make rectangle with hole in middle
    canvas.drawRect(start + (progressPercent)*(end-start)-15, top, start +
(progressPercent)*(end-start)-5, bottom, mSliderpaint);
    canvas.drawRect(start + (progressPercent)*(end-start)+5, top, start +
(progressPercent)*(end-start)+15, bottom, mSliderpaint);
    canvas.drawRect(start + (progressPercent)*(end-start)-15, top-10, start +
(progressPercent)*(end-start)+15, top,mSliderpaint);
    canvas.drawRect(start + (progressPercent)*(end-start)-15, bottom, start +
(progressPercent)*(end-start)+15, bottom+10, mSliderpaint);

}
```

The seekbar is used in the BackgroundColourFragment and StrokeColourFragments. These are dialogs shown to the user when the menu item is pressed. These include getters and setters which can be used to retrieve and set the colours in the ColourSeekBar object.

```
// Return color int
public int getCurrentBackgroundColour() {
    return convertValueToColor(currentBackgroundColour);
}

// SETTER (also sets value in colourseekbar)
public void setCurrentBackgroundColour(float value) {
    currentBackgroundColour = value;
    if(backgroundcoloursb != null) {
        backgroundcoloursb.setProgress((int) value);
        backgroundcoloursb.invalidate();
    }
}
```

The setters are called in the NewDrawingFragment to use the magnet movement to select a colour. When the user presses either the stroke colour or background colour menu item, a Boolean is set to true and drawing is disabled on the canvas. The distance along the colour bar is set using the magnitude of the MagnetPosition object.

```
if (choosingStrokeColour || choosingBgColor) {

    //Get a float from the seekbar which represents the chosen colour
    float colourValue = 100.0f - ((float) currentp.magnitude()) / 255 * 100.0f;
    if (colourValue > 100.0f) {
        colourValue = 100.0f;//set to value at end of bar
    } else if (colourValue < 1.0f) {
        colourValue = 1.0f;//set to value at start of bar
    }

    if (choosingStrokeColour) {//call to method in associated dialogfragment
        scf.setCurrentStrokeColour(colourValue);
    } else {
        bcf.setCurrentBackgroundColour(colourValue);
    }
}
```

The BackgroundColourFragment and StrokeColourFragment dialogs have associated interfaces. These interfaces define the methods called when confirm or cancel are pressed on the dialog. The NewDrawingFragment implements these interfaces and therefore uses these methods to change the background or stroke colour in the drawing view. These interfaces are also used to re-enable drawing as shown:

```
//     Callback from StrokeColourFragment- used to change colour when 'confirm' is pressed
    @Override
    public void onStrokePositiveClick(DialogFragment dialog) {
        //The movement of the magnetic stylus is used when changing colour, therefore
drawing must be disabled to ensure
        // that lines are not drawn on the canvas when the colour is being chosen.  When a
colour is chosen the dialog is dismissed
        //and drawing is enabled
        drawingactive = true;
        choosingStrokeColour = false;
        dv.initialPaint.setColor(((StrokeColourFragment)dialog).getCurrentStrokeColour());
        dv.endingPaint.setColor(((StrokeColourFragment)dialog).getCurrentStrokeColour());
    }

//     Callback from StrokeColourFragment- used to change colour when 'cancel' is pressed
and user does not wish to change strokecolour
    @Override
    public void onStrokeNegativeClick(DialogFragment dialog) {
        drawingactive = true;
        choosingStrokeColour = false;
    }
```

5.5.5 Save, View and Delete Drawings

When drawing, drawings can be saved using the menu item. The Boolean ‘saved’ checks if the drawing has been saved before. If the drawing has not been saved before then the drawing must be saved using a unique file name. This filename is created by getting the current time in milliseconds. This level of accuracy was needed to ensure, for example, that two drawings saved in the same

minute would not have the same filepath. If the drawing has been saved before the filepath is not changed and therefore any changes to the drawing are saved in the existing file.

```
public void saveDrawingToDevice(){

    boolean filecreated = false;

    //Check if the image has been saved before, if so then use the same filepath to update
the saved image
    if(!saved) {
        java.util.Calendar c = Calendar.getInstance();
        //File names are produced using milliseconds from calendar to ensure they are
unique
        mseconds = "magnetart " + c.get(Calendar.MILLISECOND) + ".png";
        saved = true;
    }

    File file = new File(getActivity().getFileStreamPath(mseconds)//use date and time to
create unique stream
            .getPath());
    if (!file.exists()) {
        try {
            filecreated = file.createNewFile();
        } catch (IOException e) {
            e.printStackTrace();
            Toast.makeText(getActivity(), "IOException", Toast.LENGTH_SHORT).show();
        }
    }
```

FileOutputStream is then used to write to file. The Bitmap is obtained from the drawing view and compressed before saving. A white rectangle is drawn in the background of the bitmap to ensure that the drawing is not transparent. If the user has set a background colour then this will remain.

```
    try {

        FileOutputStream fos = new FileOutputStream(file);

        System.out.println(fos);

        Bitmap drawingViewBitmap = dv.getBitmap();
        Bitmap bitmapToBeSaved = Bitmap.createBitmap(drawingViewBitmap.getWidth(),
drawingViewBitmap.getHeight(), Bitmap.Config.ARGB_8888);

        Paint paint = new Paint();
        paint.setColor(Color.WHITE);//Set background colour as white
        Canvas currentCanvas = new Canvas(bitmapToBeSaved);
        Rect drawingRect = new
Rect(0,0,drawingViewBitmap.getWidth(),drawingViewBitmap.getHeight());

        //draw white rectangle in the background
        currentCanvas.drawRect(drawingRect, paint);

        //draw the actual bitmap on the canvas
        currentCanvas.drawBitmap(drawingViewBitmap, drawingRect, drawingRect, null);

        //save bitmap to the file as a PNG
        bitmapToBeSaved.compress(Bitmap.CompressFormat.PNG, 100, fos);

    } catch (NullPointerException e) {
        e.printStackTrace();
        Toast.makeText(getActivity(), "NullPointerException", Toast.LENGTH_SHORT).show();
    } catch (FileNotFoundException e) {
        e.printStackTrace();
        Toast.makeText(getActivity(), "FileNotFoundException", Toast.LENGTH_SHORT).show();
    }
    //Display a toast to confirm file is saved
```

```
    Toast.makeText(getActivity(), "Saved", Toast.LENGTH_SHORT).show();

}
```

When the device back button has been pressed a dialog is displayed to ensure that the user does not lose a drawing by not having it saved. If the user presses save, the same method will be called and either a new file will be created or all changed will be added to the previously saved file.

The drawing is retrieved in the MainFragment class. A gallery with a horizontal scrollview is created along the top of this fragment. When this fragment is launched, the gallery is updated with all saved drawings.

```
private void updateGallery() {

    gallery.removeAllViews();
    //Returns absolute path to directory where files have been created using
FileOutputStream
    File targetDirector = getActivity().getFilesDir();
    //An array of pathnames denoting files in the directory
    File[] files = targetDirector.listFiles();

    //If there are files found, show the scrollview and frame to display the drawings
    if (files.length > 0) {
        scrollview.setVisibility(View.VISIBLE);
        viewdrawingframe.setVisibility(View.VISIBLE);
        //Hide text that says there are no drawings
        nodrawingstext.setVisibility(View.GONE);

        for (File file : files) {
            filepath = file.getAbsolutePath();
            gallery.addView(createLayoutForPhoto(filepath));//Iterate through array and add
files to gallery
        }

    } else {
        //If no drawings are found, hide the scrollview and gallery and display a message
to the user
        scrollview.setVisibility(View.GONE);
        viewdrawingframe.setVisibility(View.GONE);
        nodrawingstext.setVisibility(View.VISIBLE);
    }
```

If no drawings are found, the gallery is hidden and a message tells the user that no drawings have been found. To add drawings to the scrollview, the method createLayoutForPhoto() is called to create a layout to display each drawing.

```
private View createLayoutForPhoto(String path) {//called when updating gallery if images
are found

    Bitmap bm = retrieveScaledBitmap(path, 200, 200);//get Bitmap from filepath

    LinearLayout layout = new LinearLayout(getActivity());
    layout.setLayoutParams(new LinearLayout.LayoutParams(250, 250));
    layout.setGravity(Gravity.CENTER);

    ImageView imageView = new ImageView(getActivity());//place the bitmap in an imageview
    imageView.setLayoutParams(new ViewGroup.LayoutParams(200, 200));
    imageView.setScaleType(ImageView.ScaleType.CENTER_CROP);
    imageView.setImageBitmap(bm);
    imageView.setOnClickListener(new DrawingSelectedOnClickListener(path));

    layout.addView(imageView);
```

```
    return layout;
}
```

A LinearLayout is created for each drawing and each layout contains an imageview. The PNG image from the filepath is used to create a new bitmap, which is then scaled before being added. To scale the image, the following methods are called. The integers rwidth and rheight are the required width and height which in this case are set to 200 pixels. Using BitmapFactory.Options allows options to be set on how the file is scaled. inJustDecodeBounds=true avoids memory allocation.

```
private Bitmap retrieveScaledBitmap(String path, int rwidth, int rheight) {

    BitmapFactory.Options options = new BitmapFactory.Options();

    options.inJustDecodeBounds = true;//First decode with inJustDecodeBounds=true to check
dimensions
    BitmapFactory.decodeFile(path, options);

    //calculate inSampleSize
    options.inSampleSize = calculateScaledSize(options, rwidth, rheight);

    //Decode bitmap with inSampleSize set
    options.inJustDecodeBounds = false;
    Bitmap bm = BitmapFactory.decodeFile(path, options);

    return bm;
}
```

This method controls the ratio of width and height used when scaling. The height and width from options are compared to the required height and width. The smallest ratio is chosen as the inSampleSize which ensures the image will fit the requested dimensions.

```
private int calculateScaledSize(BitmapFactory.Options options, int rwidth, int rheight) {
    //raw width and height of bitmaps
    final int height = options.outHeight;
    final int width = options.outWidth;
    int scaledSize = 1;

    if (width > rwidth || height > rheight) {

        //Ratios of actual height and width to required height and width
        final int heightRatio = Math.round((float) height / (float) rheight);
        final int widthRatio = Math.round((float) width / (float) rwidth);
    //Choose the smaller ratio as scaled size
        scaledSize = width>height?heightRatio:widthRatio;
    }
    return scaledSize;
}
```

When an image is retrieved and is displayed in the scrollview, the image can be selected in order to view the image. This was achieved using a nested class to define the onClickListener for each drawing. The steps used to achieve this are similar to those used to fill the gallery. When the image is pressed, its filepath is used to retrieve a scaled bitmap. An imageview is created and the bitmap is displayed in it in a FrameLayout.

```
private class DrawingSelectedOnClickListener implements View.OnClickListener {

    public String imagePath;

    public DrawingSelectedOnClickListener(String path) {
        imagePath = path;
    }

    public void onClick(View v) {
```

```
        viewdrawingframe.removeAllViews();

        try {
            //get actual bitmap size
            final BitmapFactory.Options options = new BitmapFactory.Options();
            //avoids memory allocation-avoids memory allocations while decoding file
            //Allows us to read the dimensions and type of image data before constructing
bitmap
            options.inJustDecodeBounds = true;
            BitmapFactory.decodeFile(imagePath, options);

            //send the bitmap size to decoding sampled bitmap from Uri
            Bitmap bm = retrieveScaledBitmap(imagePath, options.outWidth,
options.outHeight);

        //save the bitmap in an image view
        final ImageView imageView = new ImageView(getActivity());
        imageView.setLayoutParams(new ViewGroup.LayoutParams(options.outWidth,
options.outHeight));
        imageView.setScaleType(ImageView.ScaleType.CENTER_CROP);
        imageView.setImageBitmap(bm);

        viewdrawingframe.addView(imageView);
        viewdrawingframe.invalidate();
        //make delete button visible when drawing is selected
        delete.setVisibility(View.VISIBLE);
        //Use delete floating action button to delete the selected drawing
        delete.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                deleteImage();
            }
        });
        }catch(Exception e){
        Toast.makeText(getActivity(), R.string.image_not_disp, Toast.LENGTH_SHORT).show();
            e.printStackTrace();
    }
    }
```

In the code snippet above, the floating action button ‘delete’ becomes visible when the image is displayed in the frame. When this button is pressed a dialog will be displayed to confirm that the user wishes to delete the image.

```
/**
 * Method used to delete the image from device storage
 */
public void deleteImage() {
    //Dialog to check if user wishes to delete image
    //Dialog built here because of nature of callbacks in inner class- did not use
interface
    AlertDialog.Builder alertDialogBuilder = new AlertDialog.Builder(getActivity());
    alertDialogBuilder.setMessage(R.string.dialog_confirmdelete)
            .setPositiveButton(R.string.dialog_yes, new DialogInterface.OnClickListener() {
                @Override
                public void onClick(DialogInterface dialog, int which) {
                    try {
                        File file = new File(imagePath);
                        file.delete();
                        Toast.makeText(getActivity(), R.string.file_deleted,
Toast.LENGTH_SHORT).show();
                        viewdrawingframe.removeAllViews();
                        updateGallery();
                        delete.setVisibility(View.INVISIBLE);
                    }catch (Exception e){
                        Toast.makeText(getActivity(), R.string.delete_error,
Toast.LENGTH_SHORT).show();
                    }
                }
```

```
            }).setNegativeButton(R.string.dialog_cancel, new
DialogInterface.OnClickListener() {
                @Override
                public void onClick(DialogInterface dialog, int which) {
                    dialog.dismiss();
                }
            }).show();
}
```

When the confirm button is pressed, the image is found from the device storage using its filepath and deleted:

```
File file = new File(imagePath);
file.delete();
```

Chapter 6: Evaluation and Conclusion

6.1 Project Evaluation

In terms of the requirements, the project can be deemed successful as the majority of requirements were successfully implemented. The project also showed that the use of a magnet as an input to mobile devices does have potential as the device can be used to track the magnet position.

However, this is not without limitation. The magnet field readings, despite applying a filter and estimating the terrestrial field, still fluctuated with each use of the application. The magnetic field readings can be affected by the device used, the magnet size and shape used and the environment tested in. This means that results are seldom repeatable when using the application. In terms of the functions developed for the application, the drawing application features were implemented successfully. However, these are not as important as the magnet positioning.

The accuracy was not to the required value of 20% between the actual screen co-ordinates and expected screen co-ordinates. This may be reduced over time by considered all possible conditions which may impact on the readings and minimising them as much as possible.

Treating the magnetic field as non-linear appears to be the correct approach as the magnet tracking was more accurate in this case.

6.2 Future Work

Future work in this project would centre around optimising the estimation of screen co-ordinates. The magnetic stylus used could be developed further and possibly an electrified stylus could be used. This stylus could control its magnetic field strength and toggle whether drawing is active or not. Other features could be implemented to the application such as the ability to share drawings or to load previously saved drawings or photos to edit.

References

1. ‘Mobile Fact Sheet’, web page, pewinternet.org/fact-sheet/mobile/, published 1.12.17, accessed on 23.6.17
2. ‘Fast Facts’, web page, https://www.ofcom.org.uk/about-ofcom/latest/media/facts, accessed 30.6.17
3. ‘Number of smartphone users worldwide from 2014 to 2020 (in billions)’, web page, www.statista.com/statistics/330695/number-of-smartphone-users-worldwide, accessed 30.6.17
4. ‘Global Tablet Penetration Forecast’, web page, https://www.statista.com/statistics/219909/global-tablet-penetration-forecast/], accessed 30.6.17
5. ‘How Art Changes Your Brain: Differential Effects of Visual Art Production and Cognitive Art Evaluation on Functional Brain Connectivity’, http://journals.plos.org/plosone/article?id=10.1371/journal.pone.0101035, accessed 2.7.17
6. ‘Sketchbook’, web page, www.autodesk.com/products/sketchbook/overview, accessed 2.7.17
7. ‘Livescribe Features’, web page, https://www.livescribe.com/uk/smartpen/ls3/features.html], accessed 2.7.17
8. ‘Best Smart Pen 2017’, web page, www.chipin.com/best-smart-pens-reviews , accessed 2.7.17
9. ‘Equil Smartpen 2 Review’, web page, http://www.toptenreviews.com/electronics/family/best-digital-pens/equil-smartpen-2-review/, accessed 3.7.17
10. ‘Hall Effect, web page, www.electronics-tutorials.ws/electromagnetism/hall-effect, accessed on 10.7.17
11. ‘Mountain Goat Software Blog’, web page, www.mountaingoatsoftware.com/blog, accessed on 12.7.17
12. ‘Agile Manifesto’, web page, agilealliance.org, accessed on 12.7.17
13. International Standard Requirements Engineering, document, http://ieeexplore.ieee.org/document/6146379/, accessed 20.7.17
14. ‘SOLID Principles’, web page, deviq.com/solid, accessed 28.8.17
15. ‘Butterknife’, android plugin, https://github.com/JakeWharton/butterknife
16. ‘AOSP Android Code Style for Contributors’, web page, https://source.android.com/source/code-style, accessed 30.8.17
17. ‘Faraday, Lenz and Newton’s Nightmare’, web page, http://www.4physics.com/phy_demo/NewtonsNightmare/NewtonsNightmare.html, accessed 20.7.17
18. ‘Android Sensor Overview’, web page, https://developer.android.com/guide/topics/sensors/sensors_overview.html, accessed 20.6.17
19. ‘Applying Low Pass Filter to Android’s Sensor Readings, web page, https://www.built.io/blog/applying-low-pass-filter-to-android-sensor-s-readings, accessed 8.7.17

20. Understanding Geographical Co-ordinates, web page, http://www.alanzucconi.com/2017/03/13/positioning-and-trilateration/, accessed 20.8.17
21. Input Techniques to the Surface around a Smartphone using a Magnet Attached on a Stylus, Tetsuya Abe et al, paper, https://dl.acm.org/citation.cfm?doid=2851581.2892376, accessed 30.7.17
22. Drawing Shapes with Fingers, Gabriele Mariotti, web page, https://gmariotti.blogspot.co.uk/2014/01/drawing-shapes-with-fingers.html], accessed 20.8.17

Appendices

Appendix A: List of User Stories

Shown below are complete tables of user stories, including those not shown in Chapter 3. The Must-Have requirements are sorted in order of sprint. The remaining user stories are in order of priority.

User Stories- Functional Requirements

ID	Date Created	Category	Description	Priority	Story Points	Acceptance Criteria	Completed	Sprint	Tested	Comments
1.1	19/06/2017	Magnet Detection	As a user, I can move the stylus to cause the application to detect changes in the magnetic field.	Must-have	2	Given the magnetic stylus is moved near the device, the app will receive callbacks from the device magnetometer due to the changing magnetic field readings.	JS	1	JS	Permissions required- see 2.2
1.3	26/06/2017	Magnet Detection	As a user, the raw readings must be corrected for the terrestrial magnetic field before I can draw.	Must-have	8	Given the application is receiving magnetic field readings, the application will use these readings to estimate the earth's magnetic field before the user can draw.	JS	1	JS	
1.4	26/06/2017	Magnet Detection	As a user, I can place the magnetic stylus in each corner of the drawing surface and store the magnetic field readings taken in order to calibrate the application.	Must-have	8	Given the drawing screen is displayed, the user can place the magnetic stylus at each corner of the drawing surface and press a button to save the magnetic field readings at each corner.	JS	1	JS	
1.5	26/06/2017	Drawing	As a user, I can open a drawing canvas on which to draw.	Must-have	3	Given that the home screen of the activity is displayed, the user can press a button which will open the new drawing screen which includes a drawing canvas.	JS	2	JS	
1.16	19/06/2017	Drawing	As a user, I can save a drawing.	Must-have	13	Given that the new drawing screen is displayed, the user can press a button which will save the drawing.	JS	3	JS	Permissions required to save an image- see 2.1
1.6	26/06/2017	Drawing	As a user, I can draw a spot on the canvas at the screen co-ordinates calculated from the stylus' position on the drawing surface.	Must-have	20	Given that the new drawing screen is displayed and the application is calibrated, by default a spot will be drawn onscreen at the screen co-ordinates of the stylus.	JS	2	JS	
1.17	19/06/2017	View Drawings	As a user, I can view the saved drawings in a gallery.	Must-have	8	Given that at least one drawing has been saved to device memory and the home screen is displayed, a scrollable gallery of the saved images will be displayed.	JS	3	JS	

1.19	04/07/2017	View Drawings	As a user, I can delete a saved image from storage	Must-have	8	Given that at least one drawing has been saved to device memory and the home screen is displayed, when an image is clicked on, the user will be able to press a button to delete the selected image.	JS	3	JS	Permissions required to delete an image- see 2.1
1.2	19/06/2017	Magnet Detection	As a user, I can view magnetic field readings on the device display.	Should-have	2	Given that the application is receiving magnetic field readings from the magnetometer, these will be displayed to the user.				
1.13	27/06/2017	Magnet Position	As a user, I can recalibrate the app while drawing	Should-have	5	Given that the new drawing screen has been displayed and the application is calibrated, the user can press a button to recalibrate the application by taking new readings at each corner of the drawing surface.				
1.7	30/06/2017	Drawing	As a user, I can draw a smooth line between consecutive magnet positions	Should-have	20	Given that the new drawing screen is displayed and the application is calibrated, the user can press a button which will change the shape being drawn to a smooth line which will follow the stylus location.				
1.15	31/07/2017	Drawing	As a user, I can clear the drawing canvas.	Should-have	3	Given the new drawing screen is displayed and the application is calibrated, the user can clear the canvas by pressing the ‘clear’ button and selecting ‘confirm’.				
1.2	26/06/2017	Help	As a user, I can view a help screen with information on how to use the application	Should-have	3	Given that the home screen is displayed, the user can access a help screen from the menu which will open a screen with information on how to use the app.				
1.18	20/06/2017	View Drawings	As a user, I can select a saved image to view an enlarged version of the image	Could-have	13	Given that at least one drawing has been saved to device memory and the home screen is displayed, when an image is clicked on, a larger version of the image will appear onscreen.				
1.14	27/06/2017	Drawing	As a user, I can choose to view or hide the sensor readings while drawing	Could-have	5	Given that the new drawing screen is displayed, the user can press a menu button to toggle whether the sensor readings are displayed or not.				
1.21	14/08/2017	Settings	As a user, I can access settings in which they can choose a calibration algorithm.	Could-have	8	Given that the home screen is displayed, the user can open a settings screen in which they can choose a calibration method using radio buttons. This algorithm will be used in the new drawing screen to calculate the screen co-ordinates.				

1.8	25/07/2017	Drawing	As a user, I can draw a circle onscreen centred on the magnet position	Could-have	13	Given that the new drawing screen is displayed and the application is calibrated, the user can press a button which will change the shape being drawn to a circle centred on the stylus position when the user places their finger onscreen and diameter given by the stylus position when the finger is removed.				
1.9	25/07/2017	Drawing	As a user, I can draw a triangle onscreen with corners determined by the magnetic position	Could-have	13	Given that the new drawing screen is displayed and the application is calibrated, the user can press a button which will change the shape being drawn to a triangle with its base drawn from the stylus position when the user places their finger onscreen to the stylus position when the finger is removed. The apex will then be drawn in the same way when the finger again is placed onscreen.				
1.1	25/07/2017	Drawing	As a user, I can draw a rectangle onscreen with corners determined by the magnet position	Could-have	13	Given that the new drawing screen is displayed and the application is calibrated, the user can press a button which will change the shape being drawn to a rectangle with one corner given by the stylus position when the user places a finger onscreen and opposite corner given by the stylus position when the finger is removed.				
1.11	24/07/2017	Drawing	As a user, I can change the stroke colour	Could-have	13	Given that the new drawing screen is displayed, the user can press a button to change the stroke colour by means of a seekbar in a dialog.				
1.12	24/07/2017	Drawing	As a user, I can change the background colour of the canvas	Could-have	13	Given that the new drawing screen is displayed, the user can press a button to change the background colour by means of a seekbar in a dialog.				

User Stories- Non-Functional Requirements

ID	Date Created	Category	Description	Priority	Story Points	Acceptance Criteria	Completed	Sprint	Tested	Comments
2.2	19/06/2017	Permissions	As a user, I can move the stylus which can be detected because the application has the required permission to access the device's magnetic field sensor	Must-have	2	The application can access the uncalibrated magnetic field sensor of the device to receive magnetic field data.	JS	1	JS	Possible duplication of 1.1, as 1.1 depends on this permission
2.5	14/08/2017	Compatibility	As a user, I can run the app on Android devices which have a magnetometer and run at least API 18.	Must-have	8	The application will run on devices which run at least API 18. This API is needed to access the uncalibrated magnetic field sensor of the device.	JS	1	JS	
2.3	19/06/2017	Magnet Detection	As a user, I can draw with the onscreen co-ordinates matching the stylus' relative position on the drawing surface to an error of less than 20%.	Must-have	20	Given that magnetic field readings have been taken in each corner of the drawing surface, an algorithm will be applied to the current magnet field readings to calculate the corresponding screen co-ordinates.		2		It is very difficult to experimentally repeat an accuracy of 20%- the environment, device and magnets used must be carefully controlled
2.1	19/06/2017	Permissions	As a user, I will be able to write to and delete from storage as the application has the required permissions.	Must-have	2	The application can save images to device storage and delete saved images.	JS	3		Possible duplication of 1.16 and 1.19 as they require this permission
2.8	26/06/2017	Saving Drawings	As a user, when I save a drawing, it will be saved to internal storage with a unique file name.	Must-have	5	Given that the save button has been pressed, the drawing will be saved using a filename given by the date and time at which it was saved.	JS	3	JS	

2.9	26/06/2017	Saving Drawings	As a user, when I save a drawing, it will be saved in PNG format.	Must-have	3	Given that the save button has been pressed, the drawing will be saved in PNG format to the device memory.	JS	3	JS	
2.4	20/08/2017	Screen Orientation	As a user, when I use the application it always be displayed in portrait orientation	Should-have	5	The application will always be displayed in portrait orientation regardless of the rotation of the device.	JS			
2.6	07/08/2017	Reliability	As a user, when I move the magnetic stylus, the app will respond to magnetic field changes in the order of 0.1μT.	Should-have	8	The device sensor can detect changes in magnetic field of less than 0.1μT, which will mean that the app responds to slight movements of the magnetic stylus.				This level of accuracy was not achieved repeatably. Possibly accuracy can be improved by future work and a controlled environment
2.7	14/08/2017	Reliability	As a user, when I move the magnetic stylus, the app will respond to magnetic field changes within a timescale of less than 0.1 seconds.	Should-have	8	The rate at which the magnetic sensor receives data is less than every 0.1 seconds.				

User Stories – User Interface Requirements

ID	Date Created	Category	Description	Priority	Story Points	Acceptance Criteria	Sprint	Completed	Tested	Comments
3.3	19/06/2017	User Interface	As a user, the UI behaves as expected	Must-have	8	The whole interface behaves as the user expects.	1	JS	JS	
3.2	26/06/2017	Navigation	As a user, I can navigate between different functions of the application using menus	Must-have	5	Given the application is running, the user can access a menu with various buttons depending on the current screen.	3	JS	JS	
3.1	26/06/2017	User Interface	As a user, I will be presented with dialogs when appropriate when navigating the application.	Should-have	5	Dialogs are shown to the user when a decision must be made or to show that a function is loading.				
3.4	19/06/2017	User Interface	As a user, the UI feels familiar.	Should-have	8	The whole interface is familiar to the user.				
3.5	19/06/2017	User Interface	As a user, the UI is intuitive.	Should-have	8	The whole interface is intuitive to the user.				

List of Function Definitions

Function	Name	Pre-Conditions	Inputs	Processing	Output	Post-conditions	Error Checking
1	Select image from gallery	Home screen displayed, at least one image has been saved	A touch event on the image in scrollView	The path of the image is used to create a new bitmap. The bitmap is scaled and placed in an imageView which is then displayed in a frameLayout	The selected image is displayed in a frameLayout below the scrollView. The delete floating action button is also displayed.	Selected image and delete floating action button are displayed, a new bitmap has been created	If the image cannot be displayed, an error message is displayed in a Toast.
2	Delete image	Image has been selected and the delete floating action button is visible.	A touch event on the delete floating action button and on delete dialog	A dialog is displayed to check that the user wishes to delete the image. The path of the selected image is used to locate and delete the image from memory.	A dialog is displayed to confirm deletion. When confirmed, a toast is displayed to confirm that the image has been deleted. The image is no longer displayed.	The image has been deleted from device memory. No image is selected and the frameLayout is empty. The dialog is dismissed.	If the image cannot be deleted, an error message is displayed in a Toast.
3	Open help screen	Home screen displayed	Touch event on menu item	A fragment transaction is used to display the help screen fragment	The help screen is displayed	The help screen fragment has been added to the view.	If the help screen cannot be displayed, an error message is displayed in a Toast.
4	Open settings	Home screen or new drawing screen displayed	Touch event on menu item	An intent is used to start the settings activity.	The settings screen is displayed	The settings activity has been launched	If the settings screen cannot be displayed, an error message is displayed in a Toast.
5	Choosing an algorithm from settings	Settings screen is displayed	Touch event on radio button	When the user presses the radio button corresponding to an algorithm, the integer corresponding to the chosen algorithm is saved in the application SharedPreferences.	A toast is displayed to confirm that an algorithm has been chosen	The chosen algorithm is saved as an integer in SharedPreferences and retrieved in the	If the algorithm cannot be saved to or retrieved from shared preferences, an error message is displayed as a toast.

						new drawing fragment.	
6	Open new drawing screen	Home screen displayed	Touch event on floating action button	A fragment transaction is used to display the new drawing screen.	The new drawing screen is displayed. A progress dialog will be displayed as the buffer fills with magnetic field readings.	The new drawing screen fragment has been added to the view	If the new drawing screen cannot be displayed, an error message is displayed as a Toast.
7	Hiding sensor readings	Drawing screen displayed, sensor readings visible	Touch event on toggle sensor readings menu item	A boolean is used to check if the readings are visible. If the boolean is true, the textviews and buttons will be made invisible.	No readings or buttons are visible on the drawing canvas.	The textVews containing the sensor readings and the buttons used to calibrate are hidden. Boolean value is false.	None implemented
8	Showing sensor readings	Drawing screen displayed, sensor readings visible	Touch event on menu item	A boolean is used to check if the readings are visible. If the boolean is false, the textviews and buttons will be made visible.	Sensor readings and buttons are visible on the drawing canvas.	The textViews containing the sensor readings and the buttons used to calibrate are displayed. Boolean value is true.	None implemented
9	Calculating corrected magnetic field values	Drawing screen displayed	Raw magnetic field readings from sensor	Readings initially are used to fill a buffer. This is used to estimate the terrestrial magnetic field. This estimate is subtracted from each subsequent reading and a low pass filter applied.	Magnetic field readings in 3 dimensions which have been corrected for the terrestrial field.	Raw sensor reading has been corrected for terrestrial field and lowpass filter applied	A low pass filter reduces noise in sensor readings.
10	Calculating screen co-ordinates	Drawing screen displayed, calibration complete	Current magnetic field readings, readings from	The current magnetic field readings and the readings taken at each corner are used to estimate the screen co-ordinates using the	The calculated screen co-ordinates based on the	Co-ordinates have been estimated and passed to the	If the calculated co-ordinates are outside the dimensions of the

			calibration, chosen calibration algorithm.	chosen calibration algorithm. The algorithm is obtained from SharedPreferences.	current magnet position.	drawing view to be used in drawing the chosen shape.	device screen, they are not sent to the drawing view.
11	Draw spot	Drawing screen displayed, calibration complete, spot setting is chosen (this is the default).	Magnetic field readings, touch event	The calculated screen co-ordinates are sent to the drawing view class and a circle of radius 10 pixels is drawn at this location.	A spot is drawn at calculated screen co-ordinates given by the magnet position.	A spot is drawn on the canvas and the drawing view class will continue to draw spots as the magnet is moved while this setting is chosen.	If the calculated co-ordinates are outside the dimensions of the device screen, they are not sent to the drawing view.
12	Draw line	Drawing screen displayed, calibration complete, line option selected	Magnetic field readings, touch event	The calculated screen co-ordinates are sent to the drawing view class. As new screen co-ordinates are calculated the previous values are saved as a variable and a line is drawn between consecutive sets of co-ordinates.	A line is drawn between consecutive sets of coordinates.	As the magnet is moved further, the co-ordinates used will be updated and a line will continue to be drawn to match the magnet position as the stylus is moved.	If the calculated co-ordinates are outside the dimensions of the device screen, they are not sent to the drawing view.
13	Draw circle	Drawing screen displayed, calibration complete, circle selected	Magnetic field readings, touch event	The calculated screen co-ordinates are sent to the drawing view class. The centre of the circle is given by the co-ordinates when a finger is placed onscreen and the radius given by the co-ordinates when the finger is removed.	Circle drawn onscreen when finger is released from screen.	Circle will remain the selected shape and no drawing will take place until a finger is placed onscreen.	If the calculated co-ordinates are outside the dimensions of the device screen, they are not sent to the drawing view.
14	Draw triangle	Drawing screen displayed, calibration complete, triangle selected	Magnetic field readings, touch event	The calculated screen co-ordinates are sent to the drawing view class. One end of the base of the traingle is given by the co-ordinates when a finger is placed onscreen and the other end given by the co-ordinates when the finger is removed. The other two sides are drawn when the finger is placed	Triangle drawn when finger is released from screen the second time.	Triangle will remain the selected shape and no drawing will take place until a finger is placed onscreen.	If the calculated co-ordinates are outside the dimensions of the device screen, they are not sent to the drawing view.

				onscreen again, with the apex location given by the calculated co-ordinates when the finger is released.			
15	Draw rectangle	Drawing screen displayed, calibration complete, rectangle selected	Magnetic field readings, touch event	The calculated screen co-ordinates are sent to the drawing view class. One corner of the rectangle is given by the co-ordinates when a finger is placed onscreen and the opposite corner given by the co-ordinates when the finger is removed.	Rectangle drawn when finger is removed from screen	Triangle will remain the selected shape and no drawing will take place until a finger is placed onscreen.	If the calculated co-ordinates are outside the dimensions of the device screen, they are not sent to the drawing view.
16	Recalibrate	New drawing screen displayed	Magnetic field readings, touch event	The calibration readings and buttons are made visible and the buttons are enabled, meaning when pressed new readings will be saved for each corner of the drawing surface.	New calibration readings will be taken when each corner button is pressed. The readings and buttons will become invisible when all 4 corners have been calibrated.	The new readings taken in each corner will be used to estimate the screen co-ordinates based on the stylus position.	The user will not be able to draw while recalibration is taking place.
17	Change Stroke Colour	New drawing screen displayed	Touch input to open the dialog, magnetic field readings to choose colour	A dialog will be displayed which includes a seekbar with a colour gradient. The user can move the slider along the seekbar using the magnet position and press the confirm button to change the stroke to the selected colour.	The stroke colour will now be the chosen colour when the user continues to draw.	The chosen colour will be saved as an integer which measures the distance along the seekbar.	If the dialog screen cannot be displayed, an error message is displayed as a Toast.
18	Change Background Colour	New drawing screen displayed	Touch input to open the dialog, magnetic field readings to choose colour	A dialog will be displayed which includes a seekbar with a colour gradient. The user can move the slider along the seekbar using the magnet position and press the confirm button to change the background of the canvas to the selected colour.	The background colour of the canvas will be changed to selected colour	The chosen colour will be saved as an integer which measures the distance along the seekbar.	If the dialog screen cannot be displayed, an error message is displayed as a Toast.

19	Clear Drawing Canvas	New drawing screen displayed, a drawing is on the canvas	Touch event on menu item and dialog	A dialog will be displayed to confirm that the user wishes to clear the drawing. When the user presses confirm, a transparent bitmap will be drawn over the canvas to erase it.	Clear bitmap will be drawn over the canvas	The drawing has been cleared	If the dialog screen cannot be displayed, an error message is displayed as a Toast.
20	Save	New drawing screen is displayed, calibrated and the user has started to draw	Touch event on menu item	A bitmap image is taken from the drawing view, compressed, given a unique filename and saved as a PNG image to the device storage.	Toast is displayed to confirm the drawing has been saved	The drawing has been saved to device storage as a PNG file with a unique filename	If the drawing cannot be saved, an error message is displayed as a Toast.
21	Save Before Exiting	New drawing screen is displayed	Device back button press	When the user presses back from the new drawing screen, a dialog will display to confirm that the user wishes to save the image before returning to the home screen. When the user presses confirm the image will be saved.	Toast is displayed to confirm the drawing has been saved	The drawing has been saved to device storage as a PNG file with a unique filename	If the drawing cannot be saved, an error message is displayed as a Toast.

Appendix B: Won’t-have-today requirements

ID	Date Created	Category	Description	Priority	Story Points	Acceptance Criteria	Sprint	Completed	Tested	Comments
1.22	19/06/2017	Drawing	As a user, I can write words onscreen using the stylus which will be recognised by a handwriting recognition API.	Won't-have	20	Given that the new drawing screen is displayed and the user has produced handwriting onscreen, a button can be pressed to display the equivalent text.				
1.23	22/06/2017	Drawing	As a user, I can press a button to toggle the magnetic stylus input and therefore turn drawing on and off	Won't-have	5	Given that the new drawing screen is displayed, when the user presses a button, the magnetic stylus input will be toggled.				
1.24	30/06/2017	Drawing	As a user, when I select a shape or line to draw, the selected shape will remain highlighted.	Won't-have	8	Given that the new drawing screen is displayed, the selected shape icon will be highlighted in the menu.				
1.25	10/07/2017	Help	As a first-time user of the application, I will be presented with a walkthrough when drawing for the first time	Won't-have	5	Given that the application is being launched for the first time, when the user opens the drawing screen, a walkthrough will explain how to draw with the application.				

Appendix C: Kanban board screenshots

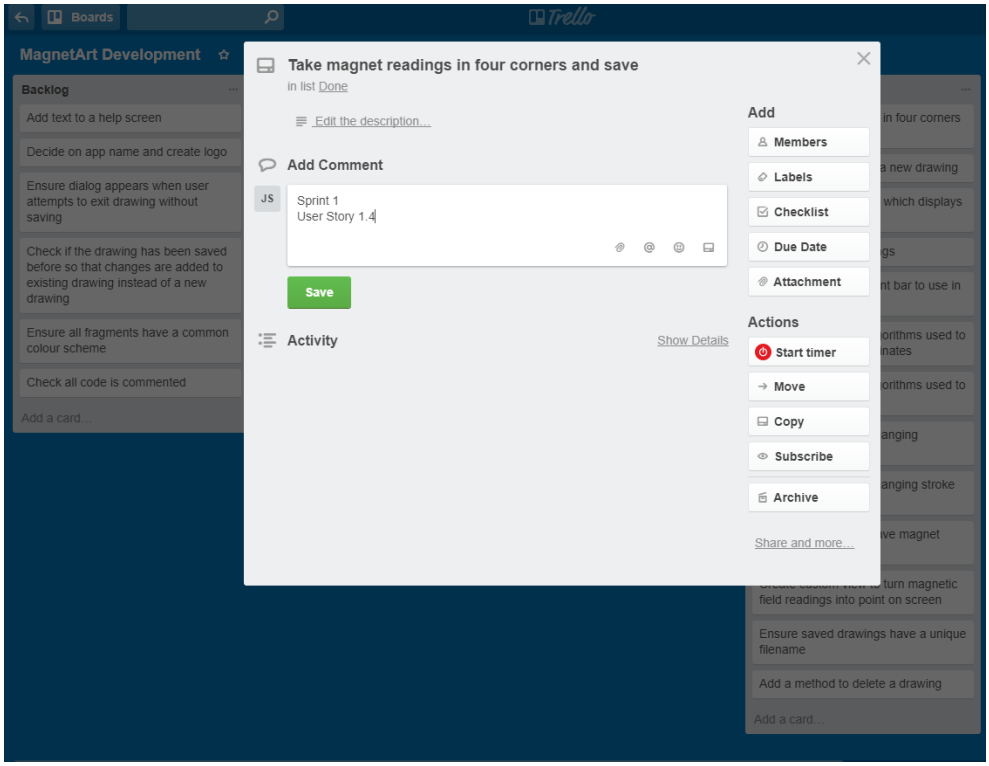


Fig.i A task written on a card

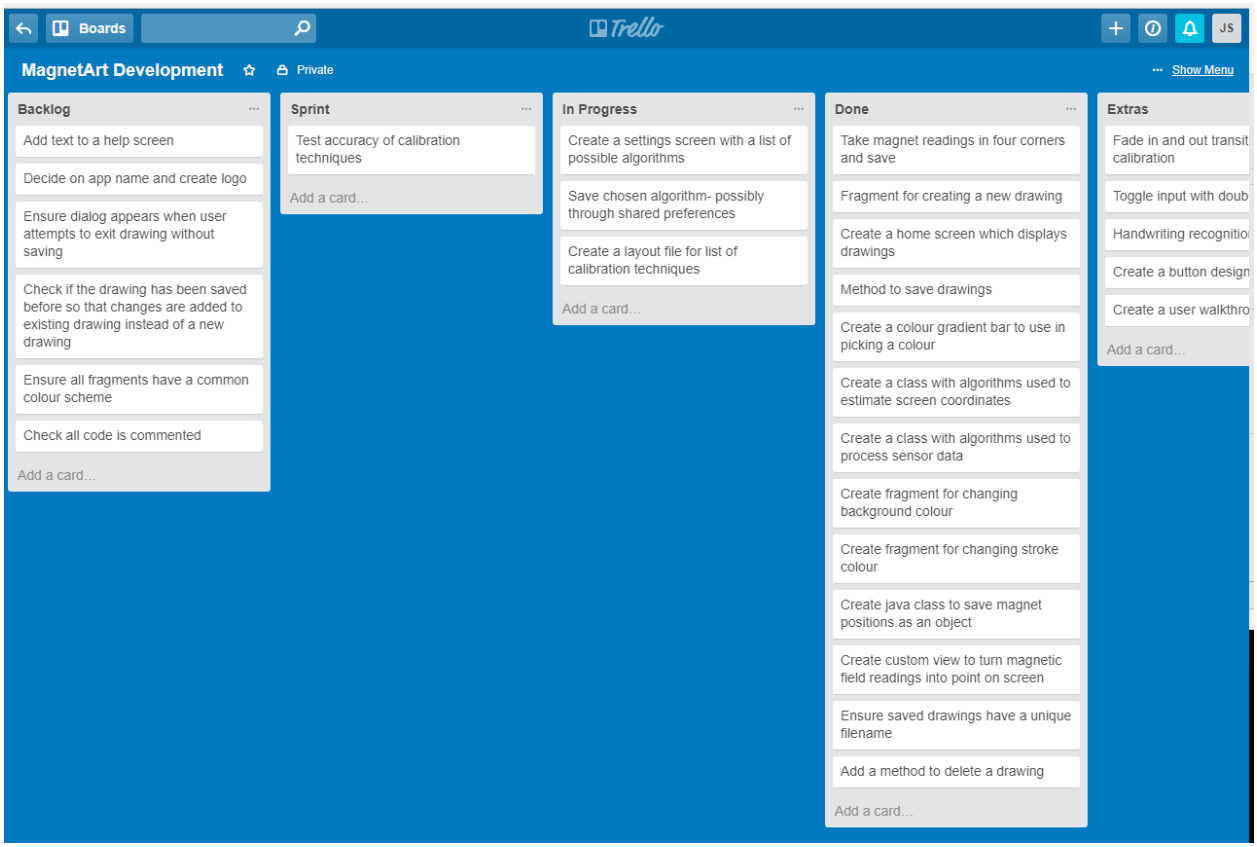


Fig.ii The Kanban board showing lists and cards

Appendix D: Implementations (Source Code)

MainActivity.java

```
package com.example.jorda.magnetart.activity;

import android.os.Bundle;
import android.support.v7.app.AppCompatActivity;
import android.widget.Toast;

import com.example.jorda.magnetart.fragment.MainFragment;
import com.example.jorda.magnetart.R;


/**
 * This main activity controls the fragments that make up the home screen- There
will be a toolbar at the top, a scrollview
 * of the saved documents and a floating action button to create a new document
 * When the application is opened, the main_fragment will be displayed
 */
public class MainActivity extends AppCompatActivity {

    //Starting point for the activity
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        //Layout associated with this activity is defined in activity_main.xml
        //This contains a container view for fragments
        setContentView(R.layout.activity_main);

        if (savedInstanceState == null) {
            try {
                getFragmentManager().beginTransaction().add(R.id.container, new
MainFragment()).commit();
            } catch (NullPointerException e) {
                e.printStackTrace();
                Toast.makeText(this, R.string.Fragment_not_Displayed,
Toast.LENGTH_SHORT).show();
            }
        }
    }

}
```

UserPreferencesActivity.java

```
package com.example.jorda.magnetart.activity;

import android.content.Context;
import android.content.SharedPreferences;
import android.os.Bundle;
import android.support.v7.app.ActionBar;
import android.support.v7.app.AppCompatActivity;
import android.view.View;
import android.widget.RadioButton;
import android.widget.RadioGroup;
import android.widget.Toast;

import com.example.jorda.magnetart.R;
```

```
/**
 * Allows the user to select an algorithm with which to calibrate the application
from a list.
 * The algorithm will be saved as an integer in SharedPreferences
 */
public class UserPreferencesActivity extends AppCompatActivity {

    private static final String KEY = "algoprefs";
    private static final String INT_KEY = "Chosen Algorithm";

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        ActionBar actionBar = getSupportActionBar();

        try {
            //Hide the arrow in actionbar, o ensure the user returns to previous
fragment on back press
            actionBar.setDisplayHomeAsUpEnabled(false);
        }catch(NullPointerException e){
            Toast.makeText(this, R.string.error_actionbar,
Toast.LENGTH_SHORT).show();
        }
        setContentView(R.layout.activity_preferences);
        createRadioButtons();
    }

    /**
     * Method which iterates through the radiogroup to create a radio button for
each algorithm named in layout file
     */
    private void createRadioButtons() {

        RadioGroup rg = (RadioGroup) findViewById(R.id.radio_group_algo);
        //algorithm names taken from resource
        String[] algorithmOptions =
getResources().getStringArray(R.array.algo_list);

        for (int i = 0; i < algorithmOptions.length; i++) {

            final String algoname = algorithmOptions[i];//Get algorithm name
            final int algo = i + 1;//Algorithm number- starts at 1
            //Radio buttons are produced by iterating through the algorithm names
            RadioButton rb = new RadioButton(this);
            rb.setText(algoname);
            rb.setTextAppearance(this,
android.R.style.TextAppearance_DeviceDefault_Medium);

            //When the button is pressed, get the integer associated with that
algorithm, and send to SharedPreferences using
            // save AlgorithmSelected method
            rb.setOnClickListener(new View.OnClickListener() {
                @Override
                public void onClick(View v) {
                    Toast.makeText(UserPreferencesActivity.this,
getString(R.string.chosen_algo) +algo + " : " + algoname,
Toast.LENGTH_SHORT).show();
                    saveAlgorithmSelected(algo);
                }
            });
            rg.addView(rb);

            //select default setting- ensures there is always a selected item
            if (algo == getAlgorithmSelected(this)) {
                rb.setChecked(true);
            }
        }
```

```
    }

    /**
     * Saves the algorithm integer in SharedPreferences
     * @param algo
     */
    private void saveAlgorithmSelected(int algo) {
        try {
            SharedPreferences sp = this.getSharedPreferences(KEY, MODE_PRIVATE);
            SharedPreferences.Editor editor = sp.edit();
            editor.putInt(INT_KEY, algo);
            editor.apply();
        } catch (ClassCastException e) {
            Toast.makeText(this, R.string.variable_wrong_type,
Toast.LENGTH_SHORT).show();
        }
    }

    /**
     * Method to get the algorithm number from SharedPreferences- this method can
be called from other fragments
     */
    public static int getAlgorithmSelected(Context context) {

        int getalgo = 0;
        SharedPreferences sp = context.getSharedPreferences(KEY, MODE_PRIVATE);
        try {
            getalgo = sp.getInt(INT_KEY, 0);//assign to value from
sharedpreferences
        } catch (NullPointerException e) {
            Toast.makeText(context, R.string.algorithm_nullpointer,
Toast.LENGTH_SHORT).show();
        }
        return getalgo;
    }

    /**
     * Method to ensure that pressing the device back button returns to previous
fragment
     */
    @Override
    public void onBackPressed() {
        super.onBackPressed();
        finish();
    }
}
```

BackgroundColourFragment.java

```
package com.example.jorda.magnetart.dialogFragment;

import android.app.AlertDialog;
import android.app.Dialog;
import android.app.DialogFragment;
import android.content.DialogInterface;
import android.os.Bundle;
import android.view.View;

import com.example.jorda.magnetart.interfaces.BackgroundColourListener;
import com.example.jorda.magnetart.R;
import com.example.jorda.magnetart.view.ColourSeekBar;
import static com.example.jorda.magnetart.view.ColourSeekBar.convertValueToColor;

/**
```

```
 * Class to define a dialog which allows the user to choose the background colour
for the application
 */
public class BackgroundColourFragment extends DialogFragment {

    private BackgroundColourListener mListener;// Instance of interface which
defines methods for handling dialog callbacks
    public ColourSeekBar backgroundcoloursb;// Instance of the ColourSeekBar custom
view object
    private float currentBackgroundColour;// The selected colour

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

        //Checks that the calling fragment implements the listener interface
        try{
            mListener = (BackgroundColourListener) getTargetFragment();
        } catch (ClassCastException e) {
            throw new
ClassCastException(getString(R.string.bcf_classcastexception));
        }

        currentBackgroundColour = 0.0f;//set the initial value of the colour

    }

    /**
     * Class to define the dialog
     * @param savedInstanceState
     * @return
     */
    public Dialog onCreateDialog(Bundle savedInstanceState){

        View rootView =
getActivity().getLayoutInflater().inflate(R.layout.fragment_changebgcolour,null,fal
se);

        AlertDialog.Builder builder = new AlertDialog.Builder(getActivity());

builder.setTitle(R.string.dialog_selectbgc).setPositiveButton(R.string.dialog_confi
rm, new DialogInterface.OnClickListener() {// Confirm selected
                    //when confirm button is pressed the associated method is
called in the calling fragment
            @Override
                    public void onClick(DialogInterface dialog, int which) {

mListener.onBackgroundPositiveClick(BackgroundColourFragment.this);
                    }
                })//when cancel button is pressed the associated method is called
in the calling fragment
                .setNegativeButton(R.string.dialog_cancel, new
DialogInterface.OnClickListener() {
                    @Override
                    public void onClick(DialogInterface dialog, int which) {//
Cancel selected

mListener.onBackgroundNegativeClick(BackgroundColourFragment.this);
                    }
                })

                .setView(rootView);

        backgroundcoloursb = (ColourSeekBar)
rootView.findViewById(R.id.backgroundcolourbar);

        return builder.create();
    }

    /**
```

```
     * Getter for the selected background colour
     * The colour is saved as an integer
     * @return
     */
    public int getCurrentBackgroundColour() {
        return convertValueToColor(currentBackgroundColour);
    }

    // SETTER (also sets value in colourseekbar)
    public void setCurrentBackgroundColour(float value) {
        currentBackgroundColour = value;
        if(backgroundcoloursb != null) {
            backgroundcoloursb.setProgress((int) value);
            backgroundcoloursb.invalidate();
        }
    }
}
```

ClearDrawingFragment.java

```
package com.example.jorda.magnetart.dialogFragment;

import android.app.AlertDialog;
import android.app.Dialog;
import android.app.DialogFragment;
import android.content.DialogInterface;
import android.os.Bundle;

import com.example.jorda.magnetart.R;
import com.example.jorda.magnetart.interfaces.ClearDrawingListener;

/**
 * Created by jordan on 04/08/2017.
 */

/**
 * Class to define a dialog called when user wishes to clear the canvas
 */
public class ClearDrawingFragment extends DialogFragment {

    private ClearDrawingListener mListener;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        try{
            mListener = (ClearDrawingListener)getTargetFragment();
        }catch (ClassCastException e){
            throw new ClassCastException("Calling fragment does not implement
interface");
        }
    }

    public Dialog onCreateDialog(Bundle savedInstanceState){

        AlertDialog.Builder builder = new AlertDialog.Builder(getActivity());
        builder.setMessage(R.string.clear_message)
                // Confirm selected
                .setPositiveButton(R.string.dialog_yes, new
DialogInterface.OnClickListener() {
                    @Override
                    public void onClick(DialogInterface dialog, int which) {
```

```
                        mListener.onClearPositiveClick(ClearDrawingFragment.this);
                    }
                })
                // Cancel selected
                .setNegativeButton(R.string.dialog_cancel, new
DialogInterface.OnClickListener() {
                    @Override
                    public void onClick(DialogInterface dialog, int which) {
                        mListener.onClearNegativeClick(ClearDrawingFragment.this);
                    }
                });

        return builder.create();
    }
}
```

StrokeColourFragment.java

```
package com.example.jorda.magnetart.dialogFragment;

import android.app.AlertDialog;
import android.app.Dialog;
import android.app.DialogFragment;
import android.content.DialogInterface;
import android.os.Bundle;
import android.view.View;

import com.example.jorda.magnetart.R;
import com.example.jorda.magnetart.interfaces.StrokeColourListener;
import com.example.jorda.magnetart.view.ColourSeekBar;

import static com.example.jorda.magnetart.view.ColourSeekBar.convertValueToColor;

public class StrokeColourFragment extends DialogFragment {

    // Listener Instance
    private StrokeColourListener mListener;

    // View Instance
    public ColourSeekBar strokecolourseekbar;

    // Current Color Value
    private float currentStrokeColour;

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

        // Make sure calling fragment uses the listener interface
        try{
            mListener = (StrokeColourListener) getTargetFragment();
        } catch (ClassCastException e) {
            throw new ClassCastException("Calling fragment must implement the
StrokeColourListener Interface");
        }

        currentStrokeColour = 0.0f;

    }

    public Dialog onCreateDialog(Bundle savedInstanceState){

        View rootView =
getActivity().getLayoutInflater().inflate(R.layout.fragment_changestrokecolour,null
```

```
,false);


        AlertDialog.Builder builder = new AlertDialog.Builder(getActivity());
        builder.setTitle(R.string.select_stroke)
                // Confirm selected
                .setPositiveButton(R.string.dialog_confirm, new
DialogInterface.OnClickListener() {
                    @Override
                    public void onClick(DialogInterface dialog, int which) {
                        mListener.onStrokePositiveClick(StrokeColourFragment.this);
                    }
                })
                // Cancel selected
                .setNegativeButton(R.string.dialog_cancel, new
DialogInterface.OnClickListener() {
                    @Override
                    public void onClick(DialogInterface dialog, int which) {
                        mListener.onStrokeNegativeClick(StrokeColourFragment.this);
                    }
                })
                .setView(rootView);
        strokecolourseekbar = (ColourSeekBar)
rootView.findViewById(R.id.strokecolourbar);
        return builder.create();
    }

    // Return color int
    public int getCurrentStrokeColour() {
        return convertValueToColor(currentStrokeColour);
    }

    // SETTER (also sets value in colourseekbar)
    public void setCurrentStrokeColour(float value) {
        currentStrokeColour = value;
        if(strokecolourseekbar != null) {
            strokecolourseekbar.setProgress((int) value);
            strokecolourseekbar.invalidate();
        }
    }
}
```

HelpFragment.java

```
package com.example.jorda.magnetart.fragment;

import android.app.Fragment;
import android.app.FragmentTransaction;
import android.os.Bundle;
import android.view.LayoutInflater;
import android.view.Menu;
import android.view.MenuInflater;
import android.view.MenuItem;
import android.view.View;
import android.view.ViewGroup;
import android.widget.Button;
import android.widget.TextView;

import com.example.jorda.magnetart.R;

/**
 * Created by jordan on 04/08/2017.
 */
```

```
/**
 * Fragment to define help screen
 */
public class HelpFragment extends Fragment {

    private TextView par1;
    private Button beginbutton;

    public View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle
SavedInstanceState){

        View rootView = inflater.inflate(R.layout.fragment_help, container, false);
        Initialise(rootView);
        setHasOptionsMenu(true);

        return rootView;
    }

    public void Initialise(View rootView){
        par1 = (TextView)rootView.findViewById(R.id.helpfirstpp);
        beginbutton = (Button)rootView.findViewById(R.id.helpreturnbutton);
        //Go to NewDrawingFragment
        beginbutton.setOnClickListener(new View.OnClickListener(){
            @Override
            public void onClick(View v) {
                FragmentTransaction ft = getFragmentManager().beginTransaction();
                ft.replace(R.id.container, new NewDrawingFragment());
                ft.addToBackStack(null);
                ft.commit();
            }
        });
        getActivity().setTitle(getString(R.string.help_title));

    }

    @Override
    public void onCreateOptionsMenu(Menu menu, MenuInflater inflater) {
        menu.clear();
        // Inflate the main menu; this adds items to the action bar if it is
present.
        inflater.inflate(R.menu.helpscreen_menu, menu);
        super.onCreateOptionsMenu(menu, inflater);
    }

    @Override
    public boolean onOptionsItemSelected(MenuItem item) {

        switch (item.getItemId()){

            case R.id.return_home://Return to main fragment- view drawings
                FragmentTransaction ft = getFragmentManager().beginTransaction();
                ft.replace(R.id.container, new MainFragment());
                ft.addToBackStack(null);
                ft.commit();
                break;
        }

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

MainFragment.java

```
package com.example.jorda.magnetart.fragment;

import android.app.Fragment;
import android.app.FragmentTransaction;
import android.os.Bundle;
import android.view.LayoutInflater;
import android.view.Menu;
import android.view.MenuInflater;
import android.view.MenuItem;
import android.view.View;
import android.view.ViewGroup;
import android.widget.Button;
import android.widget.TextView;

import com.example.jorda.magnetart.R;

/**
 * Created by jordan on 04/08/2017.
 */

/**
 * Fragment to define help screen
 */
public class HelpFragment extends Fragment {

    private TextView par1;
    private Button beginbutton;

    public View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle
SavedInstanceState){

        View rootView = inflater.inflate(R.layout.fragment_help, container, false);
        Initialise(rootView);
        setHasOptionsMenu(true);

        return rootView;
    }

    public void Initialise(View rootView){
        par1 = (TextView)rootView.findViewById(R.id.helpfirstpp);
        beginbutton = (Button)rootView.findViewById(R.id.helpreturnbutton);
        //Go to NewDrawingFragment
        beginbutton.setOnClickListener(new View.OnClickListener(){
            @Override
            public void onClick(View v) {
                FragmentTransaction ft = getFragmentManager().beginTransaction();
                ft.replace(R.id.container, new NewDrawingFragment());
                ft.addToBackStack(null);
                ft.commit();
            }
        });
        getActivity().setTitle(getString(R.string.help_title));

    }

    @Override
    public void onCreateOptionsMenu(Menu menu, MenuInflater inflater) {
        menu.clear();
        // Inflate the main menu; this adds items to the action bar if it is
present.
        inflater.inflate(R.menu.helpscreen_menu, menu);
        super.onCreateOptionsMenu(menu, inflater);
    }

    @Override
    public boolean onOptionsItemSelected(MenuItem item) {

        switch (item.getItemId()){

            case R.id.return_home://Return to main fragment- view drawings
```

```
                FragmentTransaction ft = getFragmentManager().beginTransaction();
                ft.replace(R.id.container, new MainFragment());
                ft.addToBackStack(null);
                ft.commit();
                break;
        }

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

NewDrawingFragment.java

```
package com.example.jorda.magnetart.fragment;

import android.app.AlertDialog;
import android.app.DialogFragment;
import android.app.Fragment;
import android.app.FragmentTransaction;
import android.app.ProgressDialog;
import android.content.Context;
import android.content.DialogInterface;
import android.content.Intent;
import android.graphics.Bitmap;
import android.graphics.Canvas;
import android.graphics.Color;
import android.graphics.Paint;
import android.graphics.Rect;
import android.hardware.Sensor;
import android.hardware.SensorEvent;
import android.hardware.SensorEventListener;
import android.hardware.SensorManager;
import android.os.Bundle;
import android.util.Log;
import android.view.KeyEvent;
import android.view.LayoutInflater;
import android.view.Menu;
import android.view.MenuInflater;
import android.view.MenuItem;
import android.view.View;
import android.view.ViewGroup;
import android.widget.Button;
import android.widget.RelativeLayout;
import android.widget.TextView;
import android.widget.Toast;

import com.example.jorda.magnetart.R;
import com.example.jorda.magnetart.activity.UserPreferencesActivity;
import com.example.jorda.magnetart.dialogFragment.BackgroundColourFragment;
import com.example.jorda.magnetart.dialogFragment.ClearDrawingFragment;
import com.example.jorda.magnetart.dialogFragment.StrokeColourFragment;
import com.example.jorda.magnetart.interfaces.BackgroundColourListener;
import com.example.jorda.magnetart.interfaces.ClearDrawingListener;
import com.example.jorda.magnetart.interfaces.StrokeColourListener;
import com.example.jorda.magnetart.utilities.CalibrationCalculations;
import com.example.jorda.magnetart.utilities.MagnetCalculations;
import com.example.jorda.magnetart.utilities.MagnetPosition;
import com.example.jorda.magnetart.view.NewDrawingView;

import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;
import java.util.Calendar;
```

```
public class NewDrawingFragment extends Fragment implements SensorEventListener,
ClearDrawingListener,
        BackgroundColourListener, StrokeColourListener{

    private SensorManager mSensorManager;
    private Sensor mSensor;

    private NewDrawingView dv;//Create a drawingview which will contain the canvas
the user draws on

    private TextView earthx, earthy, earthz, sensorx, sensory, sensorz;
    private TextView tlreading, trreading, blreading, brreading;
    private TextView screenx, screeny, displayalgo;
    private Button btn_tl, btn_tr, btn_bl, btn_br;
    private double[] previousValues = {0,0,0};//used in low pass filter
    private double dwidth, dheight;
    private boolean calibrationcomplete = false;
    private boolean choosingBgColor = false, choosingStrokeColour = false,
drawingactive = true;
    private boolean saved = false;
    private boolean highaccuracy = false;
    private int calibrationcounter = 0;
    private static int algorithm = 1;//chosen algorithm
    private BackgroundColourFragment bcf = new BackgroundColourFragment();
    private StrokeColourFragment scf = new StrokeColourFragment();
    private ClearDrawingFragment cdf = new ClearDrawingFragment();
    private static final String TAG = "MyActivity";
    private static final String dialog = "dialog";
    private MagnetPosition topleftp, toprightp, botleftp, botrightp, currentp,
screenp, earthp;
    private ProgressDialog pd;
    private RelativeLayout mLayout;
    private Menu menu;
    private String mseconds = "magnetart";
    private MagnetCalculations cr = new MagnetCalculations();
    private Toast t1, t2;

    public View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle
savedInstanceState) {

        View rootView = inflater.inflate(R.layout.fragment_newdraw, container,
false);
        setHasOptionsMenu(true);
        initializeVariables(rootView);
        mSensorManager =
(SensorManager)getActivity().getSystemService(Context.SENSOR_SERVICE);
        if
(mSensorManager.getDefaultSensor(Sensor.TYPE_MAGNETIC_FIELD_UNCALIBRATED) != null)
{
            mSensor =
mSensorManager.getDefaultSensor(Sensor.TYPE_MAGNETIC_FIELD_UNCALIBRATED);
            mSensorManager.registerListener(this, mSensor,
SensorManager.SENSOR_DELAY_GAME);

        } else {
            Toast.makeText(getActivity(), R.string.no_sensor,
Toast.LENGTH_SHORT).show();
            btn_tl.setEnabled(false);
            btn_tr.setEnabled(false);
            btn_bl.setEnabled(false);
            btn_br.setEnabled(false);
            pd.dismiss();
        }

        //when each button is pressed, the magnetic field reading is recorded for
that corner of the drawing area and the button is then disabled
        btn_tl.setOnClickListener(
                new Button.OnClickListener(){
                    public void onClick(View v){
```

```
                        btn_tl.setEnabled(false);
                        btn_tl.setBackgroundColor(0xff00ff00);
                        topleftp = new MagnetPosition(currentp.toDoubleArray());
                        tlreading.setText(String.format("TL X: %.3f  Y: %.3f Z:
%.3f", topleftp.xPosition, topleftp.yPosition, topleftp.zPosition));
                        Log.d(TAG, "Top Left Calibrated- TL Readings are above");
                        calibrationcounter++;
                        checkCornersCalibrated();
                    }
                }
        );
        //top right button
        btn_tr.setOnClickListener(
                new Button.OnClickListener(){
                    public void onClick(View v){
                        btn_tr.setEnabled(false);
                        btn_tr.setBackgroundColor(0xff00ff00);
                        toprightp = new MagnetPosition(currentp.toDoubleArray());
                        trreading.setText(String.format("TR X: %.3f  Y: %.3f Z:
%.3f", toprightp.xPosition, toprightp.yPosition, toprightp.zPosition));
                        Log.d(TAG, "Top Right Calibrated- TR Readings are above");
                        calibrationcounter++;
                        checkCornersCalibrated();
                    }
                }
        );
        //bottom left button
        btn_bl.setOnClickListener(
                new Button.OnClickListener(){
                    public void onClick(View v){
                        btn_bl.setEnabled(false);
                        btn_bl.setBackgroundColor(0xff00ff00);
                        botleftp = new MagnetPosition(currentp.toDoubleArray());
                        blreading.setText(String.format("BL X: %.3f  Y: %.3f Z:
%.3f", botleftp.xPosition, botleftp.yPosition, botleftp.zPosition));
                        Log.d(TAG, "Bottom Left Calibrated- BL Readings are
above");
                        calibrationcounter++;
                        checkCornersCalibrated();
                    }
                }
        );
        //bottom right button
        btn_br.setOnClickListener(
                new Button.OnClickListener(){
                    public void onClick(View v){
                        btn_br.setEnabled(false);
                        btn_br.setBackgroundColor(0xff00ff00);
                        botrightp = new MagnetPosition(currentp.toDoubleArray());
                        brreading.setText(String.format("BR X: %.3f  Y: %.3f Z:
%.3f", botrightp.xPosition, botrightp.yPosition, botrightp.zPosition));
                        Log.d(TAG, "Bottom Right Calibrated- BR Readings are
above");
                        calibrationcounter++;
                        checkCornersCalibrated();
                    }
                }
        );
    return rootView;

    }

    /**
     * When 4 corners have been calibrated, the readings are hidden and a Toast is
shown
     */
    public void checkCornersCalibrated(){
        if(calibrationcounter==4){
            toggleReadings();
            Toast.makeText(getActivity(), "Calibration Complete",
```

```
Toast.LENGTH_SHORT).show();
        }
    }

    /**
     * Initialise all the variables for the fragment
     * @param rootView
     */
    private void initializeVariables(View rootView) {

        mLayout = (RelativeLayout)rootView.findViewById(R.id.buttonsandreadings);
        mLayout.setVisibility(View.VISIBLE);
        dv = (NewDrawingView)rootView.findViewById(R.id.drawingview);
        //Default shape is spot drawn at magnet position
        dv.activeShape = NewDrawingView.DrawingShape.SPOT;
        //textviews to display the terrestrial magnetic field readings
        earthx = (TextView)rootView.findViewById(R.id.earthx);
        earthy = (TextView)rootView.findViewById(R.id.earthy);
        earthz = (TextView)rootView.findViewById(R.id.earthz);
        //textviews to display the raw sensor readings
        sensorx = (TextView)rootView.findViewById(R.id.sensorx);
        sensory = (TextView)rootView.findViewById(R.id.sensory);
        sensorz = (TextView)rootView.findViewById(R.id.sensorz);
        //textviews to display the readings in each corner used in calibration
        tlreading = (TextView)rootView.findViewById(R.id.tlreadings);
        trreading = (TextView)rootView.findViewById(R.id.trreadings);
        blreading = (TextView)rootView.findViewById(R.id.blreadings);
        brreading = (TextView)rootView.findViewById(R.id.brreadings);
        //textviews to display the current screen co-ordinates
        screenx = (TextView)rootView.findViewById(R.id.screenx);
        screeny = (TextView)rootView.findViewById(R.id.screeny);
        //display algorithm chosen in settings
        displayalgo = (TextView)rootView.findViewById(R.id.displayAlgorithm);
        //buttons in each corner of the drawing area
        btn_tl = (Button)rootView.findViewById(R.id.tlbutton);
        btn_tr = (Button)rootView.findViewById(R.id.trbutton);
        btn_bl = (Button)rootView.findViewById(R.id.blbutton);
        btn_br = (Button)rootView.findViewById(R.id.brbutton);

        bcf.setTargetFragment(this,0);//set target fragment for the dialog
fragments
        cdf.setTargetFragment(this,0);
        scf.setTargetFragment(this,0);

        //retrieve the algorithm chosen in the activity_preferences activity
        algorithm = UserPreferencesActivity.getAlgorithmSelected(getActivity());
        displayalgo.setText("Chosen algo: "+algorithm);
        //progress dialog shown when the buffer is filling with magnetic field
readings
        pd = new ProgressDialog(getActivity());
        pd.setTitle("Initialising Magnetometer");
        pd.setMessage("Please do not place magnet on drawing surface");
        pd.setProgressStyle(ProgressDialog.STYLE_SPINNER);
        pd.setCancelable(false);
        pd.show();

        //Initialise toasts for screen dimensions here so they can be referenced
outside onSensorChanged
        t1 = Toast.makeText(getActivity(), "", Toast.LENGTH_SHORT);
        t2 = Toast.makeText(getActivity(),"", Toast.LENGTH_SHORT);
    }

    /**
     * Create the options menu
     * @param menu
     * @param inflater
     */
    @Override
    public void onCreateOptionsMenu(Menu menu, MenuInflater inflater) {
        menu.clear();
```

```
        // Inflate the main menu; this adds items to the action bar if it is
present.
        inflater.inflate(R.menu.new_drawing, menu);
        this.menu=menu;
        super.onCreateOptionsMenu(menu, inflater);

    }

    @Override
    public boolean onOptionsItemSelected(MenuItem item) {

        switch (item.getItemId()){

            case R.id.sensor_readings://Toggle if sensor readings are visible or
not
                toggleReadings();
            break;

            case R.id.clear://clear the drawing area
                try {
                    cdf.show(getFragmentManager(), dialog);
                }catch (NullPointerException e){
                    Toast.makeText(getActivity(), R.string.dialog_null_pointer,
Toast.LENGTH_SHORT).show();
                }
                break;

            case R.id.action_spot://draw a spot on screen at magnet position- this
is the default setting
                dv.activeShape = NewDrawingView.DrawingShape.SPOT;
                dv.reset();
                break;

            case R.id.action_smoothline:// draw a smoothline between magnet
positions
                dv.activeShape = NewDrawingView.DrawingShape.SMOOTHLINE;
                dv.reset();
                break;

            case R.id.action_rectangle:
                dv.activeShape = NewDrawingView.DrawingShape.RECTANGLE;;
                dv.reset();
                break;

            case R.id.action_circle:
                dv.activeShape = NewDrawingView.DrawingShape.CIRCLE;
                dv.reset();
                break;

            case R.id.action_triangle:
                dv.activeShape = NewDrawingView.DrawingShape.TRIANGLE;
                dv.reset();
                break;

            case R.id.recalibrate://recalibrate the drawing area by retaking
magnetic field readings at the four corners
                reCalibrate();
                break;

            case R.id.strokecolour://change colour of the stroke/shape
                if(calibrationcounter!=4){//warn the user that app is not
calibrated
                    //User may change colour before calibration so colour will be
set before use
                    Toast.makeText(getActivity(), R.string.calib_not_complete,
Toast.LENGTH_SHORT).show();
                }
                choosingStrokeColour = true;
                drawingactive = false;
                try {
```

```
                    scf.show(getFragmentManager(), dialog);
                }catch(NullPointerException e){
                    Toast.makeText(getActivity(), R.string.dialog_null_pointer,
Toast.LENGTH_SHORT).show();
                }
                break;

            case R.id.bgcolour://change the background colour of the canvas
                if(calibrationcounter!=4){
                    Toast.makeText(getActivity(), R.string.calib_not_complete,
Toast.LENGTH_SHORT).show();
                }
                choosingBgColor = true;
                drawingactive = false;
                try {
                    bcf.show(getFragmentManager(), dialog);
                }catch(NullPointerException e){
                    Toast.makeText(getActivity(), R.string.dialog_null_pointer,
Toast.LENGTH_SHORT).show();
                }
                break;

            case R.id.savetodevice://save the drawing
                if(calibrationcounter!=4){//cannot save without calibrating first
                    Toast.makeText(getActivity(), R.string.calib_not_complete,
Toast.LENGTH_SHORT).show();
                }else {
                    saveDrawingToDevice();
                }
                break;

            case R.id.settingsfromdrawing://change calibration technique- for demo
purposes
                try {
                    Intent goToSettings = new Intent(getActivity(),
UserPreferencesActivity.class);
                    startActivity(goToSettings);
                }catch(NullPointerException e){
                    Toast.makeText(getActivity(), "Could not open settings",
Toast.LENGTH_SHORT).show();
                }
                break;
        }

        return super.onOptionsItemSelected(item);
    }

    /**
     * Method implemented due to SensorEventListener
     * @param sensor
     * @param accuracy
     */
    @Override
    public final void onAccuracyChanged(Sensor sensor, int accuracy) {
        // Check that the magnetic field sensor is of the required accuracy
        if(sensor==mSensor){

            switch(accuracy){
                case 0:
                    Log.d(TAG, getString(R.string.sensor_0));
                    highaccuracy = false;
                    break;
                case 1:
                    Log.d(TAG, getString(R.string.sensor_1));
                    highaccuracy = false;
                    break;
                case 2:
                    Log.d(TAG, getString(R.string.sensor_2));
                    highaccuracy = false;
                    break;
```

```
                case 3:
                    Log.d(TAG, getString(R.string.sensor_3));
                    highaccuracy = true;
                    break;
            }
        }
    }

    /**
     * This method will be called when the magnet is moved
     * @param event
     */
    @Override
    public final void onSensorChanged(SensorEvent event) {

        if (event.sensor.getType() == Sensor.TYPE_MAGNETIC_FIELD_UNCALIBRATED &&
highaccuracy) {
            //Progress dialog is shown while the buffer is being filled to prevent
the user attempting to draw

            MagnetPosition correct = cr.correctReadings(event.values);
//            Create an array of new corrected values
            double[] currentValues = correct.toDoubleArray();
            MagnetCalculations.lowPass(currentValues, previousValues);//apply a low
pass filter to values which are then used to create a new magnetposition object
            currentp = new MagnetPosition(previousValues);
            earthp = cr.getEarthp();

            Log.d(TAG, "X: " + currentp.xPosition + " Y: " +
currentp.yPosition);//log these values to be able to plot them over time, plot
against position in grid
            //use squared paper and plot magnet position against field readings

        }//dismiss dialog
        if (cr.isBufferfull()) {
            pd.dismiss();
        }

        earthx.setText(String.format("Earth X: %.3f", earthp.xPosition));
        earthy.setText(String.format("Earth Y: %.3f", earthp.yPosition));
        earthz.setText(String.format("Earth Z: %.3f", earthp.zPosition));

        sensorx.setText(String.format("X: %.3f ", currentp.xPosition));
        sensory.setText(String.format("Y: %.3f ", currentp.yPosition));
        sensorz.setText(String.format("Z: %.3f ", currentp.zPosition));//these are
changing correctly

        getScreenDimensions();//get the screen dimensions

        //The magnetic field readings are used when choosing a colour
        if (choosingStrokeColour || choosingBgColor) {

            //Get a float from the seekbar which represents the chosen colour
            float colourValue = 100.0f - ((float) currentp.magnitude()) / 255 *
100.0f;
            if (colourValue > 100.0f) {
                colourValue = 100.0f;//set to value at end of bar
            } else if (colourValue < 1.0f) {
                colourValue = 1.0f;//set to value at start of bar
            }

            if (choosingStrokeColour) {//call to method in associated
dialogfragment
                scf.setCurrentStrokeColour(colourValue);
            } else {
                bcf.setCurrentBackgroundColour(colourValue);
            }
        }

        //all corners have been calibrated
```

```
        if (calibrationcounter == 4) {

//                Algorithms 1 and 2 are assuming a linear field
            if(algorithm==1||algorithm==2) {
                screenp = CalibrationCalculations.Calibrate(algorithm, topleftp,
toprightp, botleftp, botrightp, currentp, dwidth, dheight);
            }
        //non-linear field, calculating geometry using technique by Tetsuya Abe
            if (algorithm == 3) {
                screenp = cr.geometryCalc(currentp);//calculate geometry of
environment
            }
            calibrationcomplete = true;
//                Log.d(TAG,"X: "+currentp.xPosition+" Y: "+currentp.yPosition+ "
ScreenX: " + screenp.xPosition + " ScreenY: " + screenp.yPosition);
            screenx.setText(Double.toString(screenp.xPosition));
            screeny.setText(Double.toString(screenp.yPosition));


        }

        if (calibrationcomplete && drawingactive) {//draw on screen by sending co-
ordinates to the drawing class
            //check that estimated coordinates are within the screen dimensions
            //boolean drawingactive prevents drawing when changing colour
            dv.screenX = (float)screenp.xPosition;
            dv.screenY = (float)screenp.yPosition;

            //Toasts are shown for two seconds while sensor updates every 0.02
seconds - therefore the toast will remain shown
            //Even when magnet is returned- need to cancel it
            if (dv.screenX > dwidth) {
                t1.setText("Outside screen width");
                t1.show();
            }else{
                t1.cancel();
            }

            if (dv.screenY > dheight) {
                t2.setText("Outside screen height");
                t2.show();
            }else{
                t2.cancel();
            }
                dv.invalidate();
            }
        }



    public void onResume() {
        super.onResume();
        mSensorManager.registerListener(this, mSensor,
SensorManager.SENSOR_DELAY_GAME);//change to game
        if (getView() == null) {
            return;
        }
        //When the phone back button is pressed, ask user if they wish to save
changes
        getView().setFocusableInTouchMode(true);
        getView().requestFocus();
        getView().setOnKeyListener(new View.OnKeyListener() {
            @Override
            public boolean onKey(View v, int keyCode, KeyEvent event) {
                //If the back button is pressed
                if (event.getAction() == KeyEvent.ACTION_UP && keyCode ==
KeyEvent.KEYCODE_BACK) {
                    //display dialog to ask if they wish to save before exiting
                    AlertDialog.Builder alertDialogBuilder = new
AlertDialog.Builder(getActivity());
```

```
                    alertDialogBuilder.setMessage("Do you wish to save before
quitting?")
                            .setPositiveButton("Save", new
DialogInterface.OnClickListener() {
                                @Override
                                public void onClick(DialogInterface dialog, int
which) {
                                    saveDrawingToDevice();//save drawing
                                    returnToMainFrag();//return to saved drawings
                                }
                            })
                            .setNegativeButton("Don't Save", new
DialogInterface.OnClickListener() {
                                @Override
                                public void onClick(DialogInterface dialog, int
which) {
                                    returnToMainFrag();//return without saving
                                }
                            }).setNeutralButton(R.string.dialog_cancel, new
DialogInterface.OnClickListener() {
                        @Override
                        public void onClick(DialogInterface dialog, int which) {
                            //dismiss dialog, do not return t
                            dialog.dismiss();
                        }
                    }).show();
                    return true;
                }
                return false;
            }
        });
    }

    /**
     * Method to return to main fragment- handles fragment transaction
     */
    public void returnToMainFrag() {
        try {
            FragmentTransaction ft = getFragmentManager().beginTransaction();
            ft.replace(R.id.container, new MainFragment());
            ft.addToBackStack(null);
            ft.commit();
        } catch (NullPointerException e) {
            Toast.makeText(getActivity(), "Could not return home",
Toast.LENGTH_SHORT).show();
        }
        HideToasts();
    }

    public void onPause() {
        super.onPause();
        mSensorManager.unregisterListener(this);
        HideToasts();
    }

    public void HideToasts(){
        //ensure Toasts are cancelled
        if(t1!=null){
            t1.cancel();
        }
        if(t2!=null){
            t2.cancel();
        }
    }


    /**
     * Method to get the screen dimensions using the size of the drawing view
     */
    public void getScreenDimensions(){
```

```
        dwidth = dv.getWidth();
        dheight = dv.getHeight();
        //1920x1080p on oneplus X
    }

    /**
     * Method to make magnetic field readings and calibration buttons visible or
invisible
     */
    public void toggleReadings() {
        //Display sensor readings and buttons
        if (mLayout.getVisibility() == View.INVISIBLE) {
            mLayout.setVisibility(View.VISIBLE);
            menu.findItem(R.id.sensor_readings).setTitle("Hide sensor readings");

        } else if (mLayout.getVisibility() == View.VISIBLE) {
            mLayout.setVisibility(View.INVISIBLE);
            menu.findItem(R.id.sensor_readings).setTitle("Show sensor readings");
        }
        if (calibrationcounter < 4) {
            Toast.makeText(getActivity(), "Calibration Not Complete",
Toast.LENGTH_SHORT).show();
            //buttons will be hidden so make user aware that calibration is not
complete and app will not draw
        }
    }

    /**
     *Reset calibration values used- enable buttons and use onclicklistener again
     */
    public void reCalibrate(){

        calibrationcounter = 0;
        //Remove readings from textviews
        tlreading.setText("TL:");
        trreading.setText("TR:");
        blreading.setText("BL:");
        brreading.setText("BR:");

        //change all corner buttons to grey and enable
        for(int i=0;i<mLayout.getChildCount();i++){
            View v = mLayout.getChildAt(i);
            if(v instanceof Button){//get all view elements which are buttons
                v.setBackgroundColor(getResources().getColor(R.color.colorGrey));
                v.setEnabled(true);
            }
        }

        //Set view to invisible first so that toggleReadings will change to VISIBLE
and update the menu item
        mLayout.setVisibility(View.INVISIBLE);
        toggleReadings();
        //removes readings from the buffers to allow the terrestrial field to be
recalculated
        cr.clearBuffers();
        pd.show();

    }

    /**
     * Method to save the drawing
     */
    public void saveDrawingToDevice(){

        boolean filecreated = false;

        //Check if the image has been saved before, if so then use the same
filepath to update the saved image
        if(!saved) {
            java.util.Calendar c = Calendar.getInstance();
```

```
            //File names are produced using milliseconds from calendar to ensure
they are unique
            mseconds = "magnetart " + c.get(Calendar.MILLISECOND) + ".png";
            saved = true;
        }

        File file = new File(getActivity().getFileStreamPath(mseconds)//use date
and time to create unique stream
                .getPath());
        if (!file.exists()) {
            try {
                filecreated = file.createNewFile();
            } catch (IOException e) {
                e.printStackTrace();
                Toast.makeText(getActivity(), R.string.ioexception,
Toast.LENGTH_SHORT).show();
            }
        }

        try {

            FileOutputStream fos = new FileOutputStream(file);

            System.out.println(fos);

            Bitmap drawingViewBitmap = dv.getBitmap();
            Bitmap bitmapToBeSaved =
Bitmap.createBitmap(drawingViewBitmap.getWidth(), drawingViewBitmap.getHeight(),
Bitmap.Config.ARGB_8888);

            Paint paint = new Paint();
            paint.setColor(Color.WHITE);//Set background colour as white
            Canvas currentCanvas = new Canvas(bitmapToBeSaved);
            Rect drawingRect = new
Rect(0,0,drawingViewBitmap.getWidth(),drawingViewBitmap.getHeight());

            //draw white rectangle in the background
            currentCanvas.drawRect(drawingRect, paint);

            //draw the actual bitmap on the canvas
            currentCanvas.drawBitmap(drawingViewBitmap, drawingRect, drawingRect,
null);

            //save bitmap to the file as a PNG
            bitmapToBeSaved.compress(Bitmap.CompressFormat.PNG, 100, fos);

        } catch (NullPointerException e) {
            e.printStackTrace();
            Toast.makeText(getActivity(), R.string.nullpointer,
Toast.LENGTH_SHORT).show();
        } catch (FileNotFoundException e) {
            e.printStackTrace();
            Toast.makeText(getActivity(), R.string.filenotfound,
Toast.LENGTH_SHORT).show();
        }
        //Display a toast to confirm file is saved
        Toast.makeText(getActivity(), "Saved", Toast.LENGTH_SHORT).show();

    }


//     Callback from ClearDrawingFragment- this clears the canvas when 'confirm'
button is pressed
    @Override
    public void onClearPositiveClick(DialogFragment dialog){
        dv.clearDrawing();
        Toast.makeText(getActivity(), "Drawing Cleared",
Toast.LENGTH_SHORT).show();
    }
```

```
//    Callback from ClearDrawingFragment- this is called when the user selects
cancel on the clear dialog, dialog is dismissed
    @Override
    public void onClearNegativeClick(DialogFragment dialog){
        dialog.dismiss();
    }

//     Callback from StrokeColourFragment- used to change colour when 'confirm' is
pressed
    @Override
    public void onStrokePositiveClick(DialogFragment dialog) {
        //The movement of the magnetic stylus is used when changing colour,
therefore drawing must be disabled to ensure
        // that lines are not drawn on the canvas when the colour is being chosen.
When a colour is chosen the dialog is dismissed
        //and drawing is enabled
        drawingactive = true;
        choosingStrokeColour = false;

dv.initialPaint.setColor(((StrokeColourFragment)dialog).getCurrentStrokeColour());

dv.endingPaint.setColor(((StrokeColourFragment)dialog).getCurrentStrokeColour());
    }

//     Callback from StrokeColourFragment- used to change colour when 'cancel' is
pressed and user does not wish to change strokecolour
    @Override
    public void onStrokeNegativeClick(DialogFragment dialog) {
        drawingactive = true;
        choosingStrokeColour = false;
    }


//      Callback from BackgroundColourFragment- used to change background colour
when 'confirm' is selected
    @Override
    public void onBackgroundPositiveClick(DialogFragment dialog) {
        drawingactive = true;
        choosingBgColor = false;

dv.setBackgroundColor(((BackgroundColourFragment)dialog).getCurrentBackgroundColour
());
    }

//      Callback from BackgroundColourFragment when 'cancel' is selected
    @Override
    public void onBackgroundNegativeClick(DialogFragment dialog) {
        drawingactive = true;
        choosingBgColor = false;
    }
    }
```

BackgroundColourListener.java

```
package com.example.jorda.magnetart.interfaces;

import android.app.DialogFragment;

/**
 * Created by jordan on 29/08/2017.
 */
```

```
// Interface for dialog callbacks
public interface BackgroundColourListener {
    void onBackgroundPositiveClick(DialogFragment dialog);
    void onBackgroundNegativeClick(DialogFragment dialog);
}
```

ClearDrawingListener.java

```
package com.example.jorda.magnetart.interfaces;

import android.app.DialogFragment;

/**
 * Created by jordan on 29/08/2017.
 */
    public interface ClearDrawingListener {
        void onClearPositiveClick(DialogFragment dialog);
        void onClearNegativeClick(DialogFragment dialog);

    }
```

StrokeColourListener.java

```
package com.example.jorda.magnetart.interfaces;

import android.app.DialogFragment;

/**
 * Created by jordan on 29/08/2017.
 */
    public interface ClearDrawingListener {
        void onClearPositiveClick(DialogFragment dialog);
        void onClearNegativeClick(DialogFragment dialog);

    }
```

CalibrationCalculations.java

```
package com.example.jorda.magnetart.utilities;

/**
 * Created by jordan on 14/08/2017.
 */

/**
 * This class contains algorithms attempted when calibrating the device.
 * Agorithms detailed are - fourcorners (assume linear and non-linear field),
trilateration, rotation matrices, projection on planes
 */
public class CalibrationCalculations {

    /**
     * Method to take sensor data and produce an estimated set of screen co-
ordinates
```

```
     * @param algorithm
     * @param topleft
     * @param topright
     * @param botleft
     * @param botright
     * @param current
     * @param dwidth
     * @param dheight
     * @return
     */
    public static MagnetPosition Calibrate (int algorithm, MagnetPosition topleft,
MagnetPosition topright, MagnetPosition botleft, MagnetPosition botright,
MagnetPosition current, double dwidth, double dheight) {
        double xLoc = 0, yLoc = 0;

        //4 corners algorithm
        if (algorithm == 1) {

            //calculate fractional distance along surface in x direction at top
        double xloc1 = (topleft.xPosition-current.xPosition)/(topleft.xPosition -
topright.xPosition);
            //calculate fractional distance along surface in x direction at bottom
        double xloc2 = (botleft.xPosition-current.xPosition)/(botleft.xPosition -
botright.xPosition);
            //take an average
        double avgx = (xloc1+xloc2)/2;
            //calculate fractional distance along y axis at left
        double yloc1 = (topleft.yPosition-current.yPosition)/(topleft.yPosition -
botleft.yPosition);
            //calculate fractional distance along y axis on right
        double yloc2 = (topright.yPosition-current.yPosition)/(topright.yPosition -
botright.yPosition);
//            take an average y
            double avgy = (yloc1 + yloc2) / 2;

            xLoc = avgx*dwidth;//multiply by screen dimensions
            yLoc = avgy*dheight;


            //trilateration
        } else if (algorithm == 2) {

            //Treat top left, top right and bottom left as reference points
            //Ignore z dimension- assume zero
            double i1 = topleft.xPosition;
            double i2 = topright.xPosition;
            double i3 = botleft.xPosition;
            double j1 = topleft.yPosition;
            double j2 = topright.yPosition;
            double j3 = botleft.yPosition;
            double x = current.xPosition;
            double y = current.yPosition;

            // Distances from reference points to current point
            double distxi1 = x - i1;
            double distyj1 = y - j1;
            double distxi2 = x - i2;
            double distyj2 = y - j2;
            double distxi3 = x - i3;
            double distyj3 = y - j3;

            double d1 = Math.sqrt(Math.pow(distxi1, 2) + Math.pow(distyj1,
2));//distance from top left
            double d2 = Math.sqrt(Math.pow(distxi2, 2) + Math.pow(distyj2,
2));//distance from top right
            double d3 = Math.sqrt(Math.pow(distxi3, 2) + Math.pow(distyj3,
2));//distance from bottom left

            //Solve simultaneous equations for x and cast as xPos to estimate the
location
```

```
            double xPos = ((Math.pow(d1, 2) - Math.pow(d2, 2) - Math.pow(i1, 2) +
Math.pow(i2, 2) - Math.pow(j1, 2) + Math.pow(j2, 2)) * (-2 * j2 + 2 * j3)
                    - (Math.pow(d2, 2) - Math.pow(d3, 2) - Math.pow(i2, 2) +
Math.pow(i3, 2) - Math.pow(j2, 2) + Math.pow(j3, 2)) * (-2 * j1 + 2 * j2)) / ((-2 *
j2 + 2 * j3) * (-2 * i1 + 2 * i2)
                    - (-2 * j1 + 2 * j2) * (-2 * i2 + 2 * i3));


//        Solve equations for y
            double yPos = (((Math.pow(d1, 2) - Math.pow(d2, 2) - Math.pow(i1, 2) +
Math.pow(i2, 2) - Math.pow(j1, 2) + Math.pow(j2, 2)) * (-2 * i2 + 2 * i3) - (-2 *
i1 + 2 * i2) *
                    (Math.pow(d2, 2) - Math.pow(d3, 2) - Math.pow(i2, 2) +
Math.pow(i3, 2) - Math.pow(j2, 2) + Math.pow(j3, 2))) / (((-2 * j1 + 2 * j2) * (-2
* i2 + 2 * i3)) - (-2 * i1 + 2 * i2) * (-2 * j2 + 2 * j3)));


            // Find estimated position as a fraction of x and distance and multiply
by screen dimensions
            xLoc = Math.abs(((i1 - xPos) / (i1 - botright.xPosition))) * dwidth;
            yLoc = Math.abs(((j1 - yPos) / (j1 - botright.yPosition))) * dheight;

        }
        return new MagnetPosition(xLoc, yLoc, 0);
    }
}
```

MagnetCalculations.java

```
package com.example.jorda.magnetart.utilities;

/**
 * Created by jordan on 03/09/2017.
 */

public class MagnetCalculations {

    private int buffsize = 500;
    private double[] xarray = new double[buffsize];
    private double[] yarray = new double[buffsize];
    private double[] zarray = new double[buffsize];
    private int iter = 1;
    private boolean bufferfull = false;
    private double dx = 0, dy = 0, dz = 0, ex = 0, ey = 0, ez = 0, bx = 0, by = 0,
bz = 0;
    private static final float ALPHA = 0.1f;
    private double l = 0.001;//magnet length- in this case 1cm
    private double dr, r, xa, ya;
    private MagnetPosition earthp = new MagnetPosition(0,0,0);//create earth
magnetposition and pass to newdrawingfrag to display earth field

    public MagnetPosition getEarthp() {
        return earthp;
    }

    public boolean isBufferfull() {
        return bufferfull;
    }

    /**
     * Method to correct readings for the terrestrial magnetic field
     * @param eventvalues
     * @return
     */
```

```
    public MagnetPosition correctReadings(float[] eventvalues) {

        double sumex = 0, sumey = 0, sumez = 0;//buffers are initially empty

        if (!bufferfull) {
            //Add new sensor readings to the buffer to fill it
            if (xarray[buffsize - 1] == 0) {
                xarray[iter] = eventvalues[0];
                yarray[iter] = eventvalues[1];
                zarray[iter] = eventvalues[2];
                iter++;

            } else {
                //Add all buffer values in x, y, and z dimensions
                for (int i = 0; i < buffsize; i++) {
                    sumex += xarray[i];
                    sumey += yarray[i];
                    sumez += zarray[i];
                }
                //Take an average value for each dimension, this is taken to be the
terrestrial magnetic field
                ex = sumex / buffsize;
                ey = sumey / buffsize;
                ez = sumez / buffsize;

                earthp = new MagnetPosition(ex,ey,ez);//the estimated terrestrial
field in 3 dimensions

                bufferfull = true;//x, y and z buffers have been filled

            }

        }

        bx = eventvalues[2];
        by = eventvalues[1];
        bz = eventvalues[0];

        //correct for the terrestrial magnetic field
        dx = bx - ex;
        dy = by - ey;
        dz = bz - ez;

        return new MagnetPosition(dx, dy, dz);
    }

    /**
     * Reset buffers to recalculate terrestrial field
     */
    public void clearBuffers(){

        xarray = new double[buffsize];
        yarray = new double[buffsize];
        zarray = new double[buffsize];
        bufferfull = false;
        iter = 0;
    }


    /**
     * Low-Pass filter to attenuate high frequency signals and ensure continuous
data
     * @param input
     * @param output
     * @return
     */
    public static double[] lowPass(double[] input, double[] output) {

        if (output == null) {
            return input;//first set of data- return input
```

```
        }
        for (int i = 0; i < input.length; i++) {
            output[i] += ALPHA * (input[i] - output[i]);//ALPHA controls how much
of the input signal is attenuated
        }
        return output;
    }

    /**
     * Method to calculate the relative co-ordinates by solving Coulomb's equation
     * @param currentpos
     * @return
     */
    public MagnetPosition geometryCalc(MagnetPosition currentpos){

        double dx = currentpos.xPosition;
        double dy = currentpos.yPosition;

        dr = - Math.sqrt(Math.pow(dx,2)+Math.pow(dy,2));//Incremental distance in
direction of magnet
        r=bisectionMethod(currentpos, dr);
        xa = r*Math.cos(Math.atan(dy/dx));
        ya = r*Math.sin(Math.atan(dy/dx));

        return new MagnetPosition(xa,ya,currentpos.zPosition);

    }

    /**
     * Vector r in direction of magnet can be estimated using the bisection method
     * @param currentpos
     * @param dr
     * @return
     */
    private double bisectionMethod(MagnetPosition currentpos, Double dr){

        double a = 0.001;//lower boundary of interval
        double tolerance = 0.00001;
        double b=1;//upper boundary of interval
        double p,fa,fp;
        int n = 500;//max number of iterations
        int iter;//iteration number

        iter = 1;
        fa=equationFunc(currentpos, a, dr);

        while (iter<=n){
            p = a + (b-a)/2;
            fp = equationFunc(currentpos, p, dr);

            if ((fp == 0) || ((b-a) < tolerance))
                return p;
            iter++;

            if (fp*fa > 0){
                a = p;
                fa = fp;
            }
            else
                b=p;
        }
        return 0.01;
    }

    /**
     * Plug values from each iteration into the equation
     * @param cp
     * @param esr
     * @param dr
     * @return
```

```
     */
    private double equationFunc(MagnetPosition cp, double esr, double dr){
        double dz = cp.zPosition;
        //Coulomb equation
        return Math.pow(l/esr,5)+3*Math.pow(l/esr,3)+(3-
Math.pow(dr/dz,2))*(l/esr)+2*dr/dz;
    }
}
```

MagnetPosition.java

```
package com.example.jorda.magnetart.utilities;

/**
 * Created by jordan on 14/08/2017.
 */

//An object which consists of magnetic field readings in the x,y and z axes
public class MagnetPosition {

        public double xPosition;
        public double yPosition;
        public double zPosition;

        //Constructor to produce an object using 3 variables type double
        public MagnetPosition(double x, double y, double z) {

            xPosition = x;
            yPosition = y;
            zPosition = z;
        }

        //Constructor to producr an object using an array of 3 doubles
        public MagnetPosition(double[] readings)
        {
            if(readings.length == 3) {
                xPosition = readings[0];
                yPosition = readings[1];
                zPosition = readings[2];
            }
        }

        //Method to create an array using 3 double variables
        public double[] toDoubleArray()
        {
            return new double[]{xPosition, yPosition, zPosition};
        }

    //calculate magnitude of position, this is used when choosing colours
    public double magnitude(){
        return Math.sqrt(Math.pow(xPosition,2) + Math.pow(yPosition,2) +
Math.pow(zPosition,2));
    }
}
```

ColourSeekBar.java

```
package com.example.jorda.magnetart.view;

import android.content.Context;
import android.graphics.Canvas;
import android.graphics.Color;
import android.graphics.Paint;
import android.util.AttributeSet;
import android.view.View;

public class ColourSeekBar extends View {


    private int progress;// 0-100 value representing distance along the seekbar
    private Paint mPaint, mSliderpaint;
    int colouratposition;// Paint settings

    // CONSTRUCTOR
    public ColourSeekBar(Context context, AttributeSet attrs){
        super(context,attrs);
        setupSlider();
    }

    /**
     *Create the slider
     */
    private void setupSlider(){
        progress = 0;
        mPaint = new Paint();
        mSliderpaint = new Paint();
        mPaint.setStyle(Paint.Style.FILL);
    }

    // Draw Method
    @Override
    protected void onDraw(Canvas canvas){

        float start = getWidth()*.1f;//10% along the width
        float end = getWidth()*.9f;//90% along the width
        float top = getHeight()*.4f;//40% of the height
        float bottom = getHeight()*.80f;//80% of the height

        // Draw the color palette- 0 is at start of rectangle, 100 is at end
        for(int i = 0; i <= 100; i++){
            colouratposition = convertValueToColor(i);//convert each position to an
rgb colour value
            mPaint.setColor(colouratposition);//
            //Draw a rectangle with colour gradient by drawing a rectangle with
each iteration
            canvas.drawRect(start + ((float)i/100.0f)*(end-start),top,start +
((float)(i+1)/100.0f)*(end-start),bottom,mPaint);
        }
        drawCursor(canvas,start,end,top,bottom);
    }

    /**Draw the cursor for the currently selected colour (this is a rectangle with
a gap in the middle to allow the user to see the
     * selected colour- made up of four smaller rectangles
     */
    protected void drawCursor(Canvas canvas, float start, float end, float top,
float bottom) {

        mSliderpaint.setColor(Color.BLACK);
        float progressPercent = progress/100.0f;

        //4 rectangles used to make rectangle with hole in middle
        canvas.drawRect(start + (progressPercent)*(end-start)-15, top, start +
(progressPercent)*(end-start)-5, bottom, mSliderpaint);
        canvas.drawRect(start + (progressPercent)*(end-start)+5, top, start +
(progressPercent)*(end-start)+15, bottom, mSliderpaint);
        canvas.drawRect(start + (progressPercent)*(end-start)-15, top-10, start +
```

```
(progressPercent)*(end-start)+15, top,mSliderpaint);
        canvas.drawRect(start + (progressPercent)*(end-start)-15, bottom, start +
(progressPercent)*(end-start)+15, bottom+10, mSliderpaint);

    }

    /**
     * Setter for the progress variable
     * @param progress 0-100
     */
    public void setProgress(int progress){
        this.progress = progress;
    }

    /**
     * Convert value along bar (0-100) to a color int (0-255)
     * @param val 0-100
     * @return colour 0-255
     */
    public static int convertValueToColor(float val){

        float red,green,blue;//Each colour is in the range 0-255 in rgb scale

        float i = 16*(val/100);
        red = (float)Math.sin(0.3*i)*127+128;//colour gradient is made using 3 out
of phase sine waves
        green = (float)Math.sin(0.3*i+2)*127+128;
        blue = (float)Math.sin(0.3*i+4)*127+128;
        //returns a colour int made up of red, green and blue components
        return Color.rgb((int) red, (int) green, (int) blue);

    }

}
```

NewDrawingView.java

```
package com.example.jorda.magnetart.view;

import android.content.Context;
import android.graphics.Bitmap;
import android.graphics.Canvas;
import android.graphics.Color;
import android.graphics.Paint;
import android.graphics.Path;
import android.util.AttributeSet;
import android.view.MotionEvent;
import android.view.View;

/**
 * This class forms a custom view which will be used for drawing with the magnetic
stylus.
 * The input is taken from the screen co-ordinates calculated in the
onSensorChanged method of the NewDrawingFragment class
 */
public class NewDrawingView extends View {

    //These are the possible shapes that the user can draw
    public enum DrawingShape{
        CIRCLE, TRIANGLE, RECTANGLE, SPOT, SMOOTHLINE
    }

    public static final float TOUCH_TOLERANCE = 4;
    public static final float TOUCH_STROKE_WIDTH = 5;
```

```
    public DrawingShape activeShape;

    protected Path pathToDraw;
    public Paint initialPaint;//must be public to allow access to change colour
    public Paint endingPaint;
    protected Bitmap bitmapImage;
    protected Canvas drawingCanvas;

    /* Check for drawing */
    protected boolean drawingActive = false;

    protected float beginX;
    protected float beginY;

    protected float xPos;
    protected float yPos;

    public float screenX = 0;//screen X and Y co-ordinates - these are set in the
NewDrawingFragment
    public float screenY = 0;

    //Constructor
    public NewDrawingView(Context context) {
        super(context);
        initialise();//initialise
    }

    public NewDrawingView(Context context, AttributeSet attrs) {
        super(context, attrs);
        initialise();
    }

    public NewDrawingView(Context context, AttributeSet attrs, int defStyleAttr) {
        super(context, attrs, defStyleAttr);
        initialise();
    }

    protected void initialise() {

        pathToDraw = new Path();

        initialPaint = new Paint(Paint.DITHER_FLAG);
        initialPaint.setAntiAlias(true);
        initialPaint.setDither(true);
        initialPaint.setColor(Color.BLACK);//default colour
        initialPaint.setStyle(Paint.Style.STROKE);
        initialPaint.setStrokeJoin(Paint.Join.ROUND);
        initialPaint.setStrokeCap(Paint.Cap.ROUND);
        initialPaint.setStrokeWidth(TOUCH_STROKE_WIDTH);

        //this remains on screen when finger has been lifted
        endingPaint = new Paint(Paint.DITHER_FLAG);
        endingPaint.setAntiAlias(true);
        endingPaint.setDither(true);;
        endingPaint.setStyle(Paint.Style.STROKE);//change to fill shape with colour
        endingPaint.setStrokeJoin(Paint.Join.ROUND);
        endingPaint.setStrokeCap(Paint.Cap.ROUND);
        endingPaint.setStrokeWidth(TOUCH_STROKE_WIDTH);
    }

    @Override
    protected void onSizeChanged(int width, int height, int prevW, int prevH) {
        super.onSizeChanged(width, height, prevW, prevH);
        bitmapImage = Bitmap.createBitmap(width, height, Bitmap.Config.ARGB_8888);
        drawingCanvas = new Canvas(bitmapImage);
    }

    @Override
    protected void onDraw(Canvas canvas) {
```

```
        super.onDraw(canvas);
        canvas.drawBitmap(bitmapImage, 0, 0, initialPaint);

        //default is to draw a spot on screen
        if(activeShape == DrawingShape.SPOT) {
            drawingCanvas.drawCircle(screenX, screenY, 10, endingPaint);
        }

        if (drawingActive){
            switch (activeShape) {
                case RECTANGLE:
                    onDrawRectangle(canvas);
                    break;
                case CIRCLE:
                    onDrawCircle(canvas);
                    break;
                case TRIANGLE:
                    onDrawTriangle(canvas);
                    break;
            }
        }
    }



    public void reset() {
        pathToDraw = new Path();
        countTouch=0;
    }

    @Override
    public boolean onTouchEvent(MotionEvent event) {

        xPos = screenX;
        yPos = screenY;
        switch (activeShape) {
            case SMOOTHLINE:
                onTouchEventSmoothLine(event);
                break;
            case RECTANGLE:
                onTouchEventRectangle(event);
                break;
            case CIRCLE:
                onTouchEventCircle(event);
                break;
            case TRIANGLE:
                onTouchEventTriangle(event);
                break;
        }
        return true;
    }

    // Drawable line
    private void onTouchEventSmoothLine(MotionEvent event) {

        switch (event.getAction()) {
            case MotionEvent.ACTION_DOWN:
                drawingActive = true;
                beginX = xPos;
                beginY = yPos;

                pathToDraw.reset();
                pathToDraw.moveTo(xPos, yPos);

                invalidate();
                break;
            case MotionEvent.ACTION_MOVE:

                float distanceX = Math.abs(xPos - beginX);
                float distanceY = Math.abs(yPos - beginY);
```

```
                if (distanceX >= TOUCH_TOLERANCE || distanceY >= TOUCH_TOLERANCE) {
                    pathToDraw.quadTo(beginX, beginY, (xPos + beginX) / 2, (yPos +
beginY) / 2);
                    beginX = xPos;
                    beginY = yPos;
                }
                drawingCanvas.drawPath(pathToDraw, initialPaint);
                invalidate();
                break;
            case MotionEvent.ACTION_UP:
                drawingActive = false;
                pathToDraw.lineTo(beginX, beginY);
                drawingCanvas.drawPath(pathToDraw, endingPaint);
                pathToDraw.reset();
                invalidate();
                break;
        }

    }

    // Two stroke triangle

    int countTouch = 0;
    float basexTriangle = 0;
    float baseyTriangle = 0;

    private void onDrawTriangle(Canvas canvas){

        if (countTouch<3){
            canvas.drawLine(beginX,beginY,xPos,yPos,initialPaint);
        }else if (countTouch==3){
            canvas.drawLine(xPos,yPos,beginX,beginY,initialPaint);
            canvas.drawLine(xPos,yPos,basexTriangle,baseyTriangle,initialPaint);
        }
    }

    private void onTouchEventTriangle(MotionEvent event) {

        switch (event.getAction()) {
            case MotionEvent.ACTION_DOWN:
                countTouch++;
                if (countTouch==1){
                    drawingActive = true;
                    beginX = xPos;
                    beginY = yPos;
                } else if (countTouch==3){
                    drawingActive = true;
                }
                invalidate();
                break;
            case MotionEvent.ACTION_MOVE:
                invalidate();
                break;
            case MotionEvent.ACTION_UP:
                countTouch++;
                drawingActive = false;
                if (countTouch<3){
                    basexTriangle=xPos;
                    baseyTriangle=yPos;
                    drawingCanvas.drawLine(beginX,beginY,xPos,yPos,endingPaint);
                } else{
                    drawingCanvas.drawLine(xPos,yPos,beginX,beginY,endingPaint);

drawingCanvas.drawLine(xPos,yPos,basexTriangle,baseyTriangle,endingPaint);
                    countTouch =0;
                }
                invalidate();
                break;
        }
    }
```

```
    // Midpoint drawable circle

    private void onDrawCircle(Canvas canvas){
        canvas.drawCircle(beginX, beginY, calculateRadius(beginX, beginY, xPos,
yPos), initialPaint);
    }

    private void onTouchEventCircle(MotionEvent event) {
        switch (event.getAction()) {
            case MotionEvent.ACTION_DOWN:
                drawingActive = true;
                beginX = xPos;
                beginY = yPos;
                invalidate();
                break;
            case MotionEvent.ACTION_MOVE:
                invalidate();
                break;
            case MotionEvent.ACTION_UP:
                drawingActive = false;
                drawingCanvas.drawCircle(beginX, beginY,
calculateRadius(beginX,beginY,xPos,yPos), endingPaint);
                invalidate();
                break;
        }
    }


    /**
     * Method to calculate radius of circle
     * @param x1
     * @param y1
     * @param x2
     * @param y2
     * @return
     */
    protected float calculateRadius(float x1, float y1, float x2, float y2) {

        return (float) Math.sqrt(
                Math.pow(x1 - x2, 2) + Math.pow(y1 - y2, 2));
    }

    // Rectangle

    private void onDrawRectangle(Canvas canvas) {
        drawRectangle(canvas,initialPaint);
    }

    private void onTouchEventRectangle(MotionEvent event) {

        switch (event.getAction()) {
            case MotionEvent.ACTION_DOWN:
                drawingActive = true;
                beginX = xPos;
                beginY = yPos;
                invalidate();
                break;
            case MotionEvent.ACTION_MOVE:
                invalidate();
                break;
            case MotionEvent.ACTION_UP:
                drawingActive = false;
                drawRectangle(drawingCanvas,endingPaint);
                invalidate();
                break;
        }
    }

    private void drawRectangle(Canvas canvas,Paint paint){
```

```
        float right = beginX > xPos ? beginX : xPos;
        float left = beginX > xPos ? xPos : beginX;
        float bottom = beginY > yPos ? beginY : yPos;
        float top = beginY > yPos ? yPos : beginY;
        canvas.drawRect(left, top , right, bottom, paint);
    }

    public void clearDrawing(){
        //erase the canvas
        bitmapImage.eraseColor(Color.WHITE);
        drawingCanvas.drawBitmap(bitmapImage, 0, 0, endingPaint);
    }

    public Bitmap getBitmap(){
        setDrawingCacheEnabled(true);
        buildDrawingCache();
        Bitmap bitmap = Bitmap.createBitmap(getDrawingCache());
        setDrawingCacheEnabled(false);
        return bitmap;
    }

}
```

activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical" android:layout_width="match_parent"
    android:layout_height="match_parent">

    <FrameLayout
        android:id="@+id/container"
        android:layout_width="match_parent"
        android:layout_height="match_parent">
    </FrameLayout>

</LinearLayout>
```

activity_preferences.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical"
    >
<TextView
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:textSize="20dp"
    android:gravity="center_horizontal"
    android:text="Choose Calibration Method"
    android:layout_marginTop="10dp"
    android:layout_marginBottom="10dp"/>

    <RadioGroup
        android:id="@+id/radio_group_algo"
        android:layout_width="match_parent"
        android:layout_height="match_parent">
    </RadioGroup>
```

```
</LinearLayout>
```

Fragment_changebgcolour.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical" android:layout_width="match_parent"
    android:layout_height="match_parent">

    <TextView
        android:id="@+id/changebackgroundcolour"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        />

    <com.example.jorda.magnetart.view.ColourSeekBar
        android:id="@+id/backgroundcolourbar"
        android:layout_width="fill_parent"
        android:layout_height="60dp"
        />


</LinearLayout>
```

Fragment_changestrokecolour.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical" android:layout_width="match_parent"
    android:layout_height="match_parent">

    <TextView
        android:id="@+id/changestrokecolour"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        />

    <com.example.jorda.magnetart.view.ColourSeekBar
        android:id="@+id/strokecolourbar"
        android:layout_width="fill_parent"
        android:layout_height="60dp"
        />


</LinearLayout>
```

Fragment_help.xml

```
<?xml version="1.0" encoding="utf-8"?>
<ScrollView xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical" android:layout_width="match_parent"
```

```
    android:layout_height="match_parent"
    android:id="@+id/helpscrollview">

    <RelativeLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:orientation="horizontal">

        <TextView
            android:id="@+id/helpfirstpp"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_marginTop="20dp"
            android:layout_marginBottom="20dp"
            android:layout_marginLeft="5dp"
            android:layout_marginRight="5dp"
            android:text="@string/help_para_1"
            />


        <ImageView
            android:layout_width="fill_parent"
            android:layout_height="314dp"
            android:id="@+id/helpimage1"
            android:layout_below="@+id/helpfirstpp"
            android:layout_marginTop="30dp"
            android:src="@drawable/screen_nodrawings"/>

        <TextView
            android:id="@+id/helpsecondpp"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_marginTop="20dp"
            android:layout_marginBottom="20dp"
            android:layout_marginLeft="5dp"
            android:layout_marginRight="5dp"
            android:layout_below="@+id/helpimage1"
            android:text="@string/help_para_2"
            />

        <ImageView
            android:layout_width="fill_parent"
            android:layout_height="314dp"
            android:id="@+id/helpimage2"
            android:layout_below="@+id/helpsecondpp"
            android:layout_marginTop="30dp"
            android:src="@drawable/screen_loading"/>
        <TextView
            android:id="@+id/helpthirdpp"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_marginTop="20dp"
            android:layout_marginBottom="20dp"
            android:layout_marginLeft="5dp"
            android:layout_marginRight="5dp"
            android:layout_below="@+id/helpimage2"
            android:text="@string/help_para_3"/>

        <ImageView
            android:layout_width="fill_parent"
            android:layout_height="314dp"
            android:id="@+id/helpimage3"
            android:layout_below="@+id/helpthirdpp"
            android:layout_marginTop="30dp"
            android:src="@drawable/screen_calibrating"/>

        <TextView
            android:id="@+id/helpfourthpp"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
```

```
            android:layout_marginTop="20dp"
            android:layout_marginBottom="10dp"
            android:layout_marginLeft="5dp"
            android:layout_marginRight="5dp"
            android:layout_below="@+id/helpimage3"
            android:text="@string/help_para_4"/>

        <TextView
            android:id="@+id/helpfifthpp"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_marginTop="10dp"
            android:layout_marginBottom="20dp"
            android:layout_marginLeft="5dp"
            android:layout_marginRight="5dp"
            android:layout_below="@+id/helpfourthpp"
            android:text="@string/help_para_5"/>

        <ImageView
            android:layout_width="fill_parent"
            android:layout_height="314dp"
            android:id="@+id/helpimage4"
            android:layout_below="@+id/helpfifthpp"
            android:layout_marginTop="30dp"
            android:src="@drawable/screen_colourbar"/>

        <TextView
            android:id="@+id/helpsixthpp"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_marginTop="20dp"
            android:layout_marginBottom="20dp"
            android:layout_marginLeft="5dp"
            android:layout_marginRight="5dp"
            android:layout_below="@+id/helpimage4"
            android:text="@string/help_para_6"/>

        <ImageView
            android:layout_width="fill_parent"
            android:layout_height="314dp"
            android:id="@+id/helpimage5"
            android:layout_below="@+id/helpsixthpp"
            android:layout_marginTop="30dp"
            android:src="@drawable/screen_viewdrawing"/>


        <TextView
            android:id="@+id/helpseventhpp"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_marginTop="20dp"
            android:layout_marginBottom="20dp"
            android:layout_marginLeft="5dp"
            android:layout_marginRight="5dp"
            android:layout_below="@+id/helpimage5"
            android:text="Now you have read the tutorial, you are ready to produce
some artwork!  Press the button below to begin drawing and explore the app
features."/>

        <Button
            android:id="@+id/helpreturnbutton"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_below="@+id/helpimage5"
            android:layout_gravity="bottom"
            android:background="@drawable/custom_button"
            android:textColor="@android:color/white"
            android:layout_marginLeft="5dp"
            android:layout_marginRight="5dp"
            android:layout_marginTop="20dp"
```

```
            android:layout_marginBottom="8dp"
            android:text="Draw"/>

    </RelativeLayout>

</ScrollView>
```

Fragment_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:background="#f5f5f5"
    >

    <TextView
        android:id="@+id/nodrawingstextview"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginTop="20dp"
        android:gravity="center_horizontal"
        android:textSize="20dp"
        android:text="No drawings found\nPress the button below to draw!"/>

    <android.support.design.widget.FloatingActionButton
        android:id="@+id/adddrawingfab"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginRight="16dp"
        android:layout_marginBottom="16dp"
        android:src="@drawable/ic_add_white_24dp"
        android:layout_alignParentRight="true"
        android:layout_alignParentBottom="true"
        app:fabSize="normal"
        />

    <android.support.design.widget.FloatingActionButton
        android:id="@+id/deletedrawingfab"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginLeft="16dp"
        android:layout_marginBottom="16dp"
        android:src="@drawable/ic_delete_white_24dp"
        android:layout_alignParentLeft="true"
        android:layout_alignParentBottom="true"
        app:fabSize="normal"
        />

    <LinearLayout
        android:orientation="vertical"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content">

    <HorizontalScrollView
        android:id="@+id/saveddocumentsscrollview"
        android:layout_width="match_parent"
        android:layout_height="match_parent">

        <LinearLayout
            android:id="@+id/gallery"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
```

```
            android:orientation="horizontal"/>

    </HorizontalScrollView>

    <FrameLayout
        android:id="@+id/viewsaveddoc"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:paddingLeft="5dp"
        android:paddingRight="5dp"
        android:paddingTop="5dp"
        android:paddingBottom="5dp">

    </FrameLayout>
</LinearLayout>
</RelativeLayout>
```

Fragment_newdraw.xml

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/container"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    tools:ignore="MergeRootFrame" >

    <com.example.jorda.magnetart.view.NewDrawingView
        android:id="@+id/drawingview"
        android:layout_height="match_parent"
        android:layout_width="match_parent">
    </com.example.jorda.magnetart.view.NewDrawingView>

<RelativeLayout
    android:id="@+id/buttonsandreadings"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginTop="100dp"
        android:id="@+id/earthx"
        android:text="Earth_mag_x" />
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/earthy"
        android:layout_below="@+id/earthx"
        android:text="Earth_mag_y" />

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/earthz"
        android:layout_below="@+id/earthy"
        android:text="Earth_mag_z!" />


    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/sensorx"
        android:layout_below="@+id/earthz"
        android:text="Sensor_x" />
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/sensory"
        android:layout_below="@+id/sensorx"
```

```
        android:text="Sensor_y" />

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/sensorz"
        android:layout_below="@+id/sensory"
        android:text="Sensor_z" />

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/tlreadings"
        android:layout_below="@+id/sensorz"

        android:text="TL:" />

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/trreadings"
        android:layout_below="@+id/tlreadings"

        android:text="TR:" />

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/blreadings"
        android:layout_below="@+id/trreadings"
        android:text="BL:" />

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/brreadings"
        android:layout_below="@+id/blreadings"
        android:text="BR:" />

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/displayAlgorithm"
        android:layout_below="@+id/brreadings"/>

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/screenx"
        android:layout_below="@+id/displayAlgorithm"

        android:text="Screen Coord X" />

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/screeny"
        android:layout_below="@+id/screenx"

        android:text="Screen Coord Y" />


    <Button
        android:layout_width="30dp"
        android:layout_height="30dp"
        android:id="@+id/tlbutton"
        android:background="@color/colorGrey"
        android:layout_alignParentTop="true"
        android:layout_alignParentStart="true"
        />
```

```
    <Button
        android:layout_width="30dp"
        android:layout_height="30dp"
        android:id="@+id/trbutton"
        android:background="@color/colorGrey"
        android:layout_alignParentTop="true"
        android:layout_alignParentEnd="true" />

    <Button
        android:layout_width="30dp"
        android:layout_height="30dp"
        android:id="@+id/blbutton"
        android:background="@color/colorGrey"
        android:layout_alignParentBottom="true"
        android:layout_alignParentStart="true" />

    <Button
        android:layout_width="30dp"
        android:layout_height="30dp"
        android:id="@+id/brbutton"
        android:background="@color/colorGrey"
        android:layout_alignParentBottom="true"
        android:layout_alignParentEnd="true" />

</RelativeLayout>
</RelativeLayout>
```

Helpscreen_menu.xml

```
<?xml version="1.0" encoding="utf-8" ?>
<menu xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto">

    <item android:id="@+id/return_home"
        android:title="Home"
        android:orderInCategory="100"
        app:showAsAction="never"/>

</menu>
```

Main_menu.xml

```
<?xml version="1.0" encoding="utf-8" ?>
<menu xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto">

    <item android:id="@+id/help_screen"
        android:title="Help"
        android:orderInCategory="100"
        app:showAsAction="never"/>

    <item android:id="@+id/settings"
        android:title="Settings"
        android:orderInCategory="100"
        app:showAsAction="never"/>

</menu>
```

new_drawing.xml

```
<?xml version="1.0" encoding="utf-8" ?>
<menu xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto">

    <item android:id="@+id/sensor_readings"
        android:title="Hide sensor readings"
        android:orderInCategory="100"
        app:showAsAction="never"/>
```

```
    <item android:id="@+id/action_spot"
        android:title="Spot"
        android:icon="@drawable/spots_white"
        android:orderInCategory="100"
        app:showAsAction="always" />

    <item android:id="@+id/action_smoothline"
        android:title="@string/action_smoothline"
        android:icon="@drawable/line_white"
        android:orderInCategory="100"
        app:showAsAction="always" />

    <item android:id="@+id/action_rectangle"
        android:title="@string/action_rectangle"
        android:icon="@drawable/rectangle_white"
        android:orderInCategory="100"
        app:showAsAction="always" />

    <item android:id="@+id/action_circle"
        android:title="@string/action_circle"
        android:orderInCategory="100"
        android:icon="@drawable/circle_white"
        app:showAsAction="always" />

    <item android:id="@+id/action_triangle"
        android:title="@string/action_triangle"
        android:icon="@drawable/triangle_white"
        android:orderInCategory="100"
        app:showAsAction="always" />

    <item android:id="@+id/recalibrate"
        android:title="Recalibrate"
        android:orderInCategory="100"
        app:showAsAction="never" />

    <item android:id="@+id/strokecolour"
        android:title="Stroke Colour"
        android:icon="@drawable/ic_palette_white_24dp"
        android:orderInCategory="100"
        app:showAsAction="always" />

    <item android:id="@+id/bgcolour"
        android:title="Background Colour"
        android:orderInCategory="100"
        app:showAsAction="never"/>

    <item android:id="@+id/clear"
        android:title="Clear"
        android:orderInCategory="100"
        app:showAsAction="never" />

    <item android:id="@+id/savetodevice"
        android:title="Save to Device"
        android:orderInCategory="100"
        app:showAsAction="never"/>

    <item android:id="@+id/settingsfromdrawing"
        android:title="Settings"
        android:orderInCategory="100"
        app:showAsAction="never"/>

</menu>
```

Appendix E: Test Cases

Test case ID	Objective	Preconditions	Test data	Expected Result	Test condition(s)	Priority	Test completion date	Status	Tester	Defect ID	Defect Severity	Open Date	Close Date	Summary	Description / Comment
Drawing_ Tcase_1	To show that the user can open a drawing canvas on which to draw	Home screen is displayed and the new drawing floating action button is pressed	N/A	The new drawing screen is displayed, which contains a drawing canvas	Drawing_Tcon _1	High	31/07/2017	Passed	Jordan Shaw						
Drawing_ Tcase_2	To show that the user can change the stroke colour	New drawing screen is displayed, surface calibrated, stroke colour menu item has been pressed and stroke colour dialog is displayed	Slider used to select a stroke colour of blue and confirm button pressed	When drawing the stroke colour will now be the selected blue colour	Drawing_Tcon _10	Low	14/08/2017	Passed	Jordan Shaw						
Drawing_ Tcase_3	To show that the user can change the background colour of the canvas	New drawing screen is displayed, surface calibrated, background colour menu item has been pressed and background colour dialog is displayed	Slider used to select a stroke colour of red and confirm button pressed	When drawing the backgroun d colour will now be the selected red colour	Drawing_Tcon _11	Low	14/08/2017	Passed	Jordan Shaw						
Drawing_ Tcase_4	To show that the user can save a drawing	New drawing screen is displayed, save menu item has been pressed	Spots drawn on the canvas	The drawing will be saved to the device's storage and a toast will be displayed to confirm this	Drawing_Tcon _2	High	14/08/2017	Passed	Jordan Shaw						

Drawing_Tcase_5	To show that the user can draw a spot on the canvas at the screen co-ordinates calculated from the stylus' position on the drawing surface.	New drawing screen is displayed, surface calibrated, user is drawing with the magnetic stylus	The magnetic field data due to the movement of the stylus	A spot will be drawn at a position onscreen that matches the relative position of the stylus on the page	Drawing_Tcon_3	High	17/07/2017	Failed	Jordan Shaw	DEF_Drawing_1	Major	17/07/2017	21/08/2017	The spot is drawn on screen but the position onscreen does not match the expected position	The calibration algorithms were checked after this error was discovered
Drawing_Tcase_6	To show that the user can draw a smooth line between consecutive magnet positions	New drawing screen is displayed, surface has been calibrated and smoothline menu item has been pressed	The magnetic field data due to the movement of the stylus	A smooth line will be drawn on screen which matches the motion of the magnetic stylus on the drawing surface	Drawing_Tcon_4	Medium	28/07/2017	Passed	Jordan Shaw						The line is drawn as expected but the position of the line depends on the calibration algorithm. Therefore there is an overlap with DEF_Drawing_1
Drawing_Tcase_7	To show that the user can clear the drawing canvas.	New drawing screen is displayed, surface has been calibrated, user has begun to draw, clear menu item has been pressed and dialog is displayed	Confirm button pressed on dialog	Any drawing that was on the canvas is now erased, the canvas background colour is white	Drawing_Tcon_5	Medium	31/07/2017	Passed	Jordan Shaw						
Drawing_Tcase_8	To show that the user can choose to view or hide the sensor readings while drawing	New drawing screen is displayed, readings are initially hidden, toggle readings menu item has been pressed	N/A	If the sensor readings are visible, they will be hidden on button press. If the readings are invisible, they will	Drawing_Tcon_6	Low	31/07/2017	Passed	Jordan Shaw						

				become visible on button press.											
Drawing_ Tcase_9	To show that the user can draw a circle onscreen centred on the magnet position	New drawing screen is displayed, surface has been calibrated and circle menu item has been pressed	The magnetic field data due to the movement of the stylus, touch input onscreen	A circle will be drawn when finger is placed on screen with diameter determined by magnet movement	Drawing_Tcon_7	Low	28/07/2017	Passed	Jordan Shaw						The line is drawn as expected but the position of the line depends on the calibration algorithm. Therefore there is an overlap with DEF_Drawing_1
Drawing_ Tcase_10	To show that the user can draw a triangle onscreen with corners determined by the magnetic position	New drawing screen is displayed, surface has been calibrated and triangle menu item has been pressed	The magnetic field data due to the movement of the stylus, touch input onscreen	The base of a triangle will be drawn when finger is placed onscreen and magnet is moved. The apex will be drawn when finger is placed onscreen again.	Drawing_Tcon_8	Low	28/07/2017	Passed	Jordan Shaw						The line is drawn as expected but the position of the line depends on the calibration algorithm. Therefore there is an overlap with DEF_Drawing_1

Drawing_Tcase_11	To show that the user can draw a rectangle onscreen with corners determined by the magnet position	New drawing screen is displayed, surface has been calibrated and rectangle menu item has been pressed	The magnetic field data due to the movement of the stylus, touch input onscreen	A rectangle will be drawn with corners determined by magnet position after a finger has been placed onscreen	Drawing_Tcon_9	Low	28/07/2017	Passed	Jordan Shaw						The line is drawn as expected but the position of the line depends on the calibration algorithm. Therefore there is an overlap with DEF_Drawing_1
Help_Tcase_1	To show that the user can view a help screen with information on how to use the application	Home screen is displayed, the help menu item has been pressed	N/A	The help screen is displayed	Help_Tcon_1	Medium	14/08/2017	Passed	Jordan Shaw						
MD_Tcase_1	To show that the user can move the stylus to cause the application to detect changes in the magnetic field.	New drawing screen is displayed	Sensor magnetic field readings	As the user moves the stylus on the drawing area, the magnetic field readings change	MD_Tcon_1	High	17/07/2017	Passed	Jordan Shaw						
MD_Tcase_2	To show that the the raw readings must be corrected for the terrestrial magnetic field before I can draw.	Home screen is displayed and the new drawing floating action button is pressed	Sensor magnetic field readings	A loading dialog will be displayed while a buffer fills with values, the estimated terrestrial field readings will be displayed on screen when finished	MD_Tcon_2	High	17/07/2017	Passed	Jordan Shaw						

MD_Tcase_3	To show that the user can place the magnetic stylus in each corner of the drawing surface and store the magnetic field readings taken in order to calibrate the application.	New drawing screen is displayed, the loading dialog has been dismissed	Sensor magnetic field readings	Values are displayed onscreen	MD_Tcon_3	High	17/07/2017	Passed	Jordan Shaw						
MD_Tcase_4	To show that the user can view magnetic field readings on the device display.	New drawing screen is displayed, readings are initially hidden, toggle readings menu item has been pressed	Sensor magnetic field readings	Values are displayed onscreen	MD_Tcon_5	Medium	17/07/2017	Passed	Jordan Shaw						Possible duplicate of MD_Tcase_3
MP_Tcase_1	To show that the user can recalibrate the app while drawing	New drawing screen is displayed, recalibrate menu item has been pressed	Sensor magnetic field readings	The buttons and sensor readings are now visible and the buttons are enabled so the user can now recalibrate the drawing area	MP_Tcon_1	Medium	31/07/2017	Passed	Jordan Shaw						
Nav_Tcase_1	To show that the user can navigate between different functions of the application using menus	Home screen is displayed	N/A	The user can use a menu on the home screen, new drawing screen and help screen	Nav_Tcon_1	High	31/07/2017	Passed	Jordan Shaw						

Set_Tcase_1	To show that the user can access settings from the new drawing screen in which they can choose a calibration algorithm.	New drawing screen is displayed, save menu item has been pressed	N/A	Settings screen is displayed	Set_Tcon_1	Low	31/07/2017	Passed	Jordan Shaw						
Set_Tcase_2	To show that the user can access settings from the home screen in which they can choose a calibration algorithm.	Home screen is displayed, settings menu item has been pressed	N/A	Settings screen is displayed	Set_Tcon_1	Low	31/07/2017	Passed	Jordan Shaw						
UI_Tcase_1	To show that the user will be presented with dialogs when appropriate when navigating the application.	Main screen displayed, an image selected and delete button pressed	N/A	A dialog will be displayed to ask the user if they wish to delete the selected drawing	UI_Tcon_2	Medium	31/07/2017	Passed	Jordan Shaw						
UI_Tcase_2	To show that the user will be presented with dialogs when appropriate when navigating the application.	New drawing screen displayed, drawing surface has been calibrated and device back button pressed	N/A	A dialog will be displayed to ask the user if they wish to save the drawing before exiting	UI_Tcon_2	Medium	31/07/2017	Passed	Jordan Shaw						
UI_Tcase_3	To show that the user will be presented with dialogs when appropriate when navigating the application.	New drawing screen displayed, drawing surface has been calibrated and clear menu item pressed	N/A	A dialog will be displayed to ask the user if they wish to clear the drawing	UI_Tcon_2	Medium	31/07/2017	Passed	Jordan Shaw						

UI_Tcase_4	To show that the user will be presented with dialogs when appropriate when navigating the application.	New drawing screen displayed, drawing surface has been calibrated and stroke colour menu item pressed	N/A	Stroke colour dialog is displayed with a seekbar to change colour	UI_Tcon_2	Medium	31/07/2017	Passed	Jordan Shaw						
UI_Tcase_5	To show that the user will be presented with dialogs when appropriate when navigating the application.	New drawing screen displayed, drawing surface has been calibrated and background colour menu item pressed	N/A	Background colour dialog is displayed with a seekbar to change colour	UI_Tcon_2	Medium	31/07/2017	Passed	Jordan Shaw						
UI_Tcase_6	To show that the UI contains no spelling or grammatical errors	Home screen is displayed	Textual content as the user navigates through the application	Application contains no spelling or grammatical errors	UI_Tcon_3	Medium	14/08/2017	Failed	Jordan Shaw	DEF_UI_1	Minor	14/08/2017	14/08/2017	Spelling error in help screen	Error has been fixed
VD_Tcase_1	To show that the user can view the saved drawings in a gallery.	Home screen is displayed, at least one image has been saved	N/A	The saved drawing will now be scaled and be visible as part of a gallery on home screen	VD_Tcon_1	High	01/08/2017	Passed	Jordan Shaw						
VD_Tcase_2	To show that the user can delete a saved image from storage	Home screen is displayed, at least one image has been saved, user presses on drawing and presses delete icon	N/A	The image is deleted from device storage, a toast is displayed to confirm deletion	VD_Tcon_2	High	01/08/2017	Passed	Jordan Shaw						

VD_Tcase_3	To show that the user can select a saved image to view an enlarged version of the image	Home screen is displayed, at least one image has been saved, user presses on drawing	N/A	A larger version of the image is displayed below the gallery	VD_Tcon_3	Low	01/08/2017	Passed	Jordan Shaw					

Appendix F: Meeting Minutes

Student Name: Jordan Shaw

Student Number: 40106564

Date of Meeting: 19.6.17

Time of Meeting: 3:30pm

Other Attendees at Meeting: Dr Darryl Stewart

Key Points Arising from Meeting:

- Confirmed that the project will take the form of an android app
- App will involve drawing/ note taking using magnet
- Considered devices that app can be tested on- phone, tablet, PC with sensors

Next Action as a result of Meeting:

- Test that the magnet is detected by the phone
- Decide on a development lifecycle- break up the project into sprints and a backlog?
- Produce a Gantt chart
- Consider where magnet will be positioned on the pen
- Where will notes be saved- online or on device?

Student Name: Jordan Shaw

Student Number: 40106564

Date of Meeting: 30.6.17

Time of Meeting: 4pm

Other Attendees at Meeting: Dr Darryl Stewart

Key Points Arising from Meeting:

- Discussed feasibility of project, based on limitations mentioned in project plan
- Decided that plan is feasible, and will serve as a concept demonstrator, not a finished market-ready product
- Discussed methods of calibrating the magnetometer for the app and saving coordinates

Next Action as a result of Meeting:

- Develop stylus (magnets ordered online)
- Calibrate the writing surface by taking magnetic field readings in all 4 corners
- Use these readings to produce a co-ordinate system
- This co-ordinate system will then be used to calculate the relative position of the magnetic stylus on the surface using it’s magnetic field reading

Student Name: Jordan Shaw

Student Number: 40106564

Date of Meeting: 26.7.17

Time of Meeting: 2:30pm

Other Attendees at Meeting: Dr Darryl Stewart

Key Points Arising from Meeting:

- When using the algorithm from sample code the magnetic field readings appear to be stable- returning to the same values in each corner
- The lines drawn on screen do not match the magnet position
- Values can be mocked to test if the magnet strength at each position matches that expected

Next Action as a result of Meeting:

- Ensure earth's magnetic field is accounted for
- Display readings to screen when magnet is moved so they can be tracked
- Check that these readings match with readings in corners (from calibration)
- Check algorithm for producing a coordinate system from readings
- Draw a line on screen when magnet is moved

Student Name: Jordan Shaw

Student Number: 40106564

Date of Meeting: 2.8.17

Time of Meeting: 11am

Other Attendees at Meeting: Dr Darryl Stewart

Key Points Arising from Meeting:

- Magnetic field reading were following the expected trend and changed when magnet moved
- Need to consider whether to filter/ round results
- Canvas can be tested by plotting points and checking they match expectation
- Only draw on screen when input is toggled

Next Action as a result of Meeting:

- Consider use of onSensorChanged method- how often are readings taken and how often do we want them to be taken
- Continue to work on filtering magnetic field readings
- Try to draw a line on screen based on plotting co-ordinates and see if it matches the expected line based off magnetic field readings
- Try to draw a dot/ line at magnet position

Student Name: Jordan Shaw

Student Number: 40106564

Date of Meeting: 9.8.17

Time of Meeting: 11am

Other Attendees at Meeting: Dr Darryl Stewart

Key Points Arising from Meeting:

- The fluctuations in magnetic field readings suggest that they may not follow a linear trend when the magnet is moved between corners
- The co-ordinates can be checked against the screen dimensions

- Calibration method may need to be changed

Next Action as a result of Meeting:

- Use logger to track readings and co-ordinates as magnet is moved, plot results
- Try other calibration algorithms and record findings in dissertation
- Sanity test the co-ordinates- check they are at the expected positions

Student Name: Jordan Shaw

Student Number: 40106564

Date of Meeting: 31.8.17

Time of Meeting: 1pm

Other Attendees at Meeting: Dr Darryl Stewart

Key Points Arising from Meeting:

- **Document the devices the app was tested on**
- **Magnetic field assumption- linear or non linear?**
- **Do not need to include all algorithms tested**
- **Function definitions should include all user inputs and movement of the stylus**

Next Action as a result of Meeting:

- **Plot readings and check calculations**
- **Fine tune calibration algorithms**
- **Create draft dissertation**

Student Name: Jordan Shaw

Student Number: 40106564

Date of Meeting: 6.9.17

Time of Meeting: 12pm

Other Attendees at Meeting: Dr Darryl Stewart

Key Points Arising from Meeting:

- Dissertation feedback up to implementation section
- Why methodology chosen over other options?
- Calculations can be unit tested
- Other methods can be black box tested
- Need to check phone specification

Next Action as a result of Meeting:

- Map user stories to tasks
- Change requirements to user stories
- Mention what principles were followed in design
- Mention what coding standards were followed
- Calibration algorithms are an important part of dissertation