DEVELOPMENT OF A TOUCHSCREEN BASED SLATE DEVICE WITH JAVA APPLICATION

A Thesis Submitted to the Department of Electrical and Electronics and Engineering

Of

BRAC University By

Tamanna Hamid	**11121029**
Sakib Muhammad Muntasir Adnan	**11121084**
Shamama Nazneen	**11121046**
Zannatun Nayeem Chowdhury	**11121031**

Thesis Supervisor: Dr. Mohammed Belal Hossain Bhuian

Thesis Co-Supervisor: Muhammad Abdur Rahman Adnan

In Partial Fulfillment of the Requirements for the Degree Of Bachelor of Science in Electrical and Electronics Engineering

April 2015

DECLARATION

We hereby declare that this thesis is based on the results found by ourselves. Materials of work found by other researcher are mentioned by reference. This thesis, neither in whole nor in part, has been previously submitted for any degree.

Signature of

Supervisor

Signature of

Tamanna Hamid

Signature of

Sakib Muhammad Muntasir Adnan

Signature of

Shamama Nazneen

Signature of

Zannatun Nayeem Chowdhury

ACKNOWLEDGMENT

First and foremost, we would like to express our sincere gratitude to our thesis supervisor Dr. Mohammed Belal Hossain Bhuian for the continuous support of our thesis project, for his patience, motivation, enthusiasm, and immense knowledge. His guidance helped us in all the time of research and writing of this thesis. We could not have imagined having a better advisor and mentor for our thesis project.

We would also like to thank our co-supervisor Muhammad Abdur Rahman Adnan, for his great help during our work. His insightful comments and encouragement have helped us in understanding the project and the role of Raspberry Pi in our thesis thoroughly.

Our sincere thanks also go to Md. Shamsul Kaonain for his great suggestions regarding our thesis project and for giving us good advice for the improvement of our thesis.

ABSTRACT

Touch panel technologies are a key theme in current digital devices, including smartphones, slate devices like the iPad and Android tablets, the screens on the backs of digital cameras, and Windows 8 devices. The term touch panel encompasses various technologies for sensing the touch of a finger or stylus. This project aims to show the work involved to create a slate/tablet with a basic touch panel sensing method which takes an input from a user, via a stylus. The data (from the etchings of the stylus) is then to be fed to other smart devices using a wireless adapter, primarily to other smart devices instantaneously. The main objective of this thesis work is to aid teacher and student interaction in the classroom. The device will provide students the drawings and lectures written on said slate instantly for a copy, and to modify said copy in any way they choose to, or receive a copy at a later time via the internet. It can also be used as a commercial communication method in long distance business conferencing.

Table of Contents

DECLARATION **1**

ACKNOWLEDGMENT **3**

ABSTRACT **4**

LIST OF FIGURES **7**

CHAPTER 1: INTRODUCTION **8**

1.1: Purpose of the project 9

1.2: Related works 10

CHAPTER 2: LITERATURE REVIEW **13**

2.1: Design Overview 14
- 2.1.1: Touchscreen panel 14
- 2.1.2: Display 20
- 2.1.3:Central Processing Unit (CPU) 24
- 2.1.4:Wireless adapter 28
- 2.1.5:Miscellaneous materials 30

2.2: Operating system 31

2.3 Java 33
- 2.3.1: The common features of Java 34

CHAPTER 3: HARDWARE DESIGN AND SYSTEMSETUP **40**

3.1: Hardware Integration 41

3.2: Raspberry Pi Setup 42

3.3: Project block diagram 45

4.1: Kernel Recompilation 47

4.2: Calibration 52

CHAPTER 5: PRELIMINARY OUTLINE OF JAVA APPLICATION 55

5.1: Server (Teacher) End Program – Java Paint 56
5.1.1: MainFrame class 57
5.1.2: Server class 58
5.1.3: The design of the Drawing Canvas 60
5.1.4: Client (Student) End 64

CHAPTER 6: BROADCASTING TECHNOLOGY 68

6.1: EDIMAX EW-7811UN Wireless Adapter Installation 69

6.2: Configuration 71

CHAPTER 7: CONCLUSION 73

REFERENCES 76

APPENDIX A 80

APPENDIX B 82

APPENDIX C 83

APPENDIX D 104

List of Figures

Fig. 1.1: Magnetic resonant circuit in a Wacom pen tablet screen 11
Fig. 2.1: Touch detection method in 4-wire resistive technology 15
Fig. 2.2: How surface capacitive technology works 16
Fig. 2.3: How projected capacitive technology works 17
Fig. 2.4: Circulation of ultrasound waves in a SAW touchscreen panel 18
Fig. 2.5: How SAW touchscreen works 18
Fig. 2.6: Detailed cross-section of an LCD panel 21
Fig. 2.7: Cross-section of a plasma screen display 22
Fig. 2.8: Layers in an OLED display 23
Fig. 2.9: Raspberry Pi model B+ layout 25
Fig. 2.10: Arduino Uno layout 26
Fig. 2.11: Package (Tree) directory structure 34
Fig. 3.1: NOOBS OS install manager for RPi 44
Fig. 3.2: Raspbian desktop after successful installation 44
Fig. 4.2: Replacing new kernel image in SD card 51
Fig. 4.3: Running xinput_calibrator from LXTerminal 53
Fig. 4.4: Calibration test window 54
Fig. 5.1: Java Paint Structure 56
Fig. 5.2: Dialog box for Server Connection Status 58
Fig. 5.3: Main method constituents and the beginning of the program 60
Fig. 5.4: General User Interface of the Drawing Canvas 62
Fig. 5.5: DrawPanel and the Canvas 63
Fig. 5.6: Client Status Dialog Box 65
Fig. 5.7: Final output from Server 67
Fig. 6.1: Checking the wireless network configuration of the Raspberry Pi system 70
Fig. 6.2: Verifying the connection and IP address of the system 72

CHAPTER 1: Introduction

Our world is coming closer everyday thanks to the wonderful advancement in communication. Numerous technologies are being developed and implemented for the purpose of information and communication (collectively called ICT), due to which we are being able to endow ourselves with proper and latest data and news instantaneously. This is helping us to better utilize our resources and develop optimum ways for a better life. This thesis work brings together some well-known technologies to achieve this end.

Regardless its uses, technology is leaving an imprint on all facets of our lives. Smart new gadgets, finer machineries and Internet are the keys of development today. The advent of touchscreen technology led to the development of numerous applications that we now take for granted. Smart touch devices have given us access to information literally on our fingertips. However, this information gain and share would have been impossible without the progress of computers and computer languages. In this work we aim to develop a touchscreen device that runs an application that can share information from a source in real time.

1.1: Purpose of the project

Our primary objective is to augment source-to-recipient interaction in a congregated environment. For this, we are developing a full-fledged device that will, for a start, replace or at least substitute the black/whiteboard concept in our education sector. Today, a typical classroom in Bangladesh uses a board or projector medium to convey lessons to students. Though simple and effective, this mode of communication has certain drawbacks. An informal survey showed that during busy or advanced lessons students often tend to miss or mislead any idea or part of an idea expressed by the teacher. Our work will enable a teacher (source) to write anything on a hand held device which will be streamed live into a student's (recipient's) end device. This will lead to ease of information accessibility at the student end and fewer tendencies to make mistakes, thus learning process will become better.

Secondly, this device can be an integral tool for distance learning purposes. Many foreign universities have distance learning programs that already use techniques like file sharing and

discussion boards to convey lessons to students all over the world. This device can be used as a digital whiteboard along with these techniques, allowing students to log in to the lesson and receive lesson materials in real time. Thus using this concept, overall classroom digitalization is possible. From a more general perspective, our project can be applied in the business sector as well. Already video conferencing is a norm in various industries and with the help of a live streaming whiteboard element, communication in e-commerce will take a further step forward.

This device is aimed to be a convenient and user friendly interface. For this reason its construction and design is done using familiar technologies. We will discuss some of the possible techniques considered for this device later in the next chapter. Needless to say, well known technologies are more accessible and reasonably priced than the rest. This will make our device a useful and affordable tool that everyone can implement with ease.

1.2: Related works

Although the concept of this project was born from purposes stated above, similar works have been discovered that employ familiar techniques or serve a correlated purpose. They were a wonderful source of inspiration for our efforts and helped to make our device better. A couple of these works are discussed below.

Java-Powered Braille Slate Talker [1]

This paper outlines the methodology of a similar device which is intended to help blind users to communicate. In this hand held device, a fixed layout of plastic guide is placed over a touch screen to take input. This input is then converted to text using a table driven state machine. The current system in this device is designed in Java. The device is portable, inexpensive and designed with a simple interface. The touch screen that takes the input acts as a virtual Braille slate with a fixed screen layout.

The Braille dots can be pressed with fingers and only one dot can be pressed at a time. The first touch of any of the Braille dots takes it as an input, giving an audio signal at the same time. A second touch on the same dot will erase the pressed dot, which is signaled by a different sound. After the end of a word formation the user may allow the device to turn it to text. In this way the user can make an accurate input. The use of Java makes it possible for the program to be used in other platforms like desktop computer or even in mobile phones. This device can help the user in many different applications like read and write documents, sending emails, and even using a simple web browser when connected to the Internet.

Products by Wacom

Wacom is a Japanese company that specializes on stylus based devices used in design purposes mainly. They develop a variety of sketch pads for different levels graphic art and each series of their products have their own specialized technology through which this is achieved. Their pen tablet collection, for instance, uses electromagnetic technology to create the etchings on the tablet screen. This relieves the tablet from the use of any cord or built-in battery power supply. It is also a non-contact based system, where the surface has a sensor board which helps to detect the movement of the pen. Weak energy is induced in the pen's resonant circuit by a magnetic field, which is generated by the sensor board. The circuit uses the magnetic power to send the signal back to the sensor board. This is seen in Figure 1.1 below.

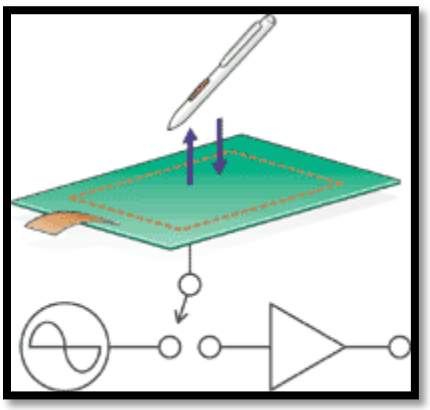


Fig. 1.1: Magnetic resonant circuit in a Wacom pen tablet screen

By repeating the movement of the pen, the sensor board gathers information on the angles and positions created by the pen and also how much speed and pressure was used on each stroke by the user. This helps the board to recreate the etching on itself. There is also a sensor attached to the magnetic field of the board turns the board on and off according to the movement of the pen. [2]

CHAPTER 2: Literature Review

There are plenty of well-developed technologies today that can be used to create an apparatus that serves our purpose. Before selecting the types of hardware and software we will use, we need to consider many factors like accessibility, compatibility and affordability of the equipment, our knowledge of the software related to them, etc. As we aim to employ our device for academic purpose primarily, cost and availability should play a huge role in this development. Each hardware and software existing has its own distinct features, pros and cons of use. We look into these technologies and setup methods that this device can be designed with below.

2.1: Design Overview

There are four main parts of hardware in this device, each having its own distinct purpose. These equipment in turn have their individual types, the working principles of which are explained below.

2.1.1: Touchscreen panel

The touchscreen panel is the main interface that the user will use in this device. Modern day gadgets and appliances make extensive use of touchscreens; hence it is taken to be a user friendly means interaction. Since our primary objective is to use this device in a classroom environment it is convenient if the device improves on the writing on a whiteboard tradition of a typical classroom. Keeping this in view the following touchscreen technologies were explored.

Resistive touchscreen

Resistive touchscreen technology implements the effect of two layers of glass, coated with a conductive material such as Indium tin oxide (ITO), coming in contact with each other. The position of a touch on this panel is determined by 4-wire technology, which involves four

conductive bus bars being attached to each glass layer pair wise. These pairs apply a uniform voltage across each axis periodically. When the top layer touches the bottom one, a voltage divider is created across one axis. The voltage at that point is measured, which is then passed through an analog-to-digital converter to get X-coordinate. Similarly, voltage is applied across the other axis and read back to get the Y-coordinate. [3] Figure 2.1 illustrates this clearly. Other than 4-wire, advanced detection technologies like 5-wire and 8-wire also exist, but for our purpose this is sufficiently accurate. Resistive touchscreens are pressure sensitive; hence using small objects like a stylus on them gives a more handwriting-like, effective touch result. They are cheap, consume less power and are more 0accurate than their capacitive counterparts. [4]However, they are less durable than capacitive touchscreens and reflect ambient light.

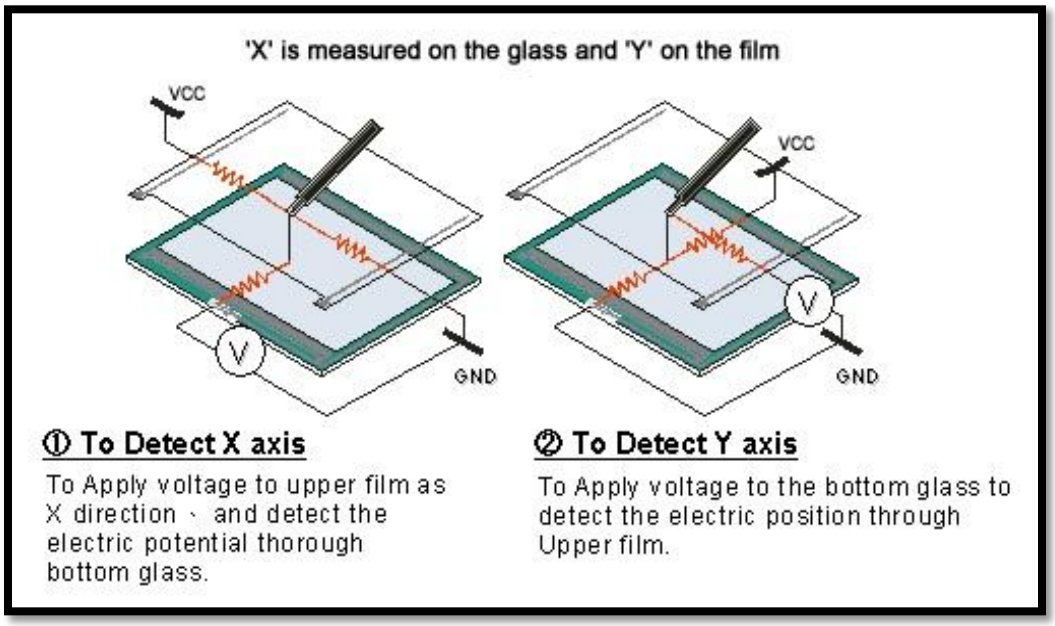


Fig. 2.1: Touch detection method in 4-wire resistive technology

Capacitive touchscreen

This is the most popular touchscreen technology used in smart devices today. Unlike resistive touchscreen, a capacitive touchscreen comprises of a single glass panel coated with conductive ITO. A small voltage is applied across this layer to create a uniform electrostatic field. When touched by another conductor, such as our finger, charges accumulate on that point on the screen, creating a capacitor. Controller circuits situated at the corners of the panel detect a change in capacitance in that position when this happens, and determines the point of touch. This method of detection is called the surface capacitive technology.

Another method called projected capacitive technology (PCT) also exists, which uses a fine grid of conductive lines or traces to detect a touch. When we touch a PCT screen with our finger, the capacitance of the nearest traces changes and this enables the controller to determine the position of contact. [5] Capacitive touchscreens are more durable and slimmer than their resistive counterparts. However, they are more expensive than resistive touchscreens and do not respond to any input from a nonconductive source, like a stylus.

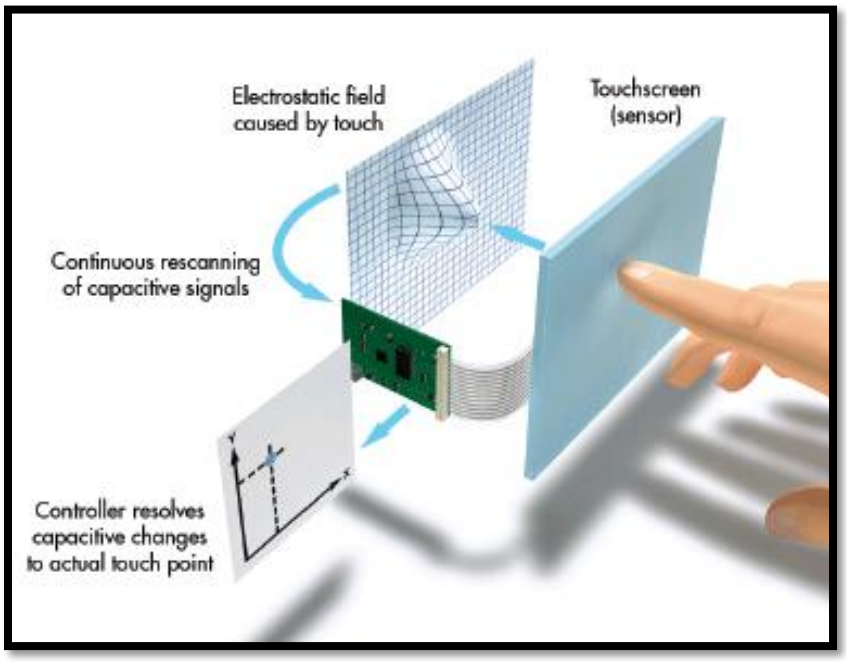


Fig. 2.2: How surface capacitive technology works

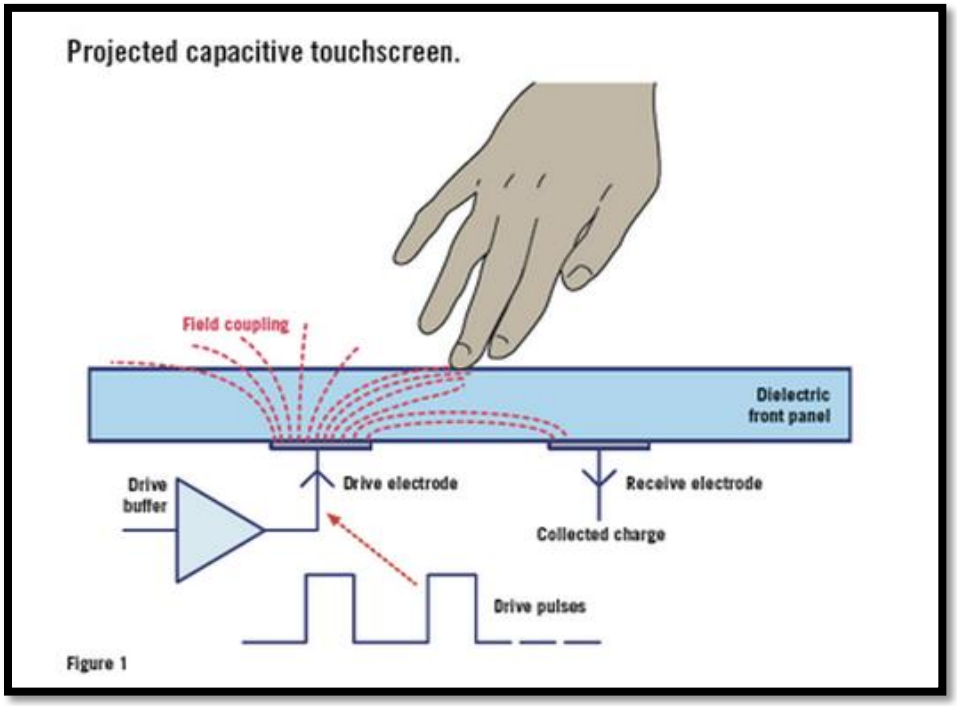


Fig. 2.3: How projected capacitive technology works

Surface acoustic wave touchscreen

Surface Acoustic Wave (SAW) is the latest technology among all the touchscreen technologies that we know of. An SAW touchscreen panel consists of a single glass body that uses ultrasonic waves to detect a touch. As Figure 2.4 illustrates, a pair of transducers (for emission and reception) is placed on either axis of the glass body. The controller sends electrical signals to the transmitting transducers which converts them to ultrasound waves. The panel is also lined with reflectors that spread the waves across it, as indicated by the dotted arrows in the diagram. The receiving transducer pair catches the waves, converts them to electrical signals again and sends them back to the controller. When the panel is touched the waves at that point get absorbed, creating an interruption that enables the controller to detect the position of touch. This is shown in Figure 2.5. As compared to the other touchscreen technologies, SAW is the clearest and most durable touchscreen available now, with the best resolution. However it is very expensive and tends to get damaged easily, owing to its sensitivity to dirt and scratch. [5]

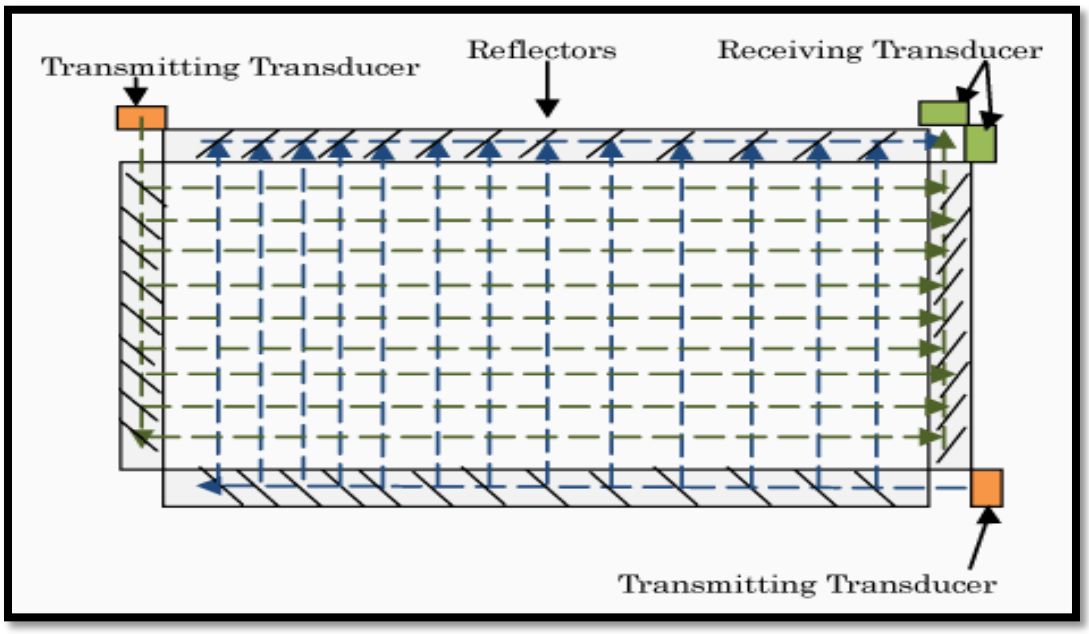


Fig. 2.4: Circulation of ultrasound waves in a SAW touchscreen panel

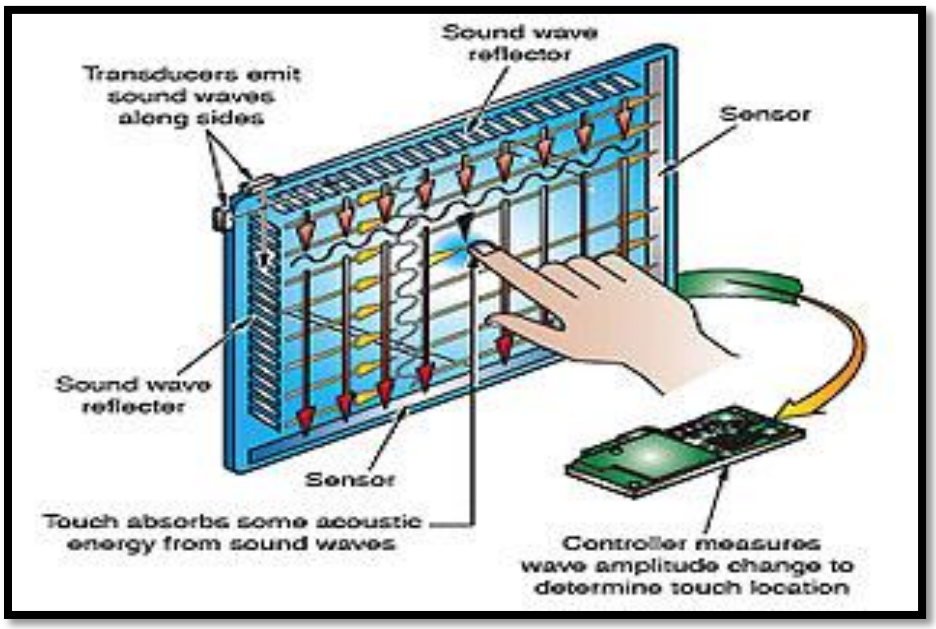


Fig. 2.5: How SAW touchscreen works

Considering everything about these technologies and especially since it complements our objective to keep the ‘pen-based writing tradition’ in the device, a 7 inch resistive touchscreen panel is used for this project. It has reasonable price and was well available hence it served our purpose of a touchscreen solution for our slate device fittingly.

2.1.2: Display

Monitors or device displays are termed as the output device of a computer system. Although modern gadgets jointly refer to them as ‘touchscreen display’, displays are in fact meticulously adhered to the touchscreen panels to appear like one unit. Even so, monitors are of several types, a few of which are explored for the purpose of use of this project.

LCD monitor display

Liquid crystal displays are the most common display solutions of the modern day. An LCD display contains a liquid crystal solution filled between two polarizing glass sheets. A current is passed through this solution, which aligns the crystals such that they serve as effective 'shutters' that close or open to block light in a varying degree. This current flow due to the voltage applied using transparent, grid-like structured electrodes placed on the sheets. The density of these grids determines the picture elements or pixels of the display. [6] Figure 2.2 illustrates the material layers of a typical LCD display in detail. A layer of silicon dioxide acts as the polarizing material while the ITO layers are the transparent electrodes. LCD displays are very slim and light; they consume less power than LEDs and cause minimum radioactive emission. [7]

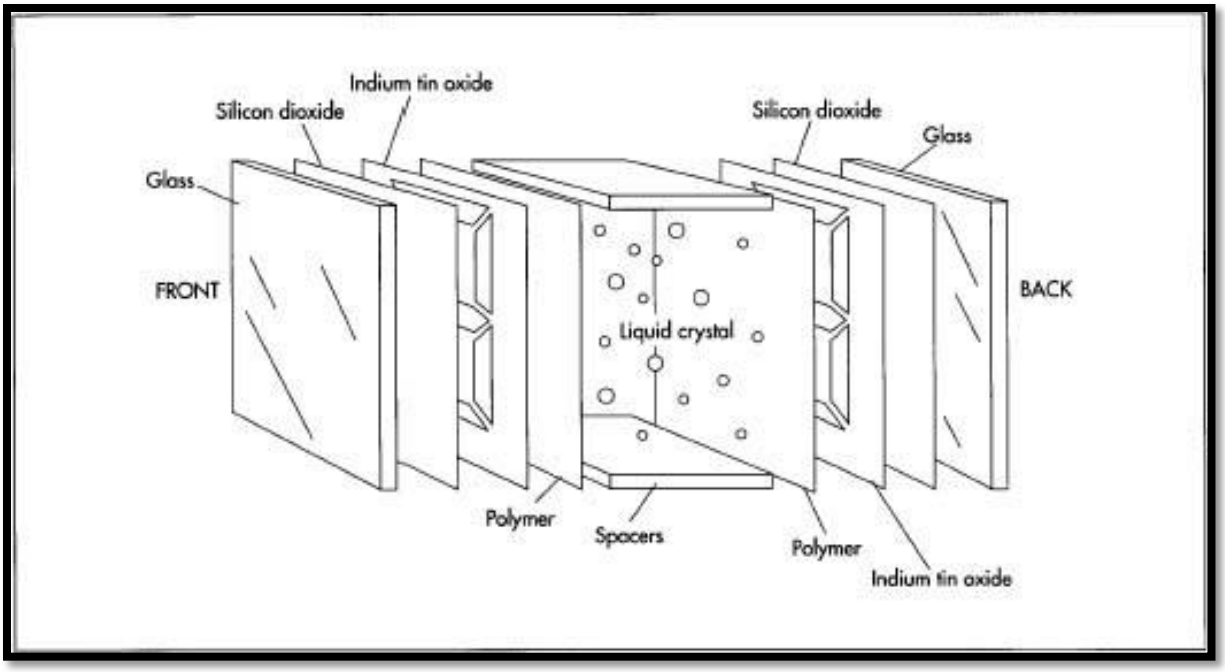


Fig. 2.6: Detailed cross-section of an LCD panel

Plasma display

Plasma technology evolved alongside LCD technology after the long dominance of Cathode Ray Tube (CRT) display solutions. Its operation principle is very much similar to that of a CRT display. As seen in Figure 2.3, a plasma monitor contains minuscule gas-filled (mostly noble gas) cells called sub-pixels, coated with blue, red and green phosphor. Electrodes on either side pass a voltage through the gas, exciting and changing it to a plasma form. The energy levels of the ionized gas lower as atoms collide with each other, releasing ultraviolet (UV) light photons. These photons appear as colored lights due to the different colored phosphor surrounding, giving us the various colors on the screen. Although plasma screens cost less and its display colors are more vibrant than that of LCD, they get easily hot and need to be cooled. [8] For the purpose of this work, we do not require the level of color vibrancy it provides. Most importantly, plasma screens are mostly manufactured in large size (more than 42 inches) which is unnecessary for this project. [9]

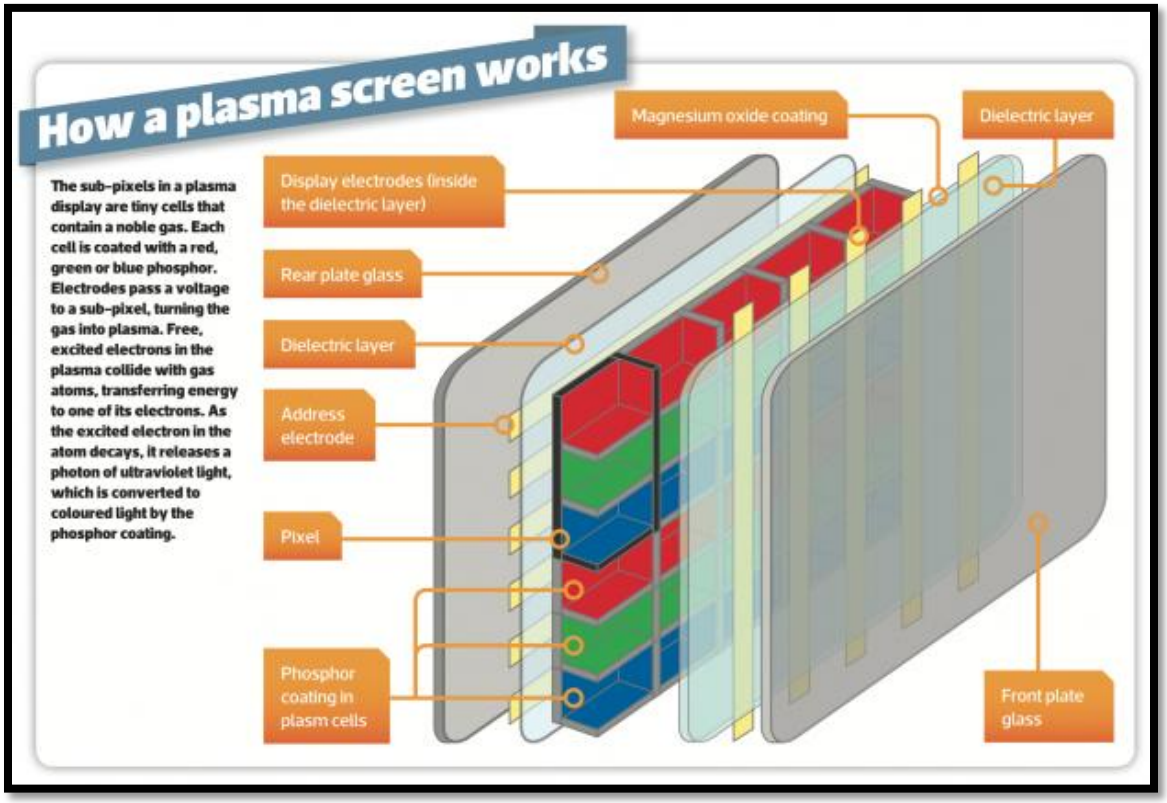


Fig. 2.7: Cross-section of a plasma screen display

OLED Display

Organic Light Emitting Diode (OLED) is the latest technology in display solutions nowadays. It can be perceived as a slimmer plasma technology, but with better color contrast and higher energy efficiency. [10] Unlike plasma displays, an OLED display consists of organic polymer layers between its electrodes, as shown in Figure 2.4. These polymers behave like the p-n junctions of a diode when voltage is passed across the pair of electrodes that encase them. Electron-hole recombination occurs between them, emitting photons of light which we utilize in the display. The different molecules of the polymers determine the color of the light emitted. [11] Although OLED displays have better video quality than their LCD counterparts, they are

very expensive. Because of this they are not readily available, and are unnecessary for the purpose of this work.

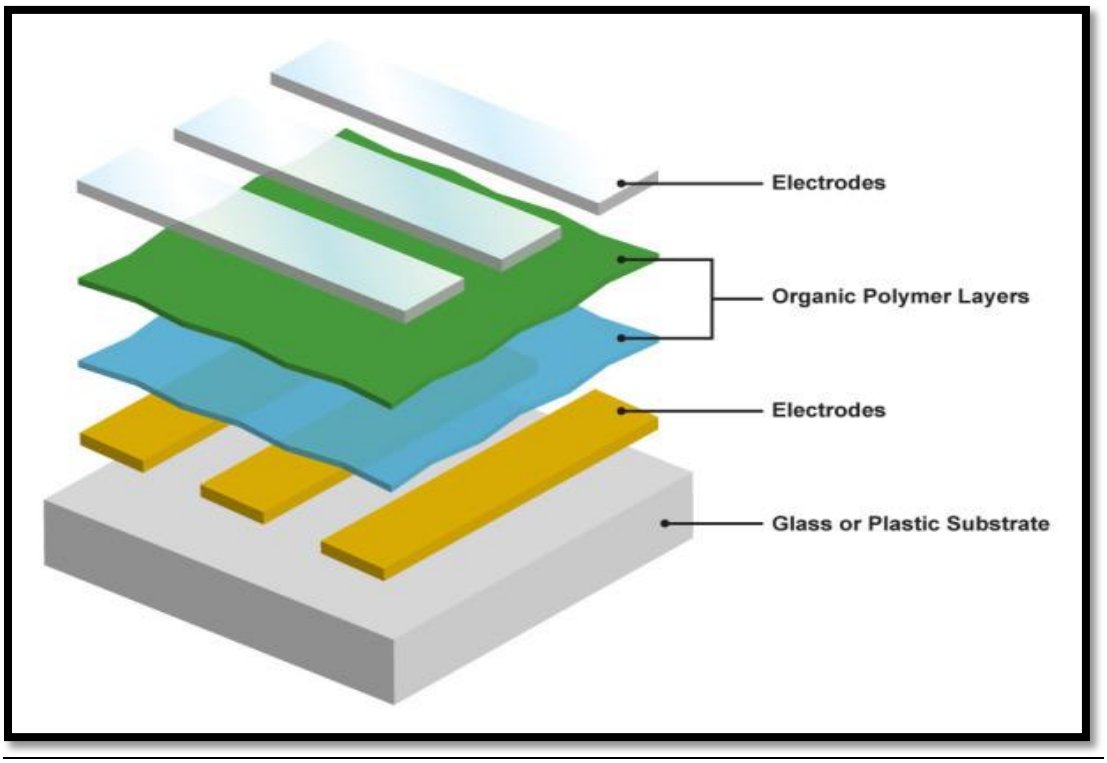


Fig. 2.8: Layers in an OLED display

Owing to its moderate cost, desirable size and availability, a 7 inch LCD panel is adapted for this work. It also has sharp resolution features and a slim body that was required for this tablet device.

2.1.3: Central Processing Unit (CPU)

In order to function at all, a device needs a computing module that can take in some data from by a user, process it according to its protocol and return an output that the user desires. This integral job is performed by a CPU. Nowadays a number of portable, easy to use mini-computers are available that can serve as the main processing unit of our device. The features of some such dominant computers are below explored.

Raspberry Pi

The Raspberry Pi is a miniature ARM-based computer board in the size of a credit card. It is developed in the UK by the Raspberry Pi Foundation to promote basic computer science in schools. It can be used as a desktop PC when a keyboard, mouse, display, microSD card and power supply is connected to it. It can run operating systems (OS) based on the Linux OS kernel, such as Raspbian, Arch Linux ARM, OpenELEC, Pidora. All types of applications that a normal PC can do, like preparing documents and spreadsheets, watching HD videos, playing games and even surfing the internet can be done with the Raspberry Pi. [12]

Over the years the Raspberry Pi Foundation has developed the Raspberry Pi in terms of its processor speed, RAM, USB ports etc. Each version was name as Model A, A+, B and so on. The very first model of the RPi, Model A, had only 256 MB of RAM, one USB port and no Ethernet port while the latest Model B+ has better processor speed, four USB ports, lower power consumption etc.

Some of the specifications of Raspberry Pi Model B+ are follows:

- Dimension: 86 x 56 x 20 mm
- Broadcom BCM2835 700MHz ARM1176JZFS processor
- 512MB SDRAM
- +5V, 2A power supply via microUSB socket
- 4 x USB 2.0 sockets
- HD 1080p video output

- Composite video (PAL/NTSC) output
- Stereo audio output
- 10/100 BaseT RJ45 Ethernet socket
- HDMI 1.3 & 1.4 video/audio socket
- MicroSD card socket
- 40-pin header for GPIO and serial buses [13]

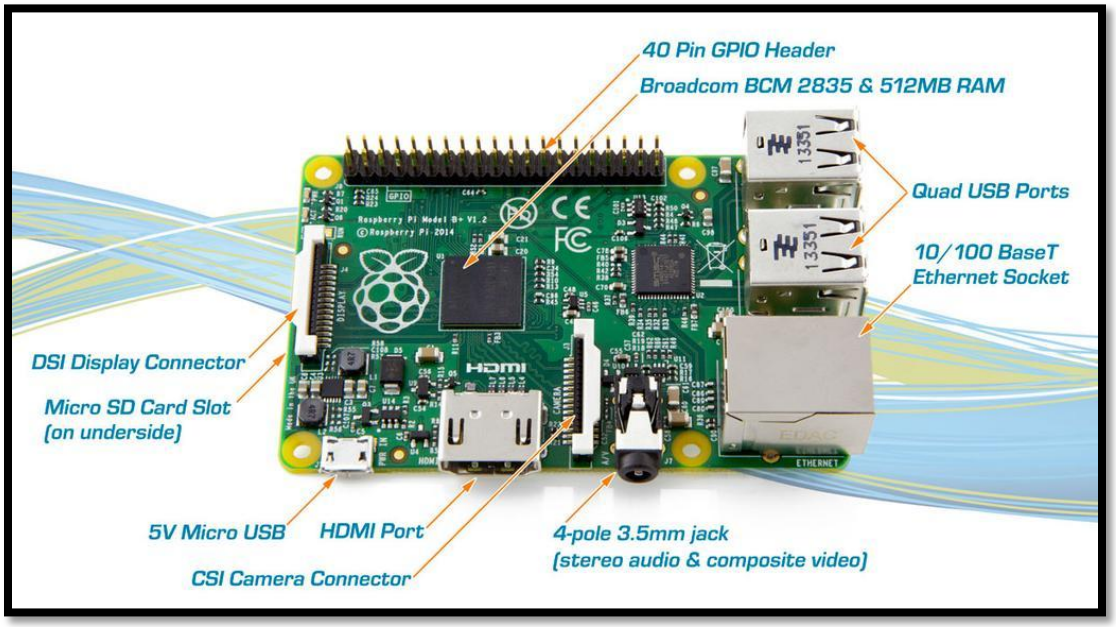


Fig. 2.9: Raspberry Pi model B+ layout

Arduino Uno

Another popular small sized computer board is the Arduino Uno. It is a simple microcontroller board with no operating system loaded. Instead, an application called Development Environment helps to write the software for the board. Arduino can take input from different switches and sensors and control different motors and other physical outputs. What makes Arduino superior to other microcontroller boards is its cheap price, and the fact that the software can be run in almost all the OS like Windows, Macintosh, and Linux. The Arduino software is also an open source software and can be extended by advanced programmers for their desired work. [14]

To use Arduino, we need to download the Arduino Development Environment application and extract its files. Then we connect the board to the computer where these files are and begin installation. After successful installation we may run the application, write desired codes in it and compile them. Then the codes can be uploaded to the Arduino module. [15]

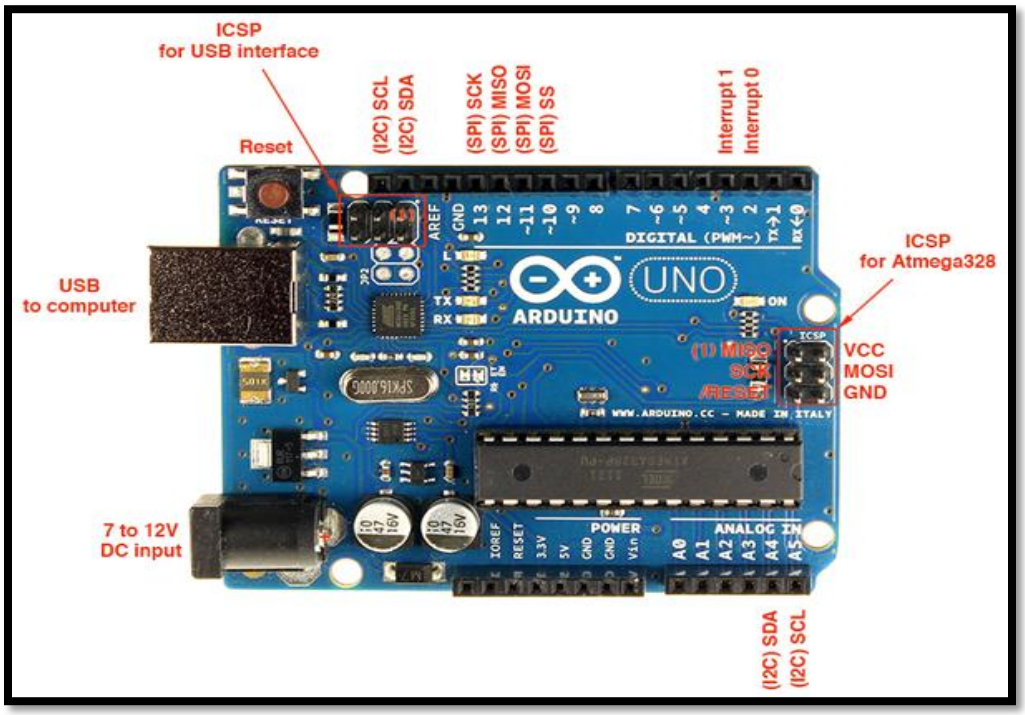


Fig. 2.10: Arduino Uno layout

For the purpose of our project, we used Raspberry Pi Model B+ rather than Arduino for some of the useful features the Pi provides. Although the prices of both modules are almost same, Raspberry Pi is far ahead than Arduino in terms of processor speed, RAM and storage memory. Raspberry Pi can multitask and can be easily connected to a network or the Internet; whereas Arduino is incapable of multitasking and connecting it to a network is quite a hassle. Since these factors are major requirements for our project, Raspberry Pi serves as the better module here. Arduino provides easier and simpler programming than the Pi; in spite of this we will use the latter for its superiority in other aspects.

Note: The terms Arduino, Uno and Arduino Uno has been used interchangeably for the single model of Arduino Uno.

2.1.4: Wireless adapter

Wireless adapters are electronic devices that allow computers to connect to the Internet and to other computers without using wires. They send data via radio waves to routers that pass it on to broadband modems or internal networks.

For efficient data transfer, they maintain a broadcast standard supervised by IEEE numbered as 802.11(applicable to wireless networks only) and a maximum speed corresponding to that standard. There are many versions of 802.11 standard developed so far, the most recent and widely used now being the 802.11n standard. Wireless adapters also encrypt their access portal to maintain security using protocols like WEP and WPA. WEP is a weak protocol whereas WPA2 has now has the strongest security. [16] Wireless adapters are of many types, as described below, based on the devices they can be connected to. [17]

Peripheral Computer Interconnect (PCI)

This is used in desktop computers. They have to be fitted in the PCI slot of the motherboard and their installation process is very easy. They have an external antenna that sticks out of the CPU.

Personal Computer Memory Card International Association (PCMCIA) or Card Bus

This is used in notebook computers and has an antenna built internally. A computer must have a PCMCIA slot to install a card bus.

Mini PCI

This is also used in notebook computers and requires a mini PCI slot to install. A computer must have a built-in wireless antenna for this type of wireless adapter.

USB wireless adapter

This can be used in any device with a USB port and has plug-and-play operability. Its internal antenna and small size makes it a portable and user friendly adapter.

Ethernet Port

This type of wireless adapter needs to be connected to a computer's Ethernet port using an Ethernet cable to function. Wireless networks can only be accessed by the computer then.

Compact Flash

This is used mainly in hand-held computers and notebook computers. They are really small in size and require a CF slot or PCMCIA adapter to install.

For the purpose of this project the EDIMAX EW-7811UN USB Wireless Adapter is used. It satisfies 802.11n standard and has a maximum speed of 150 Mbps. Having an internal chip antenna, it gives our device the portability and user friendliness that we are aiming for. Most importantly, it is compatible with the Raspberry Pi module. [18]

2.1.5: Miscellaneous materials

Other common devices, mainly for the purpose of connection and simple operation are used in the device. These include an HDMI cable, power supply cables for LCD panel and Raspberry Pi, microSD card, etc. The HDMI cable, as we will explain in next chapter, connects the Raspberry Pi to the LCD display, while the microSD card is required to load an OS on the RPi. A stylus or pen-like substitute is also used to operate on the touchscreen panel.

2.2: Operating system

The operating system is the software in a device that manages the hardware and software and brings them to a common interface. The overall strength and weakness of a device depends on its OS. Bearing this in mind installing common operating systems like Windows or Android in the Raspberry Pi may seem like a desirable and easy solution for our device. However in reality the RPi is not a powerful enough unit to run any of them. Raspberry Pi has an ARM v6 processor which does not support Windows at all. [19] An older version of Android like the Android 2.3 (nicknamed Gingerbread) can be successfully installed in the Pi, but it will make the computer extremely slow. Hence it is not feasible to use any Android OS as well. [20]On the other hand, the Linux based operating systems discussed below are compatible with the Pi.

Raspbian

This is an operating system developed by Linux specifically for the Raspberry Pi based on its Debian OS. It brings in most of Debian's software packages, but they are edited for the Pi module. It is the recommended operating system for the Pi by the Raspberry Pi Community.

Pidora

Pidora is a stable operating system for Raspberry Pi, and is based on the Fedora OS. Unfortunately in terms of performance and availability of software, Pidora lacks behind Raspbian. It also poses some operation problems that Raspbian can perform easily.

Arch Linux ARM

The Arch Linux ARM operating system has an advanced coding structure that primarily only experienced programmers are capable of handling. This structure is much more difficult than that of Raspbian.

RISC

The RISC OS is unique from the other operating systems as it has different sets of layouts and interfaces. Generally, this is not a problem for the RPi module. However, what makes RISC less appealing to users is that the teaching tools are few here in comparison to Raspbian. Also, a bit of advanced knowledge is required to use this operating system, which contradicts the Raspberry Pi's objective to be an easy-to-use computer.

Compared to the other operating systems, Raspbian is fast, easy to use and has a lot of educational and teaching software, making it quite resourceful. Evidently thus, we selected Raspbian as our device operating system. [21]

2.3 Java

The Java Platform may have been used for decades, but it is a fairly recent programming language when compared to the C Language. The project concerned here uses the Java Platform, Standard Edition (Java SE), which allows development and deployment of Java applications on various systems, such as desktops and servers, as well as in today's demanding embedded environments. It offers a rich user interface, performance, versatility, portability, and security that today's applications require and thus it was chosen for this project. The primary reason of the choice is the standard suite of APIs and services for developing a graphical interface for a drawing tool and server-side and client-side applications.

From its creation, Java was intended and designed as an object-oriented programming language that has various features that the user requires for building applications. Many features, such as *Swing* (stems from the javax.swing package) and its components help create a general user interface; *Exceptions* (required for handling anomalies that may occur in a series of events) which helps writing correct error-free applications, while *Socket* (from the *java.net.Socket* class) objects help in connecting servers to clients or to databases. [20] Initially, the software to be used in this project was decided to be built in the Android Operating System but the decision has since been altered to Java. The primary cause of using Java instead of Android is the widespread compatibility of Java over Android across multiple operating systems and hardware. Java can easily be translated to be used by other programs as the Java codes are *serializable* [22], which allows the other programs to read them easily.

2.3.1: The common features of Java

Java provides a host of features that have been used in this software but primarily, the hierarchical system of organizing data is why Java is preferred here. These features or common objects, had they been grouped in individual classes, would render a large incomprehensible list of classes for a large program and would be unorganized. Thus we group necessary programs into packages for a category of classes [23].

Java Packages

Packages in Java, as the name suggest, are a collection of classes that are of the same type or are inherently related to each other, which are grouped together in a "module" called package. It becomes essential to group multiple Java classes into packages as a project grows bigger, by splintering the code into individual classes and then packaging them according to their associative behavior or type. This allows the code to be read easily, and find out individual classes as required. This hierarchical system is essential in Java as it allows codes to follow an organized pattern and allows programmers to retrace their steps in the event of bugs or code malfunction. In laymen's terms, a package is like a directory in the local computer's file system.

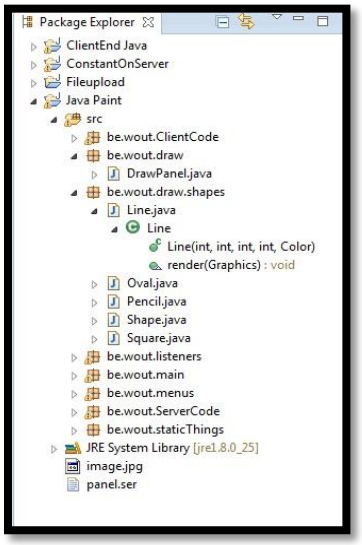


Fig. 2.11: Package (Tree) directory structure

For a Java Program, the package that is saved on a computer location is in itself a directory for the classes. The packages in turn can contain sub-packages. This coordinated structure, like a tree of packages, sub-packages with classes inside these classes is called a package structure. They are organized as directories in our local hard drives inside a zip file, called a JAR file. There is a large collection of packages that has container classes required for different circumstances, such as threads, network communications, exceptions, swings etc that are provided by Oracle in the Java Class Library. The Java Class Library (JCL) is a set of dynamically loadable libraries that Java applications call at run time. There are a number of operating systems out there with different platform-native libraries on different hardware. This inconsistency is why the Java Platform was made to be operating system independent. These common packages are a comprehensive set of standard class libraries, containing the functions common to modern operating systems. Examples of packages are the java.lang that contains the fundamental classes and java.io contains input and output function classes.

Objects and classes

As previously mentioned, Java is an Object-Oriented Language. Before details of objects and classes are discussed, a basic concept of Object-Oriented programming is necessary. In its simplest essence, a computer program can be organized in one of two ways. The program can be built around its code, its real world analogy being what is actually happening or, the program can be built with the data needed to be processed in concern, and building the program around the data. Its primary method of operation is that the data controls the access to the code. Java's basic unit of encapsulation is called a *class.* A class is a blue-print or template that describes in detail the design and make of an object. It does so by specifying both the data and the code that will operate on that data. Objects in turn are instances of classes. For example, a minivan and a car are two different instances of the category vehicle. From this analogy we can interpret objects and classes as a car being an object instance of a Vehicle class. Thus a class is essentially a set of plans that specify how to build an object.

Memory management

Java allows ease for the programmer by removing errors that can break programs and can remove unused objects. This is done by the *Stack* and the *Heap*. They are the two areas of memory, where *the heap* is where the objects reside and *the stack* is where method invocations and local variables live. [24]

Objects and JRE classes are assigned memory by Java Runtime from heap memory. The creation of a new object always starts in the Heap space. Whenever the object is no longer required, loses its relevance or goes out of scope, garbage collection runs on the heap memory to clean those objects that don't have any reference. Global access is allowed for the objects; that is, they can be called via a reference from anywhere within the application. When no longer necessary, the objects are removed automatically from the heap, freeing up memory for the programmer.

The other memory space, called Java Stack memory is where threads are executed from. This memory contains values that are transient and references to other objects in the heap that are getting called from the method. This memory is always referenced in Last-In-First-Out order. A new block or scope is created in the stack memory every time a method is invoked. This stack memory holds local variables (contained within the scope), primitive values and references to other objects in the method. The block then becomes unused when the method ends and becomes available for the next method. Stack memory size is much smaller compared to heap memory. Memory errors still do occur but they can be rectified by throwing *Exceptions.*

Exceptions

Java does not allow developers to have errors or other risky behavior in their code and forces them to address the errors using *Exceptions*. It does so by using a *try-catch block* and it is possible to overlook these errors by having an empty code block in the catch section but the program will not compile without the try-catch mechanism. The exception–handling mechanism in Java is a clean and useful way to handle "exceptional situations" that may occur at runtime. Its primary principle is based on the programmer's knowledge that a method being called is risky

(generates an exception) and thus one should write code to handle that possibility and recover from it.

Support for threads (for concurrent programs)

Threads or multiple threading are programs built right into the fabric of the Java programming language. This facility is useful for running multiple programs simultaneously across multiple computers or in this paper concerned, among the server and clients. It is created by classes extending the java.lang.Thread class or by implementing the Runnable interface. Threads allow the programmer to perform asynchronous and background processing of methods, increase the responsiveness of Java GUI applications (javax.swing) and allow us to use multiprocessor systems with ease. By having separate Thread *objects,* we can launch separate *threads of execution* by multiple clients, each with their own stack of work to do. Having more than one stack, the programmer can provide the illusion of having multiple processes working at the same time. Execution of methods in the stack moves back and forth so rapidly that it seems as though all the stacks are executing simultaneously, i.e. multitasking.

But multiple threads running concurrently bring the problem of scheduling; that is, the sequence in which threads should run. Thus, threads were created with a priority and every individual thread has its own. A higher priority thread is executed before a lower priority thread. Two paths of execution are created whenever a thread is invoked. In each of the paths, the thread itself is executed and the statement after the thread invocation is executed respectively. Each thread will possess its own separate stack and memory space. Using threads have their own inherent problems with multiple processes. When these processes contend for access to multiple methods, there is a possibility of deadlock. The term deadlock here in threads mean that when each thread goes into a frozen state when each is waiting for an action that the only the other can perform. The remedy to this problem is to synchronize the threads, using the keyword “synchronized” to allow a thread to complete before another one is executed and maintain order in all threads. The network communication can thus be conducted using multiple threads instead of innumerable individual processes, which improves parallel communication processes. Overuse of threads

must still be avoided, as it can be detrimental to the program's performance and maintenance. [26]

Network Communication

The Java Platform provides a series of classes and interfaces in a collection in the java.net package that handles the lower level (machine-level) communication details and gives the programmer the freedom to work on other higher level programming issues at hand. In network programming of our project, it is required by our program to execute across multiple devices (computers, tablets, etc. with operating systems that support Java), in which all the devices are connected to each other using a network. There are two common network protocols in the java.net package. They are the TCP (Transmission Control Protocol) and the UDP (User Datagram Protocol). In our project, TCP Protocol has been used and thus we will focus mainly on it only. Since the UDP protocol is a connection-less protocol which communicates only via packets to be exchanged between applications, it is not used here.

The TCP protocol is one of the primary protocols of the TCP/IP networks. It allows for reliable communication between two applications used over the Internet Protocol (IP). The TCP protocol, unlike the IP Protocol which communicates only via packets, establishes a connection between two hosts and exchanges streams of data. The TCP protocol is favored as it guarantees the delivery of data and also guarantees synchronization of the data sent in the form of packets. Sockets are used to establish connection and a communication channel between two devices using TCP. Sockets are obtained in Java from the java.lang.Socket class.

A socket is necessary at both ends of the program; the one situated at the teacher/server end and one situated at the student/client end. The client program creates a socket on its end of the communication and attempts to connect to that socket at the server end. The server end creates a socket object on its side of the communication and thus the two ends can then communicate by

writing to and reading from the sockets. The terms writing to and reading from socket will be explained in detail at the design of the program.

CHAPTER 3: Hardware Design and System Setup

3.1: Hardware Integration

The hardware of this device was assimilated using connection techniques that we know and use often. This makes the device more user-friendly. After the tasks described below is followed we get a tablet like device in appearance, but more has to be done to make it complete and operational.

i. Connect the touchscreen panel and LCD monitor to the Raspberry Pi module.

The touchscreen panel and the LCD monitor are both compatible with Pi. They connect to their respective driver boards with ribbon cables. The driver board of the touchscreen, then, connects to the Raspberry Pi with a USB cable directly to one of the module's ports. The LCD driver board, however, has VGA, HDMI, and composite video ports for connection to any processor. For the purpose of this project, the HDMI port was used for the Raspberry Pi.

ii. Connect all supporting devices to the Raspberry Pi.

Necessary devices like the power supply, microSD card and wireless adapter are connected to the module. The microSD card contains the operating system (Raspbian 'Wheezy') file which the Raspberry Pi will boot. This operating system (OS) is compatible with the Pi and is developed by Linux Corporation. A keyboard is used to enter the username and password for the OS to load.

iii. Connect power supply of LCD monitor.

The LCD monitor is powered using a 12V 2A power adapter connected to the main supply. When the LCD monitor is turned on, its display settings, e.g. channel select, color, brightness, etc may be altered using a mini-keyboard, included in the LCD kit.

iv. Switch on the LCD and Raspberry Pi power to turn the device on.

3.2: Raspberry Pi Setup

The Raspberry Pi is designed to be an easy computer module at user end. We need to only provide it with power supply and its software to work with it. The process described below loads an operating system (OS) in the Raspberry Pi.

i. Install NOOBS.

New Out Of the Box Software (NOOBS) is a system which is used to install one or more operating systems in the Raspberry Pi. It can be downloaded from the Raspberry pi website. For the purpose of this project the offline version of NOOBS is used, which does not require any Internet connection during installation. After downloading the .zip file of the system it is extracted.

ii. Format the SD card.

An 8 GB microSD card is used for the Raspberry Pi. Before use it is formatted using the software "Card Formatter". After installing the software, the SD card is inserted and the formatter tool is run. After selecting the appropriate drive letter for the SD card and given it a suitable name, click "Format". This formats the SD card and makes it ready for use. Then the files extracted from the NOOBS .zip file is copied and pasted in the SD card drive. The SD card is ejected from the computer and inserted in the RPi.

iii. Choose an operating system for the Raspberry Pi and install it.

There are several operating systems available for installation in Raspberry pi such as Raspbian, Arch Linux ARM, OpenELEC, Pidora etc. For this project Raspbian, also known as the Raspbian 'Wheezy' is used. A keyboard and a mouse are plugged in the RPi module before powering it up. The OS Raspbian is selected from a list of available

operating systems and the button “Install” is clicked. After successful installation, the Raspberry Pi reboots and a prompt asks for login information, which is as follows:

Username: pi
Password: raspberry

The system boots further until the command line prompt appears, where “startx” is typed in to launch the desktop. Raspbian is now fully installed and loaded in the Raspberry Pi.

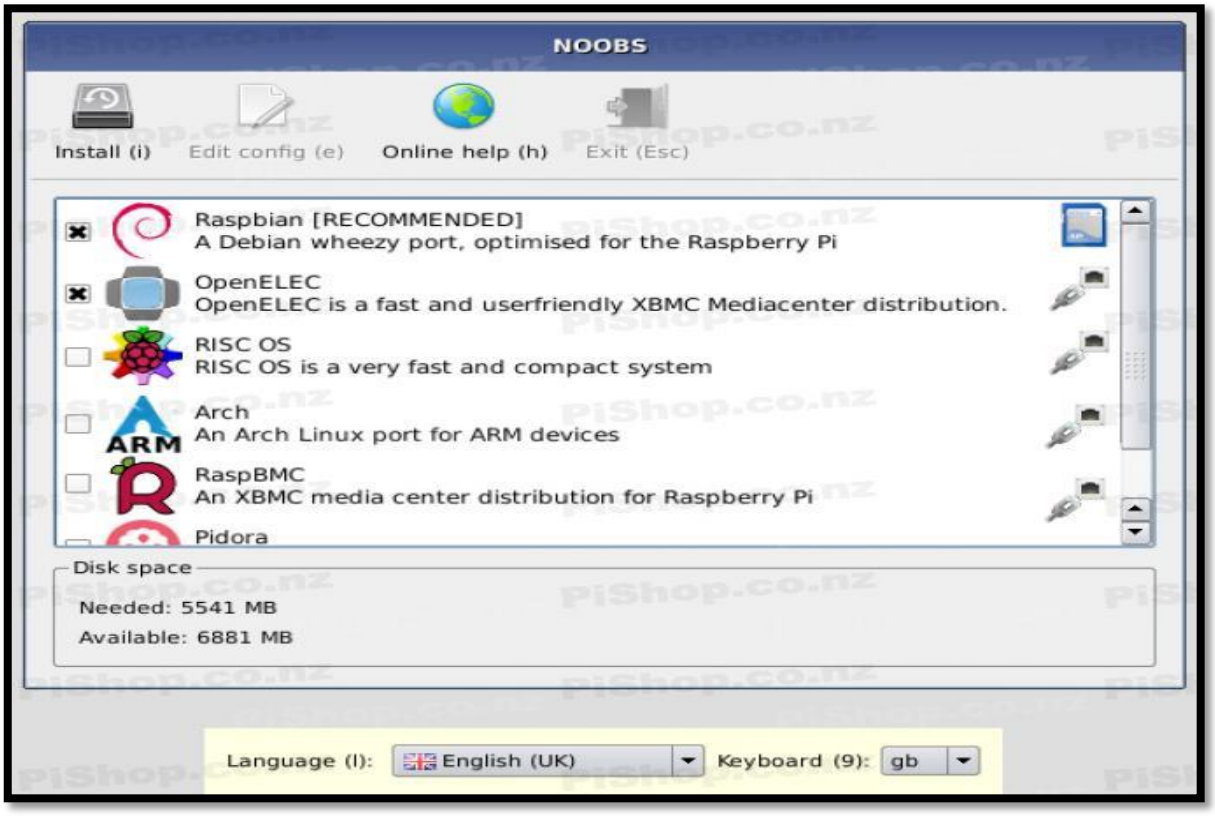
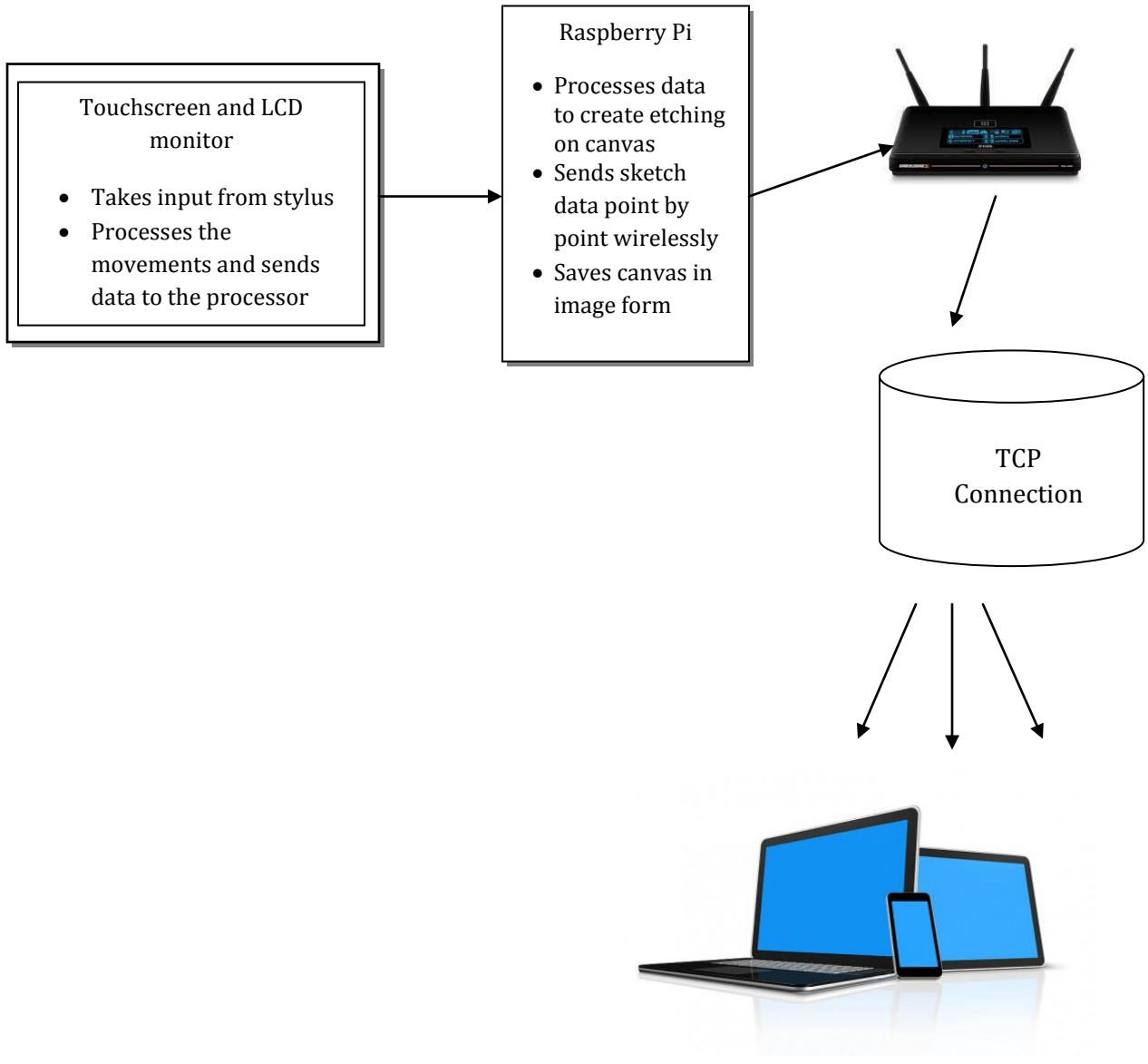


Fig. 3.1: NOOBS OS install manager for RPi

Fig. 3.2: Raspbian desktop after successful installation

3.3: Project block diagram

The diagram below summarizes our slate device and the process in which it operates. The functions each piece of hardware carries out are also stated.



CHAPTER 4: Configuration of Touchscreen Panel to Raspberry Pi Operating System

The integral part of system integration for this device is to combine the LCD monitor (the output device) with the touchscreen panel (the input device), to make a complete touchscreen unit. This incorporation requires two main tasks:

1. kernel recompilation
2. calibration

4.1: Kernel Recompilation

The Raspbian kernel does not by default contain support for a touch sensitive device. Hence it needs to be modified so that any input from the touchscreen can be processed. Since the Raspberry Pi is a small processor and the kernel of an operating system cannot be changed while it is in use, the recompilation process is done in the command line interface (CLI)terminal of a separate computer that runs Ubuntu 12.04 operating system (another Linux OS).The computer requires an active internet connection as well as super user access; otherwise many commands will not be executed.

i) Get system updates and confirm that Raspbian recognizes the touch device.

This is done in the LXTerminal application of the Raspberry Pi to ensure smooth proceedings during kernel build and the touchscreen's compatibility. LXTerminal is the command terminal of Raspbian, and the devices connected to the Pi can be viewed here using a simple command. The OS is updated and recognizes the touchscreen, but it is unable to process its input. So the RPi module is safely shut down and the SD card withdrawn from the hardware.

ii) Build a new kernel.

This is done in Ubuntu. The microSD card is inserted in the computer and the terminal is opened to start the reconstruction process, which is as follows:

(1) Acquire necessary files

The tool chains for a new kernel is downloaded using the Synaptic Package Manager (download wizard in Ubuntu, available in the Ubuntu Software Centre). These along with the kernel sources and some dependencies are downloaded and installed using the terminal window.

(2) Extract kernel sources and navigate to extracted folder.

The compressed file containing the kernel sources are extracted by typing a command in the terminal. Although this opens the extracted folder on a separate window, the terminal needs to change its path to this folder to work with these files. So, using the change directory command the terminal is navigated to this folder.

(3) Configure new kernel.

Firstly, the old configuration files and dependencies are deleted and a new directory named ‘kernel’ is created using appropriate commands. There, a new configuration file (with .config extension) is created, where a new kernel is put in order (See Appendix A). The tool chains installed earlier are used in this process, and the new kernel is programmed to have ARM architecture. After the appropriate commands have been fed, the Linux/arm 3.6.11 Kernel Compilation window opens. There, the ‘USB touchscreen driver’ is selected from the ‘Device Drivers’ list and saved. This adds touchscreen support to the kernel under construction. This window is then safely closed.

(4) Compile the kernel.

Using commands in the terminal, the new kernel is compiled (See Appendix A). This process is a little time-consuming, however, when it is successfully over, the kernel directory shows the new files created.

iii) Create kernel image.

To create the new kernel image, some tools are cloned from the internet in the default (home) directory. The image tool from there is then run to create the kernel.img file in the folder (See Appendix A).

iv) Build kernel modules.

Now that the kernel image is created, the kernel is arranged in modules (sections of programs connected to each other). A directory named module is created and using terminal commands the modules are built within that directory.

v) Acquire latest firmware.

Firmware is the control program written in the ROM of a particular device, and it enables the hardware to work with the software of a computer system. The latest firmware for the SD card is downloaded and the zipped folder extracted in the kernel folder.

vi) Replace the kernel in the SD card.

The SD card containing the operating system has two partitions: boot and root. Raspbian loads from the boot partition and the root partition contains the core files and functions of the SD card. In order to write the new kernel in the SD card, the contents in both these partitions have to be replaced. So, two folders are created, and the boot and root mounted

on each one of them. Now the configuration file, firmware and kernel image files in the kernel folder are copied to the boot mounted folder. In the second folder, the modules and other files are copied. The SD card contains the new kernel image and the replacement process is now complete (See Appendix A).

The SD card is now safely unmounted from the computer and inserted in the Raspberry Pi. The touchscreen now works on the Pi. However, its axes are opposite and the positions are not exact. This is rectified by calibration.

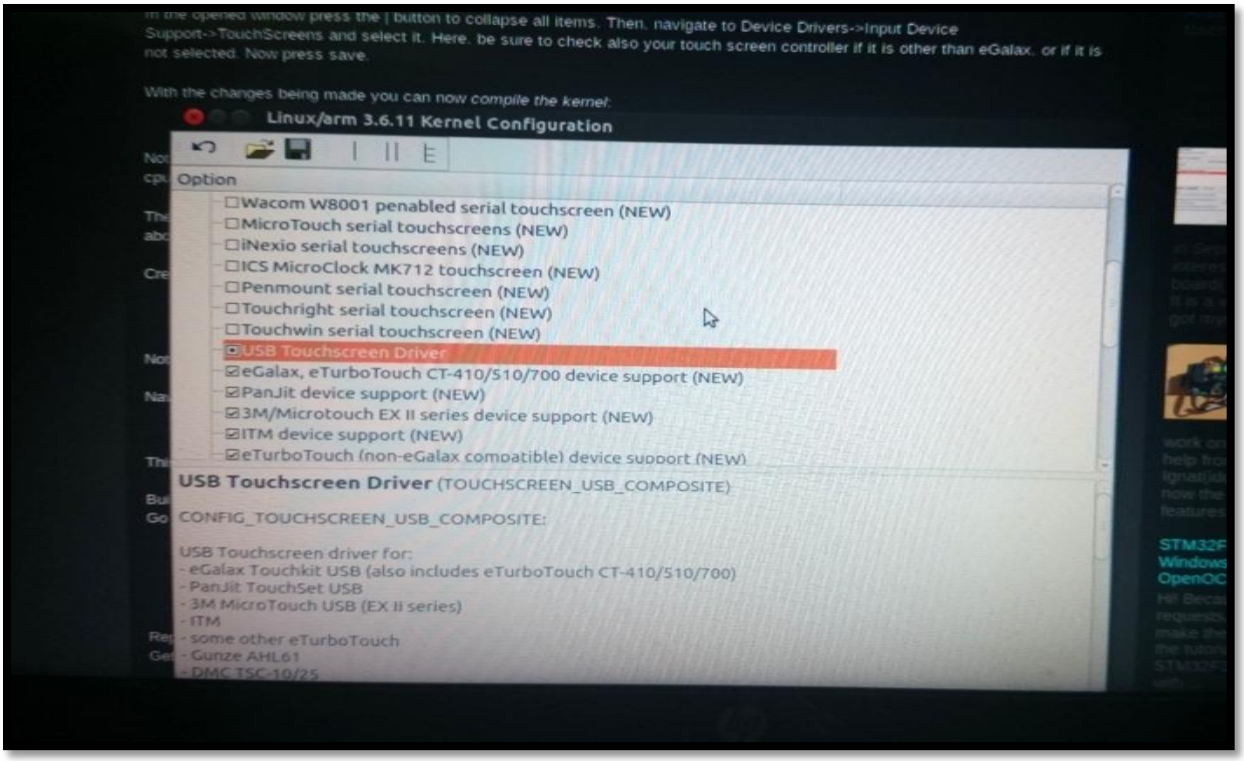


Fig. 4.1: Adding touchscreen support in Linux Kernel Configuration window

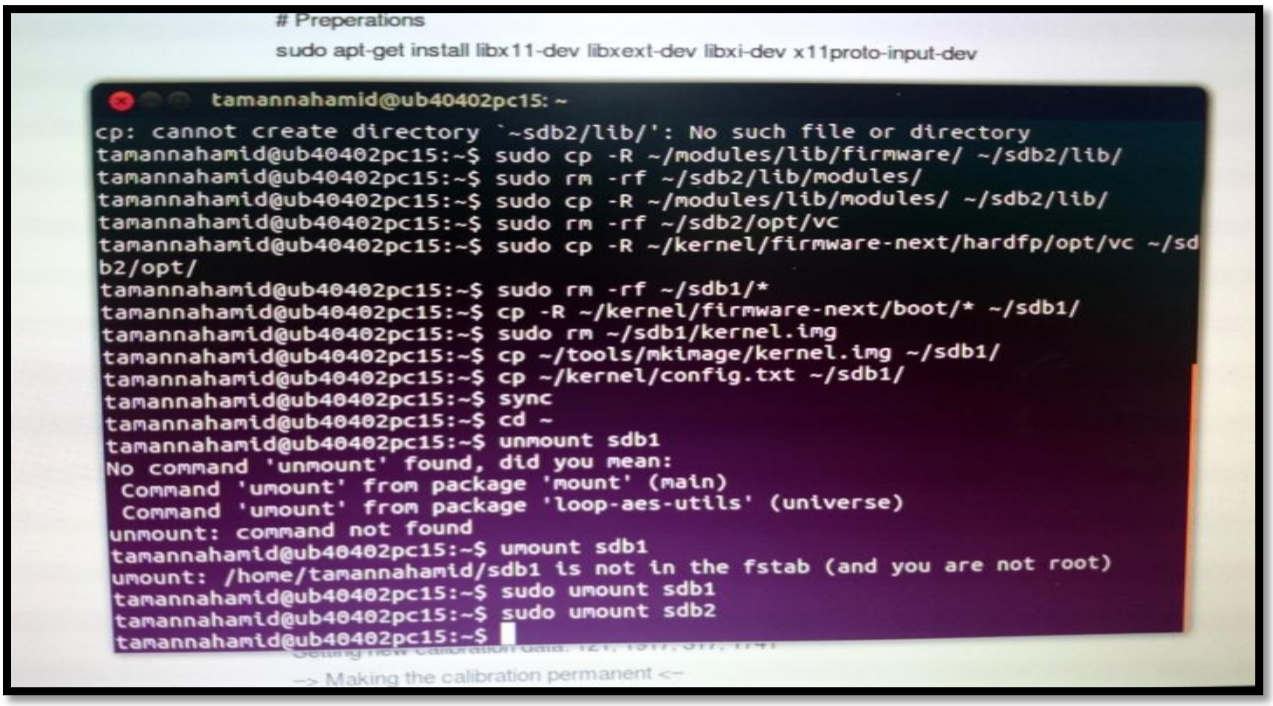


Fig. 4.2: Replacing new kernel image in SD card

4.2: Calibration

The calibration process is done in Pi using LXTerminal, and requires Internet connection to the module. It corrects the touch panel input coordinates to the cursor position on the output device (monitor) in the procedure described below.

i. Download xinput_calibrator and install some dependencies.

Xinput_calibrator is a screen calibration utility. It is downloaded through the LXTerminal, after installing some dependent tools that will be used by the module for the process (See Appendix A). These files may be saved in any location of the Raspberry Pi folder structure.

ii. Install program and run calibration test.

The xinput_calibrator file is extracted on the desktop. Then, the terminal path is navigated to the unzipped folder and the application is commanded to install. The terminal window shows the installation progress, and upon its completion the application is run from there. Then, a graphical user mode window opens and displays some test points to be touched by the user using the touchscreen panel. This is the calibration test that enables the application to match the dimension and coordinates of the Raspberry Pi's display to that of the touchscreen panel it is connected to. When all the test points are touched, the application gives a message showing the current calibration status of the system in the LXTerminal.

iii. Write snippet of calibration message on necessary file.

The test output message contains some statements that can modify the current calibration settings of the operating system. These statements are thus written in a text file in a designated directory, and saved accordingly. The simple method of copying and pasting text is used to do this. The text usually resides in the /etc/X11 or /usr/X11/ directory,

depending on the system (See Appendix A). Thus, the new calibration settings are saved in the Raspbian operating system.

iv. Exit terminal session and reboot the system.

To ensure the system makes the calibration changes permanent, the LXTerminal session is closed safely and the computer is rebooted. [28]

Thus, the touchscreen is fully configured to the Raspberry Pi module. After reboot, the system is able to process touchscreen input like other touch devices, with accuracy.

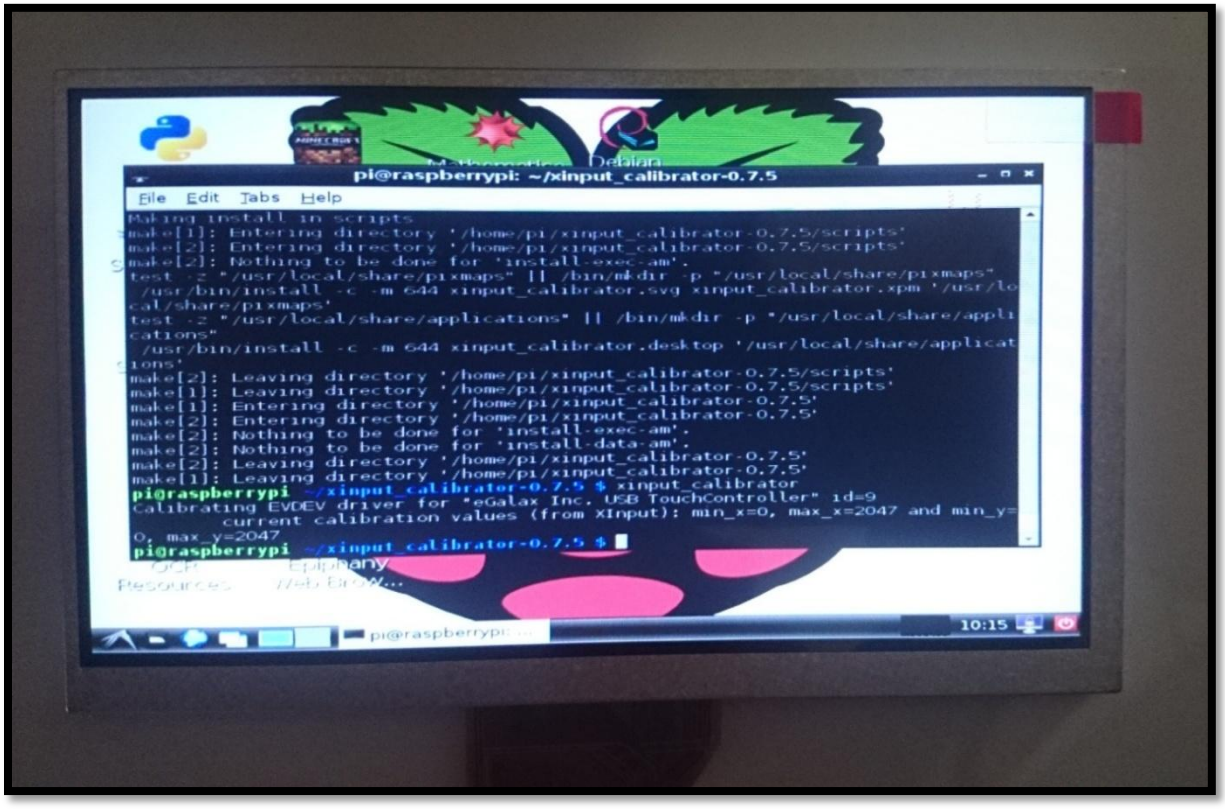


Fig. 4.3: Running xinput_calibrator from LXTerminal

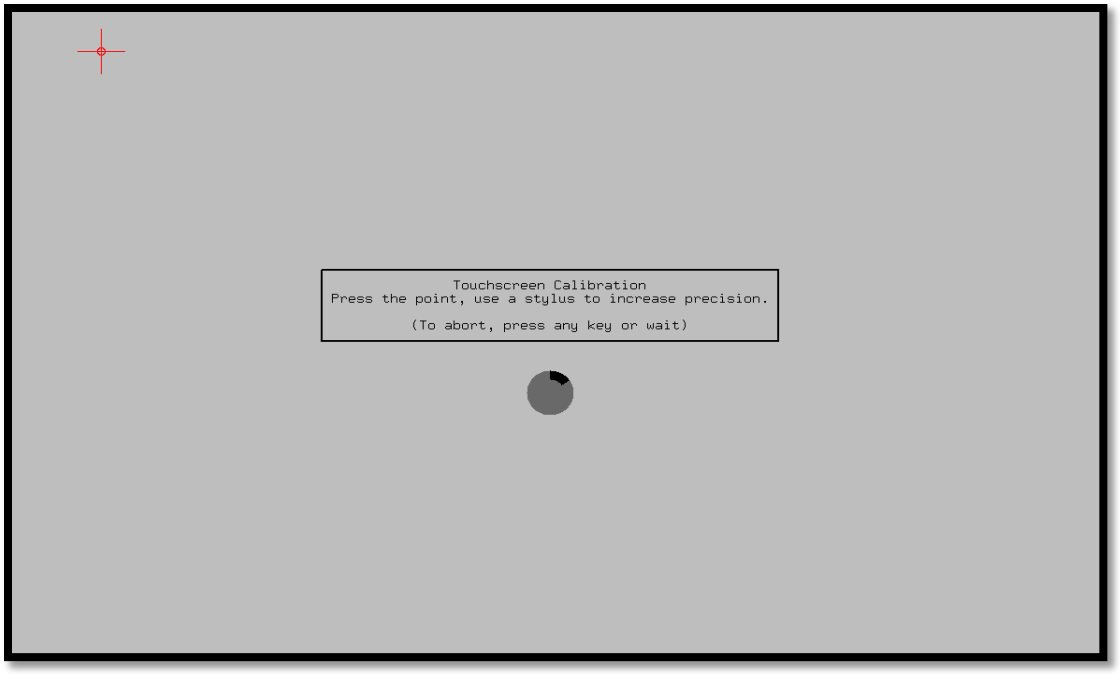


Fig. 4.4: Calibration test window

CHAPTER 5: Preliminary Outline of Java Application

The program for our project was built on the IDE (Integrated Development Environment) called Eclipse developed by the Eclipse foundation which contains a base workspace and an extensible plug-in system for customizing the environment. The library of Java used was Java SE (Standard Edition) Runtime environment 8 (jre1.8.0_25).

5.1: Server (Teacher) End Program – Java Paint

Our software contains two different variations; one for the server/teacher side and the other for the client/student side. Henceforth, server and client side will be used to denote the teacher and student side respectively. The simplest form of a Java program consists of a class declaration that defines a main() method. The method main() contains the code that is executed when the program is run. The server side has been named "Java Paint" in its file directory and contains the following packages with their corresponding inner classes:

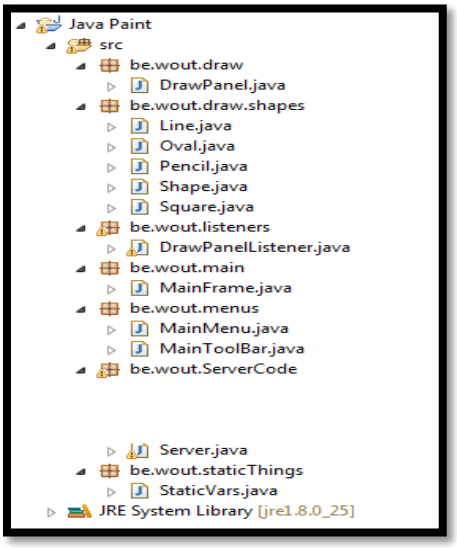


Fig. 5.1: Java Paint Structure

The procedure of how our program works will be explained a hierarchical manner in which process invokes which classes and methods. To begin with, the Server class in the package be.wout.ServerCode is where our main() method lies. The method used:

```
publicstaticvoid main(String [] args){

            newMainFrame();

            Server sally = newServer();
            sally.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
            sally.startRunning();
      }
```

5.1.1: MainFrame class

As the program begins a new MainFrame is called from the be.wout.main package. The class named “MainFrame” extends the class JFrame which will contain the JPanel, where our actual sketches will be formed. JFrame and JPanel are java swing components, or in general terms the canvas for drawing, from the javax.swing class required for the painting application. The class MainFrame simply contains the general layout of the JFrame, for example, dimensions, the BorderLayout manager which controls the design of the look and feel of the GUI (general user interface), the main menu and toolbar menu situated in the program. The main menu and toolbar menu contain options for saving the final image, closing the program and the toolbar contains other editing options. The code for the main menu and toolbar is organized into the package be.wout.menus. The entire code is provided according to the classes in the Appendix and will be further discussed in the next section.

5.1.2: Server class

Next, an instance of the server class is called and the constructor and all the methods for setting up a connection are inside this class. The constructor of the server class contains a JPanel in itself with a TextArea in it for connection status updates. The status updates will be appended to the TextArea from the methods inside the server class. The initialization of these methods will be started from the main method using the keyword sally.startRunning() which starts the startRunning() method inside the server class. Initially a new serverSocket is set up in the name "server" there. A serverSocket gives a mechanism to the server system to listen for customers and make associations/connections with them. Once that is done a socket connection is not instantly made but rather broken down into three separate methods that each perform their own separate work. They are named waitForConnection(), setupStreams() and whileSending() and are named literally as the work they do. This dialog box will show for the server along with the JPanel for showing server status.

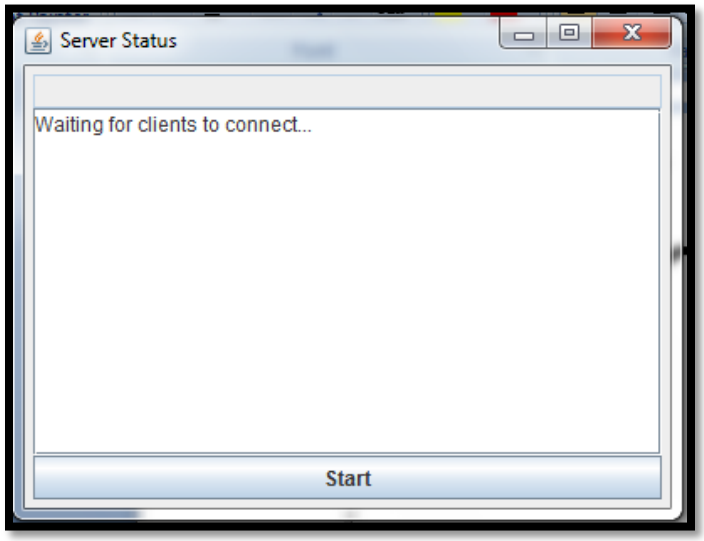


Fig. 5.2: Dialog box for Server Connection Status

The following steps occur when establishing a connection between the two devices using sockets when the startRunning() method begins:

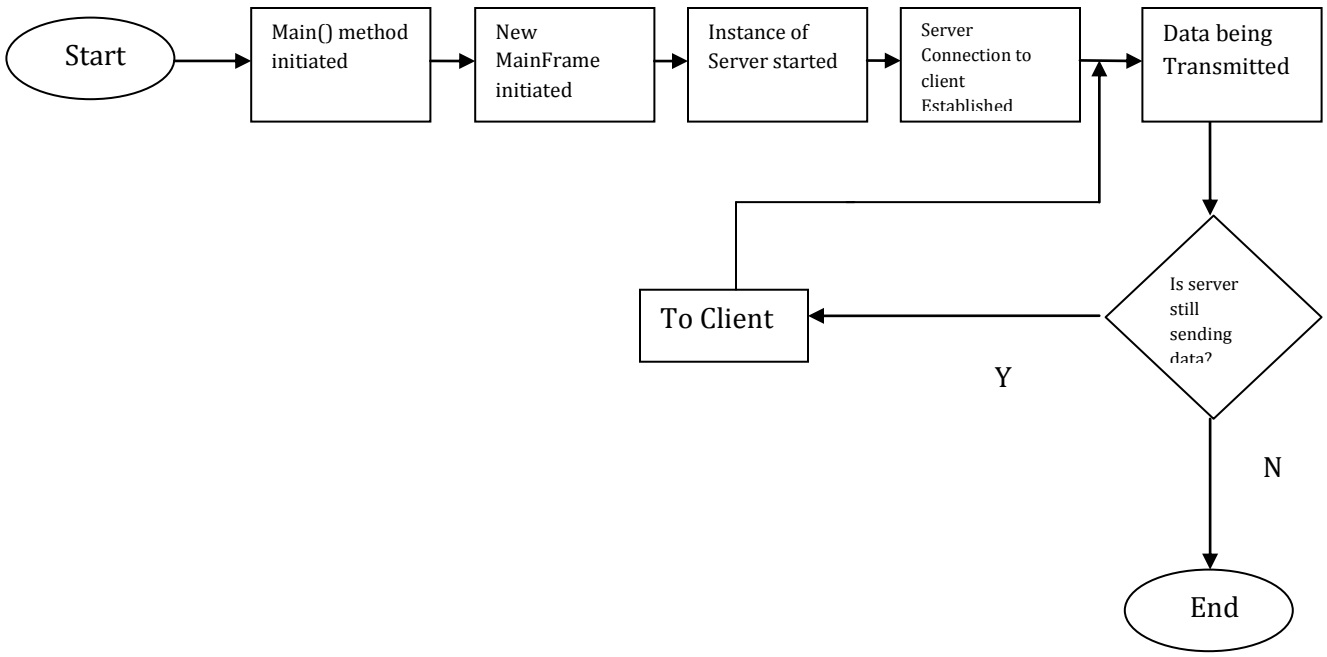
1. The server instantiates a ServerSocket object named “server”, denoting which port number communication is to occur on.
2. When the setupStreams method is invoked, the server invokes the accept() method of the ServerSocket class. This method waits until a client connects to the server on the given port of 6789.
3. After the server is waiting, at the client end a client instantiates a Socket object, specifying the server name and port number to connect to.
4. The constructor of the Socket class attempts to connect the client to the specified server and port number. If communication is established, the client now has a Socket object capable of communicating with the server.
5. On the server side, the accept() method returns a reference to a new socket on the server that is connected to the client's socket.

After the connections are established, communication can occur using I/O streams. Each socket, at the server side and the client side has an OutputStream and an InputStream respectively. The server's OutputStream is connected to the client's InputStream. The reverse connection is not made as we only require the data to be transmitted from server/teacher to client/student.

Finally, when a server-client connection is established the whileSending() method is invoked which is where new text file is created. This is where the data will be saved and this text file will be sent over the TCP connection to the client by OutputStream. The client receives the text file

and draws the input in real time to the canvas (JPanel) at its end. This will also be detailed once we are at the client end.

Fig. 5.3: Main method constituents and the beginning of the program



Note: Conventional flowchart design not used due to limitations in MS Word.

5.1.3: The design of the Drawing Canvas

The MainFrame class invokes an instance of a JPanel class within itself called DrawPanel. This is where our strokes from the stylus or any other input medium are taken from the teacher and sketches are drawn. The DrawPanel class contains an instance of a DrawPanelListener class. This DrawPanelListener class contains the MouseMotionListener and MouseListener interfaces. These two interfaces simply contain methods for mouse motion events like mousePressed, mouseDragged and mouseReleased. The purposes of each of these events are as their names

suggest. The mousePressed, mouseDragged and mouseReleased listeners wait for an input from the user and perform the action as required, that is draw the strokes. This is done by recording the input at the corresponding x and y co-ordinates of the point clicked and dragged in the arrayList Point. Each point is then passed into DrawPanel instance "drawshape" and the overridden graphics component paintComponent draws the strokes using the method fillOval(). A series of ovals are drawn from the point of origin at mousePressed, continuing through mouseDragged and ending at mouseReleased, to show a complete coherent stroke of ovals, similar to the stroke of a pen. Each x co-ordinate is also passed to an array called Newpoints and each y-co-ordinate is passed to a an array called Odpoints. The values stored in each index is appended and saved in a string called sum and written to a text file by a Printwriter instance named "out".

The DrawPanel class also contains another ArrayList of shapes that include Line, Oval, Square and Pencil. They were included as drawing options to the user and Pencil is the only shape that takes user input and traces the sketches and draws them on the canvas. The code for each of these shapes is recorded in the be.wout.shapes package. All of the codes are provided in the Appendix. The canvas and the general GUI of the program is shown below.

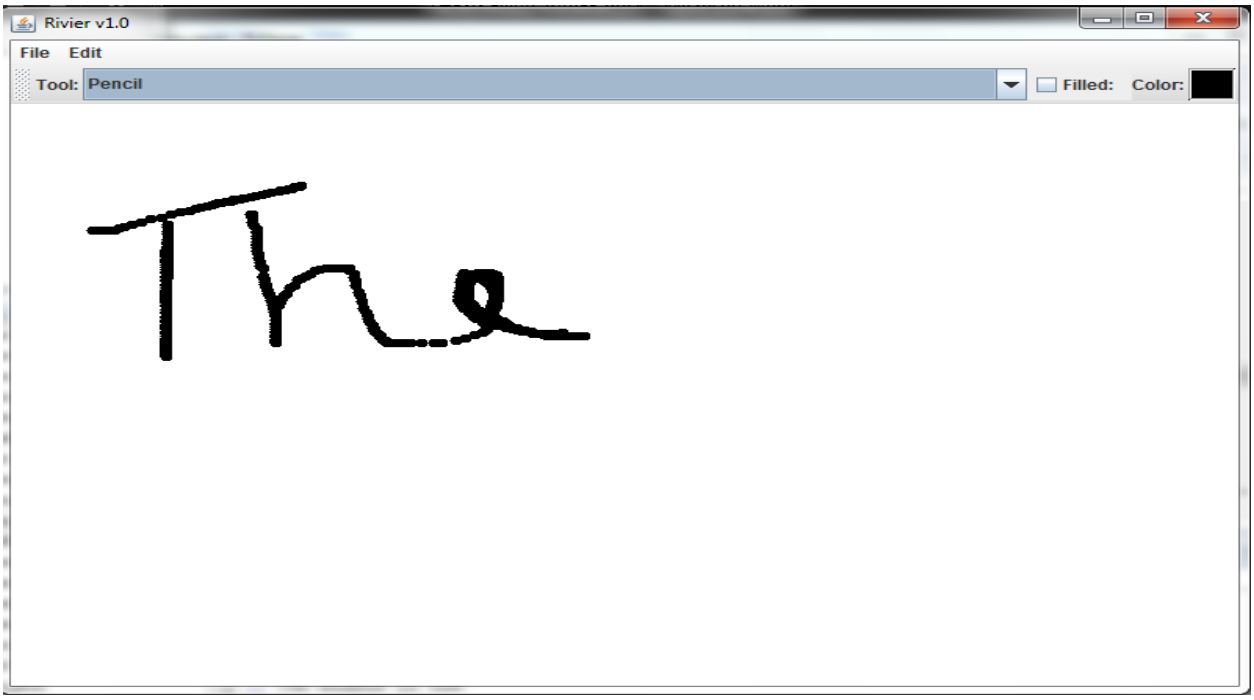
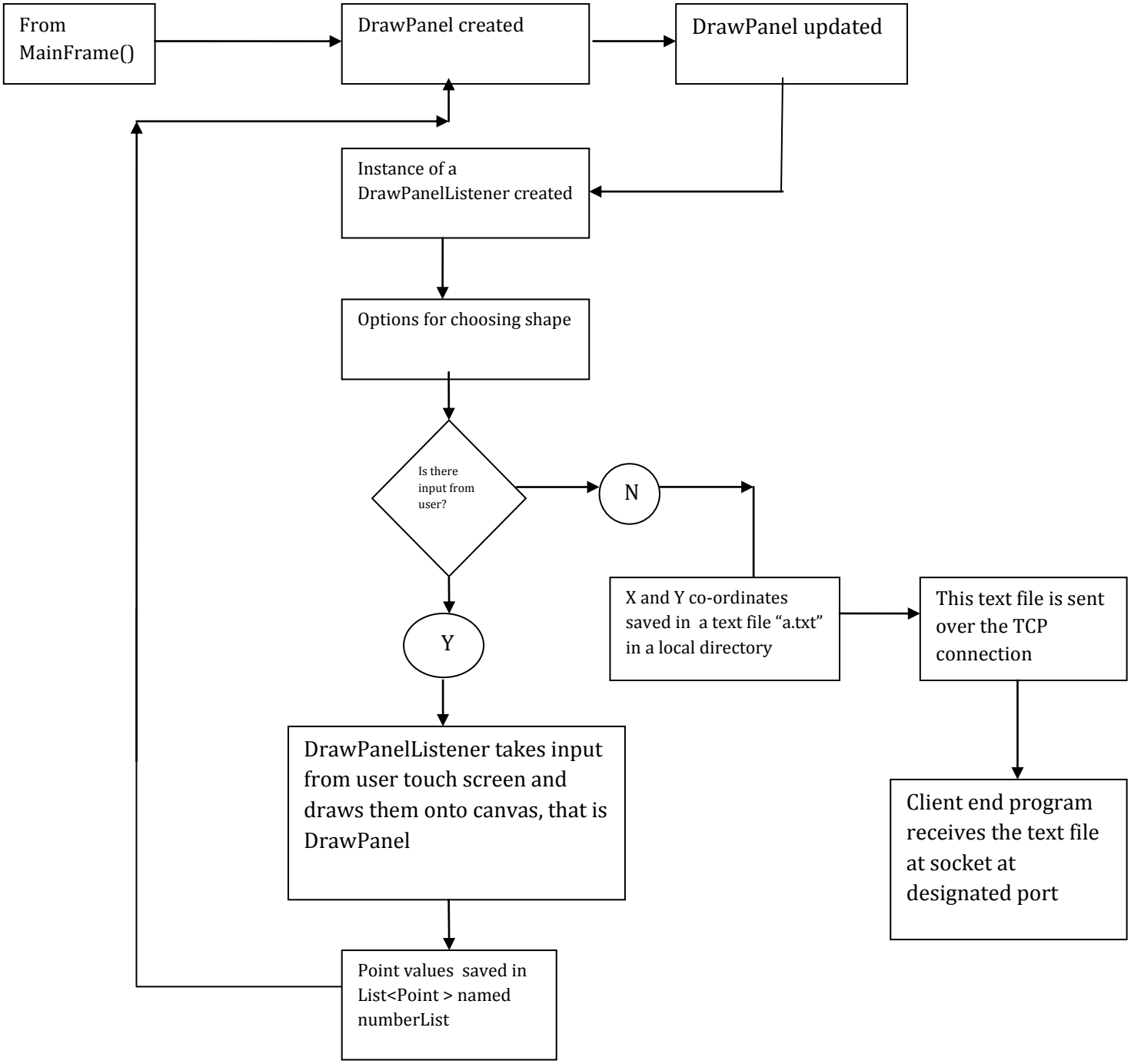


Fig. 5.4: General User Interface of the Drawing Canvas

Fig. 5.5: DrawPanel and the Canvas



5.1.4: Client (Student) End

At the client end, the program is far simpler than that for the server end. Here we have all our requirements in one package as we will need to continuously update the canvas. We save all our components, like JPanel and JFrame all in the same file. A new class of JPanel called Drawer and a new JFrame called MainFrame2 is declared here. Once the main method runs, an instance of the client is created and the method startRunning is called. Here the IP address of the client must be passed into the instance of the new client in text form. For testing purposes on our end, we used the same Raspberry Pi we are working on and hence the IP address is of the localhost.

```
publicstaticvoid main (String [] args) throws IOException{


            Client charlie;
            charlie = new Client("127.0.0.1");
            charlie.startRunning();

 }
```

The constructor and all the methods for setting up a connection are inside the class. The constructor of the client class contains a JPanel in itself with a TextArea in it for connection status updates, similar to that of the server side. The status updates will be appended to the TextArea from the methods inside the client.

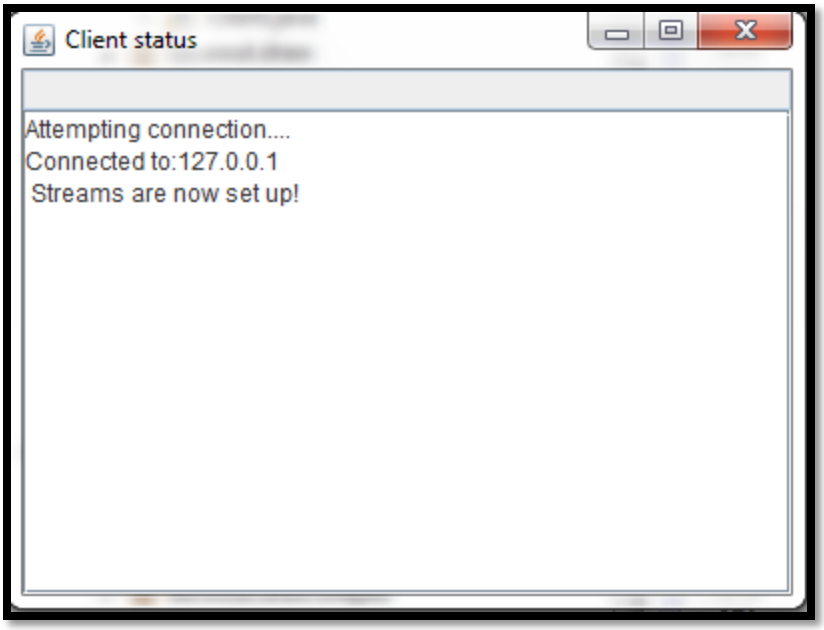


Fig. 5.6: Client Status Dialog Box

This dialog box will show for the client along with the JPanel for showing client status. The initialization of these methods will be started from the main method using the keyword charlie.startRunning() which starts the startRunning() method inside the client class. Initially the connectToServer method starts which connects to the open server that waits for a connection. While that is done the methods setupStreams (which shows the status message that streams are set up) and whileReceiving() start.

Here whileReceiving is where the actual streams are started, unlike that of the server. A byte array called mybytearray is instantiated for storing the values from the server side text file and an InputStream "is" is started to receive that file. This file is then, using a FileOutPutStream and BufferedOutputStream is copied on to a text file called "b.text" in the client end memory. Using the BufferedReader.read() method we read the contents of the file and save it into the byte array. Another BufferedReader is required to read the text file "b.text" and saved into a new String called "line". Using a StringTokenizer (which simply appends the x and y co-ordinates side by

side, separated by a comma in a text file) the String “line” is written to the text file “b.text”. Now that we have received our co-ordinates in a text file we must now separate each co-ordinate and save them again for their corresponding x and y values. The text file is parsed using Integer.parseInt and saved into a new array called “narray” which simply saves all the x and y values. We separate the x and y values into their individual arrays by using another “for” loop. The “for” loop condition is simple. Here, since the starting value was an x co-ordinate and second value was a y co-ordinate in the text file, it follows that all odd index numbers are x co-ordinates and all even index numbers are y co-ordinates. Initializing an integer index counter named “m”, all values for which a mod of “m” returns a zero is even and all that don’t are odd, or is y and x co-ordinates respectively. Each x and y co-ordinate is then stored inside new arrays “Newpoints” and “Odpoints” respectively.

Finally, since our JPanel is in the same class, the paintComponent method is overridden to draw our x and y points from Newpoints and Odpoints respectively, and passed onto the fillOval method. For each point, an oval is drawn. This is continued in a loop for all values in the arrays Newpoints and Odpoints until we have recreated the stroke from the original server end, ending our streaming output.

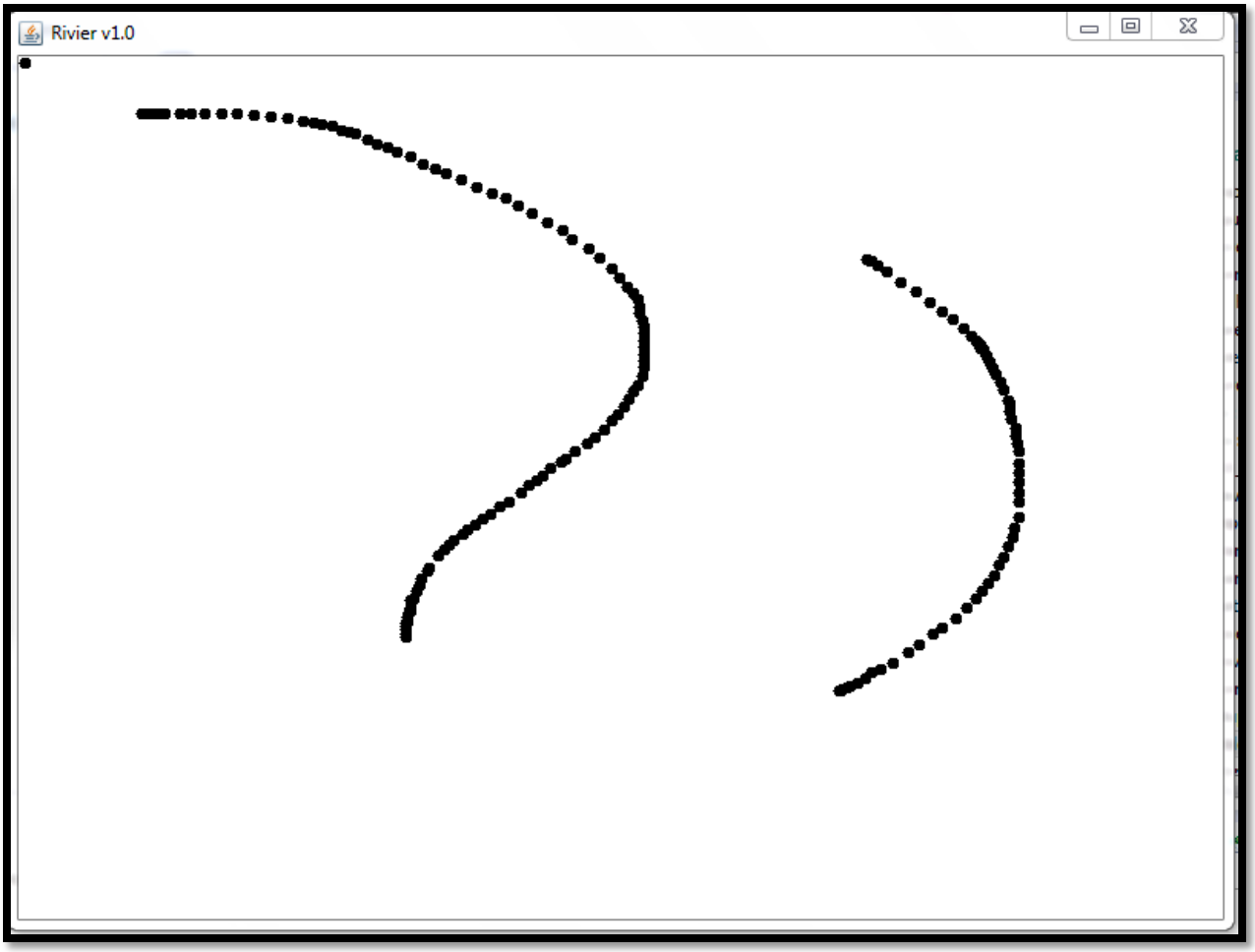


Fig. 5.7: Final output from Server

CHAPTER 6: Broadcasting Technology

While our application is capable of handling the connection and streaming issues of the source device substantially, all the devices concerned still require a network to get connected to each other. We assume for the purpose of our project that a classroom environment will have access to a LAN or similar network. For the transmission of our source device data to others, we connect a wireless adapter to the Raspberry Pi.

As stated earlier, the EDIMAX EW-7811UN Wireless USB Adapter is used for this device for its compatibility with the Raspberry Pi and its portability. Also, its driver contains the latest Raspbian distributions pre-installed in it. [18]

6.1: EDIMAX EW-7811UN Wireless Adapter Installation

A simplified procedure stated below is followed to install the wireless adapter to the Raspberry Pi.

1. Confirm that Raspberry Pi module recognizes the adapter.

   After inserting the EDIMAX USB adapter into the RPi's USB port, it is verified using the LXTerminal window that the Pi recognizes the adapter. The device name is seen in the list in the output.

2. Confirm the wireless adapter kernel driver is loaded in Raspbian.

   The kernel driver of the wireless adapter consists of files that will enable the Raspberry Pi module to transmit data through the adapter. This driver is loaded automatically into the Pi after the adapter is plugged in. To verify this, the ‘lsmod’ command is used in the LXTerminal. A list of all the kernel modules currently in the system is returned, and the module named ‘8192cu’ is checked. This is the wireless adapter kernel driver.

3. Verify the current wireless network configuration of the system.

This is done using the ‘iwconfig’ command. It gives a listing of all the wireless network setup details in the system. The ‘wlan0’ adapter in the list confirms that the EDIMAX wireless adapter is fully installed and is ready for use.

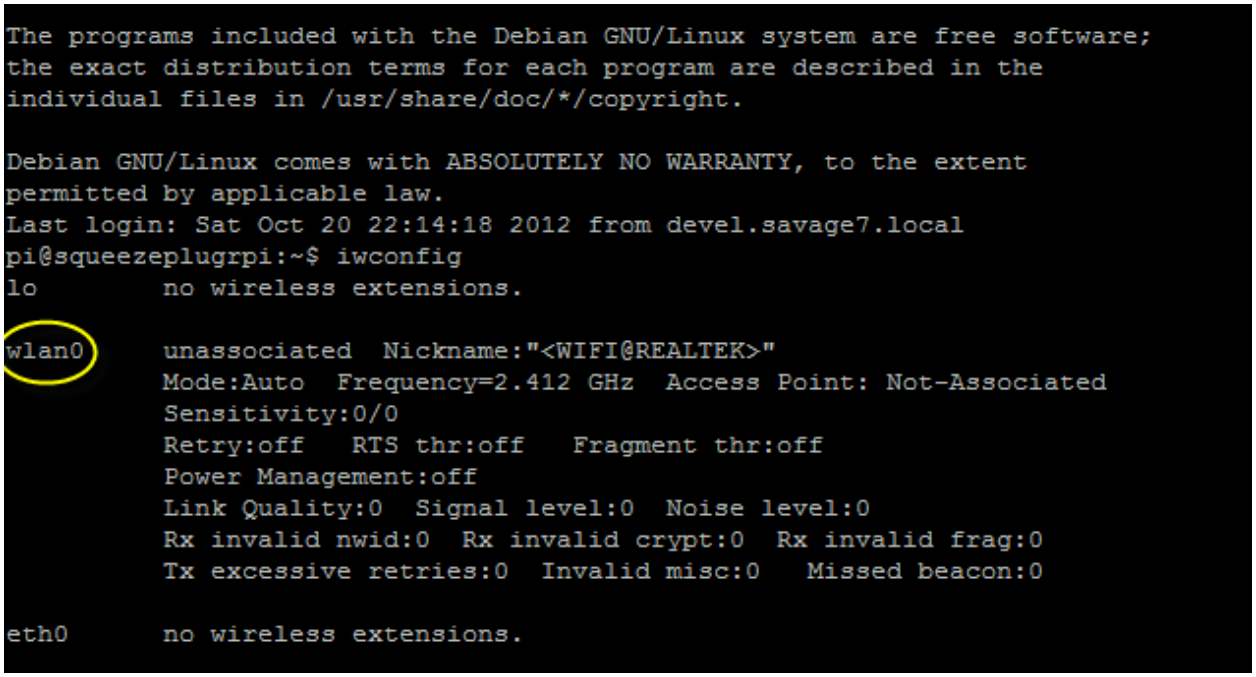


Fig. 6.1: Checking the wireless network configuration of the Raspberry Pi system

6.2: Configuration

The procedure stated below is followed to configure the wireless connection settings of our device so that it securely connects to an available wireless network.

1. Open the network interfaces configuration file and edit it.

   This is again done in LXTerminal of Raspbian. Using appropriate commands the network interfaces configuration file is opened and modified accordingly (See Appendix B).

2. Create or edit the wireless configuration file in the Raspberry Pi system.

   The wireless configuration file is a text file that contains the credentials of the network that the device is connects to. Hence using suitable commands this file is opened and appropriate information is added to it (See Appendix B). It is then saved properly and closed.

3. Restart the wireless adapter to confirm changes.

   A simple command in the LXTerminal restarts the network interface. This causes the newly defined interface and wireless settings to establish a wireless network connection. After a bit of processing, our wireless interface wlan0 is connected to the network and its IP address is acquired.

4. Verify connection and IP address of the system.

   The new network configuration can be viewed using the ‘ifconfig wlan0’ command. This shows the IP address of the device, as well as information like broadcast address, subnet mask, etc. [18]

The device is now connected to the local network properly. For the purpose of our work, we assume that the wireless network in a classroom (where our device will be used) will have a DHCP server from which student devices may lease an IP address to get connected to the network as well.

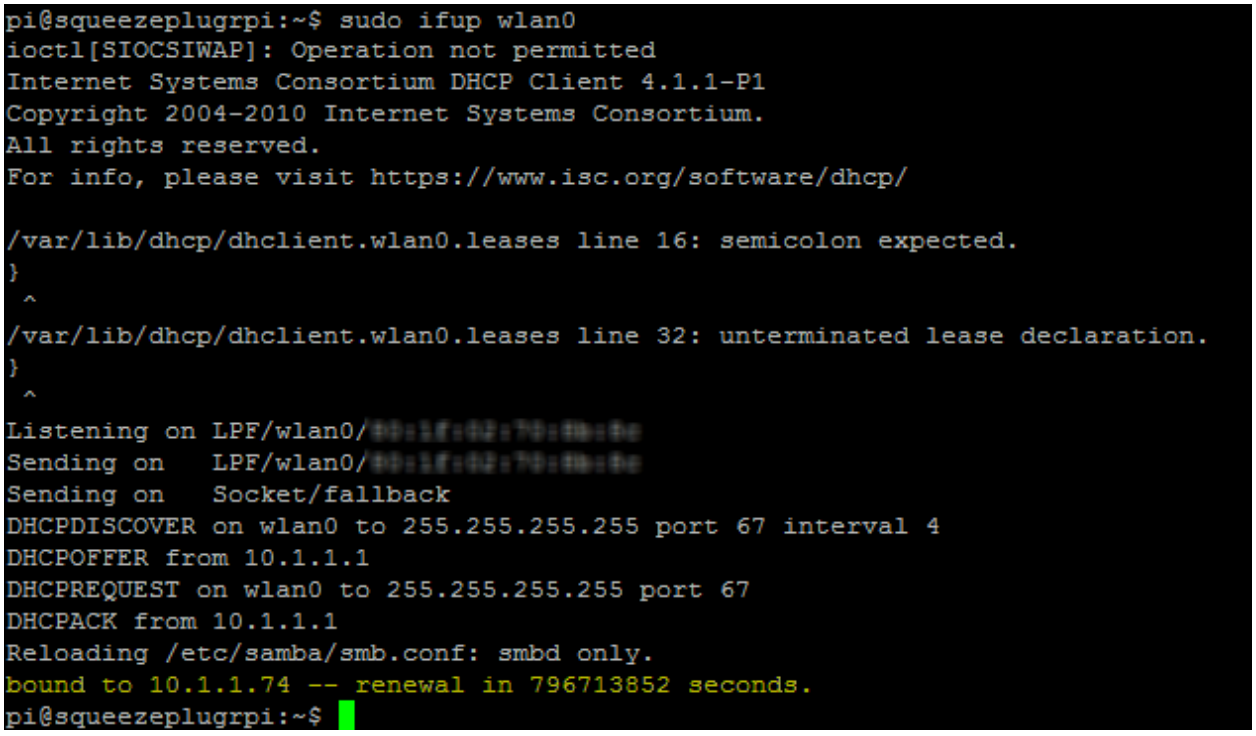


Fig. 6.2: Verifying the connection and IP address of the system

CHAPTER 7: Conclusion

7.1: Discussion

This thesis project aims to create a complete device that can transmit etchings on a canvas application in real time. Various technologies were studied for this purpose and the best ones were selected and executed effectively to build the device. Although previously we aimed to build the application with the Android OS, the Java library is more popular and available in most computer systems and mobile platforms, and has the most compatibility with most devices. Thus we decided to go with the latter.

Due to design limitations, we used a 7 inch display whereas a bigger screen would have made the device more interactive. We have also worked with the Ubuntu operating system, needed to build the kernel for our lcd panel and configuring the touchscreen with it. It required learning the inner workings of a new operating system; a challenge for any newcomer. After facing some problems we became familiar with it and were able to achieve our purpose.

7.2: Further Development and Implementation Aims

Considering the current facilities available in our classrooms, if implemented people in both rural and urban areas, our slate device can be a great tool for learning. The live-stream feature and ability to save a canvas in an image form will provide students an electronic copy of their lessons in class time.

Needless to say, distance learning programmes will also greatly benefit from this. Our recent focus has been on the remote learning campaign started by Grameenphone Limited with affiliation of JAAGO Foundation. This is a distance learning program aimed for primary education for children in Banderban and similar remote areas of the country. We believe our device can effectively contribute to this campaign. Currently Grameenphone is using video streaming to convey the lessons to those places. This involves a large amount of internet data transfer, which can be greatly reduced if our device is combined with audio streaming of the lecturer.

At present, there has been an ongoing political vision of Bangladesh Government for the year 2021 known as the Vision 2021. This will help to make our country more digitalized. Our device can contribute in one of the main sectors of this vision, which is the education sector. Concept of Digital Bangladesh can be achieved in a better way if our device is implemented on it.

References

1. Arato, A., Juhasz, Z., Blenkhorn, P., Evans, G. & Evreinov, G. (2004, July 7). *Java-powered Braille Slate Talker.* Paper presented at Computers Helping People with Special Needs: 9th International Conference, ICCHP 2004, Paris, France, July 7-9, 2004 Proceedings, Paris. DOI: 10.1007/978-3-540-27817-7_74

2. *EMR (Electro-Magnetic Resonance) Technology.* Retrieved from http://www.wacom-components.com/english/technology/emr.html

3. Baker, B. & Fang, W. (2007, May 28). *Powering resistive touchscreens efficiently.* Retrieved from http://www.planetanalog.com/document.asp?doc_id=527515

4. Lee, D. (2010, October 28). *Capacitive vs. Resistive Touchscreens.* Retrieved from http://capacitive-resistive-touchscreens.articles.r-tt.com
https://techexplainer.wordpress.com/2012/04/02/resistive-vs-capacitive-touchscreen/

5. Bhalla, M.R., &Bhalla, A.V. (2010). Comparative Study of Various Touchscreen Technologies. *International Journal of Computer Applications*, *6*(8), 13-14.

6. *How-it-Works: LCD Display Technology*. Retrieved fromhttp://www.practical-home-theater-guide.com/lcd-display.html

7. LCD Characteristics - Advantages and Disadvantages. (n.d.)*VarTech Vision*, 2 (4). Retrieved from http://www.vartechsystems.com/pressroom/aprnewsletter2003/lcd-advantages.htm

8. Morris, I. (2014, May 8). *How Plasma TV works: technology explained,vs LCD, 4K and the future of plasma.* Retrieved fromhttp://www.expertreviews.co.uk/tvs-entertainment/8078/how-plasma-tv-works-technology-explained-vs-lcd-4k-and-the-future-of-plasma

9. HowStuffWorks.com Contributors. (2011, July 22). *What's the difference between a LCD TV and a plasma TV?* Retrieved from http://electronics.howstuffworks.com/difference-between-lcd-tv-and-plasma-tv

10. Cross, J. (2012, March 18). *Digital Displays Explained.* Retrieved from http://www.techhive.com/article/251988/digital_displays_explained.html?page=2

11. Freudenrich, Ph.D., C. (2005, March 24). *How OLEDs Work*. Retrieved from http://electronics.howstuffworks.com/oled2.htm

12. *What is a Raspberry Pi?* Retrieved from https://www.raspberrypi.org/help/what-is-a-raspberry-pi/

13. *Raspberry Pi B+ Product Details*. Retrieved from http://uk.rs-online.com/web/p/processor-microcontroller-development-kits/8111284/

14. *What is Arduino?* Retrieved from http://www.Arduino.cc/en/guide/introduction

15. *Getting started with Arduino on Windows.* Retrieved from http://www.Arduino.cc/en/Guide/Windows

16. Markgraf, B. (2015). How Does a Wireless Adapter Work? *Houston Chronicle.* Retrieved from http://smallbusiness.chron.com/wireless-adapter-work-61042.html

17. *Wireless Adapters Buying Guide.* Retrieved fromhttp://www.newegg.com/Product/CategoryIntelligenceArticle.aspx?articleId=224
18. Savage, R. (2012, October 20). *Raspberry Pi - Installing the Edimax EW-7811Un USB WiFi Adapter (WiFiPi).* Retrieved from http://www.savagehomeautomation.com/projects/raspberry-pi-installing-the-edimax-ew-7811un-usb-wifi-adapte.html

19. *Is it possible for a copy of Windows to run on a Raspberry Pi?* Retrieved from http://www.element14.com/community/thread/24865/l/is-it-possible-for-a-copy-of-windows-to-run-on-a-raspberry-pi?displayFullThread=true

20. Bruce, J. (2013, June 21). *7 Operating Systems you can run with Raspberry Pi*. Retrieved from http://www.makeuseof.com/tag/7-operating-systems-you-can-run-with-raspberry-pi/

21. Zwetsloot, R. (2015, January 16). *Top 4 Raspberry Pi OS.* Retrieved from http://www.linuxuser.co.uk/reviews/top-4-raspberry-pi-os

22. Sierra, K. & Bates, B. (2005) Head First Java, 2nd Edition. O'Reilly Media

23. Oracle and/or affiliates (2014). *Interface Serializable*. Retrieved from http://docs.oracle.com/javase/7/docs/api/java/io/Serializable.html

24. Packages. (n.d.). In Wikipedia. Retrieved April 20, 2015, from http://en.wikibooks.org/wiki/Java_Programming/Packages

25. Kumar, P. (2014, August 18). *Java Heap Memory vs Stack Memory Difference*. Retrieved fromhttp://www.journaldev.com/4098/java-heap-memory-vs-stack-memory-difference

26. Singh, C. (n.d.). What are Java Threads? *Basics: All about Java Threads.* Retrieved from http://beginnersbook.com/2013/03/java-threads/

27. *NOOBS Setup*. Retrieved from https://www.raspberrypi.org/help/noobs-setup/

28. Istodorescu, A. (2013, January 13). *Adding 7 inch display with touchscreen to Raspberry Pi.* Retrieved from http://engineering-diy.blogspot.com/2013/01/adding-7inch-display-with-touchscreen.html

APPENDIX A

The following list contains the terminal line commands used for various purposes as captioned.

1. To configure new kernel:

   makemrproper
   mkdir ../kernel
   make O=../kernel/ ARCH=arm CROSS_COMPILE=/usr/bin/arm-linux-gnueabi- bcmrpi_cutdown_defconfig
   make O=../kernel/ ARCH=arm CROSS_COMPILE=/usr/bin/arm-linux-gnueabi- xconfig

2. To compile new kernel:

   make O=../kernel/ ARCH=arm CROSS_COMPILE=/usr/bin/arm-linux-gnueabi- -k -j3

3. To create kernel image:

   cd../
   git clone git://github.com/raspberrypi/tools.git
   cd tools/mkimage

   ./imagetool-uncompressed.py ../../kernel/arch/arm/boot/Image

4. Replacingnew kernel in the SD card:

   cd ~

mkdir sdb1 sdb2
sudo mount /dev/sdb1 sdb1
sudo mount /dev/sdb2 sdb2

cp ~/sdb1/config.txt ~/kernel/
*sudorm -rf ~/sdb1/**
cp -R ~/kernel/firmware-next/boot/ ~/sdb1/*
sudorm ~/sdb1/kernel.img
cp ~/tools/mkimage/kernel.img ~/sdb1/
cp ~/kernel/config.txt ~/sdb1/

sudorm -rf ~/sdb2/lib/firmware/
sudocp -R ~/modules/lib/firmware/ ~/sdb2/lib/
sudorm -rf ~/sdb2/lib/modules/
sudocp -R ~/modules/lib/modules/ ~/sdb2/lib/
sudorm -rf ~/sdb2/opt/vc
sudocp -R ~/kernel/firmware-next/hardfp/opt/vc/ ~/sdb2/opt/

sync
cd~
sudoumount sdb1
sudoumount sdb2

5. To download xinput_calibrator and install dependencies:

cd ~
wget http://github.com/downloads/tias/xinput_calibrator/xinput_calibrator-0.7.5.tar.gz

sudo apt-get install libx11-dev libxext-devlibxi-dev x11proto-input-dev

6. To make text file to copy calibration message to:

sudomkdir /usr/X11/xorg.conf.d
sudonano /usr/share/X11/xorg.conf.d/01-input.conf

Appendix B

The following list contains the terminal line commands used for various purposes as captioned.

1. To open the network interface configuration file:

   sudonano /etc/network/interfaces

   Add the following lines (*or un-comment in*) to our file:

   auto wlan0
   allow-hotplugwlan0
   iface wlan0 inet manual
   wpa-roam /etc/wpa_supplicant/wpa_supplicant.conf

2. To open wireless configuration file:

   sudonano /etc/wpa_supplicant/wpa_supplicant.conf

   Add the following data to this wireless configuration file:

   Replace the "_SSID_" and "_WPA_SHARED_KEY_" text with actual SSID and WPA key values of our network.

   network={

ssid="_SSID_"
proto=RSN
key_mgmt=WPA-PSK
pairwise=CCMP TKIP
group=CCMP TKIP
psk="_WPA_SHARED_KEY_"
}

Appendix C

Server End Code:

Provided in the order in which it is in the source file:

1. **Be.wout.draw package:**

Drawpanel class:

```
package be.wout.draw;

import java.awt.Color;
import java.awt.Graphics;
import java.awt.Point;
import java.util.ArrayList;
import java.util.List;

import javax.swing.JPanel;

import be.wout.draw.shapes.Shape;
import be.wout.listeners.DrawPanelListener;


public class DrawPanel extends JPanel {
private static final long serialVersionUID = 1L;



public DrawPanelListener dpl;
private List<Shape> shapes = new ArrayList<>();

private Shape tmpShape;


public DrawPanel() {
//init colour
setOpaque(true);
setBackground(Color.WHITE);
```

```
//listeners
dpl = new DrawPanelListener(this );
addMouseListener(dpl);
addMouseMotionListener(dpl);


}



public void paintComponent (Graphics g){
super.paintComponent(g);
for(Shape s : shapes){
if (s != null)
s.render(g);
}


if (tmpShape != null)
tmpShape.render(g);
}


public List<Shape> getShapes() {
return shapes;
}


public void setShapes(List<Shape> shapes) {
this.shapes = shapes;
}


public Shape getTmpShape() {
return tmpShape;
}


public void setTmpShape(Shape tmpShape) {
this.tmpShape = tmpShape;
}

}
```

2. Be.wout.draw.shapes package:

a. Line class:

```
package be.wout.draw.shapes;

import java.awt.Color;
import java.awt.Graphics;

public class Line extends Shape {
```

```
	public Line(int x, int y, int x2, int y2, Color c) {
		super(x, y, x2, y2, c);

	}

	@Override
	public void render(Graphics g) {
		g.setColor(getColor());
		g.drawLine(getX(), getY(), getX2(), getY2());

	}

}
```

b. Oval class:

```
package be.wout.draw.shapes;

import java.awt.Color;
import java.awt.Graphics;

import be.wout.staticThings.StaticVars;

public class Oval extends Shape {

	public Oval(int x, int y, int x2, int y2, Color c) {
		super(x, y, x2, y2, c);

	}

	@Override
	public void render(Graphics g) {
		g.setColor(getColor());
		int w = calcWidth();
		int h = calcHeight();

		if ( w < 0 && h < 0) {
			w = Math.abs(w);
			h = Math.abs(h);
			if (StaticVars.shapeFilled)
			g.fillOval(getX2(), getY2(), w, h);
			else
				g.drawOval(getX2(), getY2(), w, h);
			}
		else if ( w < 0 && h >= 0) {
			w = Math.abs(w);
			h = Math.abs(h);
			if (StaticVars.shapeFilled)
				g.fillOval(getX2(), getY(), w, h);
			else
				g.drawOval(getX2(), getY(), w, h);
		}
		else if ( w >= 0 && h < 0) {
			w = Math.abs(w);
```

```
                h = Math.abs(h);
                if (StaticVars.shapeFilled)
                g.fillOval(getX(), getY2(), w, h);
                else
                        g.drawOval(getX(), getY2(), w, h);
        }
        else {
                if (StaticVars.shapeFilled)
        g.fillOval(getX(), getY(), w, h);
                else
                        g.drawOval(getX(), getY(), w, h);

        }
    }



    }
```

c. Pencil Class:

```
package be.wout.draw.shapes;

import java.awt.Color;
import java.awt.Graphics;
import java.awt.Point;
import java.util.ArrayList;
import java.util.List;

public class Pencil extends Shape{

    private List<Point> points = new ArrayList<>();

    public Pencil(int x, int y, int x2, int y2, Color c) {
        super(x, y, x2, y2, c);

        points.add(new Point(x, y));
    }


    @Override
    public void render(Graphics g) {
        //temp square with string

        g.setColor(getColor());

    for(Point p : points){
                g.fillOval((int)p.getX(), (int)p.getY(), 8, 8);

        }

    }

    public void addPoint(Point p){
        points.add(p);
```

```
	}


}
```

d. Shape class:

```
package be.wout.draw.shapes;

import java.awt.Color;
import java.awt.Graphics;

public abstract class Shape {

		private int x,y;
		private int x2,y2;
		private Color color;

		public Shape(int x, int y, int x2, int y2, Color c) {
			this.x = x;
			this.y = y;
			this.x2 = x2;
			this.y2 = y2;
			color = c;
		}

		public abstract void render(Graphics g);

		public int calcWidth() {

				return x2 - x;

			}

		public int calcHeight() {
				return y2 - y;
			}

		public int getX() {
			return x;
		}

		public void setX(int x) {
			this.x = x;
		}


		public int getY() {
			return y;
		}

		public void setY(int y) {
			this.y = y;
		}
```

```
	public int getX2() {
		return x2;
	}

	public void setX2(int x2) {
		this.x2 = x2;
	}

	public int getY2() {
		return y2;
	}

	public void setY2(int y2) {
		this.y2 = y2;
	}

	public Color getColor() {
		return color;
	}

	public void setColor(Color color) {
		this.color = color;
	}



}
```

e. Square class

```
package be.wout.draw.shapes;

import java.awt.Color;
import java.awt.Graphics;

import be.wout.staticThings.StaticVars;

public class Square extends Shape {

	public Square(int x, int y, int x2, int y2, Color c) {
		super(x, y, x2, y2, c);

	}

	public void render(Graphics g) {
		g.setColor(getColor());
		if (StaticVars.shapeFilled)
		g.fillRect(getX(), getY(), calcWidth(), calcHeight());
		else
			g.drawRect(getX(), getY(), calcWidth(), calcHeight());

	}
}
```

3. Be.wout.listeners package:

DrawPanelListener class:

```
package be.wout.listeners;


import java.awt.Point;
import java.awt.event.MouseEvent;
import java.awt.event.MouseListener;
import java.awt.event.MouseMotionListener;
import java.io.FileNotFoundException;
import java.io.PrintWriter;
import java.io.Serializable;
import java.util.ArrayList;
import java.util.List;

import be.wout.draw.DrawPanel;
import be.wout.draw.shapes.Line;
import be.wout.draw.shapes.Oval;
import be.wout.draw.shapes.Pencil;
import be.wout.draw.shapes.Shape;
import be.wout.draw.shapes.Square;
import be.wout.staticThings.StaticVars;


public class DrawPanelListener implements MouseListener, MouseMotionListener,
Serializable {

	private static final long serialVersionUID = 1L;
	private DrawPanel panel;
	private Shape drawShape;
	private List<Point> numberList = new ArrayList<>() ;
	public  static int[] Newpoints = new int [10000];
	public  static int[] Odpoints = new int [10000];


	public DrawPanelListener(DrawPanel dp) {
		panel = dp;

	}




	@Override
	public void mouseClicked(MouseEvent e) {


	}
	@Override
	public void mouseEntered(MouseEvent e) {
```

```
	}
	@Override
	public void mouseExited(MouseEvent e) {


	}
	@Override
	public void mousePressed(MouseEvent e) {
		drawShape = getTmpShape(e.getX(),e.getY(), 8, 8);
		panel.setTmpShape(drawShape);

		numberList.add(new Point(e.getX(),e.getY()));
		numberList.indexOf(new Point (e.getX(),e.getY()));

		//System.out.println("The start is " +size1);

	}
	@Override
	public void mouseReleased(MouseEvent e) {
		drawShape.setX2(e.getX());
		drawShape.setY2(e.getY());

		numberList.add(new Point(e.getX(),e.getY()));
		numberList.indexOf( new Point (e.getX(),e.getY()));


		List<Shape> shapes = panel.getShapes();
		shapes.add(drawShape);
		panel.setTmpShape(null);
		panel.setShapes(shapes);
		drawShape = null;

		//panel.repaint();



	}


public String sum="";

	public void mouseDragged(MouseEvent e) {
		if (drawShape instanceof Pencil) {



			((Pencil) drawShape).addPoint(new Point(e.getX(), e.getY()));
			int i = 0;

			if(i<10000)
			{
			Newpoints[i]= e.getX();
			Odpoints[i] = e.getY();
			sum= sum +"," + Newpoints[i]+ "," + Odpoints[i];
```

```
                try {
                       PrintWriter out = new PrintWriter("D:\\a.txt");

                              out.println(sum);
                              out.close();
                } catch (FileNotFoundException e1) {
                       // TODO Auto-generated catch block
                       e1.printStackTrace();
                }
                i++;

                }



        } else {
        drawShape.setX2(e.getX());
        drawShape.setY2(e.getY());

    }
        panel.setTmpShape(drawShape);
        panel.repaint();
    }


    public void mouseMoved(MouseEvent e) {


    }

    private Shape getTmpShape(int x, int y, int x2, int y2) {
        switch(StaticVars.shapeType) {
        case "square":
              return new Square(x, y, x2, y2, StaticVars.shapeColor);
        case "oval":
              return new Oval(x, y, x2, y2, StaticVars.shapeColor);
        case "pencil":
              return new Pencil(x, y, x2, y2, StaticVars.shapeColor);
        case "line":
              return new Line(x, y, x2, y2, StaticVars.shapeColor);
        default:
              return new Square(x, y, x2, y2, StaticVars.shapeColor);

        }
    }


}
```

4. **Be.wout.main package:**

MainFrame Class:

```
package be.wout.main;


import java.awt.BorderLayout;
import java.awt.Dimension;
import java.awt.Point;
import java.util.List;

import javax.swing.JFrame;
import be.wout.draw.DrawPanel;
import be.wout.menus.MainMenu;
import be.wout.menus.MainToolBar;

public class MainFrame extends JFrame {
	private static final long serialVersionUID = 1L;
	private static final Dimension SIZE = new Dimension(800, 600);


	public DrawPanel panel ;

	public MainFrame() {


		super("Rivier v1.0");
		panel = new DrawPanel();


		setLayout(new BorderLayout());

		//add panel
		add(panel, BorderLayout.CENTER);


		//add mainmenu
		MainMenu topMenu = new MainMenu(this);
		setJMenuBar(topMenu);

		//add Toolbar
		MainToolBar topTool = new MainToolBar();
		add(topTool, BorderLayout.NORTH);


		setSize(SIZE);
		setResizable(true);
		setLocationRelativeTo(null); //center frame
		setDefaultCloseOperation(EXIT_ON_CLOSE);//change to windowlistener later
		setVisible(true);
	}



	public DrawPanel getDrawPanel() {
		return panel;
	}
```

```
}
```

5. **Be.wout.menus package:**

MainMenu class:

```
package be.wout.menus;

import java.awt.Graphics2D;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.image.BufferedImage;
import java.io.File;
import java.io.IOException;
import javax.imageio.ImageIO;
import javax.swing.JFileChooser;
import javax.swing.JMenu;
import javax.swing.JMenuBar;
import javax.swing.JMenuItem;
import javax.swing.JOptionPane;
import javax.swing.filechooser.FileNameExtensionFilter;

import be.wout.draw.DrawPanel;
import be.wout.main.MainFrame;

public class MainMenu extends JMenuBar {


	private static final long serialVersionUID = 1L;

	private MainFrame mf;


	//menu file items
		private	JMenuItem newAction = new JMenuItem("New...");
		private	JMenuItem openAction = new JMenuItem("Open...");
		private	JMenuItem saveAction = new JMenuItem("Save");
		private	JMenuItem saveAsAction = new JMenuItem("Save As...");
		private	JMenuItem sendAction = new JMenuItem("Send...");
		private	JMenuItem receiveAction = new JMenuItem("Receive...");
		private	JMenuItem stopAction = new JMenuItem("Stop Session");
		private	JMenuItem printAction = new JMenuItem("Print...");
		private	JMenuItem exitAction = new JMenuItem("Exit");

			//menu edit items
		private	JMenuItem undoAction = new JMenuItem("Undo");
		private	JMenuItem redoAction = new JMenuItem("Redo");
		private	JMenuItem cutAction = new JMenuItem("Cut");
		private	JMenuItem copyAction = new JMenuItem("Copy");
		private	JMenuItem pasteAction = new JMenuItem("Paste");

	public MainMenu(MainFrame mf) {
```

```
        this.mf = mf;

        JMenu fileMenu = new JMenu("File");
        JMenu editMenu = new JMenu("Edit");

        //add menus
        this.add(fileMenu);
        this.add(editMenu);



        //add menu items to menu

        //file
        fileMenu.add(newAction);
        fileMenu.add(openAction);
        fileMenu.add(saveAction);
        fileMenu.add(saveAsAction);
        fileMenu.addSeparator();
        fileMenu.add(sendAction);
        fileMenu.add(receiveAction);
        fileMenu.add(stopAction);
        fileMenu.addSeparator();
        fileMenu.add(printAction);
        fileMenu.addSeparator();
        fileMenu.add(exitAction);

        //edit
        editMenu.add(undoAction);
        editMenu.add(redoAction);
        editMenu.addSeparator();
        editMenu.add(cutAction);
        editMenu.add(copyAction);
        editMenu.add(pasteAction);

        initFileActions();
        initEditActions();
}



private void initFileActions() {

        newAction.addActionListener(new ActionListener() {

                public void actionPerformed(ActionEvent e) {

                }
        });

        openAction.addActionListener(new ActionListener() {

                public void actionPerformed(ActionEvent e) {
```

```
		}
	});

	saveAction.addActionListener(new ActionListener() {

		public void actionPerformed(ActionEvent e) {

		}
	});

	saveAsAction.addActionListener(new ActionListener() {

		public void actionPerformed(ActionEvent e) {
			DrawPanel p = mf.getDrawPanel();
			int w = p.getWidth();
			int h = p.getHeight();

			BufferedImage bi = new BufferedImage(w, h,
BufferedImage.TYPE_INT_RGB);
			Graphics2D g2d = bi.createGraphics();
			p.paint(g2d);

			JFileChooser chooser = new JFileChooser();

			chooser.addChoosableFileFilter(new
FileNameExtensionFilter("PNG (.png)", "png"));
			chooser.addChoosableFileFilter(new
FileNameExtensionFilter("JPG (.jpg)", "jpg"));
			chooser.addChoosableFileFilter(new
FileNameExtensionFilter("JPEG (.jpeg)", "jpeg"));

	chooser.setFileFilter(chooser.getChoosableFileFilters()[1]);

			int r = chooser.showSaveDialog(mf);

			if (r== JFileChooser.APPROVE_OPTION) {
			File out = chooser.getSelectedFile();

			String ext =
out.getName().substring(out.getName().lastIndexOf(".") + 1);

			if (ext.equalsIgnoreCase("png") ||
ext.equalsIgnoreCase("jpg") || ext.equalsIgnoreCase("jpeg")) {
			try {
			ImageIO.write(bi, ext, out);
		} catch (IOException e1) {

			e1.printStackTrace();
		}
			}
			else {
				JOptionPane.showMessageDialog(null, "Wrong
extension", "Error", JOptionPane.ERROR_MESSAGE);
			}
	}
```

```
}

});

	sendAction.addActionListener(new ActionListener() {

		public void actionPerformed(ActionEvent e) {

			//	t.startRunning();
			}

		});

	receiveAction.addActionListener(new ActionListener() {

		public void actionPerformed(ActionEvent e) {
		//	s.startRunning();
			}
	});

	stopAction.addActionListener(new ActionListener() {

		public void actionPerformed(ActionEvent e) {

		}
	});

	printAction.addActionListener(new ActionListener() {

		public void actionPerformed(ActionEvent e) {

		}
	});

	exitAction.addActionListener(new ActionListener() {

		public void actionPerformed(ActionEvent e) {
		System.exit(0); //check if saved in future
		}
	});
}

private void initEditActions() {
	undoAction.addActionListener(new ActionListener() {

		public void actionPerformed(ActionEvent e) {

		}
	});

	redoAction.addActionListener(new ActionListener() {

		public void actionPerformed(ActionEvent e) {

		}
```

```
		});

		cutAction.addActionListener(new ActionListener() {

			public void actionPerformed(ActionEvent e) {

			}
		});

		copyAction.addActionListener(new ActionListener() {

			public void actionPerformed(ActionEvent e) {

			}
		});

		pasteAction.addActionListener(new ActionListener() {

			public void actionPerformed(ActionEvent e) {

			}
		});


	}

}
```

MainToolbar class:

```
package be.wout.menus;

import java.awt.Color;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.ItemEvent;
import java.awt.event.ItemListener;

import javax.swing.JButton;
import javax.swing.JCheckBox;
import javax.swing.JColorChooser;
import javax.swing.JComboBox;
import javax.swing.JLabel;
import javax.swing.JToolBar;

import be.wout.staticThings.StaticVars;

public class MainToolBar extends JToolBar{
	private static final long serialVersionUID = 1L;

	private JComboBox<String> combo;
	private JLabel comboLabel = new JLabel("Tool: ");

	private JLabel colorChooser = new JLabel("Color: ");
	private JButton colorBut = new JButton("      ");
```

```
	private JCheckBox fillBox = new JCheckBox("Filled:  ");

	private Color c = StaticVars.shapeColor;
	public MainToolBar() {
		combo = new JComboBox<>();
		combo.addItem("Square");
		combo.addItem("Oval");
		combo.addItem("Pencil");
		combo.addItem("Line");

		setOpaque(true);
		colorBut.setForeground(c);
		colorBut.setBackground(c);

		fillBox.setSelected(false);

		//action Listener
		combo.addActionListener(new ActionListener() {
			@Override
			public void actionPerformed(ActionEvent e) {
			switch (combo.getSelectedIndex()) {
			case 0:
				StaticVars.shapeType = "square";
				break;
			case 1:
				StaticVars.shapeType = "oval";
				break;
			case 2:
				StaticVars.shapeType = "pencil";
				break;
			case 3:
				StaticVars.shapeType = "line";
				break;
			default:
				StaticVars.shapeType = "square";
			}

			}
		});

		colorBut.addActionListener(new ActionListener() {
			@Override
			public void actionPerformed(ActionEvent e) {
				c = JColorChooser.showDialog(MainToolBar.this, "Choose
your color", c);
				StaticVars.shapeColor = c;
				colorBut.setForeground(c);
				colorBut.setBackground(c);
			}
		});

		fillBox.addItemListener(new ItemListener() {

			@Override
```

```
                public void itemStateChanged(ItemEvent arg0) {
                StaticVars.shapeFilled = fillBox.isSelected();

                }
        });

        this.add(comboLabel);
        this.add(combo);
        this.add(fillBox);
        this.add(colorChooser);
        this.add(colorBut);

        }

}
```

6. **Be.wout.server package:**

```
        package be.wout.ServerCode;

        import java.awt.BorderLayout;
import java.awt.Point;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.MouseListener;
import java.awt.event.MouseMotionListener;
import java.io.BufferedInputStream;
import java.io.BufferedOutputStream;
import java.io.BufferedReader;
import java.io.EOFException;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.FileReader;
import java.io.IOException;
import java.io.InputStream;
import java.io.ObjectOutputStream;
import java.io.OutputStream;
import java.net.ServerSocket;
import java.net.Socket;
import java.util.ArrayList;
import java.util.List;

        import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.JScrollPane;
import javax.swing.JTextArea;
import javax.swing.JTextField;
import javax.swing.SwingUtilities;
import javax.swing.border.EmptyBorder;

        import org.w3c.dom.events.MouseEvent;
```

```
	import be.wout.draw.DrawPanel;
import be.wout.draw.shapes.Pencil;
import be.wout.draw.shapes.Shape;
import be.wout.listeners.DrawPanelListener;
import be.wout.main.MainFrame;


	public class Server extends JFrame {

		private static final long serialVersionUID = 1L;

		private static ObjectOutputStream output, output1;
		private JTextField userText;
		private ServerSocket server;
		private Socket connection;
		private JTextArea statusWindow;
		private FileOutputStream fs;
		public static DrawPanel panel;
		public JPanel contentPane;
		public boolean newmPress;
		FileInputStream fis = null;
	    BufferedInputStream bis = null;
	    OutputStream os = null;




		public static void main(String [] args) throws IOException{

			//List <Point> globalnumberList = new ArrayList<>();
			new MainFrame();



				Server sally = new Server();

	sally.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);


			sally.startRunning();


		}



		public Server() {

			super("Server Status");


			contentPane = new JPanel();
	contentPane.setBorder(new EmptyBorder(5, 5, 5, 5));
```

```
        contentPane.setLayout(new BorderLayout(0, 0));
        setContentPane(contentPane);

                    userText = new JTextField();
                    userText.setEditable(false);

                    add(userText, BorderLayout.NORTH);
                    statusWindow = new JTextArea();
                    statusWindow.setEditable(false);
                    add(new JScrollPane(statusWindow));
                    setSize(400,300);
                    setVisible(true);


                }




                public void startRunning() {
                    try {
                            server = new ServerSocket(6789, 100); //ServerSocket(int
port, int backlog) ,, backlog= how many clients can connect ,, int port = client port
no
while (true){
try {
//connect and send
waitForConnection(); //wait for clients to connect
setupStreams(); //sets up output and input stream
whileSending();


} catch (EOFException eofException) {
showMessage("\n Server ended the connection!");
} finally {
closeRiver();
}
}
} catch (IOException ioException) {
ioException.printStackTrace();
}

}

                //wait for connection and display connection information
                private void waitForConnection() throws IOException{
                    showMessage("Waiting for clients to connect...");
                    connection = server.accept(); //variable connection is connected
if there is a connection
                    showMessage("Connection made to " +
connection.getInetAddress().getHostName());

                }
```

```
            //get the streams to send and receive data

            private void setupStreams() throws IOException{
                  //output = new ObjectOutputStream(connection.getOutputStream());
//creating pathway that allows us to connect to another is connected to.
                  //output.flush();//leftover data to that is cleaned off, bytes of
info that are unnecessary



                  showMessage("\n Streams are now set yup! \n");
            }



            //during streaming session


            private void whileSending() throws IOException {
                   File myFile = new File ("D:\\a.txt");
      byte [] mybytearray  = new byte [(int)myFile.length()];
      fis = new FileInputStream(myFile);
      bis = new BufferedInputStream(fis);
      bis.read(mybytearray,0,mybytearray.length);
      os = connection.getOutputStream();
      System.out.println("Sending " + "D:\\a.txt" + "(" + mybytearray.length + "
bytes)");

      os.write(mybytearray,0,mybytearray.length);
      os.flush();
      System.out.println("Done.");


                        }







            private void closeRiver(){
                  showMessage("\n Closing connections..\n");
            //    ableToDraw(false);
                  try {

                        //output.close();
                        if (bis != null) bis.close();
            if (os != null) os.close();
            if (connection!=null) connection.close();
```

```
			} catch (IOException ioException) {
				ioException.printStackTrace();
			}
		}


		private void showMessage(final String text) {
			SwingUtilities.invokeLater(
					new Runnable() {

						@Override
						public void run() {
							statusWindow.append(text);

						}
					}
					);
		}}
```

7. **Be.wout.staticThings package:**

```
package be.wout.staticThings;
import java.awt.Color;

publicclass StaticVars {

	publicstatic String shapeType = "square";
	publicstatic Color shapeColor = Color.BLACK;
	publicstaticbooleanshapeFilled = false;
}
```

Appendix D

Client End Code:

```
package be.wout.ClientCode;

import java.io.*;
import java.net.*;
import java.util.StringTokenizer;
import java.awt.*;
import javax.swing.*;

import be.wout.draw.shapes.Shape;

publicclass Client extends JFrame {
privatestaticfinallongserialVersionUID = 1L;

privatestatic JTextArea statusWindow2;
private JTextField userText;
privatestatic ObjectInputStream input;
public Shape shapes;
private ObjectInputStream input1;
private String serverIP;
privatestatic Socket connection;
private FileInputStream fis;
publicstatic Drawer panel2;
intbytesRead;
intcurrent = 0;
FileOutputStream fos = null;
BufferedOutputStream bos = null;
publicfinalstaticintFILE_SIZE = 6022386;


publicstaticint [] Newpoints = newint [3000];
publicstaticint [] Odpoints = newint [3000];

//constructor
public Client(String host){
```

```
//give it ip address of the server
super("Client status");
serverIP = host;

userText = new JTextField();
userText.setEditable(false);

add(userText, BorderLayout.NORTH);
statusWindow2 = newJTextArea();
add(new JScrollPane(statusWindow2));
setSize(400, 300);
setVisible(true);
}


publicstaticvoid main (String [] args) throws IOException{



Client charlie;
charlie = new Client("127.0.0.1");
charlie.startRunning();

}

//connect to server
publicvoid startRunning() throws IOException{
//connectToServer();
//setupStreams();

try {
//connect and receive
//sets up output and input stream

// while(true){
connectToServer();
setupStreams();
whileReceiving();
new MainFrame2();


//}

} catch (EOFException eofException) {
showMessage("\n Client ended the connection!");
} catch (IOException ioException) {
ioException.printStackTrace();
} catch (ClassNotFoundException e) {
// TODO Auto-generated catch block
e.printStackTrace();
}

finally {
//closeRiver();
}
```

```
}

privatevoidcloseRiver(){
showMessage("\n Closing connections..\n");
//     ableToDraw(false);
try {

connection.close();

} catch (IOException ioException) {
ioException.printStackTrace();
}
}


privatestaticvoid showMessage(final String string) {
SwingUtilities.invokeLater(
new Runnable() {

@Override
publicvoid run() {
statusWindow2.append(string);//use Popupbox instead

}
}
);

}


//connect to server
privatevoid connectToServer() throws IOException{
showMessage("Attempting connection.... \n");
connection = new Socket(InetAddress.getByName(serverIP), 6789); // inetaddress
returns an InetAddress object that contains the IP address of the remote machine.
showMessage("Connected to:" + connection.getInetAddress().getHostName());
//gethostname returns ip address as string
}

//set up to receive messages
privatevoid setupStreams() throws IOException{
//input = new ObjectInputStream(connection.getInputStream());


showMessage("\n Streams are now set up!\n");
}

//while receiving from server


privatevoid whileReceiving() throws IOException, ClassNotFoundException,
EOFException{

try {
byte [] mybytearray  = newbyte [FILE_SIZE];
```

```
InputStream is = connection.getInputStream();
fos = new FileOutputStream("E:\\b.txt");
bos = new BufferedOutputStream(fos);
bytesRead = is.read(mybytearray,0,mybytearray.length);
current = bytesRead;

do {
bytesRead =
is.read(mybytearray, current, (mybytearray.length-current));
if(bytesRead>= 0) current += bytesRead;
} while(bytesRead> -1);

bos.write(mybytearray, 0 , current);
bos.flush();
System.out.println("File " +
" downloaded (" + current + " bytes read)");

} catch (Exception e) {
e.printStackTrace();
}


 BufferedReader br = newBufferedReader(new FileReader("E:\\b.txt"));
 String line = br.readLine();
br.close();
int i=0;
int [] narray = newint [6000];
StringTokenizer tokenizer = newStringTokenizer(line, ",");
while(tokenizer.hasMoreTokens()){

narray[i++] = Integer.parseInt(tokenizer.nextToken());
//System.out.println(narray[i-1]);
}
for(int m =0, j=0, k=0; m<narray.length; m++){
if (m%2==0) {
Newpoints[j++]= narray[m];
System.out.println("x is "+ Newpoints[j-1]);
} else {
Odpoints[k++]= narray[m];
System.out.println("y is "+ Odpoints[k-1]);
}

             }

publicstaticclass MainFrame2 extends JFrame    {
privatestaticfinallongserialVersionUID = 1L;
privatestaticfinal Dimension SIZE = new Dimension(800, 600);
publicstatic Drawer sapanel ;
public MainFrame2( ) {

super("Rivier v1.0");
sapanel = new Drawer();
setLayout(new BorderLayout());
//add panel
add(sapanel, BorderLayout.CENTER);
```

```
setSize(SIZE);
setResizable(true);
setLocationRelativeTo(null); //center frame
setDefaultCloseOperation(EXIT_ON_CLOSE);//change to windowlistener later
setVisible(true);
}
}

publicstaticclass Drawer extends JPanel{
privatestaticfinallongserialVersionUID = 1L;
privatestaticfinal Dimension SIZE2 = new Dimension(800, 600);
public Drawer() {
//new JPanel();
setBackground(Color.WHITE);
setSize(SIZE2);
}


publicvoid paintComponent (Graphics g){
super.paintComponent(g);
g.setColor(Color.black);

int i,j;
for ( i =0, j = 0; i<1000; i++, j++ ){
g.fillOval(Newpoints[i] , Odpoints[j], 8, 8);

}

}


}


}
```