Android App for Scribbler Robot

Master's Project

Department of Computer Science

Montclair State University

Nirajan Thapa

Advisor: Dr. Stefan Robila

May 2015

Table of Contents

LIST OF FIGURES

ABSTRACT

The goal of the project is to develop an Android application to control a Scribbler robot. Scribbler is an intelligent robot equipped with a BASIC Stamp 2 microcontroller and multiple sensors. It can be programmed to interact with its environment and perform many actions like navigation and taking picture. The app can be used by students as well as faculty members in the field of Robotics. The project utilizes the app to demonstrate various capabilities of the Scribbler robot. The app offers a rich user-interface that allows the users to experiment and learn more about interacting with robots. This easy-to-use app attracts the interest of students towards the ever expanding robotics field. The project will also help in gaining a deeper understanding of the capabilities of Android smartphones and their effectiveness in enhancing robot manipulation techniques.

ACKNOWLEDGMENT

This project is made possible through the help and support from everyone including parents, teachers, family and friends.

First I would like to thank Dr. Stefan Robila for not only providing me with an opportunity to work with him, but also for his support and encouragement. His mentorship and guidance contributed immensely to the fruition of this project.

I would also like to extend my gratitude towards Dr. Jerry A. Fails and Dr. George Antoniou for their support and guidance throughout my studies and project at Montclair State University.

Lastly, I would like to thank the faculty of the Department of Computer Science for their support during my Master's program.

1. INTRODUCTION

The robotics field is constantly evolving. There are new innovations every day that offer an insight to the huge potential of human-robot relationship. The researchers at Oregon State University recently field-tested successful locomotion abilities of a two-legged robot. Their experiments with the “ATRIAS” robot demonstrated that it could move around keeping its balance and withstand mild blows from a rubber ball. The robot walked in the grass, up and down hill and other varying terrain at a normal walking speed [13]. In a similar research with a walking robot, Beilefeld University have developed a software architecture that endowed the robot Hector with a simple form of consciousness. This software allows the robot to see himself as others see him, providing him with reflexive consciousness [14]. Large companies are also investing in robotics. Separately from pursuing self-driving car projects, Google also recently purchased Boston Dynamics, a company known for the design and development of biped and quadruped robots [18], while Amazon seeks new faster package delivery options through quadropter bots [19] . Autonomous robotics, i.e. robotics fields that aim to endow the robots with a high degree of autonomy continue, thus to expand.

To get a high degree of autonomy one has to consider implementing complex machine learning and data processing algorithms while supporting data collection from a variety of sensors. There are several types of software available to interact with robots. They are used to program the robots to perform tasks and are often provided with interfaces that allow users to program in high level languages. However, in robotics there is no one common language such as Java or C. Each robot manufacturer

develops their own specialized language to interact with their robots. Some of the industrial robot programming languages available are RAPID (ABB), Inform (Yaskawa), AS (Kawasaki), URScript (Universal Robots) and many more [15]. Most of these programming languages have a steep learning curve and they are mainly used with desktop based software applications.

The Finch is one of the robots available for computer science education designed by Carnegie Mellon's CREATE lab. It is designed to support an engaging introduction to the art of programming. The features on Finch includes Light, temperature, and obstacle sensors, Accelerometers, Motors, Buzzer, LED, Pen mount and USB port. It supports several programming languages such as Java, Javascript, Scala, C++, C#, Calico and many others [20].

Arduino is another robotics platform that designs devices that can sense and control the physical world. The computing platform is based on a simple microcontroller board and a development environment for writing software. The Arduino devices can be stand-alone or interact with software running on the computer such as Flash, Processing or MaxMSP. Arduino Uno is one of the microcontrollers developed based on this platform. It has 14 digital input/output pins, 6 analog inputs, 16 MHz resonator, a USB connection, a power jack and a reset button. The software development environment for Arduino consists of a text editor, message area, text console and a toolbar with buttons of common functions. The integrated development environment (IDE) itself is written in Java, derived from the IDE for Processing programming language and Wiring projects. The Arduino programs are written in C or C++ [21].

In an effort to bring autonomous robotics into K-12 and college classrooms, several simpler platforms were introduced in recent years. Scribbler (described below) is one of them, and through the effort of various academic groups can now be programmed in a variety of languages. Yet, simple interaction with the robot is not easy and requires the user to download and install an IDE, install libraries and then type commands in a programming language. The key inspiration behind the project is to create an application that will demonstrate an easy interface to communicate with the robot, and generate incentives for people to explore more possibilities in the field. The app can be downloaded instantly from the Android Google Play Store and can be used right away to interact with Scribbler. This application can be used by professors for demonstration purposes in the classrooms. Through the app the students can have a tangible experience with the robot during the early learning stages instead of being limited only to theory.

This report is organized as follows.

Chapter 1 provides a brief introduction of the Robotics field.

Chapter 2 states structural and functional details of a Scribbler robot, the scope of the project and the motivation behind its completion.

Chapter 3 lists out all the functional, system and deployment requirements of the project. It also explains the core Android architecture.

Chapter 4 explains the implementation of various user-interface Android components in the app.

Chapter 5 provides a detailed look into the programming structure of the app.

Chapter 6 offers necessary guidance to assist first time users.

Chapter 7 describes the conclusions drawn from the project.

Chapter 8 talks about the future work.

Chapter 9 lists all the references materials.

2. BACKGROUND

2.1. Scribbler

Scribbler (pictured in Fig. 1) is a fully programmable robot that is suited for a range of programming skills. It is commercialized by Parallax Inc (http://www.parallax.com/) as an inexpensive robotics platform. The base platform includes a powerful quad core CPU, light sensors, independent DC wheel motors, stall sensors and wheel encoders only). It also has an integrated pen port, speaker and hacker port for connection to external sensors [1].

Figure 1 - Image of a Scribbler Robot

Scribbler robots are further enhanced with additional sensors through a Fluke board developed by the Institute for Personal Robots in Education (IPRE) a joint venture between Georgia Tech and Bryn Mawr College (http://www.roboteducation.org/). The Fluke is a small electronic board that contains wireless Bluetooth, camera, IR (infrared) sensors, LEDs and an ARM microprocessor. Figure 2, provided by the manufacturer, is an image of the board with the various components' location emphasized. Fluke has an

IR emitter that transmits infrared signals and an IR Receiver which receives the generated infrared signals. It also includes a USB Host Port, a Memory Card, an Auxiliary Power Connection and Servo Motor Expansion Port. It has a Battery Power Indicator and a CPU Load Indicator as well [17]. Fluke does not have its own power or source or CPU and instead relies on Scribbler's.

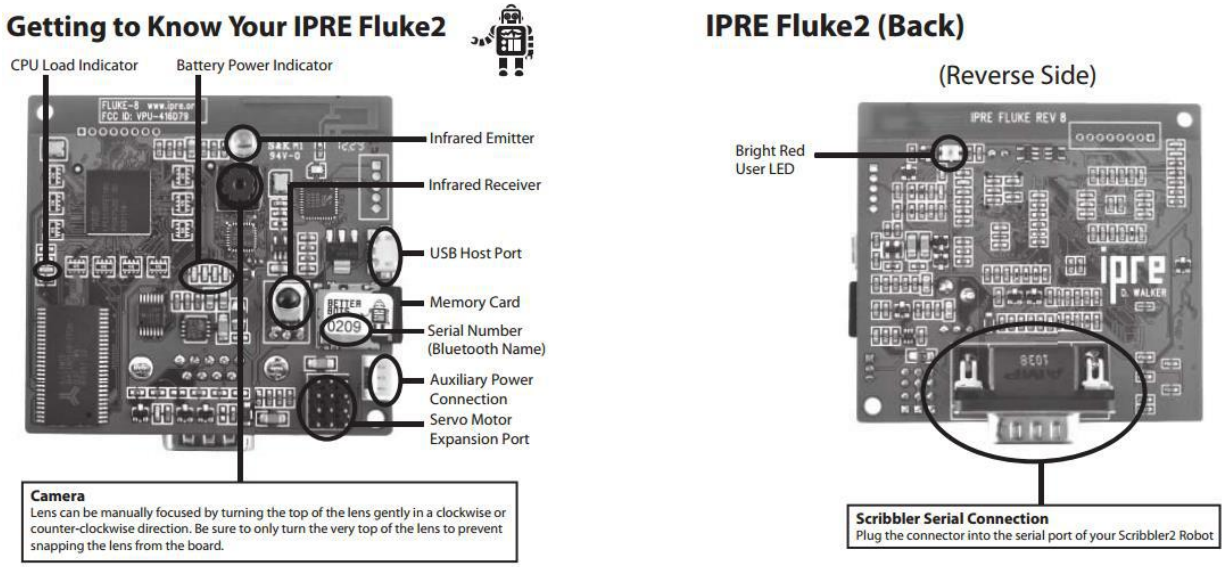

Figure 2 - Image of IPRE Fluke [17]

The software library used to control the Scribbler is called Myro (for My Robot). Myro is a framework for programming robots, which is written in Python language. The most common software used to communicate with the Scribbler is Calico software. Calico supports multiple programming languages and includes the required Myro library [16].

Combined with Scribbler, Fluke is capable of receiving simple commands and send back IR readings or camera images. It sits between the software running the Myro API and Scribbler. It intercepts byte-code commands and returns appropriate

responses. The Fluke handles camera, obstacle and brightness commands. The other commands and their responses are relayed to and from Scribbler through the Fluke.

The simplicity of the robot makes it a good candidate to be used for educational programs. At Montclair it is used in the introductory robotics courses.

2.2. Programming Environment

The commercial Scribbler robot comes with hardware and software configured for programming. The most common software used to control the robot is the Myro library. In addition, a newer environment, Calico software, which supports multiple programming languages, includes the Myro library. In order to start interacting with Scribbler through a computer, first you need to install Calico. Next, you have to establish a Bluetooth connection between the computer and Scribbler. Once these initial set ups are accomplished, one can start programming.

Python is the most common language used for programming Scribbler. The Myro library needs to be imported before writing any commands to Scribbler. This can be achieved by entering the following command in Python shell.

python> from Myro import *

This informs the shell that the Myro library is to be imported and made available for use. After that you need to initiate the Bluetooth connection with Scribbler by issuing the following command.

python> initialize("comX")

where X is the port number in use for communication with Scribbler

Once a successful connection is established, you can start sending commands to Scribbler using Myro functions. Myro has several functions to control the movement of Scribbler, retrieve values of all the sensors and even take picture through the camera on the Fluke.

Movement Functions

The following functions are used to control the movement of Scribbler. In these commands, SPEED can be a value between -1, 0 and 1. The stop() command stops all movement.

forward(SPEED), forward(SPEED, SECONDS)

backward(SPEED), backward(SPEED, SECONDS)

turnLeft(SPEED), turnLeft(SPEED, SECONDS)

turnRight(SPEED), turnRight(SPEED, SECONDS)

The robot runs on 6 AA batteries. Myro provides a function to get the internal battery-level, which can be accomplished by entering the command getBattery(). This function returns a value between 0 and 9, where 0 represents the battery being completely drained while 9 represents fully charged batteries.

Sensors Functions

Myro also provides several functions to fetch data from each sensor device.

- **Picture**

The following commands are associated with taking pictures through the camera.

python> p= takePicture()

python> p=takePicture(“gray”)

The first command above takes a picture and returns a colored picture object. The second command takes a grayscale picture.

python> savePicture(p, “test-picture.jpg”) command can be issued to save the picture. This function takes a picture object as a first parameter and a filename as the second parameter. The picture is saved in the Calico installation directory in the computer.

- **Light**

The following functions are available to obtain light sensors.

getLight()

returns all the three values of light sensors.

getLight(position)

returns the value of light at that position. The position can be 0, 1 and 2 or ‘left’, ‘center’ and ‘right’.

The returned values are in the range of 0 and 5000, where low values indicate bright and high values indicate darkness.

- **Obstacle**

The following functions retrieve data from the two sets of infrared (IR) sensors on the front of the robot, and three additional IR sensors on the Fluke.

getIR()

returns a list for all IR sensors

getIR(position)

returns the data for the IR sensor in that position. The position can either be 0 and 1 or ‘left’ and ‘right’. The value returned by the sensors is either a 0 or a 1. A value of 1 indicates there is no obstacle in close proximity whereas value of 0 indicates a presence of an obstacle.

2.3. Motivation

The study of robotics requires constant experiments with robots to learn more about their capabilities and test new discoveries. Even inside a classroom setting, study of introduction to robotics requires regular interaction with robots to supplement the theory behind it. There are many software available that allow you to program and manipulate robots. In the case of Scribbler, although it is manufactured to offer an easy configuration for programming, it still requires a good understanding of several components associated with it. The process requires the installation of Calico software on a computer and a basic knowledge of Python language in order to issue commands to Scribbler to perform certain functions. The problem with this approach is that people with no programming background who are interested in learning about Scribbler will not be able to experience any interaction without learning the usage of Python and Calico. This limitation can deter people away from motivating themselves to get involved in the field of robotics.

With the increasing significance of mobile computing in the field of technology, it is essential to be able to use mobile devices to perform communication with the robots. These days people are able to perform complicated technology related actions through their smartphones without having any prior knowledge of computer programming. They

are able to control TV, mirror phone on TV or control house thermostat through the simple use of an app. This capability of providing people a platform to perform technical work with ease can be used to attract more people and more interest towards robotics. There is a great potential in making use of mobile technology to lead the innovation in the robotics field. Therefore, creating a mobile app that allows people to interact with Scribbler with a touch of a button is vital.

2.4. Scope

The main objective of this project is to develop an Android application that can be used to interact with the Scribbler robot. This is intended to enhance the students as well as faculty experience while teaching robotics. The app allows teachers to demonstrate Scribbler's capabilities with ease through the use of the app. It saves an immense amount of time since you would not be required to write a lot of codes to do a simple demonstration in class. They can save previously used execution samples on the app and easily refer to them in the future. With the app, the students can instantly experience interacting with Scribbler.

The app along with the source code will be made available to the Montclair State University's Department of Computer Science which allows students to make further progress and enhance the app to include more capabilities.

2.5. Comparison of Other Available Apps

Even though there has been a tremendous increase in the number of apps available in all fields, there are only a handful of apps designed for the Scribbler robot. This shows

that there is still very limited contributions towards making use of the Android platform in the robotics field. The two Android apps available for Scribbler on the Google Play Store are described below.

Scribby: It is an Android app listed on the play store as developed by Karl Kim. The app provides four tabs for navigation and allows users to move the robot in any direction using the arrows on the first tab named Drive. The second Photo tab provides the option to take a picture, the third tab displays an image of a keyboard, where a user can press buttons to produce sound of various intensity. The last Status tab provides basic information on the sensor values of the Scribbler [2].

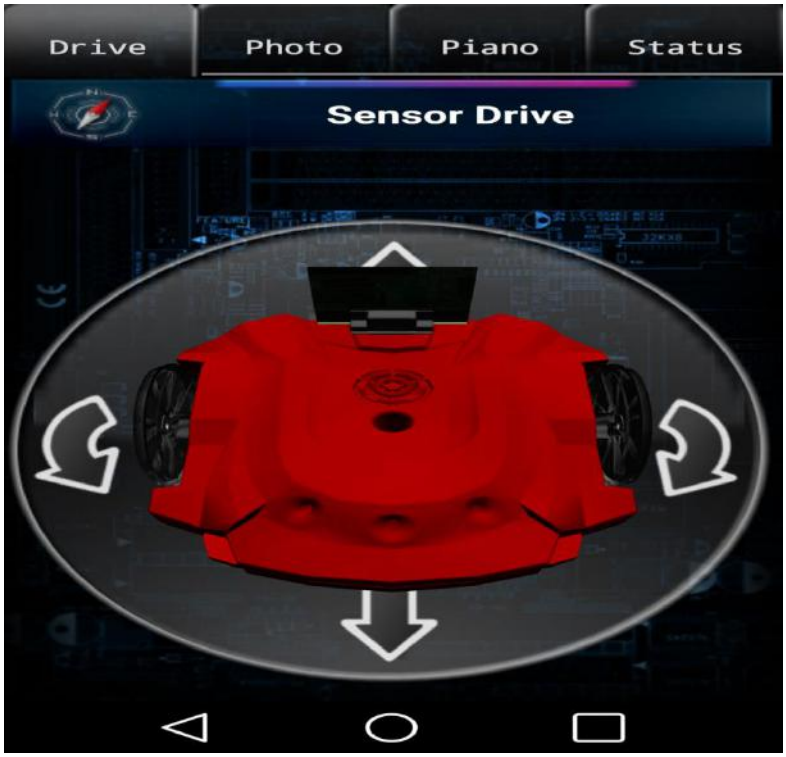

Figure 3 - Image of Scribby app home page

Although the app offers the use of much basic functionality associated with Scribbler, it comes along with several issues. As soon as you open the app, it offers you the option to go to the main UI page or discover the surrounding Bluetooth devices. If

you skip the device discovery process and go the main UI page, then there is no option to start finding devices from that page itself. You will be forced to go back to the starting screen and start the whole process again. There is also a significant delay in switching from one tab to another, which could be because of a poor Android implementation. It seems to be doing a lot of work each time you switch the tab which is freezing the app for a few seconds. The last available app update was on April 8, 2013. The slow responsiveness of the app could also be a result of not being optimized for the newest Android versions. The photo feature of the app is not working at all. When you start the image capturing process, it displays a progress for the operation but after a while it stops because of an error.

Scribdroid: This is another app available on the Play Store for Scribbler listed as developed by Sahhm. It also offers tabular navigation system like Scribby, which includes controller, sensor values and photo gallery. This too comes with very similar problems. The app has not been updated since January 2013 [3].

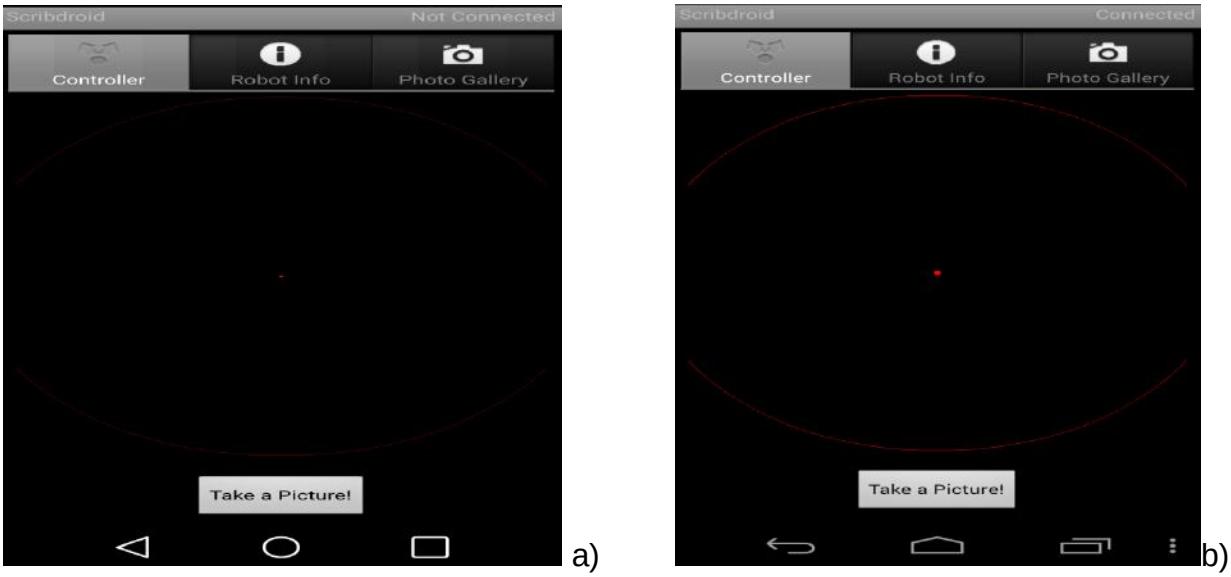

Figure 4 - Image of Scribdroid app a) (Android 5.0), b) (Android 4.4)

When you use Scribdroid on the latest Android Lollipop platform version, then the process of locating the option to find Bluetooth devices is very difficult. By installing the app on earlier Android platforms, I was able to find a menu bar at the bottom right of the page which includes the options for connecting to Scribbler. In order to open this menu bar in the newest platform you have to tap and hold the Android device's Recents button. Since there are no proper guidelines or in-app help, it makes a very poor user-friendly experience.

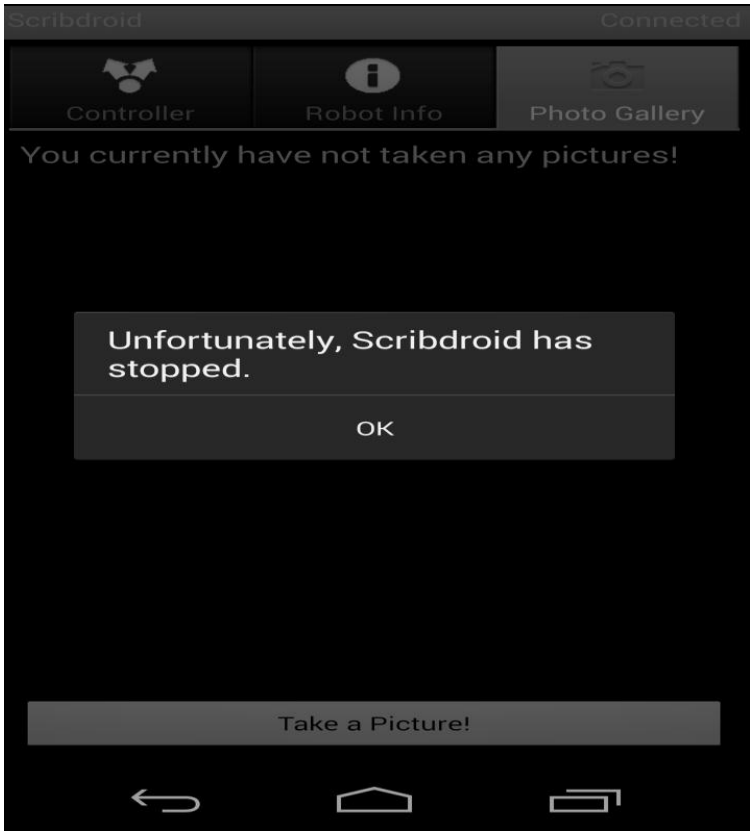

Figure 5 - Image of Scribdroid Not Responding

This app fails to take a picture as well. Once you start the image capturing process, the app stops responding and will be forced to close it.

3. APPLICATION REQUIREMENTS AND ARCHITECTURE

3.1 Functional Requirements

The functional requirements of the application are listed below.

1. *Connectivity:*

    a. It should be able to enable Bluetooth available on the device and search for remote Bluetooth devices.

    b. It should query the nearby Bluetooth devices and display them in a list to the user. Upon selection of a Bluetooth enabled Scribbler, it should be able to establish a connection.

2. *Robot Control:*

    a. It should be able to move the robot using the user-interface buttons provided on the swipeable tabular navigation.

    b. It should read the robot sensor's values and display them

    c. It should be able to take a picture through the camera mounted on the Fluke.

3. *Image Management:*

    a. It should be able to save the picture on the Android device.

    b. It should be able to scan the internal storage of the phone and display all the images associated with the application on the screen.

4. *Command Handling:*

    a. It should be able to run a list of commands in succession.

    b. It should be able to save a list of commands to a file on the device.

It should be able to open up the saved commands file, and list all the commands in the file on the screen.

3.2 Systems Requirements

The system requirements of the application are listed below.

- Mobile device running Android mobile operating system
- Scribbler Robot
- Tools: Android Software Development Kit (SDK), Android Support Library
- Eclipse/Android Studio IDE

3.3 Deployment Requirements

The application is published on the Google Play Store and is available for download on all Android devices. All future implementations can be pushed to the source and submit the updated APK file to the Play Store to publish the updated version.

3.4 Android Architecture

All Android devices run on the Android operating system, which is a Linux-based kernel system. Figure 6 provides an overview of the architecture.

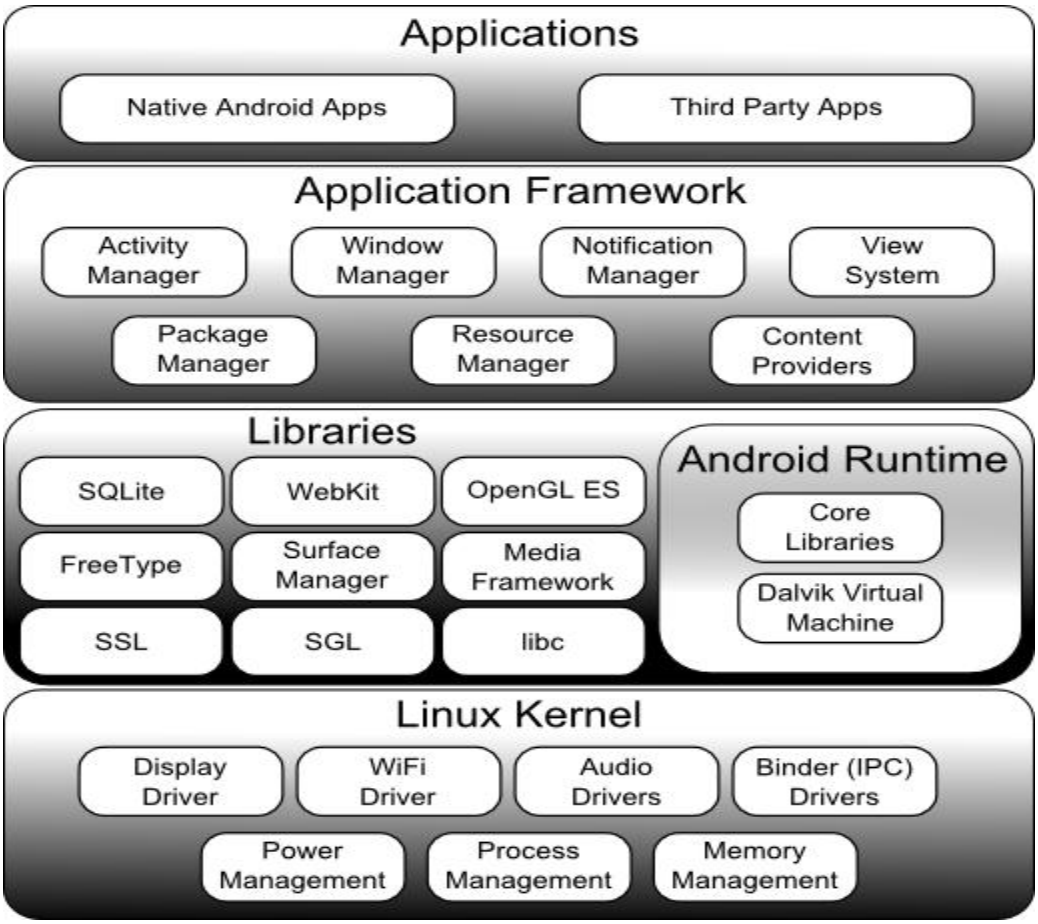

Figure 6 - Android Architecture [11]

The Android architecture consists of four layers: Linux kernel, Libraries and Android Runtime, Application Framework and Applications [4]. The applications layer consists of all the applications including system apps such as contacts, SMS client, calendar, maps and others. The application framework provides the development platform that developers can use to build applications. The Android Runtime consists of Java Core Library and Dalvik virtual machine. The Library provides access to core Java library with most functions. The Android includes set of C/C++ system libraries, which serve as a connection between application framework and Linux kernel. These libraries can be utilized by developers through the Android Native Development Kit (NDK). The kernel

system provides the innermost layer based on Linux 2.6. All the important system operations like internal storage, process management, internet protocol, and other core services are based on Linux kernel [4].

There have been several released versions of Android since November 2007. The most recent released versions are Eclair, Froyo, Gingerbread, Honeycomb, IceCream and Lollipop.

4. APPLICATION DESIGN

4.1. Material Design Pattern

The app is designed by implementing the latest design pattern by Google known as Material Design which was announced at the Google I/O conference on June, 2014. Material design is a comprehensive guide for visual, motion and interaction design across all platforms. It uses grid-based layouts, responsive animations and transitions, padding, and depth effects such as lighting and shadows. The goal of the Material Design is to be presented as a visual language that synthesizes classic principles of good design with the innovation and possibility of technology and science. This design principle is developed to offer a single underlying system that allows for a unified experience. Most of the Google's Android mobile apps such as Gmail, Google Play Store, Google Maps and many other apps have already incorporated the design pattern. The new Material design can be used in Android version 2.1 and up through the use of Android v7 appcompat support library [12].

4.2. Graphical User Interface

This app provides a simple, user-friendly design following the Material Design principles. There are many different types of Android devices with various configurations, display densities, and customization options, which makes it very important to implement rich and adaptive user interfaces.

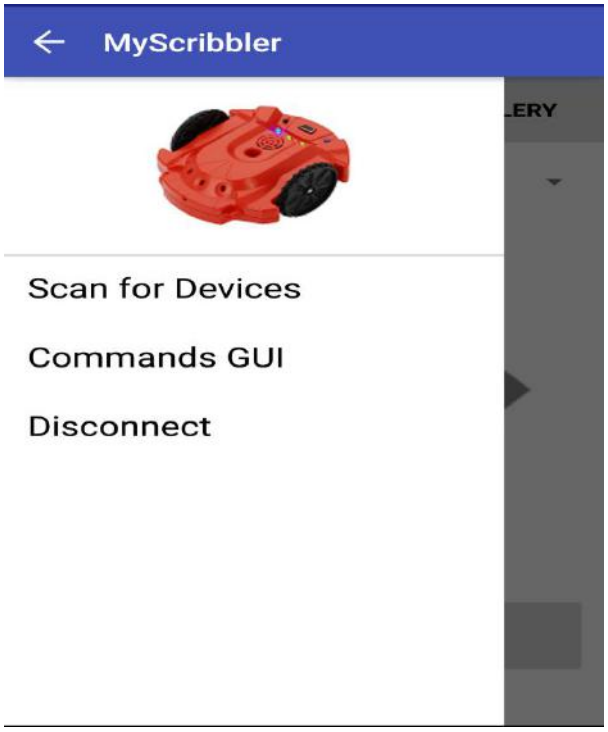

Figure 7 - Image of Navigation Drawer

4.2.1. Navigation Drawer

The navigation drawer is a panel that transitions from the left edge of the screen, and displays all the app's main navigation options. The drawer in the app provides three main options: Scan for Devices, Commands User Interface and Disconnect. The users can bring the navigation drawer by swiping from the left edge of the screen or by touching the three horizontal lines icon on the top left of the screen. Upon expanding, the drawer overlays the content on the screen.

4.2.2. Swipeable Tabs

The main page of the application provides the user with a page consisting of three tabs. The swipe view provides lateral navigation between the tabs with a horizontal finger

gesture. It is the most recommended design to allow easy navigation between multiple pages on the same screen.

The swipeable tabs are implemented with the usage of ViewPager class, which is one of the Layout Managers available in the Android platform, included in the Android Support Library package. The support library package ensures the backward-compatibility of the features that are only available through the library APIs. The ViewPager is a layout widget in which each child view is a separate page/tab on the screen. ViewPagers are often used in conjunction with Fragments, which simplifies the process of managing or retaining state/lifecycle of each page. The pages or the views associated with a ViewPager are provided through an adapter. The two types of adapters that you can use with ViewPager are FragmentPagerAdapter and FragmentStatePagerAdapter. Even though both types of adapters work similarly, they have a few key differences. FragmentPagerAdapter is best used for navigating between a fixed, small number of pages. This allows the adapter to retain the pages on the memory, which avoids reloading of page each time during navigation. However, FragmentStatePagerAdapter is most suitable for navigating across undetermined number of pages. This adapter destroys each page as the user navigates to other pages, thus minimizing memory usage [5].

Since this project's application offers only three main tabs for navigation, FragmentPagerAdapter is implemented to provide the data and layout for each tab.

Figure 8 - Image of Swipeable Tab Navigation

- **Home Tab:**

Home is the first display screen on the application. It contains a self-explanatory easy-to-use buttons that represent motions in all four directions. A stop image in the middle allows the users to stop the robot. The page also has a button at the bottom of the screen labeled as Take Picture, which will initiate the process to capture an image through the camera available on the IPRE fluke. The directions buttons' implementation detect both a short tap and a long tap and hold gesture. A short tap on any arrow button will move the robot to that direction for half a second, while a tap and hold on a button will move the robot continuously in that direction till the user lifts the finger off the button.

Tapping the Take Picture button displays another page where the image is captured. When the image capturing process is initiated, a spinning circle is displayed on the screen. The spinning circle is known as a Progress Bar, which is a visual indicator of progress in an operation. The progress bar's property is set to indeterminate to show a cyclic animation without an indication of progress. This mode is applied when the length of the operation is unknown. The use of the progress bar is to inform the user of the ongoing operation instead of displaying a blank page with to content while the operation is still in progress. This allows the user to be constantly aware of actions taking place.

There are three toggle buttons on the screen which represent the three LED lights on the Scribbler. The user can tap on or off on any of those buttons to manipulate the corresponding LED lights. On the top right hand corner of the page, there is a drop down option to select different modes available to move the robot. The options include the use of the default buttons on the screen, tilting the phone or using voice commands. The voice command is accomplished through the use of Google's Voice to Text API backed by the Google servers. The voice is captured and sent to the server which returns a list of possible matches. Those returned texts are then compared to the predefined command texts, and should there be a match, the action is performed on the robot accordingly.

- **Info Tab**

This is the second tab available of the screen for the users. It includes all the data related to the robot's sensors such as battery, light values, obstacle detection, infrared, and line sensors values. There is also an image of a battery which will change colors based on the current level of battery power in the robot. A single bar filled with red color denotes very low battery power, whereas more bars filled with green color signifies sufficient power. The button present on the middle of the screen provides the option to update the values. The users can press the button any time to get the most recent values associated with all the above sensors.

- **Gallery Tab**

The final tab is the Gallery tab, which displays all the images taken through the application. When the image capturing process is complete, the users have the option to save the image or discard it. If they decide to save the image by supplying a filename

and pressing ‘Save’, then the image is saved in the device. The page displays images in a GridView, and clicking any picture shows a full screen view of that picture.

Android platform provides several options to store data. Sharedpreferences, Internal Storage, External Storage, SQLite Database and Network Connection are the available options. In this application, the images are saved on external storage, which can be a removable storage media like SD card or an internal storage. The data saved on external storage is world-readable and can be modified by user if they enable USB mass storage. Since the intended use of the application is mostly for teaching purposes and the image data are not sensitive, all the images are stored on the external storage [6].

Manipulating images is a very memory-intensive work, and give that Android device are working with limited memory, a very special care has to be taken while dealing with images or bitmaps. As all the different types of Android devices available have various density and resolutions, large bitmaps should be properly processed without exceeding memory limit. The BitmapFactory class provided allows calculating dimension of a bitmap before decoding the image and allocating memory. Knowing the dimension allows calculating whether the image should be loaded without any changes or be scaled down which depends on the device’s screen resolution, memory as well the space allocated on the screen for the image to be displayed [7].

These image processing tasks are memory intensive as well unpredictable, hence should never be executed on the main application thread. If any of the tasks unusually long time then it will block the main User-Interface(UI) thread, and eventually the application will stop responding. The Application Not Responding(ANR) should be

avoided at all cost as it does not provide a good user experience. To ensure these guidelines are met, in this application the memory intensive image tasks are done through the Android's AsyncTask class. This class provides two key methods doInBackground() and onPostExecute(). The background method executes the intensive tasks in another background thread and passes the result to the onPostExecute method. Then in the onPostExecute method the data is updated and displayed on the screen.

One of the key concepts while working with images is caching. When a page is displaying a large set of images then continual processes of many images will slow down the responsiveness of the page. Android platform provides many caching techniques including memory and disk bitmap cache. Memory cache offers fast access to bitmaps but takes up considerable application memory size. This type of caching is suitable for limited number of bitmaps and if recently viewed bitmaps are to be referenced fast periodically. Disk caching is more suited for large sets of bitmaps to be displayed on a page. The use of memory cache in this case would fill the application's memory very quickly. In disk caching bitmaps are stored in device's internal storage. If the bitmap is no longer available in the memory, then accessing it through the disk helps decrease the loading time.

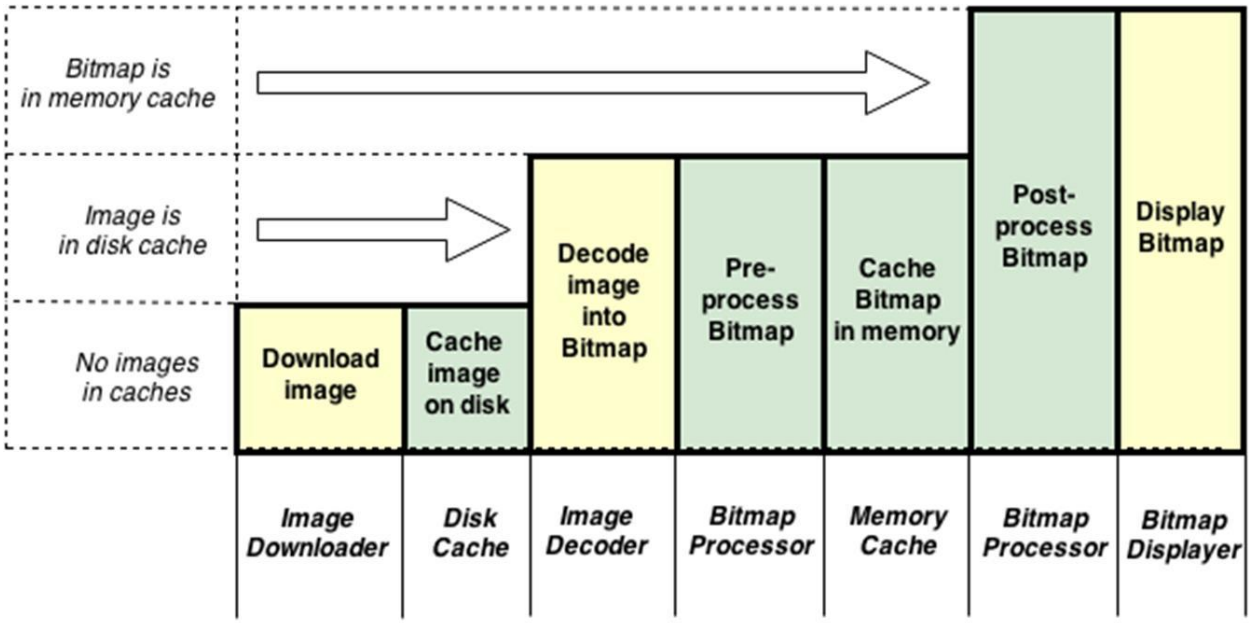

Figure 9 - Universal Image-Loader Library Load and Display Image Task Flow [8]

This project uses Android Universal Image Loader, a powerful, flexible and highly customizable library by Sergey Tarasevich. It implements multi-threaded image loading which ensures the application main thread is not blocked. It provides options to enable both memory and disk caching [8].

4.3. Scribbler Communication

The primary communication to Scribbler robot is performed through bytes messages. All the communication messages are categorized into two types: **SET** commands and **GET** commands. Each message sent to the Scribbler consists of a constant 9 bytes long. The first byte is the message type, the next bytes are the required command data if any and the rest of the messages are filled with null

characters until a total of 9 bytes. Upon receiving the commands, the Scribbler robot sends byte messages as a response. The response messages from the robot vary in length depending on the type of message. The first bytes are the message data. The last byte is the message type. If the message was sent through a GET command, then just the requested data is sent. On the other hand, if the message was through a SET command then the binary state of the robot is included in the response message [9].

4.3.1. Android Scribbler Connection

The Android platform provides an extensive support for Bluetooth network that allows devices to wirelessly exchange data with other Bluetooth devices. The Scribbler robot combined with the IPRE Fluke provides Bluetooth connectivity. This establishes a channel for communication between Android devices and the Scribbler robot through a wireless Bluetooth connection.

In order to use the Bluetooth feature in an Android application, the Bluetooth permission must be declared in the Android Manifest file. The Android Bluetooth API provides access to Bluetooth Adapter, which can be used to find remote Bluetooth devices through device discovery or querying list of paired (bonded) devices. In order to create a connection, one device must implement server-side and another client-side mechanism. In this project, the Scribbler robot will be implementing the server-side by allowing socket connection, and the Android application will use the Scribbler's MAC address to initiate a connection on the same RFCOMM channel. The established

connection will then allow each device to transfer data through the obtained input and output streams [10].

4.4 Commands Interface

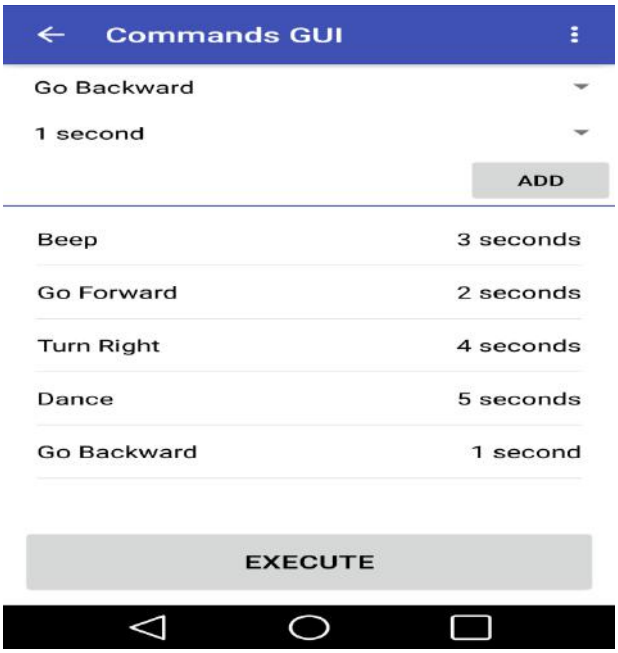

Figure 10 - Image of Commands Interface

The app provides a commands interface page, where you can select a list of commands and execute them together. The available commands are Beep, Dance, Go Backward, Go Forward, Turn Left and Turn Right. Each command is associated with a time period that it is to be executed. The drop-down menus allows easily selecting commands and adding them to the list. The Execute button present at the bottom of the page implements all the selected commands sequentially. This interface allows for an easy demonstration of all the capabilities of Scribbler.

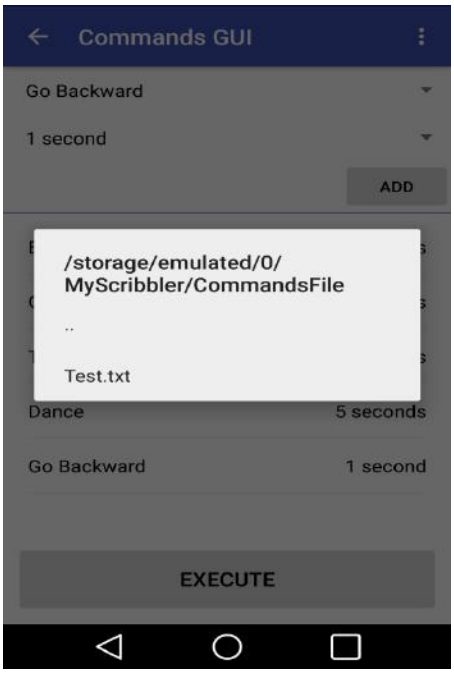

Figure 11 - Image of Open File

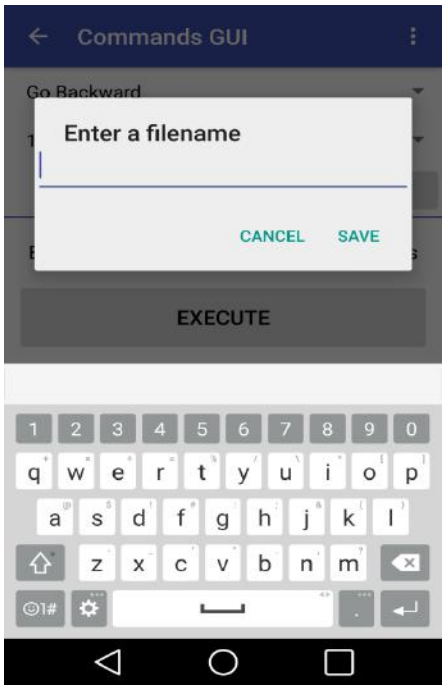

Figure 12 - Image of Save Commands File

The interface also offers the ability of save or open a previously saved list of commands on the device. When the top-right options menu on the page is selected, it provides the

user with options to Open or Save. When you click the save option, it prompts you to enter a filename, and then saves the created list of commands in a file under that name inside the app's directory in the phone's internal storage. The open option displays a dialog box which displays all the saved files inside that directory in the phone. The user can select a filename and it displays all the commands in the file on the screen.

This interface allows to easily interact with Scribbler through the use of an already available list of commands. This saves time and effort of creating a list each time during demonstration.

5. IMPLEMENTATION

The implementation of the app comprises of around 32 classes grouped in different packages. The classes are written in Java. Figure 13 illustrates a high level overview of the key classes involved in the app. The code for all classes is available as appendix.

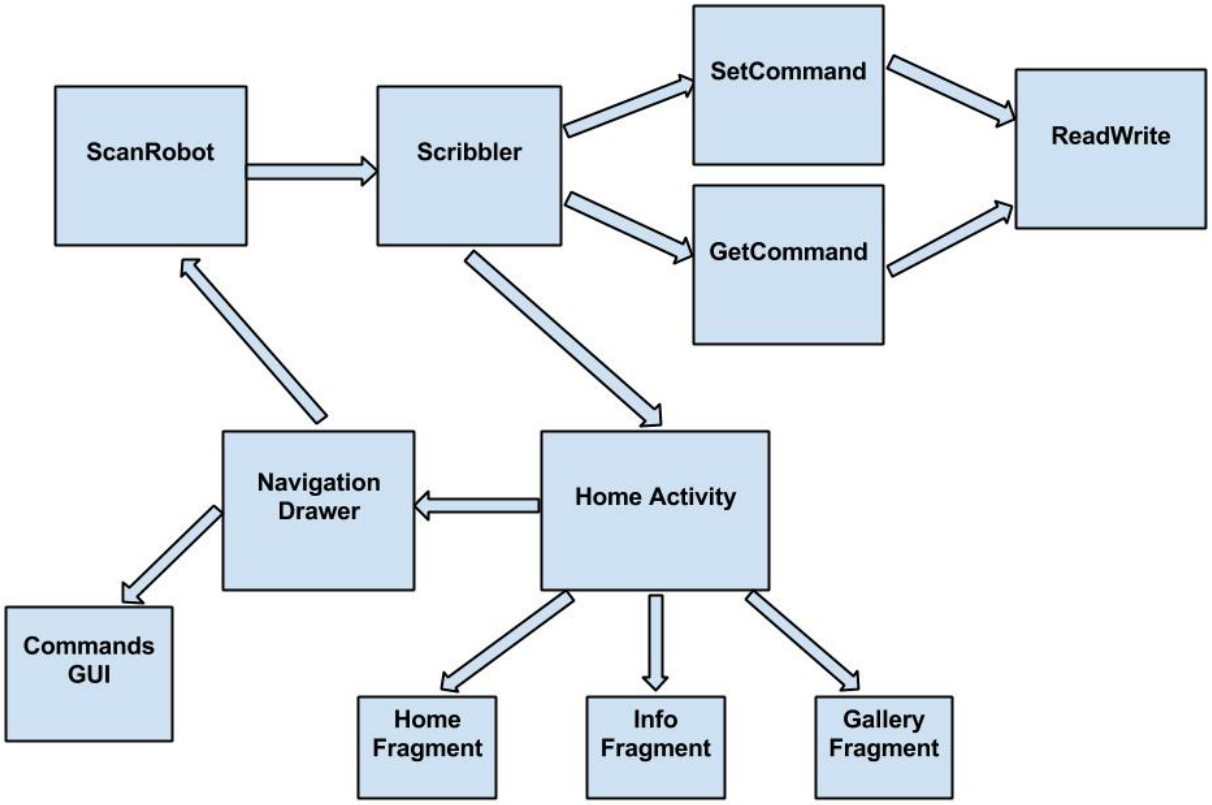

Figure 13- Class Overview

Home Activity

This is the main entry point of the app. An activity class is an application component that provides a screen with user interface. When the app is launched, the layout associated with this activity is inflated and displayed on the screen. This particular layout consists

of three tabs Home, Info and Gallery. It also consists of a navigation drawer, which can be opened by tapping on the top left icon on the screen. The navigation drawer includes the Scan Device, Commands GUI and Disconnect option. You can click on Scan Device option to start the Bluetooth enabled devices discovery process.

ScanRobot

This activity provides the user with the interface to start searching for Bluetooth devices. The layout consists of a button and a list to display the names of the devices. Once the discovery process is started by clicking the button, it displays the information about nearby Bluetooth devices and lists them on the screen. In order to pair with any device, you can touch the device's name on the list.

Scribbler

This class acts as a model to represent a Scribbler object. It's connect() method is called when the user initiates the pairing process with the device. The connected device is then configured to act as a Scribbler object and all the commands interaction with the robot is implemented through the methods of this class. Some of the methods in this class are turnLeft(), turnRight(), forward(), backward(), getBattery(), getIR() and takePicture().

SetCommands and GetCommands

These two classes represent the two types of commands issued to the Scribbler robot. The commands that request the robot to perform certain actions are classified as

SetCommands. The methods of the Scribbler class like forward() and backward() are implemented through an instance of SetCommands class object. On the other hand, if the commands request data from the robot then those actions are categorized as GetCommands. The Scribbler class methods such as getIR() and getLight() are issued through an instance of GetCommands class object.

ReadWrite

Both SetCommands and GetCommands types of commands have to perform read and write operations on the Scribbler robot. This class provides a common method to facilitate both read and write operations through the Bluetooth socket. Since there is no need to instantiate an object of this class, all of its methods are of static type.

6. USER GUIDE

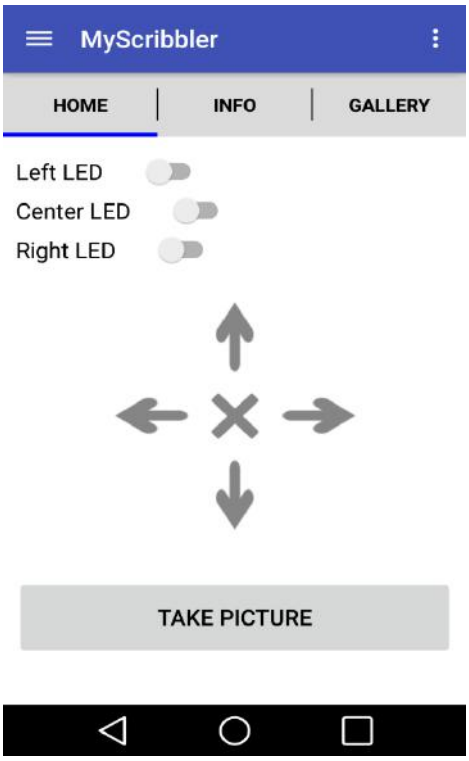

Figure 14 - Image of the app main UI

When the app is launched by touching the launcher icon on the phone, the main page of the app similar to Figure 13 is displayed to the user. In order to view the app's main options, the user can click on the top-left navigation icon and a drawer will slide out which contains the three options: Scan Devices, Commands GUI and Disconnect.

Scan Devices

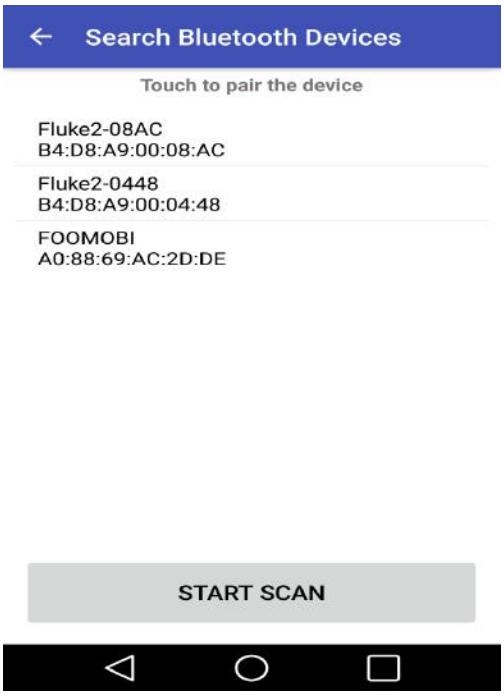

Figure 15 - Image of Bluetooth enabled devices list

The Scan Devices option is to locate the nearest Bluetooth enabled devices. When you go to the page, it displays a Start Scan button at the bottom. Upon clicking on button, it will scan for Bluetooth devices and list them on the page. The user is now able to pair their Android phone with any of the device listed by tapping on its name.

After a successful connection with a Scribbler robot, the user can interact with it by clicking on buttons on the Home Tab or take a picture by pressing the Take Picture button. The Info tab displays all the sensor values of the robot. The Update button in that tab retrieves the recent values of all the sensors available. The Gallery tab holds all the pictures taken through the camera present on the Fluke.

7. CONCLUSIONS

The project returned many positive results. We were able to develop an Android application that can communicate with Scribbler by establishing a Bluetooth connection. The application provides a very easy-to-use interface for users to control the robot and utilize several of its capabilities. We were also able to capture images and save them on the device storage. The successful development of an Android application will allow students and professors to experiment with Scribbler robot easily in classrooms. The app can be easily downloaded from the Google Play Store and students can start interacting with Scribbler right away without having to spend a lot of time on setting up the environment which would include installation of several software and learning programming languages.

8. FUTURE WORK

The project can be enhanced in several areas to improve the efficiency and capabilities of the application. One of the key areas that could be improved is the Commands Interface, which allows selecting multiple commands and executing them simultaneously. This also provides the users the option to save the list to a file and be able to retrieve it in future.

9. REFERENCES

[1] Parallax.com, 'Scribbler 2 (S2) Robot - USB | 28136 | Parallax Inc', 2015. [Online]. Available: http://www.parallax.com/product/28136. [Accessed: March-2015].

[2] Play.google.com, 2015. [Online]. Available: https://play.google.com/store/apps/details?id=com.felicekarl.scribby_project_1&hl=en. [Accessed: March-2015].

[3] Play.google.com, 2015. [Online]. Available: https://play.google.com/store/apps/details?id=com.scribdroid.android&hl=en. [Accessed: March-2015].

[4] L. Ma, L. Gu, J. Wang. Research and Development of Mobile Application for Android Platform. [Online]. Available: http://www.sersc.org/journals/IJMUE/vol9_no4_2014/20.pdf. [Accessed: March-2105]

[5] Google, 'Creating Swipe Views with Tabs | Android Developers', 2015. [Online]. Available: http://developer.android.com/training/implementing-navigation/lateral.html. [Accessed: March- 2015].

[6] Google, 'Storage Options | Android Developers', 2015. [Online]. Available: http://developer.android.com/guide/topics/data/data-storage.html#filesExternal. [Accessed: March-2015].

[7] Google, 'Caching Bitmaps | Android Developers', 2015. [Online]. Available: http://developer.android.com/training/displaying-bitmaps/cache-bitmap.html. [Accessed: March-2015].

[8] GitHub, 'nostra13/Android-Universal-Image-Loader', 2015. [Online]. Available: https://github.com/nostra13/Android-Universal-Image-Loader. [Accessed: April- 2015].

[9] Institute for Personal Robots in Education, 'Scribbler Wire Protocol - IPRE Wiki', 2015. [Online]. Available: http://wiki.roboteducation.org/Scribbler_Wire_Protocol. [Accessed: April- 2015].

[10] Google, 'Bluetooth | Android Developers', 2015. [Online]. Available: http://developer.android.com/guide/topics/connectivity/bluetooth.html#ConnectingDevices. [Accessed: April-2015].

[11] Techotopia, 'An Overview of the Kindle Fire Android Architecture - Techotopia', 2015. [Online]. Available: http://www.techotopia.com/index.php/An_Overview_of_the_Kindle_Fire_Android_Architecture. [Accessed: May-2015].

[12] Google, 'Introduction - Material design - Google design guidelines', 2015. [Online]. Available: http://www.google.com/design/spec/material-design/introduction.html#introduction-goals. [Accessed: May-2015].

[13] ScienceDaily, 'Human-inspired robot takes a brisk walk in the grass', 2015. [Online]. Available: http://www.sciencedaily.com/releases/2015/05/150503101622.htm. [Accessed: April-2015].

[14] ScienceDaily, 'A robot prepared for self-awareness: Expanded software architecture for walking robot Hector', 2015. [Online]. Available: http://www.sciencedaily.com/releases/2015/03/150331113454.htm. [Accessed: April-2015].

[15] Robot programming languages – Robotics blog, 2015. [Online]. Available: http://fabryka-robotow.pl/2015/01/programming-languages-to-control-robot. [Accessed: April- 2015].

[16] D. Kumar, 'Learning Computing With Robots'. [Online]. Available: http://cs.brynmawr.edu/~dkumar/Myro/Text/Fall2011Calico/PDF/LCRCalicoV2.pdf. [Accessed: April-2015].

[17] Institute for Personal Robots in Education, 'Betterbots', 2015. [Online]. Available: http://www.betterbots.com/download/fluke2usermanualv3.pdf. [Accessed: April-2015].

[18] S. Gibbs, 'What is Boston Dynamics and why does Google want robots?', *The Guardian*, 2013. [Online]. Available: http://www.theguardian.com/technology/2013/dec/17/google-boston-dynamics-robots-atlas-bigdog-cheetah. [Accessed: April-2015].

[19] Amazon, 'Amazon Prime Air', 2015. [Online]. Available: http://www.amazon.com/b?node=8037720011. [Accessed: April-2015].

[20] Finch Robot, 'The Finch', 2015. [Online]. Available: http://www.finchrobot.com/. [Accessed: April-2015].

[21] Arduino, 'Arduino - Environment', 2015. [Online]. Available: http://www.arduino.cc/en/Guide/Environment. [Accessed: April-2015].

APPENDIX – SOURCE CODE

Home Activity

This class represents the entry point of the application. When the app is launched, this activity is created and the layout associated with it is displayed on the screen.

```
package com.msu.myscribbler;
import java.io.File;
import com.msu.myscribbler.commands.Scribbler;
import com.msu.myscribbler.commands.SetCommands.LED;
import com.msu.myscribbler.fragments.GalleryFragment;
import com.msu.myscribbler.fragments.MainFragment;
import com.msu.myscribbler.interfaces.GalleryFragmentCallback;
import com.msu.myscribbler.interfaces.NavigationDrawerCallbacks;
import com.msu.myscribbler.models.AppModel;
import com.msu.myscribbler.navigationdrawer.NavigationDrawerFragment;
import com.msu.myscribbler.utils.Constants;
import com.msu.myscribbler.utils.Log;
import android.os.Bundle;
import android.os.Environment;
import android.os.Handler;
import android.app.Activity;
import android.app.ProgressDialog;
import android.content.Intent;
import android.os.AsyncTask;
import android.support.v4.app.Fragment;
import android.support.v4.app.FragmentTransaction;
import android.support.v4.widget.DrawerLayout;
import android.support.v7.app.ActionBarActivity;
import android.support.v7.widget.Toolbar;
import android.view.Menu;
import android.view.MenuInflater;
import android.widget.Toast;


public class HomeActivity extends ActionBarActivity
implements NavigationDrawerCallbacks, GalleryFragmentCallback {

    private static final String TAG="HomeActivity";

    private Toolbar toolbar;
    private NavigationDrawerFragment mNavigationDrawerFragment;
    private AppModel appModel;
```

```
    public final static String imageFilePath =
Environment.getExternalStorageDirectory()  + "/MyScribbler/Images/";
    public final static String
commandsFilePath=Environment.getExternalStorageDirectory()  +
"/MyScribbler/CommandsFile/";

    public static final int SCAN_DEVICE  = 1;
    public static final int TAKE_PHOTO=2;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.home_activity);

       // call navigation drawer init method
        initializeDrawer();

        // create folders in internal storage of the phone
        File ImgDir = new File(imageFilePath);
        if(!ImgDir.exists()  || !ImgDir.isDirectory()){
            ImgDir.mkdirs();
        }
        File commandsDir=  new File(commandsFilePath);
        if(!commandsDir.exists()  || !commandsDir.isDirectory()){
            commandsDir.mkdirs();
        }

        // instantiating a Scribbler model on an application context level
        appModel=(AppModel)getApplicationContext();
        appModel.setScribbler(new  Scribbler());

        if (savedInstanceState  == null) {
          FragmentTransaction transaction =
getSupportFragmentManager().beginTransaction();
          MainFragment fragment = new MainFragment();
          transaction.replace(R.id.content_fragment,  fragment);
          transaction.commit();
       }
    }


    /**
    * Method to initialize the navigation drawer
    */
    protected void initializeDrawer(){
        // initialize navigation drawer
        toolbar = (Toolbar) findViewById(R.id.tool_bar);
       setSupportActionBar(toolbar);
        getSupportActionBar().setDisplayShowHomeEnabled(true);
```

```
        mNavigationDrawerFragment = (NavigationDrawerFragment)
getSupportFragmentManager().findFragmentById(R.id.fragment_drawer);
        mNavigationDrawerFragment.setup(R.id.fragment_drawer, (DrawerLayout)
findViewById(R.id.drawer), toolbar);

    }

    @Override
    public void onBackPressed() {
        if (mNavigationDrawerFragment.isDrawerOpen())
            mNavigationDrawerFragment.closeDrawer();
        else{
            super.onBackPressed();
        }
    }

    @Override
    public void onNavigationDrawerItemSelected(int position, final String title) {
        if(!title.isEmpty()){
            new Handler().postDelayed(new Runnable(){
                @Override
                public void run() {
                    Log.d(TAG, "Item Title: "+title);
                Intent intent = null;
                if(Constants.SCAN.equalsIgnoreCase(title)){
                    intent= new Intent(HomeActivity.this, ScanRobot.class);
                    startActivityForResult(intent, SCAN_DEVICE);
                }else if(Constants.GUI.equalsIgnoreCase(title)){
                    intent= new Intent(HomeActivity.this, CommandsGUI.class);
                    startActivity(intent);
                    //startActivityForResult(intent, TAKE_PHOTO);
                } else if(Constants.DISCONNECT.equalsIgnoreCase(title)){
                    //intent= new Intent(getActivity(), AddNewFriends.class);
                    if(appModel.getScribbler().isConnected()){
                        boolean result=appModel.getScribbler().disconnect();
                        if(result)
                            Toast.makeText(getApplicationContext(), "Connection
disabled...", Toast.LENGTH_SHORT).show();
                        else
                            Toast.makeText(getApplicationContext(), "Error
disconnecting...", Toast.LENGTH_SHORT).show();
                    }else{
                        Toast.makeText(getApplicationContext(), "Not connected to
any device...", Toast.LENGTH_SHORT).show();
                    }
                }
            }
            }, 300);
        }
    }
```

```
    @Override
    public void updateData() {

    }


    /**
    * Result from the ScanRobot activity
    */
    @Override
    public void onActivityResult(int requestCode, int resultCode, Intent data) {
        //super.onActivityResult(requestCode, resultCode, data);

        switch (requestCode) {
       case SCAN_DEVICE:
          if (resultCode == Activity.RESULT_OK) {
             // Get the selected device's MAC address
             String name= data.getExtras().getString(ScanRobot.DEVICE_NAME);
            String address =
data.getExtras().getString(ScanRobot.DEVICE_ADDRESS);
            Log.d(TAG, "Connected Device Address " + address);
             Scribbler myScribbler= new Scribbler(address);
             // try connecting to the device
             try{
                if(myScribbler.connect()){
                   Log.d(TAG, "Connection successfull");
                   //myScribbler.setDeviceAddress(address);
                   //myScribbler.setDeviceName(name);
                   appModel.setScribbler(myScribbler);
                   // set the appModel status as connected
                   appModel.setConnected(true);

                   Scribbler check= appModel.getScribbler();
                   check.beep(784, .03f);
                   Toast.makeText(getApplicationContext(),
                         "Successful Connection to " + name,
Toast.LENGTH_SHORT).show();
                }else{
                   Toast.makeText(getApplicationContext(),
                         "Error connecting to " + name,
Toast.LENGTH_SHORT).show();
                }
             }catch(Exception e){
                e.printStackTrace();
             }
          }
          break;

       case HomeActivity.TAKE_PHOTO:
```

```
            if (resultCode == Activity.RESULT_OK) {
                Log.d(TAG, "TAKE PHOTO");
                /*FragmentTransaction transaction =
getSupportFragmentManager().beginTransaction();
                Fragment galleryFragment=
getSupportFragmentManager().findFragmentByTag(GalleryFragment.TAG);
                transaction.detach(galleryFragment);
                transaction.attach(galleryFragment);
                transaction.commit();*/
            }
            break;
        }
    }
}
```

Scan Robot

This activity represents the page where the device scans for nearby Bluetooth devices and displays them in a list.

```
package com.msu.myscribbler;

import java.util.Set;
import com.msu.myscribbler.R;
import com.msu.myscribbler.utils.Log;
import android.app.Activity;
import android.app.ProgressDialog;
import android.bluetooth.BluetoothAdapter;
import android.bluetooth.BluetoothDevice;
import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
import android.content.IntentFilter;
import android.os.Bundle;
import android.support.v7.app.ActionBarActivity;
import android.support.v7.widget.Toolbar;
import android.view.MenuItem;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.AdapterView;
import android.widget.ArrayAdapter;
import android.widget.Button;
import android.widget.ListView;
import android.widget.TextView;
import android.widget.Toast;
import android.widget.AdapterView.OnItemClickListener;

public class ScanRobot extends ActionBarActivity{

    private static final String TAG= "ScanRobot";

    public static String DEVICE_NAME="scan_device_name";
    public static String DEVICE_ADDRESS  = "scan_device";

    private static final int REQUEST_ENABLE_BT  = 1;

    private Toolbar toolbar;
    private Button startScanButton;
    private TextView infoView;
    private BluetoothAdapter  myBluetoothAdapter;
    private Set<BluetoothDevice>  pairedDevice;
    private ArrayAdapter<String>  myAdapter;
    private ListView myListView;
```

```
    private ProgressDialog progressDialog;

    @Override
    protected void onCreate(Bundle savedInstanceState) {

        super.onCreate(savedInstanceState);

        setContentView(R.layout.scan_robot);

        toolbar = (Toolbar) findViewById(R.id.tool_bar);
       setSupportActionBar(toolbar);
        getSupportActionBar().setDisplayShowHomeEnabled(true);
        getSupportActionBar().setDisplayHomeAsUpEnabled(true);

        myListView= (ListView) findViewById(R.id.listview_devices);
        myAdapter= new
ArrayAdapter<String>(this,android.R.layout.simple_list_item_1 );
        myListView.setAdapter(myAdapter);

        infoView= (TextView) findViewById(R.id.touch_search_view);

        // Register for broadcasts when a device is discovered
       IntentFilter filter = new IntentFilter(BluetoothDevice.ACTION_FOUND);
       this.registerReceiver(myReceiver,  filter);

       // Register for broadcasts when discovery has finished
       filter = new
IntentFilter(BluetoothAdapter.ACTION_DISCOVERY_FINISHED);
       this.registerReceiver(myReceiver,  filter);

       startScanButton= (Button)findViewById(R.id.scan_start);

        myBluetoothAdapter=  BluetoothAdapter.getDefaultAdapter();

        if(myBluetoothAdapter.isEnabled()){
           }else{
              Log.d(TAG, "bluetoothadapter disabled");
              startScanButton.setEnabled(false);
              Intent turnOnIntent = new
Intent(BluetoothAdapter.ACTION_REQUEST_ENABLE);
              startActivityForResult(turnOnIntent,  REQUEST_ENABLE_BT);
           }

       if(myBluetoothAdapter==null){
           Log.d(TAG, "Bluetooth null");
           startScanButton.setEnabled(false);

           Toast.makeText(getApplicationContext(),"Bluetooth  not supported.
Please check your settings!",
```

```
                Toast.LENGTH_LONG).show();
        }else{
            Log.d(TAG, "bluetoothadapter not null");
            Log.d(TAG, "bluetoothadapter enabled");

                startScanButton.setOnClickListener(new  OnClickListener() {
                @Override
                public void onClick(View v) {
                    findDevices(v);
                    listDevices(v);
                    //myAdapter.clear();
                    //myBluetoothAdapter.startDiscovery();
                    }
                });


        }

        myListView.setOnItemClickListener(deviceClickListener);

    }

    @Override
    protected void onDestroy(){
        super.onDestroy();
        unregisterReceiver(myReceiver);
    }

    @Override
    public boolean onOptionsItemSelected(MenuItem  item) {
      switch (item.getItemId())  {
      // Respond to the action bar's Up/Home button
      case android.R.id.home:
          finish();
          return true;
      }
      return super.onOptionsItemSelected(item);
    }

    private OnItemClickListener  deviceClickListener = new OnItemClickListener()
{
        @Override
        public void onItemClick(AdapterView<?>  parent, View view, int position,
                long id) {
            Log.d(TAG, "Adapter item on click listener");
            myBluetoothAdapter.cancelDiscovery();
            // Get the device MAC address, which is the last 17 chars in the
           // View
           String deviceName = ((TextView) view).getText().toString();
           Log.d(TAG, "Device name: "+deviceName);
```

```
            String address = deviceName.substring(deviceName.length()  - 17);
            Log.d(TAG, "device address: "+address);
            // Create the result Intent and include the MAC address
              Intent intent = new Intent();
              intent.putExtra(DEVICE_ADDRESS,  address);
              intent.putExtra(DEVICE_NAME,  deviceName);
              // Set result and finish this Activity
              setResult(Activity.RESULT_OK,  intent);
              finish();
          }


    };

    /**
    * Method to find bluetooth devices
    */
    public void findDevices(View  view){


        if(myBluetoothAdapter.isDiscovering()){
            myBluetoothAdapter.cancelDiscovery();
            Log.d(TAG, "Cancelling discovery");
        }else{
            Log.d(TAG, "Starting discovery");
            myAdapter.clear();
            myBluetoothAdapter.startDiscovery();
            //progressBar.setVisibility(View.VISIBLE);
            registerReceiver(myReceiver,  new
IntentFilter(BluetoothDevice.ACTION_FOUND));
            ScanRobot.this.runOnUiThread(new  Runnable() {
                @Override
                public void run() {
                    progressDialog = ProgressDialog.show(ScanRobot.this,  "",
                            "Scanning devices", false, false);
                }
            });
        }
    }


    /**
    * Method to list bluetooth devices
    * @param view
    */
    public void listDevices(View  view){
        infoView.setVisibility(View.VISIBLE);
        pairedDevice= myBluetoothAdapter.getBondedDevices();

        for(BluetoothDevice  device: pairedDevice)
```

```
            myAdapter.add(device.getName()+"\n"+device.getAddress());

        Toast.makeText(getApplicationContext(), "Listing Paired Devices",
Toast.LENGTH_SHORT).show();
    }

    /**
    * Activity result if prompted to turn Bluetooth on
    */
    @Override
    protected void onActivityResult(int requestCode, int resultCode, Intent data) {
        if(requestCode == REQUEST_ENABLE_BT){
            if(myBluetoothAdapter.isEnabled()){
                startScanButton.setEnabled(true);
                Toast.makeText(getApplicationContext(),"Bluetooth Enabled..." ,
                        Toast.LENGTH_SHORT).show();
            }
        }

    }
    final BroadcastReceiver myReceiver= new BroadcastReceiver(){

        @Override
        public void onReceive(Context context, Intent intent) {
            String action= intent.getAction();
            if(BluetoothDevice.ACTION_FOUND.equals(action)){
                BluetoothDevice device=
intent.getParcelableExtra(BluetoothDevice.EXTRA_DEVICE);
                if (device.getBondState() != BluetoothDevice.BOND_BONDED){
                    Log.d(TAG, "device not bonded, adding");
                    myAdapter.add(device.getName()+"\n"+device.getAddress());
                    myAdapter.notifyDataSetChanged();
                }
            }else
if(BluetoothAdapter.ACTION_DISCOVERY_FINISHED.equals(action)){

                Log.d(TAG, "discovery finished");
                //progressBar.setVisibility(View.INVISIBLE);
                ScanRobot.this.runOnUiThread(new Runnable() {
                    @Override
                    public void run() {
                        //if(!(ScanRobot.this).isFinishing()){
                            progressDialog.dismiss();
                        //}

                    }
                });
            }}
        };
}
```

CommandsGUI

This activity is the interface that provides the options to select multiple commands and execute them sequentially.

```
package com.msu.myscribbler;

import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.FileWriter;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.io.OutputStreamWriter;
import java.util.ArrayList;
import java.util.List;
import java.util.Timer;
import java.util.TimerTask;
import java.util.concurrent.Executors;
import java.util.concurrent.Future;
import java.util.concurrent.ScheduledExecutorService;
import java.util.concurrent.TimeUnit;
import com.msu.myscribbler.adapters.CommandItemAdapter;
import com.msu.myscribbler.models.AppModel;
import com.msu.myscribbler.models.CommandItem;
import com.msu.myscribbler.utils.Constants;
import com.msu.myscribbler.utils.FileDialog;
import com.msu.myscribbler.utils.Log;
import android.app.Activity;
import android.app.AlertDialog;
import android.content.Context;
import android.content.DialogInterface;
import android.content.Intent;
import android.os.AsyncTask;
import android.os.Bundle;
import android.os.CountDownTimer;
import android.os.Environment;
import android.os.Handler;
import android.support.v7.app.ActionBarActivity;
import android.support.v7.widget.Toolbar;
import android.view.Menu;
import android.view.MenuInflater;
import android.view.MenuItem;
import android.view.View;
import android.view.View.OnClickListener;
```

```
import android.view.WindowManager;
import android.widget.AdapterView;
import android.widget.AdapterView.OnItemClickListener;
import android.widget.AdapterView.OnItemLongClickListener;
import android.widget.AdapterView.OnItemSelectedListener;
import android.widget.Button;
import android.widget.EditText;
import android.widget.ListView;
import android.widget.Spinner;
import android.widget.Toast;

public class CommandsGUI extends ActionBarActivity{

   private static final String TAG="CommandsGUI";

   private Toolbar toolbar;
   private AppModel appModel;

   FileOutputStream outputStream;
   FileDialog fileDialog;
   private ListView commandsListView;
   private CommandItemAdapter myAdapter;
   private ArrayList<CommandItem> listItems;

   private Spinner commandSpinner, timeSpinner;
   private Button addButton;
   private Button executeButton;

   @Override
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.commands_gui);

      toolbar = (Toolbar) findViewById(R.id.tool_bar);
     setSupportActionBar(toolbar);
      getSupportActionBar().setDisplayShowHomeEnabled(true);
      getSupportActionBar().setDisplayHomeAsUpEnabled(true);

      appModel=(AppModel)getApplicationContext();
      timeSpinner=(Spinner)findViewById(R.id.time_spinner);
      addButton=(Button)findViewById(R.id.add_button);

      commandSpinner=(Spinner)findViewById(R.id.commands_spinner);
      //commandSpinner.setSelection(0, false);
      // Set up listener for commands spinner
      commandSpinner.setOnItemSelectedListener(new
OnItemSelectedListener() {
         @Override
         public void onItemSelected(AdapterView<?> parent, View view,
               int position, long id) {
```

```
            String itemSelected= parent.getItemAtPosition(position).toString();
            Log.d(TAG, "ITEM SELECTED: "+itemSelected);
            // Check if to display time spinner or not
            boolean flag= displayTimeSpinner(itemSelected);
            if(flag){
               timeSpinner.setVisibility(View.VISIBLE);
            }else
               timeSpinner.setVisibility(View.GONE);
            addButton.setVisibility(View.VISIBLE);

         }
         @Override
         public void onNothingSelected(AdapterView<?> parent) {
         }
      });



      commandsListView= (ListView)findViewById(R.id.commands_listview);
      listItems= new ArrayList<CommandItem>();
      myAdapter= new CommandItemAdapter(this, listItems);
      commandsListView.setAdapter(myAdapter);
      commandsListView.setOnItemLongClickListener(new
OnItemLongClickListener() {
         @Override
         public boolean onItemLongClick(AdapterView<?> parent, View view,
               int position, long id) {
            CommandItem clickedItem= (CommandItem)
myAdapter.getItem(position);
            displayCommandDialog(clickedItem, position);
            return true;
         }
      });



      addButton.setOnClickListener(new OnClickListener() {
         @Override
         public void onClick(View v) {
            String
selectedCommand=commandSpinner.getItemAtPosition(commandSpinner.getS
electedItemPosition()).toString();
            String
selectedTime=timeSpinner.getItemAtPosition(timeSpinner.getSelectedItemPosit
ion()).toString();
            Log.d(TAG, "SELECTED COMMAND: "+selectedCommand);
            Log.d(TAG, "SELECTED TIME: "+selectedTime);
            boolean flag= displayTimeSpinner(selectedCommand);
            CommandItem cmd=null;
            if(flag){
```

```
                Log.d(TAG, "Creating item with time");
                cmd= new CommandItem(selectedCommand, selectedTime);
            }else{
                Log.d(TAG, "Creating item WITHOUT time");
                cmd= new CommandItem(selectedCommand);
            }

            //myAdapter.addItem(new CommandItem(selectedCommand,
selectedTime));
            myAdapter.addItem(cmd);
            myAdapter.notifyDataSetChanged();
        }
    });

    executeButton=(Button)findViewById(R.id.execute_button);
    executeButton.setOnClickListener(new OnClickListener() {
        @Override
        public void onClick(View v) {
            executeAllCommands();
        }
    });
}

@Override
protected void onResume() {
    super.onResume();
}

@Override
public boolean onCreateOptionsMenu(Menu menu) {
    MenuInflater inflater = getMenuInflater();
    inflater.inflate(R.menu.main, menu);
    MenuItem settingItem=menu.findItem(R.id.action_settings);
    settingItem.setVisible(false);
    MenuItem scanItem=menu.findItem(R.id.scan);
    scanItem.setVisible(false);
    MenuItem disconnectItem=menu.findItem(R.id.disconnect);
    disconnectItem.setVisible(false);
    MenuItem gui= menu.findItem(R.id.dashboard);
    gui.setVisible(false);
    MenuItem openItem= menu.findItem(R.id.open);
    openItem.setVisible(true);
    MenuItem saveItem= menu.findItem(R.id.save);
    saveItem.setVisible(true);

    return true;
}

@Override
public boolean onOptionsItemSelected(MenuItem item) {
```

```
        switch(item.getItemId()){
            case android.R.id.home:
                finish();
                return true;


            case R.id.save:
                displayFilenameDialogBox();
                return true;

            case R.id.open:
                File mPath = new File(Environment.getExternalStorageDirectory() +
"/MyScribbler/CommandsFile/");
                fileDialog = new FileDialog(this, mPath);
                fileDialog.setFileEndsWith(".txt");
                fileDialog.addFileListener(new FileDialog.FileSelectedListener() {
                    public void fileSelected(File file) {
                        Log.d(getClass().getName(), "selected file " + file.toString());
                        try {
                            readFile(file.toString());
                        } catch (IOException e) {

                            e.printStackTrace();
                        }
                    }
                });

                fileDialog.showDialog();
                return true;

            default:
                return super.onOptionsItemSelected(item);
        }
    }


    private void readFile(String filename) throws IOException{
        InputStream instream = null;
        listItems.clear();
        try {
            // open the file for reading
            instream= new FileInputStream(filename);

            // if file the available for reading
            if (instream != null) {
             // prepare the file for reading
             InputStreamReader inputreader = new InputStreamReader(instream);
             BufferedReader buffreader = new BufferedReader(inputreader);

             String line;
```

```
            // read every line of the file into the line-variable, on line at the time
            while ((line = buffreader.readLine()) != null){
                Log.d(TAG, "READ FILE LINE: "+line);
                String[] splited= line.split("\\s+");
                Log.d(TAG, "FIRST "+splited[0]);
                Log.d(TAG, "SECOND "+splited[1]);
                Log.d(TAG, "THIRD "+splited[2]);
                CommandItem item= new CommandItem(splited[0]+" "+splited[1],
splited[2]);

                listItems.add(item);
            }
            }
               myAdapter.notifyDataSetChanged();
            } catch (Exception ex) {
              // print stack trace.
            } finally {
            // close the file.
            instream.close();

            }
    }

    private void displayFilenameDialogBox(){
        // Use the Builder class for convenient dialog construction
       AlertDialog.Builder builder = new AlertDialog.Builder(this);
       builder.setTitle("Enter a filename");
        //builder.setMessage(title);
        // Set an EditText view to get user input
        final EditText input = new EditText(this);
        builder.setView(input);
        builder.setPositiveButton("Save",
              new DialogInterface.OnClickListener() {
                 public void onClick(DialogInterface dialog,    int which) {
                    Log.d(TAG, "File Name Entered");
                    try{
                       String
filename=MainActivity.commandsFilePath+input.getText().toString()+".txt";
                       //File file= new File(filename);
                       outputStream = new FileOutputStream(new File(filename));
                       //OutputStreamWriter outputWriter= new
OutputStreamWriter(outputStream);
                       ArrayList<CommandItem> allCommands=
myAdapter.getAllItems();
                       String separator = System.getProperty("line.separator");

                       //FileWriter writer=new FileWriter(new File(filename),true);
                       //BufferedWriter buf= new BufferedWriter(writer);
```

```
                    BufferedWriter buf= new BufferedWriter(new
OutputStreamWriter(new  FileOutputStream(filename),  "UTF-8"));
                    for(CommandItem  c: allCommands){
                        //outputWriter.write(c.getmCommand()+"  "+c.getmTime());
                        String data=c.getmCommand()+"  "+c.getmTime();
                        //outputStream.write(data.getBytes());
                        buf.write(data);
                        buf.newLine();
                    }
                    buf.close();
                    //outputStream.flush();
                    //outputStream.close();
                    //outputWriter.close();
                }catch(Exception  e){
                    e.printStackTrace();
                }
                dialog.dismiss();
            }

        });
    builder.setNegativeButton("Cancel",  new DialogInterface.OnClickListener()  {
            public void onClick(DialogInterface  dialog, int id) {
              dialog.dismiss();
            }
        });
    // Create the AlertDialog object and return it
    AlertDialog alert = builder.create();
    alert.getWindow().setSoftInputMode(
        WindowManager.LayoutParams.SOFT_INPUT_STATE_ALWAYS_VISI
BLE);
     alert.show();
  }

  public void displayCommandDialog(CommandItem  item, final int position){
     AlertDialog.Builder  builder1 = new AlertDialog.Builder(this);
     String options[] = { "Remove  Command",  "Clear All"};
     builder1.setTitle("Edit");
     builder1.setCancelable(true)
          .setItems(options,  new DialogInterface.OnClickListener()  {

            @Override
            public void onClick(DialogInterface  dialog, int which) {
               if(which==0){
                  //listItems.remove(position);
                  myAdapter.removeItem(position);
                  myAdapter.notifyDataSetChanged();
               }else if(which==1){
                  // Ask for confirmation
                  AlertDialog.Builder confirm= new
AlertDialog.Builder(CommandsGUI.this);
```

```
                    confirm.setTitle("Delete All Commands?");
                    confirm.setCancelable(true);
                    confirm.setPositiveButton("Yes", new
DialogInterface.OnClickListener() {
                        @Override
                        public void onClick(DialogInterface dialog, int which) {
                            //listItems.clear();
                            myAdapter.removeAll();
                            myAdapter.notifyDataSetChanged();
                        }
                    });
                    confirm.setNegativeButton("Cancel", new
DialogInterface.OnClickListener() {
                        @Override
                        public void onClick(DialogInterface dialog, int which) {
                            dialog.dismiss();
                        }
                    });
                    AlertDialog confirmAlert= confirm.create();
                    confirm.show();

                } // end of if clear conversation
            }
        });
    AlertDialog alert1= builder1.create();
    alert1.show();

}

/**
* Method that executes all the commands selected
*/
private void executeAllCommands(){
    ArrayList<CommandItem>allCommands=myAdapter.getAllItems();
    if(appModel.getScribbler()!=null && appModel.isConnected()){

        //for(CommandItem item: allCommands){
        for(int i=0; i<allCommands.size(); i++){
            CommandItem item= allCommands.get(i);
            //final double time=0.5;
            //final long setTime= Long.parseLong(item.getmTime());
            String[] time= item.getmTime().split(" ");
            String setTime= time[0];
            Log.d(TAG, "COMMAND "+item.getmCommand());
            //Log.d(TAG, "TIME MULTIPLY: "+String.valueOf(time));
            Log.d(TAG, "TIME "+String.valueOf(setTime));
            String command= item.getmCommand();
            try{
                if(command.equals(Constants.GO_FORWARD)){
                    appModel.getScribbler().forward(0.5);
```

```
                }else
if(command.equals(Constants.GO_BACKWARD)){
                    appModel.getScribbler().backward(0.5);
                }else if(command.equals(Constants.TURN_LEFT)){
                    appModel.getScribbler().turnLeft(0.5);
                }else if(command.equals(Constants.TURN_RIGHT)){
                    appModel.getScribbler().turnRight(0.5);
                }else if(command.equals(Constants.TAKE_PHOTO)){
                    Intent pictureIntent= new Intent(this, TakePicture.class);
                    startActivity(pictureIntent);
                }else if(command.equals(Constants.STOP)){
                    appModel.getScribbler().stop();
                }else if(command.equals(Constants.BEEP))
                    appModel.getScribbler().beep(784,
Float.parseFloat(setTime));

            }catch(Exception e){
                e.printStackTrace();
            }
            try {
                Log.d(TAG, "SLEEPING");
                if(setTime!=null && !setTime.isEmpty())
                    Thread.sleep(Integer.parseInt(setTime)*1000);
                else
                    Thread.sleep(2000);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }

          }
          // Always stop after executing
          try{
              appModel.getScribbler().stop();
          }catch(Exception e){
              e.printStackTrace();
          }
       }else{
          Toast.makeText(this,"Not connected to any device...",
               Toast.LENGTH_SHORT).show();
       }


   }

   /**
   * Method to check whether or not to display Time spinner after command is
selected
   */
   private boolean displayTimeSpinner(String command){
       boolean result=false;
```

```
        if(command.equals(Constants.GO_FORWARD))
            result=true;
        else if(command.equals(Constants.GO_BACKWARD))
            result=true;
        else if(command.equals(Constants.TURN_LEFT))
            result=true;
        else if(command.equals(Constants.TURN_RIGHT))
            result=true;
        else if(command.equals(Constants.BEEP))
            result=true;
        else if(command.equals(Constants.DANCE))
            result=true;

        return result;
    }

}
```

Take Picture

This activity handles the operation to take picture using the camera on the Fluke. It displays the picture and provides the option to save it.

```
package com.msu.myscribbler;

import java.io.BufferedOutputStream;
import java.io.FileOutputStream;

import com.msu.myscribbler.interfaces.AsyncResponse;
import com.msu.myscribbler.models.AppModel;
import com.msu.myscribbler.utils.Log;

import android.app.Activity;
import android.content.Context;
import android.content.Intent;
import android.graphics.Bitmap;
import android.os.AsyncTask;
import android.os.Bundle;
import android.support.v7.app.ActionBarActivity;
import android.support.v7.widget.Toolbar;
import android.view.MenuItem;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.EditText;
import android.widget.ImageView;
import android.widget.ProgressBar;
```

```
import android.widget.TextView;
import android.widget.Toast;



public class TakePicture extends ActionBarActivity  implements AsyncResponse{
   private static final String TAG = "TakePicture";


   private Toolbar toolbar;

    private ImageView iv;

    // taking picture resources
    private ProgressBar pb;
    private TextView loadingMessage;
    private TextView errorMessage;

    // save/cancel resources
    private Button buttonSave;
    private Button buttonCancel;

    // Edit name resources
    private TextView textViewPName;
    private EditText editTextName;

    TakePictureTask takePicture= new TakePictureTask();
    private Bitmap image;

    private AppModel appModel;

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

      setContentView(R.layout.take_picture);
       toolbar = (Toolbar) findViewById(R.id.tool_bar);
      setSupportActionBar(toolbar);
       getSupportActionBar().setDisplayShowHomeEnabled(true);
       getSupportActionBar().setDisplayHomeAsUpEnabled(true);

      appModel=(AppModel)this.getApplicationContext();


      iv = (ImageView)  findViewById(R.id.imageView_picture);
      pb = (ProgressBar) findViewById(R.id.progressBar_picture);
      loadingMessage = (TextView) findViewById(R.id.textView_takingPicture);
      errorMessage= (TextView) findViewById(R.id.takingPicture_error);
      buttonSave = (Button) findViewById(R.id.button_savePicture);
      buttonCancel = (Button) findViewById(R.id.button_cancelPicture);
```

```
        textViewPName = (TextView) findViewById(R.id.textView_pictureName);
        editTextName = (EditText) findViewById(R.id.editText_pictureName);

        // image is null until taken
        image = null;

        // implement cancel onClick
        buttonCancel.setOnClickListener(new OnClickListener() {
          @Override
          public void onClick(View v) {
            finish();
          }
        });

        // implement save onClick
        buttonSave.setOnClickListener(new OnClickListener() {
          @Override
          public void onClick(View v) {
            if (image != null) {
              // save image to internal storage
              try {
                String fileName = editTextName.getText().toString() + ".jpg";
                if(editTextName.getText().toString()!=null &&
!editTextName.getText().toString().isEmpty()){
                    FileOutputStream fos= new
FileOutputStream(MainActivity.imageFilePath+fileName);
                    BufferedOutputStream bos = new BufferedOutputStream(fos);

                    image.compress(Bitmap.CompressFormat.JPEG, 90, fos);
                    bos.flush();
                    fos.close();

                    // Let the user know picture saved successfully and
                    // finish
                    Toast.makeText(getBaseContext(), "Picture saved successfully",
Toast.LENGTH_LONG)
                        .show();
                    Log.i(TAG, "Saved Picture");
                    Intent intent= new Intent();
                    setResult(Activity.RESULT_OK, intent);
                    finish();

                }else{
                    Toast.makeText(getBaseContext(), "Please enter a file name",
Toast.LENGTH_LONG)
                        .show();
                }

              } catch (Exception e) {
```

```
            Toast.makeText(getBaseContext(),"Picture  could not be saved",
Toast.LENGTH_LONG)
                .show();
            Log.e(TAG, e.getMessage());
           }
         } else {
           Toast.makeText(getBaseContext(),"Picture  could not be saved",
Toast.LENGTH_LONG)
               .show();
           Log.e(TAG, "There does not seem to be an image to save");
         }
       }
      });

      takePicture.delegate=this;
      if (appModel.getScribbler().isConnected()){
         pb.setVisibility(View.VISIBLE);
         loadingMessage.setVisibility(View.VISIBLE);
         takePicture.execute();
      }else{
         errorMessage.setVisibility(View.VISIBLE);
         Toast.makeText(TakePicture.this,"Not  connected to any device...",
               Toast.LENGTH_SHORT).show();
      }


    }

    public void onResume() {
      super.onResume();
    }

    @Override
    public boolean onOptionsItemSelected(MenuItem  item) {
         switch (item.getItemId()) {
         // Respond to the action bar's Up/Home button
         case android.R.id.home:
            finish();
            return true;
         }
         return super.onOptionsItemSelected(item);
       }


    @Override
    public void asyncReturn(Bitmap  bitmap) {
       image=bitmap;
      }
```

```
/**
 * AsyncTask to take picture.
 */
private class TakePictureTask extends AsyncTask<Void, Void, Bitmap> {

   public AsyncResponse delegate=null;

  protected Bitmap doInBackground(Void... params) {
     Log.d(TAG, "doInBackgroudn");
     if(appModel.getScribbler()!=null)
        Log.d(TAG, "scribbler get not null");
     else
        Log.d(TAG, "scribbler get null");
     Bitmap picture=appModel.getScribbler().takePicture(1280,800);
     //Bitmap picture= appModel.getScribbler().takePicture(256, 192);

   return picture;
  }

  @Override
  protected void onPreExecute() {
   super.onPreExecute();

   // Show the ProgressBar and associated text
   pb.setVisibility(View.VISIBLE);
   loadingMessage.setVisibility(View.VISIBLE);
  }

  protected void onPostExecute(Bitmap bm) {
     // Hide ProgressBar and associated text
     pb.setVisibility(View.INVISIBLE);
     loadingMessage.setVisibility(View.INVISIBLE);

     if (bm != null) {
        Log.i(TAG, "Picture Success");
        // Show image and picture options
        iv.setVisibility(View.VISIBLE);
        // set the image in the imageview
        iv.setImageBitmap(bm);

        delegate.asyncReturn(bm);
        // assign the newly created bitmap so we can save
        image = bm;
        buttonSave.setVisibility(View.VISIBLE);
        buttonCancel.setVisibility(View.VISIBLE);
        textViewPName.setVisibility(View.VISIBLE);
        editTextName.setVisibility(View.VISIBLE);
     }else{
        Log.e(TAG, "Error taking picture...");
     }
```

```
    }

   }

}
```

Full Screen

This activity displays the saved pictures on the entire phone display.

```
package com.msu.myscribbler;

import com.msu.myscribbler.utils.Log;
import com.nostra13.universalimageloader.core.DisplayImageOptions;
import com.nostra13.universalimageloader.core.ImageLoader;
import com.nostra13.universalimageloader.core.assist.FailReason;
import com.nostra13.universalimageloader.core.assist.ImageScaleType;
import
com.nostra13.universalimageloader.core.listener.ImageLoadingProgressListene
r;
import
com.nostra13.universalimageloader.core.listener.SimpleImageLoadingListener;
import android.graphics.Bitmap;
import android.os.Bundle;
import android.support.v7.app.ActionBarActivity;
import android.support.v7.widget.Toolbar;
import android.view.MenuItem;
import android.view.View;
import android.widget.ImageView;
import android.widget.ProgressBar;

public class FullScreenActivity extends ActionBarActivity {
  private static final String TAG="FullScreenActivity";

  private Toolbar toolbar;

  private ImageView fullView;
  private ProgressBar spinner;

  DisplayImageOptions options;
  private int loadingImage=R.drawable.ic_launcher;

  @Override
  protected void onCreate(Bundle savedInstanceState) {
     super.onCreate(savedInstanceState);
     setContentView(R.layout.activity_fullscreen);

     toolbar = (Toolbar) findViewById(R.id.tool_bar);
     setSupportActionBar(toolbar);
```

```
        getSupportActionBar().setDisplayShowHomeEnabled(true);
        getSupportActionBar().setDisplayHomeAsUpEnabled(true);

       fullView= (ImageView)  findViewById(R.id.full_screen);
       spinner = (ProgressBar) findViewById(R.id.loading);

       options = new DisplayImageOptions.Builder()
        .showImageOnLoading(R.drawable.img_default)
        .showImageForEmptyUri(R.drawable.img_default)
        .showImageOnFail(R.drawable.img_default)
        .cacheInMemory(true)
        .cacheOnDisk(true)
        .considerExifParams(true)
        .bitmapConfig(Bitmap.Config.RGB_565)
        .build();

       Bundle extras= getIntent().getExtras();
       String url="";
       if(extras!=null){
          url= extras.getString("url");
          Log.d(TAG, "FullScreen URL " + url);
       }
       //imageLoader.DisplayImage(url,  loadingImage,  fullView);
       ImageLoader.getInstance()
        .displayImage("file://"+url,  fullView, options, new
SimpleImageLoadingListener()  {
           @Override
           public void onLoadingStarted(String  imageUri, View view) {
              spinner.setVisibility(View.VISIBLE);
           }
           @Override
           public void onLoadingFailed(String imageUri, View view, FailReason
failReason) {
              spinner.setVisibility(View.GONE);
           }
           @Override
           public void onLoadingComplete(String  imageUri, View view, Bitmap
loadedImage)  {
              spinner.setVisibility(View.GONE);
           }
        }, new ImageLoadingProgressListener()  {
           @Override
           public void onProgressUpdate(String  imageUri, View view,
                 int current, int total) {
           }
        });
    }

    @Override
     public boolean onOptionsItemSelected(MenuItem  item) {
```

```
        switch (item.getItemId()) {
        // Respond to the action bar's Up/Home button
        case android.R.id.home:
            finish();
            return true;
        }
        return super.onOptionsItemSelected(item);
    }

}
```

Voice Command

This activity provides the option to use voice to control the robot.

```
package com.msu.myscribbler;

import java.util.ArrayList;
import java.util.List;
import com.msu.myscribbler.models.AppModel;
import com.msu.myscribbler.utils.Log;
import android.app.Activity;
import android.content.Intent;
import android.content.pm.PackageManager;
import android.content.pm.ResolveInfo;
import android.os.Bundle;
import android.os.Handler;
import android.speech.RecognizerIntent;
import android.support.v7.app.ActionBarActivity;
import android.support.v7.widget.Toolbar;
import android.view.KeyEvent;
import android.view.MenuItem;
import android.view.View;
import android.widget.ArrayAdapter;
import android.widget.Button;
import android.widget.ListView;
import android.widget.Toast;

public class VoiceCommand  extends  ActionBarActivity{
    private static final int REQUEST_CODE = 1234;
    private ListView resultList;
    Button speakButton;

    private Toolbar toolbar;
    private AppModel appState;

    @Override
```

```
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.voice_command);

        toolbar = (Toolbar) findViewById(R.id.tool_bar);
       setSupportActionBar(toolbar);
        getSupportActionBar().setDisplayShowHomeEnabled(true);
        getSupportActionBar().setDisplayHomeAsUpEnabled(true);

        appState = (AppModel) getApplicationContext();

        speakButton = (Button) findViewById(R.id.speakButton);
        resultList = (ListView) findViewById(R.id.list);

        // Disable button if no recognition service is present
        PackageManager pm = getPackageManager();
        List<ResolveInfo> activities = pm.queryIntentActivities(new
Intent(RecognizerIntent.ACTION_RECOGNIZE_SPEECH), 0);

        if (activities.size() == 0) {

            speakButton.setEnabled(false);
            Toast.makeText(getApplicationContext(), "Recognizer Not Found",
Toast.LENGTH_LONG).show();
        }

        speakButton.setOnClickListener(new View.OnClickListener() {

            @Override
            public void onClick(View v) {
                startVoiceRecognitionActivity();
            }
        });
}

    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
      switch (item.getItemId()) {
      // Respond to the action bar's Up/Home button
      case android.R.id.home:
          finish();
          return true;
      }
      return super.onOptionsItemSelected(item);
    }


    /**
    * Starts the intent which communicates with the Google servers
    * to decipher the audio file just recorded.
```

```
    */
    private void startVoiceRecognitionActivity() {
        Intent intent = new
Intent(RecognizerIntent.ACTION_RECOGNIZE_SPEECH);
        intent.putExtra(RecognizerIntent.EXTRA_LANGUAGE_MODEL,Recognize
rIntent.LANGUAGE_MODEL_FREE_FORM);
        intent.putExtra(RecognizerIntent.EXTRA_PROMPT, "Speak now...");
        startActivityForResult(intent, REQUEST_CODE);
    }

    /**
    * Once the startVoiceRecognitionActivity() has finished, this function
    * takes all of the Strings (presented in an ArrayList) and then
    * figures out if the user said forward, backward, turn left, turn right,
    * or beep. If one of those conditions are met, then the Scribbler performs
    * the action. If not, there is a Toast print with an error telling the user
    * what was just said was unknown.
    */
    @Override
    protected void onActivityResult(int requestCode, int resultCode, Intent data) {
        if (requestCode == REQUEST_CODE && resultCode == RESULT_OK) {
            ArrayList<String> matches =
data.getStringArrayListExtra(RecognizerIntent.EXTRA_RESULTS);

            resultList.setAdapter(new
ArrayAdapter<String>(this,android.R.layout.simple_list_item_1, matches));

            for(int i = 0; i < matches.size(); i++){

                // FORWARD
                if(matches.get(i).trim().equalsIgnoreCase("forward")){

                    appState.getScribbler().forward(0.5);
                    final Handler handler= new Handler();
                    handler.postDelayed(new Runnable(){
                        @Override
                        public void run() {
                            appState.getScribbler().stop();
                        }
                    },500);


                }
                else if(matches.get(i).trim().equalsIgnoreCase("backward")){

                    appState.getScribbler().backward(0.5);
                    final Handler handler= new Handler();
                    handler.postDelayed(new Runnable(){
                        @Override
                        public void run() {
```

```
                    appState.getScribbler().stop();
                }
            },500);

        }
        else if(matches.get(i).trim().equalsIgnoreCase("turn  Left")){

            appState.getScribbler().turnLeft(0.5);
            final Handler handler= new Handler();
            handler.postDelayed(new  Runnable(){
                @Override
                public void run() {
                    appState.getScribbler().stop();
                }
            },500);

        }
        else if(matches.get(i).trim().equalsIgnoreCase("turn  Right")){

            appState.getScribbler().turnRight(0.5);
            final Handler handler= new Handler();
            handler.postDelayed(new  Runnable(){
                @Override
                public void run() {
                    appState.getScribbler().stop();
                }
            },500);
        }
        else if(matches.get(i).trim().equalsIgnoreCase("Beep")){

            appState.getScribbler().beep(800,  2);
        }
      }
    }

    super.onActivityResult(requestCode,  resultCode, data);

    }

    /**
    * Used to help maintain clean activity across the app. If someone is
connected
    * to the Scribbler, they should disconnect before leaving the app. This
function
    * ensures that happens.
    */
    @Override
   public boolean dispatchKeyEvent(KeyEvent  event)
   {
        if (event.getAction()  == KeyEvent.ACTION_DOWN)  {
```

```
            switch (event.getKeyCode()) {
            case KeyEvent.KEYCODE_HOME:
               if(appState.getScribbler().isConnected()){
                  appState.getScribbler().disconnect();
                  Toast.makeText(getApplicationContext(), "Disconnected from
Scribbler", Toast.LENGTH_LONG).show();
               }
          return true;
            }
         }
         return super.dispatchKeyEvent(event);   // let the default handling take care
of it
   }


}
```

```
package com.msu.myscribbler.adapters;

import java.util.ArrayList;

import com.msu.myscribbler.R;
import com.msu.myscribbler.models.CommandItem;
import com.msu.myscribbler.utils.Log;

import android.content.Context;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.BaseAdapter;
import android.widget.TextView;

public class CommandItemAdapter extends BaseAdapter{
   private static final String TAG="CommandItemAdapter";

   ArrayList<CommandItem> myList=new ArrayList<CommandItem>();
   LayoutInflater inflater;
   Context context;

   public CommandItemAdapter(Context context, ArrayList<CommandItem>
myList){
      this.myList=myList;
      this.context=context;
      inflater=LayoutInflater.from(this.context);
   }

   public void addItem(CommandItem item){
```

```
        myList.add(item);
    }

    public void removeItem(int position){
        myList.remove(position);
    }

    public void removeAll(){
        myList.clear();
    }

    public ArrayList<CommandItem> getAllItems(){
        return myList;
    }

    @Override
    public int getCount() {

        return myList.size();
    }

    @Override
    public Object getItem(int position) {

        return myList.get(position);
    }

    @Override
    public long getItemId(int position) {

        return 0;
    }

    @Override
    public View getView(int position, View convertView, ViewGroup parent) {
        MyViewHolder mViewHolder;
        if(convertView==null){
            convertView= inflater.inflate(R.layout.command_list_items, null);
            mViewHolder=new MyViewHolder();
            mViewHolder.commandView=
(TextView)convertView.findViewById(R.id.commands_view);
            mViewHolder.timeView=(TextView)convertView.findViewById(R.id.time_
view);
            convertView.setTag(mViewHolder);
        }else{
            mViewHolder=(MyViewHolder)convertView.getTag();

        }
        CommandItem rowItem= myList.get(position);
        if(rowItem!=null){
```

```
            mViewHolder.commandView.setText(rowItem.getmCommand());
            // check if the command  item is associated with time
            if(rowItem.isHasTime()){
               Log.d(TAG, "ITEM HAS TIME");
               mViewHolder.timeView.setVisibility(View.VISIBLE);
               mViewHolder.timeView.setText(rowItem.getmTime());
            }else{
               Log.d(TAG, "ITEM DOESN'T HAVE TIME");
               mViewHolder.timeView.setVisibility(View.GONE);
            }

         }
         return convertView;
      }


      private class MyViewHolder {
         TextView commandView;
         TextView timeView;
      }


}
```

```
package com.msu.myscribbler.adapters;

import com.msu.myscribbler.fragments.GalleryFragment;
import com.msu.myscribbler.fragments.HomeFragment;
import com.msu.myscribbler.fragments.InfoFragment;

import android.support.v4.app.Fragment;
import android.support.v4.app.FragmentManager;
import android.support.v4.app.FragmentPagerAdapter;

public class TabsPagerAdapter extends FragmentPagerAdapter {

   private static final int NUMBER_OF_TABS=3;


   public TabsPagerAdapter(FragmentManager fm) {
      super(fm);

   }

   @Override
   public Fragment getItem(int index) {
```

```
        switch(index){
        case 0:
            return new HomeFragment();

        case 1:
            return new InfoFragment();

        case 2:
            return new GalleryFragment();


        }

        return null;
    }

    @Override
    public int getCount() {

        return NUMBER_OF_TABS;
    }




}



/*
* Code taken from project: https://github.com/sahhhm/Scribdroid
*/

package com.msu.myscribbler.commands;

import com.msu.myscribbler.utils.Log;



public class GetCommands {

 private static final String TAG = "GetCommands";
 private static final boolean D = true;

 private Scribbler s;

 private static final int PACKET_LENGTH = 9;

 private static final int GET_ALL = 65;
 private static final int GET_LIGHT_ALL = 70;
```

```
 private static final int GET_IR_ALL = 73;
 private static final int GET_NAME1 = 78;
 private static final int GET_NAME2 = 64;
 private static final int GET_IMAGE = 83;
 private static final int GET_DONGLE_L_IR = 85;
 private static final int GET_DONGLE_C_IR = 86;
 private static final int GET_DONGLE_R_IR = 87;
 private static final int GET_BATTERY = 89;

 public GetCommands(Scribbler aScrib) {
   s = aScrib;
 }

 public byte[] getPictureArray() {
   byte[] line;

   int width = 1280;
   int height = 800;
   int size = width * height;

   ReadWrite
       ._writeFluke(s.getSocket(), s.isConnected(), new byte[] { (byte)
GET_IMAGE });

   /** READS THE INPUTSTREAM TO GET IMAGE BYTES **/
   line = ReadWrite._read(s.getSocket(), s.isConnected(), size);
   //line = ReadWrite._readImage(s.getSocket(), s.isConnected(), size);
   if (D) Log.d(TAG, "Finished Reading--getArray");

   return line;
 }

 /**
  * Function which returns all the scribbler sensors (ir, light, line, stall)
  *
  * @return an int[] of size 8. idx[0,1]: ir left, ir right... idx[2,3,4]:
  *         light left, light center, light right... idx[5,6]: line left, line
  *         right... idx[7]: stall
  */
 public int[] getAll() {
   int[] temp;
   int numBytes = 11;
   int[] values = new int[8];

   // Get the Raw Bytes
   temp = _get(new byte[] { (byte) GET_ALL }, numBytes, "byte");
   if (temp == null) return null;
   Log.d(TAG, "All Values: "+ int2str(temp));
   // IR Values
   values[0] = temp[0];
```

```
  values[1] = temp[1];

  // Light Values
  values[2] = (temp[2] & 0xFF) << 8 | temp[3] & 0xFF;
  values[3] = (temp[4] & 0xFF) << 8 | temp[5] & 0xFF;
  values[4] = (temp[6] & 0xFF) << 8 | temp[7] & 0xFF;

  // Line Values
  values[5] = temp[8];
  values[6] = temp[9];

  // Stall Value
  values[7] = temp[10];

  return values;
}

/**
 *
 * @return - byte[] consisting of the 16 bytes that make up the robot name
 */
public int[] getName() {
  int[] ba, ba1, ba2;
  int retSize = 8;

  // Get both halves of the name
  ba1 = _get(new byte[] { (byte) GET_NAME1 }, retSize, "byte");
  ba2 = _get(new byte[] { (byte) GET_NAME2 }, retSize, "byte");

  // Combine both halves of the name
  ba = new int[ba1.length + ba2.length];
  System.arraycopy(ba1,  0, ba, 0, ba1.length);
  System.arraycopy(ba2,  0, ba, ba1.length, ba2.length);
  return ba;
}

public byte[] getBattery() {
  byte[] battval;
  int retSize = 2;

  ReadWrite._writeFluke(s.getSocket(),  s.isConnected(),
     new byte[] { (byte) GET_BATTERY });
  battval = ReadWrite._read(s.getSocket(),  s.isConnected(), retSize);

  if (D) Log.i(TAG, "Battery Done");
  return battval;

}

/**
```

```
 * Returns the left, center, right obstacle values of the robot. Note: this is
 * a FLUKE command.
 *
 * @return An integer array of size 3 containing left, center, right obstacle
 *         values
 */
public int[] getObstacle() {
  int[] obstacleValues = new int[3];
  int[] obstacleCodes = { GET_DONGLE_L_IR, GET_DONGLE_C_IR,
GET_DONGLE_R_IR };
  int retSize = 2;
  byte[] temp;

  // Populate the left, center, right obstacle values
  for (int i = 0; i < obstacleCodes.length; i++) {
    ReadWrite._writeFluke(s.getSocket(), s.isConnected(),
        new byte[] { (byte) obstacleCodes[i] });
    temp = ReadWrite._read(s.getSocket(), s.isConnected(), retSize);
    obstacleValues[i] = temp[0] << 8 & 0xFF | temp[1] & 0xFF;
  }

  Log.d(TAG, "Obstacle Return "+int2str(obstacleValues));
  return obstacleValues;
}

public int[] getIR() {
  int[] ba = null;
  int numBytes = 2;

  ba = _get(new byte[] { (byte) GET_IR_ALL }, numBytes, "byte");

  if (D) Log.i(TAG, "IR Done");
  return ba;
}

public int[] getLight() {
  int[] ba = null;
  int numBytes = 6;

  ba = _get(new byte[] { (byte) GET_LIGHT_ALL }, numBytes, "word");

  if (D) Log.i(TAG, "LIGHT Done");
  return ba;
}

/**
 * Function that gets contents of a request to the robot
 *
 * @param ba
 *        Byte array containing the message to be sent to the robot
```

```
 * @param numBytes
 *          the number of bytes in the robots message
 * @param getType
 *          the way the read bytes need to be formatted (byte or word)
 * @return an int[] containing the requested message from the robot
 */
private int[] _get(byte[] ba, int numBytes, String getType) {
  int[] ret;
  byte[] temp;

  // Write Message
  ReadWrite._write(s.getSocket(),  s.isConnected(), ba);

  try {
    Thread.sleep(200);
  } catch (InterruptedException  e) {
    // TODO Auto-generated catch block
    e.printStackTrace();
  }

  // Read the Message Echo
  temp = ReadWrite._read(s.getSocket(),  s.isConnected(), PACKET_LENGTH);
  if (temp == null) Log.d(TAG,"TEMP IS NULL");
  if (D) Log.d(TAG, "ECHO READ:  " + temp.length  + " -> " + ba2s(temp));

  // Read contents of what's desired
  temp = ReadWrite._read(s.getSocket(),  s.isConnected(), numBytes);

  // Convert the received bytes if needed
  if (getType.toLowerCase().equals("byte"))  {
    ret = new int[temp.length];
    for (int i = 0; i < ret.length; i++) {
      ret[i] = temp[i];
    }
  } else if (getType.toLowerCase().equals("word"))  {
    int c = 0;
    ret = new int[numBytes  / 2];

    if (D) Log.d(TAG, "GET[before word modify]: " + ba2s(temp));

    for (int i = 0; i < numBytes; i = i + 2) {
      ret[c] = (temp[i] & 0xFF) << 8 | temp[i + 1] & 0xFF;
      c++;
    }
  } else {
    if (D) Log.e(TAG, "Cannot _get type: " + getType);
    ret = new int[] {};
  }

  if (D) Log.d(TAG, "GET: " + int2str(ret));
```

```
    return ret;
  }

  private static String ba2s(byte[] ba) {
    StringBuilder sb = new StringBuilder("[ ");
    for (byte b : ba) {
      sb.append(Integer.toHexString(b  & 0xff)).append(" ");
    }
    sb.append("]");
    return sb.toString();
  }

  private static String int2str(int[] ba) {
    StringBuilder sb = new StringBuilder("[ ");
    for (int b : ba) {
      sb.append(Integer.toHexString(b)).append("  ");
    }
    sb.append("]");
    return sb.toString();
  }
}


package com.msu.myscribbler.commands;

import java.io.ByteArrayInputStream;
import java.io.ByteArrayOutputStream;
import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
import java.nio.BufferOverflowException;
import java.nio.ByteBuffer;
import java.nio.IntBuffer;

import com.msu.myscribbler.utils.Log;

import android.bluetooth.BluetoothSocket;


public class ReadWrite {
  private static final String TAG = "ReadWrite";
  private static final boolean D = true;

  private static final int PACKET_LENGTH = 9;

  public static void _write(BluetoothSocket sock, boolean connected, byte[]
values) {
    if (connected) {
      OutputStream out = null;
      try {
```

```
      out = sock.getOutputStream();
    } catch (IOException e) {
      if (D) Log.e(TAG, "Error getting output stream!");
    }

    // Ensure correctly-sized packet
    ByteBuffer b = ByteBuffer.allocate(PACKET_LENGTH).put(values);
    while (b.position() < b.limit()) {
      b.put((byte) 0);
    }

    try {
      out.write(b.array());
      if (D) Log.d(TAG, "Wrote: " + ba2s(b.array()));
    } catch (IOException e) {
      if (D) Log.e(TAG, "Error Writing: " + ba2s(b.array()));
    }

  }
 }

 public static void _writeFluke(BluetoothSocket sock, boolean connected, byte[]
values) {
   if (connected) {
     OutputStream out = null;
     try {
       out = sock.getOutputStream();
     } catch (IOException e) {
       if (D) Log.e(TAG, "Error getting output stream!");
     }

     // Only write what the code-- no message size specified
     ByteBuffer b = ByteBuffer.allocate(values.length).put(values);

     try {
       out.write(b.array());
       if (D) Log.d(TAG, "Wrote[Fluke]: " + ba2s(b.array()));
     } catch (IOException e) {
       if (D) Log.e(TAG, "Error Writing: " + ba2s(b.array()));
     }
   }
 }

 public static byte[] _read(BluetoothSocket sock, Boolean connected, int
numBytes) {
   ByteBuffer buf = ByteBuffer.allocate(numBytes);

   if (connected) {
     InputStream in = null;
     try {
```

```
      in = sock.getInputStream();
    } catch (IOException e1) {
      if (D) Log.d(TAG, "Error Opening input stream");
    }

    byte[] fake = new byte[numBytes];
    for (int i = 0; i < numBytes; i++)
      fake[i] = 0;

    byte[] buffer = new byte[numBytes];
    Log.d(TAG, "BUFFER SIZE: "+String.valueOf(numBytes));
    int read = 0;

    try {
      while (buf.hasRemaining())  {
        read = in.read(buffer);
        if(D) Log.i(TAG, "Read Int: "+String.valueOf(read));
        for (int i = 0; i < read; i++) {
          int b = buffer[i] & 0xff;
          //Log.d(TAG, "Int b: "+String.valueOf(b));
          try {
              buf.put((byte) b);
          } catch (BufferOverflowException be) {
              be.printStackTrace();
              Log.d(TAG, "buf position "+String.valueOf(buf.position()) );
              // Fix bug where robot sends too much information back
              //if (D) Log.e(TAG, be.getMessage().toString());
              if (D) Log.i(TAG, "Trying to gracefully disconnect rather than crash");
              return buf.array();
              }
           }
      }
      if (D) Log.d(TAG, "Read " + buf.position() + " bytes: " + ba2s(buf.array()));
    } catch (IOException e) {
      if (D) Log.e(TAG, "Error Reading" + e.getMessage());
    }
  }
  return buf.array();
 }

 /** Reading picture byte **/
 public static byte[] _readImage(BluetoothSocket sock, Boolean connected, int
numBytes) {
    //ByteBuffer buf = ByteBuffer.allocate(0x20000);
    ByteArrayOutputStream outStream=new ByteArrayOutputStream();
    int bytesRead =0;
    int size=5120;
    byte[]buf = new byte[size];
    byte[]data=new byte[0];
      if (connected) {
```

```
 InputStream in = null;
 try {
  in = sock.getInputStream();
  if(in instanceof ByteArrayInputStream){
     Log.d(TAG, "instance of bytearrayinputstream");
     /*size=in.available();
     buf=new byte[size];
     len=in.read(buf,0,size);*/
  }else{
     byte[]newData=new byte[data.length+bytesRead];
     // copy data previously read
     System.arraycopy(data,  0, newData, 0, data.length);
     // append data newly read
     System.arraycopy(buf,  0, newData, data.length, bytesRead);
     // discard the old array in favour of the new one
     data = newData;
  }
 } catch (IOException  e1) {
  if (D) Log.d(TAG, "Error Opening input stream");
 }


/*  byte[] fake = new byte[numBytes];
 for (int i = 0; i < numBytes; i++)
  fake[i] = 0;

 byte[] buffer = new byte[numBytes];
 Log.d(TAG, "BUFFER SIZE: "+String.valueOf(numBytes));
 int read = 0;

 try {
  while (buf.hasRemaining()) {
   read = in.read(buffer);
   if(D) Log.i(TAG, "Read Int: "+String.valueOf(read));
   for (int i = 0; i < read; i++) {
    int b = buffer[i] & 0xff;
    //Log.d(TAG, "Int b: "+String.valueOf(b));
    try {
       buf.put((byte) b);
    } catch (BufferOverflowException  be) {
       be.printStackTrace();
       Log.d(TAG, "buf position "+String.valueOf(buf.position())  );
       // Fix bug where robot sends too much information back
       //if (D) Log.e(TAG, be.getMessage().toString());
       if (D) Log.i(TAG, "Trying to gracefully disconnect rather than crash");
       return buf.array();
       }
     }
  }
  if (D) Log.d(TAG, "Read " + buf.position() + " bytes: " + ba2s(buf.array()));
```

```
        } catch (IOException e) {
          if (D) Log.e(TAG, "Error Reading" + e.getMessage());
        }*/
      }
      return data;
    }


 private static String ba2s(byte[] ba) {
   StringBuilder sb = new StringBuilder("[ ");
   for (byte b : ba) {
     sb.append(Integer.toHexString(b  & 0xff)).append(" ");
   }
   sb.append("]");
   return sb.toString();
 }
}
```

```
package com.msu.myscribbler.commands;

import java.io.IOException;
import java.lang.reflect.Method;
import java.util.HashMap;
import android.annotation.SuppressLint;
import android.bluetooth.BluetoothAdapter;
import android.bluetooth.BluetoothDevice;
import android.bluetooth.BluetoothSocket;
import android.graphics.Bitmap;
import android.graphics.Color;

import com.msu.myscribbler.commands.SetCommands.LED;
import com.msu.myscribbler.utils.Log;

@SuppressLint("DefaultLocale")
public class Scribbler {
 // Debugging
 private static final String TAG = "Scribbler";
 private static final boolean D = true;

 private String macAddress;
 private boolean connected;
 private byte[] lastSensors;
 private SetCommands setCommands;
 private GetCommands getCommands;
 private BluetoothSocket sock;
 private boolean isMoving;
```

```
 //private static Scribbler

 public Scribbler() {
   this(null);
 }

 public Scribbler(String aMac) {
   macAddress = aMac;
   setConnected(false);
   sock = null;
   setCommands = null;
   getCommands = null;
 }

 public boolean connect() throws Exception {
   boolean ret = false;
   BluetoothDevice scrib =
BluetoothAdapter.getDefaultAdapter().getRemoteDevice(
       macAddress);
   Method m = scrib.getClass()
       .getMethod("createRfcommSocket", new Class[] { int.class });
   sock = (BluetoothSocket) m.invoke(scrib, Integer.valueOf(1));
   if (D) Log.d(TAG, "Connecting");
   try {
     sock.connect();
     setCommands = new SetCommands(this);
     getCommands = new GetCommands(this);
     setConnected(true);
     ret = true;
     if (D) Log.d(TAG, "Connected");
   } catch (Exception e) {
     setConnected(false);
     if (D) Log.e(TAG, "Error Connecting");
     try {
       sock.close();
     } catch (Exception e2) {
       if (D) Log.e(TAG, "Error closing socket after error connecting");
     }
   }
   return ret;
 }

 public boolean disconnect() {
     boolean result=false;
   try {
     if (sock != null) {
       sock.close();
       setConnected(false);
       result=true;
```

```
      Log.d(TAG, "Socket Closed. Now Disconnected.");
    }
  } catch (IOException e) {
    Log.e(TAG, "Error closing socket while disconnecting");
  }
  return result;
}

/**
 * @param connected
 *          the connected to set
 */
public void setConnected(boolean connected) {
  this.connected = connected;
}

/**
 * @return the connected
 */
public boolean isConnected() {
  return connected;
}

public void setSocket(BluetoothSocket aSock) {
  sock = aSock;
}

public BluetoothSocket getSocket() {
  return sock;
}

/**
 * @param lastSensors
 *          the lastSensors to set
 */
public void setLastSensors(byte[] lastSensors) {
  this.lastSensors = lastSensors;
}

/**
 * @return the lastSensors
 */
public byte[] getLastSensors() {
  return lastSensors;
}

public void beep(float frequency, float duration) {
  if (setCommands != null)
    setCommands._setSpeaker((int) frequency, (int) (duration * 1000));
}
```

```
public void beep(float frequency1, float frequency2, float duration) {
  if (setCommands != null)
    setCommands._setSpeaker2((int)  frequency1, (int) frequency2,
        (int) (duration * 1000));
}

public void setLED(LED position, boolean on) {
  if (setCommands != null) setCommands._setLED(position,  on);
}

public void move(double translate, double rotate) {
  if (setCommands != null) setCommands._move(translate,  rotate);
}

public void turnLeft(double amount) {
  if (setCommands != null) setCommands._move(0,  -amount);
}

public void turnRight(double amount) {
  if (setCommands != null) setCommands._move(0,  amount);
}

public void forward(double amount) {
   // synchronized(setCommands){
        if (setCommands != null) setCommands._move(amount,  0);
    //}

}

public void backward(double amount) {
   // synchronized(setCommands){
        if (setCommands != null) setCommands._move(-amount,  0);
    //}

}

public void stop() {
      if (setCommands != null) setCommands._move(0,  0);
    }

public void setMoving(boolean isMoving) {
  this.isMoving = isMoving;
}

public boolean isMoving() {
  return isMoving;
}

public Bitmap takePicture(int paramInt1,  int paramInt2) {
```

```
  Bitmap bm = null;
  byte[] ba;

  if (getCommands != null) {
    ba = getCommands.getPictureArray();

    bm = Bitmap.createBitmap(paramInt1, paramInt2,
Bitmap.Config.ARGB_8888);
    int w = 1280;
    int h = 800;
    int vy, vu, y1v, y1u, uy, uv, y2u, y2v;
    int V = 0, Y = 0, U = 0;

    for (int i = 0; i < h; i++) {
      for (int j = 0; j < w; j++) {
        if (j >= 3) {
          vy = -1;
          vu = 2;
          y1v = -1;
          y1u = -3;
          uy = -1;
          uv = -2;
          y2u = -1;
          y2v = -3;
        } else {
          vy = 1;
          vu = 2;
          y1v = 3;
          y1u = 1;
          uy = 1;
          uv = 2;
          y2u = 3;
          y2v = 1;
        }
        if ((j % 4) == 0) {
          V = ba[i * w + j] & 0xff;
          Y = ba[i * w + j + vy] & 0xff;
          U = ba[i * w + j + vu] & 0xff;
        } else if ((j % 4) == 1) {
          Y = ba[i * w + j] & 0xff;
          V = ba[i * w + j + y1v] & 0xff;
          U = ba[i * w + j + y1u] & 0xff;
        } else if ((j % 4) == 2) {
          U = ba[i * w + j] & 0xff;
          Y = ba[i * w + j + uy] & 0xff;
          V = ba[i * w + j + uv] & 0xff;
        } else if ((j % 4) == 3) {
          Y = ba[i * w + j] & 0xff;
          U = ba[i * w + j + y2u] & 0xff;
          V = ba[i * w + j + y2v] & 0xff;
```

```
        }
        U = U - 128;
        V = V - 128;
        // Y = Y;

        bm.setPixel(j,  i, Color.rgb((int) Math.max(Math.min(Y  + 1.13983 * V, 255),
0),
            (int) Math.max(Math.min(Y  - 0.39466 * U - 0.58060 * V, 255), 0),
            (int) Math.max(Math.min(Y  + 2.03211 * U, 255), 0)));
      }
    }
  }
        /*localBitmap=Bitmap.createBitmap(paramInt1,paramInt2,Bitmap.Config.A
RGB_8888);
        int i = 0;
        int j = 0;
        int k = 0;
        int n;
        for (int m = 0;; m++){
            if (m >= paramInt2){
                return localBitmap;
            }
            n = 0;
            if (n < paramInt1){
                break;
            }
        }
        int i1;
      int i2;
      int i3;
      int i4;
      int i5;
      int i6;
      int i7;
      if (n >= 3){
        i1 = -1;
        i2 = -1;
        i3 = -3;
        i4 = -1;
        i5 = -2;
        i6 = -1;
        i7 = -3;
        label78:
        if (n % 4 != 0) {
          break label256;
        }
        i = 0xFF & paramArrayOfByte[(n  + m * paramInt1)];
        j = 0xFF & paramArrayOfByte[(i1  + (n + m * paramInt1))];
        k = 0xFF & paramArrayOfByte[(2  + (n + m * paramInt1))];
      }
```

```
      for (;;)
      {
        k -= 128;
        i -= 128;
        localBitmap.setPixel(n,  m, Color.rgb((int)Math.max(Math.min(j  + 1.13983D
* i, 255.0D), 0.0D), (int)Math.max(Math.min(j  - 0.39466D * k - 0.5806D * i,
255.0D), 0.0D), (int)Math.max(Math.min(j  + 2.03211D * k, 255.0D), 0.0D)));
        n++;
        break;
        i1 = 1;
        i2 = 3;
        i3 = 1;
        i4 = 1;
        i5 = 2;
        i6 = 3;
        i7 = 1;
        break label78;
        label256:
        if (n % 4 == 1)
        {
          j = 0xFF & paramArrayOfByte[(n  + m * paramInt1)];
          i = 0xFF & paramArrayOfByte[(i2  + (n + m * paramInt1))];
          k = 0xFF & paramArrayOfByte[(i3  + (n + m * paramInt1))];
        }
        else if (n % 4 == 2)
        {
          k = 0xFF & paramArrayOfByte[(n  + m * paramInt1)];
          j = 0xFF & paramArrayOfByte[(i4  + (n + m * paramInt1))];
          i = 0xFF & paramArrayOfByte[(i5  + (n + m * paramInt1))];
        }
        else if (n % 4 == 3)
        {
          j = 0xFF & paramArrayOfByte[(n  + m * paramInt1)];
          k = 0xFF & paramArrayOfByte[(i6  + (n + m * paramInt1))];
          i = 0xFF & paramArrayOfByte[(i7  + (n + m * paramInt1))];
        }
      }*/




  return bm;
}

public float getBattery() {
  byte[] ba = null;
  int unmodified;
  float value = 0;
```

```
  if (getCommands != null) {
    ba = getCommands.getBattery();
      unmodified = (ba[0] & 0xFF) << 8 | ba[1] & 0xFF;
      value = unmodified / 20.9813f;

    if (D) Log.d(TAG, "getBattery -> " + value);
  }
  return value;
 }

 @SuppressLint("DefaultLocale")
public int[] getIR(String type) {
  int[] ba, ret = null;

  type = type.toLowerCase();
  if (getCommands != null) {
    ba = getCommands.getIR();
    if (type.equals("left")) {
      ret = new int[1];
      ret[0] = ba[0];
    } else if (type.equals("right")) {
      ret = new int[1];
      ret[0] = ba[1];
    } else {
      ret = ba;
    }
  }
  return ret;
 }

 @SuppressLint("DefaultLocale")
public int[] getLight(String type) {
  int[] ba, ret = null;

  type = type.toLowerCase();
  if (getCommands != null) {
    ba = getCommands.getLight();
    if (type.equals("left")) {
      ret = new int[1];
      ret[0] = ba[0];
    } else if (type.equals("center")) {
      ret = new int[1];
      ret[0] = ba[1];
    } else if (type.equals("right")) {
      ret = new int[1];
      ret[0] = ba[2];
    } else {
      ret = ba;
    }
  }
```

```
  return ret;
 }

@SuppressLint("DefaultLocale")
public int[] getObstacle(String type) {
  int[] ba, ret = null;

  type = type.toLowerCase();
  if (getCommands != null) {
    ba = getCommands.getObstacle();

    if (type.equals("left")) {
      ret = new int[1];
      ret[0] = ba[0];
    } else if (type.equals("center")) {
      ret = new int[1];
      ret[0] = ba[1];
    } else if (type.equals("right")) {
      ret = new int[1];
      ret[0] = ba[2];
    } else {
      ret = ba;
    }
  }

  return ret;
}

 /**
  * Function properly gets and converts the robots name
  *
  * @return - String representing the robots trimmed name
  */
 public String getName() {
  int[] ba;
  StringBuilder build;
  String name = null;

  if (getCommands != null) {
    // Get the proper bytes and convert them to characters
    ba = getCommands.getName();

    build = new StringBuilder(ba.length);

    for (int i = 0; i < ba.length; i++)
      build.append((char) ba[i]);
    name = build.toString().trim();

  }
  if (D) Log.d(TAG, "Scribbler Name Read: " + name);
```

```
  return name;
 }

 /**
  * Function returns all of the scribbler sensors (not fluke) in a hashmap.
  * Each associated array for a key is of variable length, depending on the
  * type of sensor. IR array has size 2; LIGHT array has size 3; LINE array has
  * size 2; STALL array has size 1;
  *
  * @return hashmap with keys: "IR", "LINE", "LIGHT", and "STALL"
  */
 public HashMap<String, int[]> getAll() {
   HashMap<String, int[]> hm = new HashMap<String, int[]>();
   int[] v;

   if (getCommands != null) {
     v = getCommands.getAll();
     hm.put("IR",  new int[] { v[0], v[1] });
     hm.put("LIGHT",  new int[] { v[2], v[3], v[4] });
     hm.put("LINE",  new int[] { v[5], v[6] });
     hm.put("STALL",  new int[] { v[7] });
   }
   return hm;
 }
}



/*
* Code taken from project: https://github.com/sahhhm/Scribdroid
*/

package com.msu.myscribbler.commands;

public class SetCommands {
 private Scribbler s;

 private static final int PACKET_LENGTH = 9;
 private static final int SENSOR_PACKET_SIZE = 11;

 private static final int SET_SPEAKER = 113;
 private static final int SET_SPEAKER_2 = 114;
 private static final int SET_LED_LEFT_ON = 99;
 private static final int SET_LED_LEFT_OFF = 100;
 private static final int SET_LED_CENTER_ON = 101;
 private static final int SET_LED_CENTER_OFF = 102;
 private static final int SET_LED_RIGHT_ON = 103;
 private static final int SET_LED_RIGHT_OFF = 104;
 private static final int SET_LED_ALL_ON = 105;
 private static final int SET_LED_ALL_OFF = 106;
```

```
 private static final int SET_MOTORS_OFF = 108;
 private static final int SET_MOTORS = 109;

 public SetCommands(Scribbler aScrib) {
   s = aScrib;
 }

 public void _set(byte[] values) {
   byte[] ba = null;

   ReadWrite._write(s.getSocket(), s.isConnected(), values);

   try {
     Thread.sleep(200);
   } catch (InterruptedException e) {
     // TODO Auto-generated catch block
     e.printStackTrace();
   }

   ba = ReadWrite._read(s.getSocket(), s.isConnected(), PACKET_LENGTH);

   ba = ReadWrite._read(s.getSocket(), s.isConnected(),
SENSOR_PACKET_SIZE);
   s.setLastSensors(ba);

 }

 public void _setSpeaker(int frequency, int duration) {
   byte[] values = { ((byte) SET_SPEAKER), (byte) (duration >> 8),
       (byte) (duration % 256), (byte) (frequency >> 8), (byte) (frequency % 256) };

   _set(values);
 }

 public void _setSpeaker2(int frequency1, int frequency2, int duration) {
   byte[] values = { ((byte) SET_SPEAKER_2), (byte) (duration >> 8),
       (byte) (duration % 256), (byte) (frequency1 >> 8), (byte) (frequency1 %
256),
       (byte) (frequency2 >> 8), (byte) (frequency2 % 256) };

   _set(values);
 }

 public void _move(double translate, double rotate) {
   double left = Math.min(Math.max(translate - rotate, -1), 1);
   double right = Math.min(Math.max(translate + rotate, -1), 1);

   byte[] values = { ((byte) SET_MOTORS), (byte) ((left + 1) * 100), // Need
                                                                     // this
                                                                     // math
```

```
                                                    // to
      (byte) ((right + 1) * 100) // get 0, 100, or 200
    };
    _set(values);
  }

  public void _stop() {
    _set(new byte[] { (byte) SET_MOTORS_OFF });
  }

  public void _setLED(LED position, boolean on) {
    byte[] ba;
    switch (position) {
    case ALL:
      ba = on ? new byte[] { (byte) SET_LED_ALL_ON }
          : new byte[] { (byte) SET_LED_ALL_OFF };
      break;
    case LEFT:
      ba = on ? new byte[] { (byte) SET_LED_LEFT_ON }
          : new byte[] { (byte) SET_LED_LEFT_OFF };
      break;
    case CENTER:
      ba = on ? new byte[] { (byte) SET_LED_CENTER_ON }
          : new byte[] { (byte) SET_LED_CENTER_OFF };
      break;
    case RIGHT:
      ba = on ? new byte[] { (byte) SET_LED_RIGHT_ON }
          : new byte[] { (byte) SET_LED_RIGHT_OFF };
      break;
    default:
      ba = new byte[] { (byte) SET_LED_ALL_OFF };
      break;
    }
    _set(ba);
  }

  public enum LED {
    ALL, LEFT, CENTER, RIGHT
  }
}


package com.msu.myscribbler.fragments;

import java.io.File;
import java.util.ArrayList;

import com.msu.myscribbler.FullScreenActivity;
import com.msu.myscribbler.MainActivity;
import com.msu.myscribbler.R;
```

```
import com.msu.myscribbler.R.id;
import com.msu.myscribbler.R.layout;
import com.msu.myscribbler.interfaces.GalleryFragmentCallback;
import com.msu.myscribbler.utils.Log;
import com.nostra13.universalimageloader.core.DisplayImageOptions;
import com.nostra13.universalimageloader.core.ImageLoader;
import com.nostra13.universalimageloader.core.assist.FailReason;
import
com.nostra13.universalimageloader.core.listener.ImageLoadingProgressListene
r;
import
com.nostra13.universalimageloader.core.listener.SimpleImageLoadingListener;

import android.content.Context;
import android.content.Intent;
import android.graphics.Bitmap;
import android.graphics.BitmapFactory;
import android.os.Bundle;
import android.support.annotation.Nullable;
import android.support.v4.app.Fragment;
import android.view.LayoutInflater;
import android.view.View;
import android.view.View.OnClickListener;
import android.view.ViewGroup;
import android.widget.AdapterView;
import android.widget.AdapterView.OnItemLongClickListener;
import android.widget.BaseAdapter;
import android.widget.GridView;
import android.widget.ImageView;

public class GalleryFragment extends Fragment implements
GalleryFragmentCallback{

    public static final String TAG="GalleryFragment";

    private static final String KEY_TITLE = "title";
   private static final String KEY_INDICATOR_COLOR  = "indicator_color";
   private static final String KEY_DIVIDER_COLOR  = "divider_color";

   DisplayImageOptions  options;

    File[] listFile;
    ArrayList<String>  f = new ArrayList<String>();// list of file paths
    private ImageAdapter  imageAdapter;

    /**
    * @return a new instance of {@link ContentFragment}, adding the parameters
into a bundle and
    * setting them as arguments.
    */
```

```
    public static GalleryFragment newInstance(CharSequence title, int
indicatorColor,
                                      int dividerColor) {
        Bundle bundle = new Bundle();
        bundle.putCharSequence(KEY_TITLE, title);
        bundle.putInt(KEY_INDICATOR_COLOR, indicatorColor);
        bundle.putInt(KEY_DIVIDER_COLOR, dividerColor);

        GalleryFragment fragment = new GalleryFragment();
        fragment.setArguments(bundle);

        return fragment;
    }

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

        options = new DisplayImageOptions.Builder()
        .showImageOnLoading(R.drawable.img_default)
        .showImageForEmptyUri(R.drawable.img_default)
        .showImageOnFail(R.drawable.img_default)
        .cacheInMemory(true)
        .cacheOnDisk(true)
        .considerExifParams(true)
        .bitmapConfig(Bitmap.Config.RGB_565)
        .build();
    }

    @Override
    public void onResume() {
        super.onResume();
        Log.d(TAG, "Resume");
        getFromSdcard();
        imageAdapter.notifyDataSetChanged();
    }

    public void getFromSdcard(){
        File file= new File(MainActivity.imageFilePath);
        f.clear();
        if (file.isDirectory()){
            listFile = file.listFiles();
            for (int i = 0; i < listFile.length; i++){
                f.add(listFile[i].getAbsolutePath());
            }
        }
    }

    @Override
    public View onCreateView(LayoutInflater inflater,
```

```
        @Nullable ViewGroup container, @Nullable Bundle savedInstanceState)
{
      Log.d(TAG, "onCreateView called");
      View rootView= inflater.inflate(R.layout.gallery_fragment,  container,false);
      rootView.setTag(TAG);
      getFromSdcard();
      GridView imagegrid= (GridView) rootView.findViewById(R.id.grid);
      imageAdapter  = new ImageAdapter();
      imagegrid.setAdapter(imageAdapter);

      imagegrid.setOnItemLongClickListener(new  OnItemLongClickListener() {
         @Override
         public boolean onItemLongClick(AdapterView<?>  parent, View view,
               int position, long id) {

            return false;
         }
      });
      return rootView;

      }


   public class ImageAdapter  extends BaseAdapter {
      private LayoutInflater mInflater;

      public ImageAdapter()  {
         //mInflater  = (LayoutInflater)
getSystemService(Context.LAYOUT_INFLATER_SERVICE);
         mInflater=  LayoutInflater.from(getActivity());
      }

      public int getCount() {
         return f.size();
      }

      public Object getItem(int  position) {
         return position;
      }

      public long getItemId(int  position) {
         return position;
      }

      public View getView(final  int position, View convertView,  ViewGroup  parent)
{
         ViewHolder  holder;
         if (convertView  == null) {
            holder = new ViewHolder();
            convertView = mInflater.inflate(R.layout.galleryitem,  null);
```

```
            holder.imageview = (ImageView)
convertView.findViewById(R.id.thumbImage);

            convertView.setTag(holder);
        }
        else {
            holder = (ViewHolder) convertView.getTag();
        }


        //Bitmap myBitmap = BitmapFactory.decodeFile(f.get(position));
        //holder.imageview.setImageBitmap(myBitmap);
        ImageLoader.getInstance()
         .displayImage("file://"+f.get(position), holder.imageview, options, new
SimpleImageLoadingListener() {
            @Override
            public void onLoadingStarted(String imageUri, View view) {

            }
            @Override
            public void onLoadingFailed(String imageUri, View view, FailReason
failReason) {

            }
            @Override
            public void onLoadingComplete(String imageUri, View view, Bitmap
loadedImage) {

            }
         }, new ImageLoadingProgressListener() {
            @Override
            public void onProgressUpdate(String imageUri, View view,
                    int current, int total) {
            }
         });
        holder.imageview.setOnClickListener(new OnClickListener() {
            @Override
            public void onClick(View v) {
                Log.d(TAG, "Image Clicked");
                Intent fullScreenIntent = new Intent(getActivity(),
FullScreenActivity.class);
                fullScreenIntent.putExtra("url", f.get(position));
                getActivity().startActivity(fullScreenIntent);
            }
         });

        return convertView;
    }
  }
  class ViewHolder {
```

```
        ImageView imageview;
    }
    @Override
    public void updateData() {
        Log.d(TAG, "UPDATE DATA METHOD");
        getFromSdcard();
        imageAdapter.notifyDataSetChanged();

    }


}
```

```
package com.msu.myscribbler.fragments;

import com.msu.myscribbler.CommandsGUI;
import com.msu.myscribbler.HomeActivity;
import com.msu.myscribbler.R;
import com.msu.myscribbler.TakePicture;
import com.msu.myscribbler.VoiceCommand;
import com.msu.myscribbler.R.id;
import com.msu.myscribbler.R.layout;
import com.msu.myscribbler.commands.SetCommands.LED;
import com.msu.myscribbler.models.AppModel;
import com.msu.myscribbler.utils.Constants;
import com.msu.myscribbler.utils.Log;

import android.app.Activity;
import android.app.AlertDialog;
import android.content.Context;
import android.content.DialogInterface;
import android.content.Intent;
import android.hardware.Sensor;
import android.hardware.SensorEvent;
import android.hardware.SensorEventListener;
import android.hardware.SensorListener;
import android.hardware.SensorManager;
import android.os.AsyncTask;
import android.os.Bundle;
import android.os.Handler;
import android.support.annotation.Nullable;
import android.support.v4.app.Fragment;

import android.view.LayoutInflater;
import android.view.Menu;
import android.view.MenuInflater;
import android.view.MenuItem;
import android.view.MotionEvent;
```

```
import android.view.View;
import android.view.View.OnClickListener;
import android.view.View.OnLongClickListener;
import android.view.View.OnTouchListener;
import android.view.ViewGroup;
import android.widget.AdapterView;
import android.widget.AdapterView.OnItemSelectedListener;
import android.widget.ArrayAdapter;
import android.widget.Button;
import android.widget.CompoundButton;
import android.widget.CompoundButton.OnCheckedChangeListener;
import android.widget.ImageView;
import android.widget.Spinner;
import android.widget.Switch;
import android.widget.TextView;
import android.widget.Toast;

public class HomeFragment extends Fragment implements
OnItemSelectedListener, OnTouchListener, SensorEventListener{

    private static final String TAG="HomeFragment";

    private static final String KEY_TITLE = "title";
    private static final String KEY_INDICATOR_COLOR = "indicator_color";
    private static final String KEY_DIVIDER_COLOR = "divider_color";

    private AppModel appModel;

    private Spinner mySpinner;
    private String[] entries={"Buttons", "Phone Tilt", "Voice"};

    private Switch leftSwitch, centerSwitch, rightSwitch;

    private SensorManager sensorManager;
    private Sensor acceleroMeter;


    private boolean tiltControl=false;

    private double threshold = 20.0;
    private double negThreshold = -20.0;
    private float x, y;
    private boolean allEnabled = false;
    private boolean upEnabled = false;
    private boolean downEnabled = false;
    private boolean leftEnabled = false;
    private boolean rightEnabled = false;

    private MyCompassView mCustomDrawableView;
```

```
    /**
     * @return a new instance of {@link ContentFragment}, adding the parameters
into a bundle and
     * setting them as arguments.
     */
    public static HomeFragment newInstance(CharSequence title, int
indicatorColor,
                                           int dividerColor) {
        Bundle bundle = new Bundle();
        bundle.putCharSequence(KEY_TITLE, title);
        bundle.putInt(KEY_INDICATOR_COLOR, indicatorColor);
        bundle.putInt(KEY_DIVIDER_COLOR, dividerColor);

        HomeFragment fragment = new HomeFragment();
        fragment.setArguments(bundle);

        return fragment;
    }

    @Override
    public void onCreate(Bundle savedInstanceState) {

        super.onCreate(savedInstanceState);
        Log.d(TAG, "oncreate called");

        setHasOptionsMenu(true);
        mCustomDrawableView= new MyCompassView(getActivity());

        appModel=(AppModel)getActivity().getApplicationContext();
        // initialize sensor manager
        sensorManager=(SensorManager)
getActivity().getSystemService(Context.SENSOR_SERVICE);
        acceleroMeter=sensorManager.getDefaultSensor(Sensor.TYPE_ACCELE
ROMETER);
    }

    @Override
    public void onResume() {

        super.onResume();
        Log.d(TAG, "on Resume");
        sensorManager.registerListener(this, acceleroMeter,
SensorManager.SENSOR_DELAY_UI);
        //sensorManager.registerListener(this, magnetometer,
SensorManager.SENSOR_DELAY_UI);
    }

    @Override
    public void onPause() {
        super.onPause();
```

```
        Log.d(TAG, "on Pause");
        sensorManager.unregisterListener(this);
    }


    @Override
    public void onCreateOptionsMenu(Menu menu, MenuInflater inflater) {
        super.onCreateOptionsMenu(menu, inflater);
        inflater.inflate(R.menu.home_fragment_menu, menu);

    }


    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
        switch(item.getItemId()){
        case R.id.switch_ctrl:
            Log.d(TAG, "SWITCH CONTROL MENU ITEM CLICKED");
            AlertDialog.Builder builder1 = new AlertDialog.Builder(getActivity());
            String options[] = {Constants.BUTTONS,Constants.PHONE_TILT,
Constants.VOICE};
            builder1.setTitle("Choose Control Type");
            builder1.setCancelable(true)
                    .setItems(options, new DialogInterface.OnClickListener() {
                        @Override
                        public void onClick(DialogInterface dialog, int which) {
                            // Stop the Scribbler if it is moving
                            if(appModel.getScribbler()!=null && appModel.isConnected()){
                                try {
                                    appModel.getScribbler().stop();
                                } catch (Exception e) {
                                    e.printStackTrace();
                                }
                            }
                            if(which==0){
                                unregisterSensor();
                            }else if(which==1){
                                Log.d(TAG, "Phone tilt selected");
                                registerSensor();
                            }else if(which==2){
                                unregisterSensor();
                                Intent i= new Intent(getActivity(), VoiceCommand.class);
                                startActivity(i);
                            }
                        }
                    });
            AlertDialog alert1= builder1.create();
            alert1.show();
            return true;
        default:
```

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

    }


    @Override
    public View onCreateView(LayoutInflater  inflater, ViewGroup container,
            Bundle savedInstanceState) {
        Log.d(TAG, "onCreateView called");
        return inflater.inflate(R.layout.home_fragment,  container, false);
    }

    @Override
    public void onViewCreated(View  rootView, Bundle savedInstanceState) {

        super.onViewCreated(rootView,  savedInstanceState);
        Bundle args= getArguments();
        Log.d(TAG, "onViewCreated called");
        Log.d(TAG, "Device Name: "+appModel.getScribbler().getName());
        ImageView leftButton=(ImageView)rootView.findViewById(R.id.left);
        leftButton.setOnTouchListener(this);
        ImageView rightButton=(ImageView)  rootView.findViewById(R.id.right);
        rightButton.setOnTouchListener(this);
        ImageView forwardButton=(ImageView)
rootView.findViewById(R.id.forward);
        forwardButton.setOnTouchListener(this);
        ImageView backwardButton=(ImageView)
rootView.findViewById(R.id.backward);
        backwardButton.setOnTouchListener(this);
        ImageView stopButton=(ImageView)  rootView.findViewById(R.id.stop);
        stopButton.setOnTouchListener(this);
        mySpinner= (Spinner) rootView.findViewById(R.id.controller);
        ArrayAdapter<String>adapter=new  ArrayAdapter<String>(getActivity(),
android.R.layout.simple_spinner_item,entries);
        adapter.setDropDownViewResource(android.R.layout.simple_spinner_drop
down_item);
        mySpinner.setAdapter(adapter);
        mySpinner.setOnItemSelectedListener(this);

        leftSwitch= (Switch) rootView.findViewById(R.id.left_switch);
        leftSwitch.setOnCheckedChangeListener(new
OnCheckedChangeListener() {
            @Override
            public void onCheckedChanged(CompoundButton  buttonView, boolean
isChecked) {
                if(isChecked)
                    appModel.getScribbler().setLED(LED.LEFT,  true);
                else
                    appModel.getScribbler().setLED(LED.LEFT,  false);
```

```
            }
        });

        centerSwitch= (Switch) rootView.findViewById(R.id.center_switch);
        centerSwitch.setOnCheckedChangeListener(new
OnCheckedChangeListener() {
            @Override
            public void onCheckedChanged(CompoundButton  buttonView,
boolean isChecked) {
                if(isChecked)
                    appModel.getScribbler().setLED(LED.CENTER,  true);
                else
                    appModel.getScribbler().setLED(LED.CENTER,
false);
            }
        });

        rightSwitch= (Switch) rootView.findViewById(R.id.right_switch);
        rightSwitch.setOnCheckedChangeListener(new
OnCheckedChangeListener() {
            @Override
            public void onCheckedChanged(CompoundButton  buttonView,
boolean isChecked) {
                if(isChecked)
                    appModel.getScribbler().setLED(LED.RIGHT,  true);
                else
                    appModel.getScribbler().setLED(LED.RIGHT,  false);
            }
        });


        Button takePicture= (Button)rootView.findViewById(R.id.take_picture);
        takePicture.setOnClickListener(new  OnClickListener() {
            @Override
            public void onClick(View v) {
                Intent pictureIntent= new Intent(getActivity(),  TakePicture.class);
                //startActivity(pictureIntent);
                getActivity().startActivityForResult(pictureIntent,
HomeActivity.TAKE_PHOTO);
            }
        });
    }


    /**
    * OnTouch listeners for the buttons
    * Continuously moves the robot according to the button held
    */
    @Override
```

```
    public boolean onTouch(View v, MotionEvent event) {
        Log.d(TAG, "onTouch listener called");
        // Null pointer check for Scribbler and connected status
        if(appModel.getScribbler()!=null && appModel.isConnected()){
            switch(v.getId()){
            case R.id.left:
                if(event.getAction() == MotionEvent.ACTION_SCROLL){
                    return false;
                }else if(event.getAction() == MotionEvent.ACTION_DOWN){
                        Log.d(TAG, "left button pressed");
                        try {
                            appModel.getScribbler().turnLeft(.5);
                        } catch (Exception e) {
                            e.printStackTrace();
                        }
                    return true;
                }
                else if(event.getAction() == MotionEvent.ACTION_UP ||
event.getAction() == MotionEvent.ACTION_CANCEL){
                        Log.d(TAG, "left button released");
                        try {
                            appModel.getScribbler().stop();
                        } catch (Exception e) {
                            e.printStackTrace();
                        }
                    return true;
                }
                break;
            case R.id.right:
                if(event.getAction() == MotionEvent.ACTION_SCROLL){
                    return false;
                }else if(event.getAction() == MotionEvent.ACTION_DOWN){
                    Log.d(TAG, "right button pressed");
                    try {
                        appModel.getScribbler().turnRight(.5);
                    } catch (Exception e) {
                        e.printStackTrace();
                    }
                    return true;
                }else if(event.getAction() == MotionEvent.ACTION_UP ||
event.getAction() == MotionEvent.ACTION_CANCEL){
                    Log.d(TAG, "right button released");
                    try {
                        appModel.getScribbler().stop();
                    } catch (Exception e) {
                        e.printStackTrace();
                    }
                    return true;
                }
                break;
```

```
            case R.id.forward:
                if(event.getAction() == MotionEvent.ACTION_SCROLL){
                    return false;
                }else if(event.getAction() == MotionEvent.ACTION_DOWN){
                    Log.d(TAG, "forward button pressed");
                    try {
                        appModel.getScribbler().forward(.5);
                    } catch (Exception e) {
                        e.printStackTrace();
                    }
                    return true;
                }else if(event.getAction() == MotionEvent.ACTION_UP ||
event.getAction() == MotionEvent.ACTION_CANCEL){
                    Log.d(TAG, "forward button released");
                    try {
                        appModel.getScribbler().stop();
                    } catch (Exception e) {
                        e.printStackTrace();
                    }
                    return true;
                }
                break;
            case R.id.backward:
                if(event.getAction() == MotionEvent.ACTION_SCROLL){
                    return false;
                }else if(event.getAction() == MotionEvent.ACTION_DOWN){
                    Log.d(TAG, "backward button pressed");
                    try {
                        appModel.getScribbler().backward(.5);
                    } catch (Exception e) {
                        e.printStackTrace();
                    }
                    return true;
                }else if(event.getAction() == MotionEvent.ACTION_UP ||
event.getAction() == MotionEvent.ACTION_CANCEL){
                    Log.d(TAG, "backward button released");
                    try {
                        appModel.getScribbler().stop();
                    } catch (Exception e) {
                        e.printStackTrace();
                    }
                    return true;
                }
                break;
            case R.id.stop:
                Log.d(TAG, "stop button clicked");
                if(event.getAction() == MotionEvent.ACTION_SCROLL){
                    return false;
                }else if(event.getAction() == MotionEvent.ACTION_DOWN){
                    try{
```

```
                    appModel.getScribbler().stop();
                    allEnabled=false;
                }catch(Exception e){
                    e.printStackTrace();
                }
                return true;
            }

            break;
        }

    }else{
        Toast.makeText(getActivity(),"Not  connected to any device...",
                Toast.LENGTH_SHORT).show();
    }

    //v.performClick();
    return false;
}


/**
* Register the Sensor
*/
private void registerSensor(){
    sensorManager.registerListener(this,  acceleroMeter,
SensorManager.SENSOR_DELAY_NORMAL);
    tiltControl=true;
    allEnabled=true;
}

/**
* Unregister the Sensor
*/
private void unregisterSensor(){
    sensorManager.unregisterListener(this);
    tiltControl=false;
}

/**
* methods for the SensorEventListener  Interface
* @param event
*/
float[] mGravity;
float[] mGeomagnetic;
@Override
public void onSensorChanged(final  SensorEvent event) {
    // check sensor type
        if(allEnabled == true &&
event.sensor.getType()==Sensor.TYPE_ACCELEROMETER){
```

```
// assign directions
x = event.values[0];
y = event.values[1];

// STOP: within 2 or -2 of any direction while its moving
if(appModel.getScribbler().isMoving()  &&
        appModel.getScribbler().isConnected()  &&
        x > negThreshold &&
        x < threshold){

        appModel.getScribbler().move(0,  0);
        appModel.getScribbler().setMoving(false);

        if(upEnabled == true){
            upEnabled = false;
        }
        else if(downEnabled == true){
            downEnabled = false;
        }
        else if(rightEnabled == true){
            rightEnabled = false;
        }
        else if(leftEnabled == true){
            leftEnabled = false;
        }
}

 // Move the Scribbler BACKWARDS
if (appModel.getScribbler().isConnected()  &&
        x < negThreshold &&
        !(x > negThreshold)) {

  // describing the speed based on what x is
  if(x < -2.0 && x > -2.5){
     appModel.getScribbler().backward(0.2);
     appModel.getScribbler().setMoving(true);
     downEnabled = true;
  }
  else if(x < -2.6 && x > -3.0){
     appModel.getScribbler().backward(0.3);
     appModel.getScribbler().setMoving(true);
     downEnabled = true;
  }
  else if(x < -3.1 && x > -3.5){
     appModel.getScribbler().backward(0.4);
     appModel.getScribbler().setMoving(true);
     downEnabled = true;
  }
  else if(x < -3.6 && x > -4.0){
     appModel.getScribbler().backward(0.5);
```

```
    appModel.getScribbler().setMoving(true);
    downEnabled = true;
  }
  else if(x < -4.1 && x > -4.5){
    appModel.getScribbler().backward(0.6);
    appModel.getScribbler().setMoving(true);
    downEnabled = true;
  }
  else if(x < -4.6 && x > -5.0){
    appModel.getScribbler().backward(0.7);
    appModel.getScribbler().setMoving(true);
    downEnabled = true;
  }
  else if(x < -5.1 && x > -5.5){
    appModel.getScribbler().backward(0.8);
    appModel.getScribbler().setMoving(true);
    downEnabled = true;
  }
  else if(x < -5.6 && x > -6.0){
    appModel.getScribbler().backward(0.9);
    appModel.getScribbler().setMoving(true);
    downEnabled = true;
  }
  else if(x > -6.1){
    appModel.getScribbler().backward(1.0);
    appModel.getScribbler().setMoving(true);
    downEnabled = true;
  }
 }
     // Move the Scribbler FORWARDS
 else if (appModel.getScribbler().isConnected()  &&
       x > threshold &&
       !(x < negThreshold)) {

   // describing the speed based on what x is

   if(x > 2.0 && x < 2.5){
      appModel.getScribbler().forward(0.2);
      appModel.getScribbler().setMoving(true);
    upEnabled = true;
   }
   else if(x > 2.6 && x < 3.0){
      appModel.getScribbler().forward(0.3);
      appModel.getScribbler().setMoving(true);
      upEnabled = true;
   }
   else if(x > 3.1 && x < 3.5){
      appModel.getScribbler().forward(0.4);
      appModel.getScribbler().setMoving(true);
      upEnabled = true;
```

```
    }
    else if(x > 3.6 && x < 4.0){
        appModel.getScribbler().forward(0.5);
        appModel.getScribbler().setMoving(true);
        upEnabled = true;
    }
    else if(x > 4.1 && x < 4.5){
        appModel.getScribbler().forward(0.6);
        appModel.getScribbler().setMoving(true);
        upEnabled = true;
    }
    else if(x > 4.6 && x < 5.0){
        appModel.getScribbler().forward(0.7);
        appModel.getScribbler().setMoving(true);
        upEnabled = true;
    }
    else if(x > 5.1 && x < 5.5){
        appModel.getScribbler().forward(0.8);
        appModel.getScribbler().setMoving(true);
        upEnabled = true;
    }
    else if(x > 5.6 && x < 6.0){
        appModel.getScribbler().forward(0.9);
        appModel.getScribbler().setMoving(true);
        upEnabled = true;
    }
    else if(x > -6.1){
        appModel.getScribbler().forward(1.0);
        appModel.getScribbler().setMoving(true);
        upEnabled = true;
    }
  }

 // Move the Scribbler RIGHT
 if (appModel.getScribbler().isConnected()  &&
        y < negThreshold &&
        !(y > negThreshold)) {

   // describing the speed based on what x is
   if(y > -2.5){
      appModel.getScribbler().turnRight(0.2);
      appModel.getScribbler().setMoving(true);
      rightEnabled = true;
   }
   else if(y < -2.6 && y > -3.0){
      appModel.getScribbler().turnRight(0.3);
      appModel.getScribbler().setMoving(true);
      rightEnabled = true;
   }
   else if(y < -3.1 && y > -3.5){
```

```
    appModel.getScribbler().turnRight(0.4);
    appModel.getScribbler().setMoving(true);
    rightEnabled = true;
  }
  else if(y < -3.6 && y > -4.0){
    appModel.getScribbler().turnRight(0.5);
    appModel.getScribbler().setMoving(true);
    rightEnabled = true;
  }
  else if(y < -4.1 && y > -4.5){
    appModel.getScribbler().turnRight(0.6);
    appModel.getScribbler().setMoving(true);
    rightEnabled = true;
  }
  else if(y < -4.6 && x > -5.0){
    appModel.getScribbler().turnRight(0.7);
    appModel.getScribbler().setMoving(true);
    rightEnabled = true;
  }
  else if(y < -5.1 && y > -5.5){
    appModel.getScribbler().turnRight(0.8);
    appModel.getScribbler().setMoving(true);
    rightEnabled = true;
  }
  else if(y < -5.6 && y > -6.0){
    appModel.getScribbler().turnRight(0.9);
    appModel.getScribbler().setMoving(true);
    rightEnabled = true;
  }
  else if(y > -6.1){
    appModel.getScribbler().turnRight(1.0);
    appModel.getScribbler().setMoving(true);
    rightEnabled = true;
  }
 }
     // Move the Scribbler LEFT
 else if (appModel.getScribbler().isConnected()  &&
       y > threshold &&
       !(y < negThreshold)) {

   // describing the speed based on what x is

   if(y > 2.0 && y < 2.5){
      appModel.getScribbler().turnLeft(  0.2);
      appModel.getScribbler().setMoving(true);
    leftEnabled = true;
   }
   else if(y > 2.6 && y < 3.0){
      appModel.getScribbler().turnLeft(  0.3);
      appModel.getScribbler().setMoving(true);
```

```
                leftEnabled = true;
               }
               else if(y > 3.1 && y < 3.5){
                  appModel.getScribbler().turnLeft(  0.4);
                  appModel.getScribbler().setMoving(true);
                leftEnabled = true;
               }
               else if(y > 3.6 && y < 4.0){
                  appModel.getScribbler().turnLeft(  0.5);
                  appModel.getScribbler().setMoving(true);
                leftEnabled = true;
               }
               else if(y > 4.1 && y < 4.5){
                  appModel.getScribbler().turnLeft(0.6);
                  appModel.getScribbler().setMoving(true);
                leftEnabled = true;
               }
               else if(y > 4.6 && y < 5.0){
                  appModel.getScribbler().turnLeft(0.7);
                  appModel.getScribbler().setMoving(true);
                leftEnabled = true;
               }
               else if(y > 5.1 && y < 5.5){
                  appModel.getScribbler().turnLeft(0.8);
                  appModel.getScribbler().setMoving(true);
                leftEnabled = true;
               }
               else if(y > 5.6 && y < 6.0){
                  appModel.getScribbler().turnLeft(0.9);
                  appModel.getScribbler().setMoving(true);
                leftEnabled = true;
               }
               else if(y > 6.1){
                  appModel.getScribbler().turnLeft(1.0);
                  appModel.getScribbler().setMoving(true);
                leftEnabled = true;
               }
            }
         }


}

@Override
public void onAccuracyChanged(Sensor  sensor, int accuracy) {
   Log.d(TAG, "onAccuracyChanged  called");

}

@Override
```

```
    public void onItemSelected(AdapterView<?> parent, View view, int position,
            long id) {
        Log.d(TAG, "Spinner item selected");
        // unregister sensor listener
        //sensorManager.unregisterListener(this);

        if(appModel.getScribbler()!=null && appModel.isConnected()){
            try {
                appModel.getScribbler().stop();
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
        mySpinner.setSelection(position);
        String selected=(String)mySpinner.getSelectedItem();
        if(selected.equals("Phone Tilt")){
            Log.d(TAG, "Phone tilt selected");
            sensorManager.registerListener(this, acceleroMeter,
SensorManager.SENSOR_DELAY_NORMAL);
            tiltControl=true;
            allEnabled=true;
        }else if(selected.equals("Buttons")){
            Log.d(TAG, "Buttons selected");
            sensorManager.unregisterListener(this);
            tiltControl=false;
        }else if(selected.equals("Voice")){
            Log.d(TAG, "Voice command selected");
            sensorManager.unregisterListener(this);
            Intent i= new Intent(getActivity(), VoiceCommand.class);
            startActivity(i);
        }

    }

    @Override
    public void onNothingSelected(AdapterView<?> parent) {
        Log.d(TAG, "Nothing selected");

    }

    public boolean isTiltControl() {
        return tiltControl;
    }

    public void setTiltControl(boolean tiltControl) {
        this.tiltControl = tiltControl;
    }


}
```

```
package com.msu.myscribbler.fragments;

import java.util.HashMap;

import com.msu.myscribbler.R;
import com.msu.myscribbler.models.AppModel;
import com.msu.myscribbler.utils.Log;
import android.content.SharedPreferences;
import
android.content.SharedPreferences.OnSharedPreferenceChangeListener;
import android.content.res.Resources;
import android.graphics.Color;
import android.os.Bundle;
import android.os.Handler;
import android.preference.PreferenceManager;
import android.support.annotation.Nullable;
import android.support.v4.app.Fragment;
import android.view.LayoutInflater;
import android.view.View;
import android.view.View.OnClickListener;
import android.view.ViewGroup;
import android.widget.Button;
import android.widget.ImageView;
import android.widget.TextView;
import android.widget.Toast;
import android.widget.ToggleButton;

public class InfoFragment extends Fragment {

    private static final String TAG="InfoFragment";

    private static final String KEY_TITLE = "title";
   private static final String KEY_INDICATOR_COLOR = "indicator_color";
   private static final String KEY_DIVIDER_COLOR = "divider_color";

   private static final boolean D = true;

    private AppModel appModel;

    private TextView name, battery;
    private TextView obstacleLeft, obstacleCenter, obstacleRight;
    private TextView lightLeft, lightCenter, lightRight;
    private TextView irLeft, irRight;
    private TextView lineLeft, lineRight;
    private float batteryValue;
    private int[] obstacleValues, lightValues, irValues, lineValues;
    private String nameValue;
```

```
    private SharedPreferences settings;
    private Resources res;
    private Handler handler;

    private ToggleButton toggleButton;
    private Button manualButton;
    private Runnable r;

    private ImageView battery_level;
    // Must be instance variable to avoid garbage collection!
    private OnSharedPreferenceChangeListener preferenceListener;

    /**
    * @return a new instance of {@link ContentFragment}, adding the parameters
into a bundle and
    * setting them as arguments.
    */
    public static InfoFragment newInstance(CharSequence title, int
indicatorColor,
                                           int dividerColor) {
        Bundle bundle = new Bundle();
        bundle.putCharSequence(KEY_TITLE, title);
        bundle.putInt(KEY_INDICATOR_COLOR, indicatorColor);
        bundle.putInt(KEY_DIVIDER_COLOR, dividerColor);

        InfoFragment fragment = new InfoFragment();
        fragment.setArguments(bundle);

        return fragment;
    }

    @Override
    public void onCreate(Bundle savedInstanceState) {

        super.onCreate(savedInstanceState);
        Log.d(TAG, "on Create");
        appModel=(AppModel)getActivity().getApplicationContext();
        settings = PreferenceManager.getDefaultSharedPreferences(getActivity());
        res = getResources();
        handler = new Handler();
    }


    public void retrieveAllValues(){
        Log.d(TAG, "retrieve values");
        HashMap<String, int[]> hm = appModel.getScribbler().getAll();
        lineValues = hm.get("LINE");
        irValues = hm.get("IR");
        lightValues = hm.get("LIGHT");
```

```
        //nameValue = appModel.getScribbler().getDeviceName();
        //batteryValue = appModel.getScribbler().getBattery();
        obstacleValues = appModel.getScribbler().getObstacle("all");
    }



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

        View rootView= inflater.inflate(R.layout.info_fragment, container,false);

        // Initialize the textviews that will be updated when requested
        name = (TextView) rootView.findViewById(R.id.name_value);
        battery = (TextView) rootView.findViewById(R.id.battery_value);
        obstacleLeft = (TextView) rootView.findViewById(R.id.obstacle_left_value);
        obstacleCenter = (TextView)
rootView.findViewById(R.id.obstacle_center_value);
        obstacleRight = (TextView)
rootView.findViewById(R.id.obstacle_right_value);
        lightLeft = (TextView) rootView.findViewById(R.id.light_left_value);
        lightCenter = (TextView) rootView.findViewById(R.id.light_center_value);
        lightRight = (TextView) rootView.findViewById(R.id.light_right_value);
        irRight = (TextView) rootView.findViewById(R.id.ir_right_value);
        irLeft = (TextView) rootView.findViewById(R.id.ir_left_value);
        lineLeft = (TextView) rootView.findViewById(R.id.line_left_value);
        lineRight = (TextView) rootView.findViewById(R.id.line_right_value);
        toggleButton = (ToggleButton) rootView.findViewById(R.id.toggleTimer);
        manualButton = (Button)
rootView.findViewById(R.id.button_manual_refresh);
        battery_level = (ImageView)
rootView.findViewById(R.id.battery_level_image);

        // set the correct control button based on previous preference
        if (settings.getBoolean(res.getString(R.string.autoRefresh_pref), false)) {
          toggleButton.setChecked(false);
          toggleButton.setVisibility(View.VISIBLE);
          manualButton.setVisibility(View.GONE);
        } else {
          toggleButton.setVisibility(View.GONE);
          manualButton.setVisibility(View.VISIBLE);
        }
        // Listener that will change the control button as needed
        preferenceListener = new
SharedPreferences.OnSharedPreferenceChangeListener() {
          @Override
          public void onSharedPreferenceChanged(SharedPreferences prefs, String
key) {
```

```
        if (key.equals(res.getString(R.string.autoRefresh_pref))) {
          boolean auto =
settings.getBoolean(res.getString(R.string.autoRefresh_pref), false);

          if (auto) {
            toggleButton.setVisibility(View.VISIBLE);
            manualButton.setVisibility(View.GONE);
          } else {
            toggleButton.setChecked(false);
            toggleButton.setVisibility(View.GONE);
            manualButton.setVisibility(View.VISIBLE);
          }
        }
      }
    };
    settings.registerOnSharedPreferenceChangeListener(preferenceListener);


    r = new Runnable() {
      @Override
      public void run() {
        // Update only of scribbler is connected
        if (appModel.getScribbler().isConnected()) {
          Log.i(TAG, "Populating Values");
          updateValues();
          populate();
        } else {
          Log.e(TAG, "Disconnected unexpectedly...");
          toggleButton.setChecked(false);
          Toast.makeText(getActivity(),"Not connected to any device...",
                  Toast.LENGTH_SHORT).show();
        }
        // Keep polling until user decides to stop or leaves activity
        if (toggleButton.isChecked()) {
          handler.postDelayed(
              this,
              Integer.parseInt(settings.getString(
                  res.getString(R.string.refresh_rate_pref),
                  res.getString(R.string.default_refresh_rate))));
        }
      }
    };

    toggleButton.setOnClickListener(new OnClickListener() {
      @Override
      public void onClick(View v) {
        if (appModel.getScribbler().isConnected()) {
          if (toggleButton.isChecked())
            handler.postDelayed(
```

```
                r,
                Integer.parseInt(settings.getString(
                    res.getString(R.string.refresh_rate_pref),
                    res.getString(R.string.default_refresh_rate))));
            } else {
              toggleButton.setChecked(false);
            }

          }
        });

        // update information once
        manualButton.setOnClickListener(new OnClickListener() {
          @Override
          public void onClick(View v) {
            if (appModel.getScribbler().isConnected())
              r.run();
            else
              Toast.makeText(getActivity(),"Not connected to any device...",
                  Toast.LENGTH_SHORT).show();
          }
        });

         return rootView;

         }

    /**
    * Function that properly updates variables containing information for the
    * sensors.
    */
    private void updateValues() {
        HashMap<String, int[]> hm = appModel.getScribbler().getAll();
        lineValues = hm.get("LINE");
        irValues = hm.get("IR");
        lightValues = hm.get("LIGHT");
        nameValue = appModel.getScribbler().getName();
        batteryValue = appModel.getScribbler().getBattery();
        obstacleValues = appModel.getScribbler().getObstacle("all");
        //obstacleValues = appModel.getScribbler().getIR("all");
     }


    /**
    * Function that properly populates the UI with the current sensors values.
    */
    private void populate() {
        // Update Name
        name.setText(nameValue);
```

```
        // Update Battery
        battery.setText(Float.toString(batteryValue));
        if (batteryValue < 6.2) {
            battery.setTextColor(Color.RED);
              battery_level.setImageResource(R.drawable.battery_low);

        } else if (batteryValue >= 6.3 && batteryValue <= 6.8) {
            battery.setTextColor(Color.YELLOW);
              battery_level.setImageResource(R.drawable.battery_half);

        } else if(batteryValue >= 6.9 && batteryValue <= 7.4 ){
            battery.setTextColor(Color.YELLOW);
            battery_level.setImageResource(R.drawable.battery_3_4);
        }

        else{
            battery.setTextColor(Color.GREEN);
            battery_level.setImageResource(R.drawable.battery_full);
        }

        // Update Obstacle Values
        obstacleLeft.setText(Integer.toString(obstacleValues[0]));
        Log.d(TAG, "obstacle left: "+String.valueOf(obstacleValues[0]));
        obstacleCenter.setText(Integer.toString(obstacleValues[1]));
        Log.d(TAG, "obstacle center: "+String.valueOf(obstacleValues[1]));
        obstacleRight.setText(Integer.toString(obstacleValues[2]));
        Log.d(TAG, "obstacle right: "+String.valueOf(obstacleValues[2]));

        // Update Light Values
        lightLeft.setText(Integer.toString(lightValues[0]));
        lightCenter.setText(Integer.toString(lightValues[1]));
        lightRight.setText(Integer.toString(lightValues[2]));

        // Update IR Values
        irLeft.setText(Integer.toString(irValues[0]));
        irRight.setText(Integer.toString(irValues[1]));

        // Update Line Values
        lineLeft.setText(Integer.toString(lineValues[0]));
        lineRight.setText(Integer.toString(lineValues[1]));
      }



}

package com.msu.myscribbler.fragments;

import android.graphics.Color;
import android.os.Bundle;
```

```
import android.support.v4.app.Fragment;
import android.support.v4.app.FragmentManager;
import android.support.v4.app.FragmentPagerAdapter;
import android.support.v4.view.ViewPager;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;

import java.util.ArrayList;
import java.util.List;

import com.msu.myscribbler.R;
import com.msu.myscribbler.utils.Constants;
import com.msu.myscribbler.utils.Log;

import common.googlecode.SlidingTabLayout;


public class MainFragment extends Fragment {

  /**
   * This class represents a tab to be displayed by {@link ViewPager} and it's
associated
   * {@link SlidingTabLayout}.
   */
  static class SamplePagerItem {
      private final CharSequence mTitle;
      private final int mIndicatorColor;
      private final int mDividerColor;


      SamplePagerItem(CharSequence title, int indicatorColor, int dividerColor) {
          mTitle = title;
          mIndicatorColor = indicatorColor;
          mDividerColor = dividerColor;
      }

      /**
       * @return A new {@link Fragment} to be displayed by a {@link ViewPager}
       */
      Fragment createFragment(int i) {
          switch(i){
          case 0:
              return HomeFragment.newInstance(mTitle, mIndicatorColor,
mDividerColor);

          case 1:
              return InfoFragment.newInstance(mTitle, mIndicatorColor,
mDividerColor);
```

```
            case 2:
                return GalleryFragment.newInstance(mTitle,  mIndicatorColor,
mDividerColor);

              }
            // return ContentFragment.newInstance(mTitle,  mIndicatorColor,
mDividerColor);
            return null;
        }

        /**
         * @return the title which represents this tab. In this sample this is used
directly by
         * {@link android.support.v4.view.PagerAdapter#getPageTitle(int)}
         */
        CharSequence getTitle() {
            return mTitle;
        }

        /**
         * @return the color to be used for indicator on the {@link
SlidingTabLayout}
         */
        int getIndicatorColor() {
            return mIndicatorColor;
        }

        /**
         * @return the color to be used for right divider on the {@link
SlidingTabLayout}
         */
        int getDividerColor() {
            return mDividerColor;
        }

    }

    static final String TAG = "SlidingTabsColorsFragment";

    /**
     * A custom {@link ViewPager} title strip which looks much like Tabs present
in Android v4.0 and
     * above, but is designed to give continuous feedback to the user when
scrolling.
     */
    private SlidingTabLayout mSlidingTabLayout;

    /**
     * A {@link ViewPager} which will be used in conjunction with the {@link
SlidingTabLayout} above.
```

```
     */
    private ViewPager mViewPager;

    /**
     * List of {@link SamplePagerItem} which represent this sample's tabs.
     */
    private List<SamplePagerItem> mTabs = new ArrayList<SamplePagerItem>();


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

        /**
         * Populate our tab list with tabs. Each item contains a title, indicator color
and divider
         * color, which are used by {@link SlidingTabLayout}.
         */
        mTabs.add(new SamplePagerItem(
                Constants.TAB_HOME, // Title
                Color.BLUE, // Indicator color
                Color.BLACK // Divider color
        ));

        mTabs.add(new SamplePagerItem(
                Constants.TAB_INFO, // Title
                Color.BLUE, // Indicator color
                Color.BLACK // Divider color
        ));

        mTabs.add(new SamplePagerItem(
                Constants.TAB_GALLERY, // Title
                Color.BLUE, // Indicator color
                Color.BLACK // Divider color
        ));

        // END_INCLUDE (populate_tabs)
    }

    /**
     * Inflates the {@link View} which will be displayed by this {@link Fragment},
from the app's
     * resources.
     */
    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
                             Bundle savedInstanceState) {
        return inflater.inflate(R.layout.fragment_main, container, false);
    }
```

```
  // BEGIN_INCLUDE (fragment_onviewcreated)
  /**
   * This is called after the {@link #onCreateView(LayoutInflater, ViewGroup,
Bundle)} has finished.
   * Here we can pick out the {@link View}s we need to configure from the
content view.
   *
   * We set the {@link ViewPager}'s adapter to be an instance of
   * {@link SampleFragmentPagerAdapter}. The {@link SlidingTabLayout} is
then given the
   * {@link ViewPager} so that it can populate itself.
   *
   * @param view View created in {@link #onCreateView(LayoutInflater,
ViewGroup, Bundle)}
   */
  @Override
  public void onViewCreated(View view, Bundle savedInstanceState) {
      // BEGIN_INCLUDE (setup_viewpager)
      // Get the ViewPager and set it's PagerAdapter so that it can display items
      mViewPager = (ViewPager) view.findViewById(R.id.viewpager);
      mViewPager.setOffscreenPageLimit(3);
      mViewPager.setAdapter(new
SampleFragmentPagerAdapter(getChildFragmentManager()));
      // END_INCLUDE (setup_viewpager)

      // BEGIN_INCLUDE (setup_slidingtablayout)
      // Give the SlidingTabLayout the ViewPager, this must be done AFTER the
ViewPager has had
      // it's PagerAdapter set.
      mSlidingTabLayout = (SlidingTabLayout)
view.findViewById(R.id.sliding_tabs);
      mSlidingTabLayout.setDistributeEvenly(true);
      mSlidingTabLayout.setViewPager(mViewPager);

      // BEGIN_INCLUDE (tab_colorizer)
      // Set a TabColorizer to customize the indicator and divider colors. Here we
just retrieve
      // the tab at the position, and return it's set color
      mSlidingTabLayout.setCustomTabColorizer(new
SlidingTabLayout.TabColorizer() {

          @Override
          public int getIndicatorColor(int position) {
              return mTabs.get(position).getIndicatorColor();
          }


      });
      // END_INCLUDE (tab_colorizer)
      // END_INCLUDE (setup_slidingtablayout)
```

```
    }
    // END_INCLUDE (fragment_onviewcreated)

    /**
     * The {@link FragmentPagerAdapter} used to display pages in this sample.
Each page is
     * created by the relevant {@link SamplePagerItem} for the requested position.
     * <p>
     * The important section of this class is the {@link #getPageTitle(int)} method
which controls
     * what is displayed in the {@link SlidingTabLayout}.
     */
    class SampleFragmentPagerAdapter extends FragmentPagerAdapter {

        SampleFragmentPagerAdapter(FragmentManager fm) {
            super(fm);
        }

        /**
         * Return the {@link android.support.v4.app.Fragment} to be displayed at
{@code position}.
         */
        @Override
        public Fragment getItem(int i) {

            return mTabs.get(i).createFragment(i);
        }

        @Override
        public int getItemPosition(Object object) {
            if (object instanceof GalleryFragment) {
                Log.d(TAG, "Gallery Fragment getItemPosition");
                ((GalleryFragment) object).updateData();
            }
            //don't return POSITION_NONE, avoid fragment recreation.
            return super.getItemPosition(object);
        }


        @Override
        public int getCount() {
            return mTabs.size();
        }

        // BEGIN_INCLUDE (pageradapter_getpagetitle)
        /**
         * Return the title of the item at {@code position}. This is important as what
this method
         * returns is what is displayed in the {@link SlidingTabLayout}.
         * <p>
```

```
        * Here we return the value returned from {@link
SamplePagerItem#getTitle()}.
        */
       @Override
       public CharSequence getPageTitle(int position) {
          return mTabs.get(position).getTitle();
       }
       // END_INCLUDE (pageradapter_getpagetitle)

    }



}



package com.msu.myscribbler.interfaces;

import android.graphics.Bitmap;

public interface AsyncResponse {

    void asyncReturn(Bitmap bitmap);

}



package com.msu.myscribbler.interfaces;

public interface GalleryFragmentCallback {
    void updateData();
}




package com.msu.myscribbler.interfaces;

public interface NavigationDrawerCallbacks {
    void onNavigationDrawerItemSelected(int position, String title);

}



package com.msu.myscribbler.models;

import com.msu.myscribbler.commands.Scribbler;
```

```
import
com.nostra13.universalimageloader.cache.disc.naming.Md5FileNameGenerator
;
import com.nostra13.universalimageloader.core.ImageLoader;
import com.nostra13.universalimageloader.core.ImageLoaderConfiguration;
import com.nostra13.universalimageloader.core.assist.QueueProcessingType;

import android.app.Application;
import android.content.Context;
import android.os.Build;
import android.os.StrictMode;

/**
* Represents a Scribbler on an application context
* @author Nirajan
*
*/
public class AppModel extends Application{

   private Scribbler scribbler;
   private boolean isConnected=false;

   private static int[] rawImage;


   @SuppressWarnings("unused")
   @Override
   public void onCreate() {
      if (Config.DEVELOPER_MODE  && Build.VERSION.SDK_INT  >=
Build.VERSION_CODES.GINGERBREAD) {
         StrictMode.setThreadPolicy(new
StrictMode.ThreadPolicy.Builder().detectAll().penaltyDialog().build());
         StrictMode.setVmPolicy(new
StrictMode.VmPolicy.Builder().detectAll().penaltyDeath().build());
      }

      super.onCreate();

      initImageLoader(getApplicationContext());
   }


   public Scribbler getScribbler() {
     return scribbler;
    }

   public void setScribbler(Scribbler scribbler) {
       this.scribbler = scribbler;
    }
```

```
    public boolean isConnected() {
        return isConnected;
    }

    public void setConnected(boolean isConnected) {
        this.isConnected = isConnected;
    }

    public int[] getRawImage() {
        return rawImage;
    }

    public void setRawImage(int[] rawImage) {
        this.rawImage = rawImage;
    }

    public static void initImageLoader(Context context) {
        // This configuration tuning is custom. You can tune every option, you may
tune some of them,
        // or you can create default configuration by
        //  ImageLoaderConfiguration.createDefault(this);
        // method.
        ImageLoaderConfiguration config = new
ImageLoaderConfiguration.Builder(context)
                .threadPriority(Thread.NORM_PRIORITY - 2)
                .denyCacheImageMultipleSizesInMemory()
                .diskCacheFileNameGenerator(new Md5FileNameGenerator())
                .diskCacheSize(50 * 1024 * 1024) // 50 Mb
                .tasksProcessingOrder(QueueProcessingType.LIFO)
                .build();
        // Initialize ImageLoader with configuration.
        ImageLoader.getInstance().init(config);
    }

    public static class Config {
        public static final boolean DEVELOPER_MODE = false;
    }

}



package com.msu.myscribbler.models;

public class CommandItem {

    private String mCommand;
    private String mTime;
```

```
    // flag to check if command has time associated
    private boolean hasTime=false;

    public CommandItem(){

    }

    public CommandItem(String  command, String time){
        this.mCommand=command;
        this.mTime=time;
        this.hasTime=true;
    }

    public CommandItem(String  command){
        this.mCommand=command;
        this.hasTime=false;
    }

    public String getmCommand() {
        return mCommand;
    }
    public void setmCommand(String  mCommand) {
        this.mCommand  = mCommand;
    }
    public String getmTime() {
        return mTime;
    }
    public void setmTime(String  mTime) {
        this.mTime = mTime;
    }
    public boolean isHasTime() {
        return hasTime;
    }
    public void setHasTime(boolean  hasTime) {
        this.hasTime = hasTime;
    }


}



package com.msu.myscribbler.navigationdrawer;

import java.util.List;
import com.msu.myscribbler.R;
import com.msu.myscribbler.interfaces.NavigationDrawerCallbacks;
import com.msu.myscribbler.utils.Log;
import android.content.Context;
import android.view.LayoutInflater;
```

```
import android.view.View;
import android.view.ViewGroup;
import android.widget.BaseAdapter;
import android.widget.TextView;


public class NavDrawerAdapter extends BaseAdapter {

    private List<NavigationItem> mData;
    private NavigationDrawerCallbacks mNavigationDrawerCallbacks;
    private int mSelectedPosition;
    private int mTouchedPosition = -1;

    private Context mContext;
    private static final String TAG= "NavAdapter";


    public NavDrawerAdapter(Context context, List<NavigationItem> data) {
        mData = data;
        mContext=context;
    }

    public NavigationDrawerCallbacks getNavigationDrawerCallbacks() {
        return mNavigationDrawerCallbacks;
    }

    public void setNavigationDrawerCallbacks(NavigationDrawerCallbacks
navigationDrawerCallbacks) {
        mNavigationDrawerCallbacks = navigationDrawerCallbacks;
    }

    @Override
    public int getCount() {
        return mData != null ? mData.size() : 0;
    }


    @Override
    public Object getItem(int position) {
        return mData.get(position);
    }


    @Override
    public long getItemId(int position) {
        return mData.indexOf(getItem(position));
    }


    @Override
```

```
    public View getView(final int position, View convertView, ViewGroup parent) {

        View itemView = convertView;
        final NavigationItem rowItem= this.mData.get(position);

        if(itemView==null){
            DrawerItemHolder drawerHolder = new DrawerItemHolder();
            itemView =
LayoutInflater.from(parent.getContext()).inflate(R.layout.drawer_row, parent,
false);
            drawerHolder.title= (TextView) itemView.findViewById(R.id.title);
            itemView.setTag(drawerHolder);
        }

        DrawerItemHolder holder= (DrawerItemHolder) itemView.getTag();
        holder.title.setText(rowItem.getTitle());

       /*holder.title.setOnClickListener(new View.OnClickListener() {
          @Override
          public void onClick(View v) {
              Log.d(TAG, "CLICKED "+String.valueOf(position));
                if(rowItem.isItem()){
                    if (mNavigationDrawerCallbacks != null)
                        mNavigationDrawerCallbacks.onNavigationDrawerItemSelected
(position, rowItem.getText());
                }
            }
        });*/

        return itemView;
    }

    private static class DrawerItemHolder{
      public TextView title;

    }

}

package com.msu.myscribbler.navigationdrawer;

import android.app.Activity;
import android.content.Context;
import android.content.Intent;
import android.content.SharedPreferences;
import android.content.res.Configuration;
import android.os.Bundle;
import android.os.Handler;
import android.support.v4.app.Fragment;
import android.support.v4.widget.DrawerLayout;
```

```
import android.support.v7.app.ActionBarDrawerToggle;
import android.support.v7.widget.Toolbar;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.AdapterView;
import android.widget.AdapterView.OnItemClickListener;
import android.widget.ListView;
import java.util.ArrayList;
import java.util.List;
import com.msu.myscribbler.CommandsGUI;
import com.msu.myscribbler.R;
import com.msu.myscribbler.ScanRobot;
import com.msu.myscribbler.interfaces.NavigationDrawerCallbacks;
import com.msu.myscribbler.utils.Constants;
import com.msu.myscribbler.utils.Log;


public class NavigationDrawerFragment  extends Fragment implements
NavigationDrawerCallbacks {

    private static final String TAG="NavigationDrawerFragment";

    private static final String PREF_USER_LEARNED_DRAWER  =
"navigation_drawer_learned";
   private static final String STATE_SELECTED_POSITION  =
"selected_navigation_drawer_position";
   private static final String PREFERENCES_FILE  = "my_app_settings";  //TODO:
change this to your file

   private NavigationDrawerCallbacks  mCallbacks;

   private ListView mDrawerList;
   private NavDrawerAdapter  adapter;
   private View mFragmentContainerView;
   private DrawerLayout mDrawerLayout;
   private ActionBarDrawerToggle  mActionBarDrawerToggle;
   private boolean mUserLearnedDrawer;
   private boolean mFromSavedInstanceState;
   private int mCurrentSelectedPosition;




   @Override
   public View onCreateView(LayoutInflater  inflater, ViewGroup container,
Bundle savedInstanceState) {
       Log.d(TAG, "onCreateView");
       View view = inflater.inflate(R.layout.fragment_navigation_drawer,  container,
false);
```

```
        mDrawerList = (ListView) view.findViewById(R.id.drawerList);
        final List<NavigationItem> navigationItems = getMenu();
        adapter= new NavDrawerAdapter(getActivity(), navigationItems);
        adapter.setNavigationDrawerCallbacks(this);
        mDrawerList.setAdapter(adapter);
        mDrawerList.setOnItemClickListener(new OnItemClickListener() {
            @Override
            public void onItemClick(AdapterView<?> parent, View view,
                    int position, long id) {
                NavigationItem clickedItem= (NavigationItem)
adapter.getItem(position);
                selectItem(position, clickedItem.getTitle());
            }
        });
        selectItem(mCurrentSelectedPosition, "");
        return view;
    }

    public void refreshView(){
        adapter.notifyDataSetChanged();
    }

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        mUserLearnedDrawer = Boolean.valueOf(readSharedSetting(getActivity(),
PREF_USER_LEARNED_DRAWER, "false"));
        Log.d(TAG, "onCreate");
        if (savedInstanceState != null) {
            mCurrentSelectedPosition =
savedInstanceState.getInt(STATE_SELECTED_POSITION);
            Log.d(TAG, "current selection index: "+mCurrentSelectedPosition);
            mFromSavedInstanceState = true;
        }
    }

    @Override
    public void onAttach(Activity activity) {
        super.onAttach(activity);
        try {
            mCallbacks = (NavigationDrawerCallbacks) activity;
        } catch (ClassCastException e) {
            throw new ClassCastException("Activity must implement
NavigationDrawerCallbacks.");
        }
    }

    public ActionBarDrawerToggle getActionBarDrawerToggle() {
        return mActionBarDrawerToggle;
    }
```

```
  public void setActionBarDrawerToggle(ActionBarDrawerToggle
actionBarDrawerToggle) {
      mActionBarDrawerToggle  = actionBarDrawerToggle;
  }

  public void setup(int fragmentId,  DrawerLayout drawerLayout,  Toolbar toolbar)
{
      mFragmentContainerView  = getActivity().findViewById(fragmentId);
      mDrawerLayout  = drawerLayout;
      mActionBarDrawerToggle  = new ActionBarDrawerToggle(getActivity(),
mDrawerLayout,  toolbar, R.string.drawer_open,  R.string.drawer_close) {
          @Override
          public void onDrawerClosed(View  drawerView) {
              super.onDrawerClosed(drawerView);
              if (!isAdded()) return;
              getActivity().invalidateOptionsMenu();
          }

          @Override
          public void onDrawerOpened(View  drawerView) {
              super.onDrawerOpened(drawerView);
              if (!isAdded()) return;
              if (!mUserLearnedDrawer) {
                  Log.d(TAG, "UserLearnedSetting
"+String.valueOf(mUserLearnedDrawer));
                  mUserLearnedDrawer  = true;
                  saveSharedSetting(getActivity(),
PREF_USER_LEARNED_DRAWER,  "true");
              }

              getActivity().invalidateOptionsMenu();
          }
      };

      if (!mUserLearnedDrawer  && !mFromSavedInstanceState)
          mDrawerLayout.openDrawer(mFragmentContainerView);

      // changed the shared preference to user learnt it
      saveSharedSetting(getActivity(),  PREF_USER_LEARNED_DRAWER,
"true");

      mDrawerLayout.post(new  Runnable() {
          @Override
          public void run() {
              mActionBarDrawerToggle.syncState();
          }
      });

      mDrawerLayout.setDrawerListener(mActionBarDrawerToggle);
```

```
    }

    public void openDrawer() {
        mDrawerLayout.openDrawer(mFragmentContainerView);
    }

    public void closeDrawer() {
        mDrawerLayout.closeDrawer(mFragmentContainerView);
    }

    @Override
    public void onDetach() {
        super.onDetach();
        mCallbacks = null;
    }

    public List<NavigationItem> getMenu() {
       List<NavigationItem> items = new ArrayList<NavigationItem>();
       items.add(new NavigationItem(Constants.SCAN));
       items.add(new NavigationItem(Constants.GUI));
       items.add(new NavigationItem(Constants.DISCONNECT));

       return items;
    }

    void selectItem(int position, String title) {
        mCurrentSelectedPosition = position;
        final String selectedTitle=title;
        if (mDrawerLayout != null) {
            mDrawerLayout.closeDrawer(mFragmentContainerView);
        }
        if (mCallbacks != null) {
            mCallbacks.onNavigationDrawerItemSelected(position, title);
        }

    }

    public boolean isDrawerOpen() {
        return mDrawerLayout != null &&
mDrawerLayout.isDrawerOpen(mFragmentContainerView);
    }

    @Override
    public void onConfigurationChanged(Configuration newConfig) {
        super.onConfigurationChanged(newConfig);
        mActionBarDrawerToggle.onConfigurationChanged(newConfig);
    }

    @Override
    public void onSaveInstanceState(Bundle outState) {
```

```
        super.onSaveInstanceState(outState);
        outState.putInt(STATE_SELECTED_POSITION,
mCurrentSelectedPosition);
    }

    @Override
    public void onNavigationDrawerItemSelected(int  position, String title) {
        selectItem(position,  title);
    }

    public DrawerLayout getDrawerLayout() {
        return mDrawerLayout;
    }

    public void setDrawerLayout(DrawerLayout  drawerLayout) {
        mDrawerLayout  = drawerLayout;
    }

    public static void saveSharedSetting(Context  ctx, String settingName, String
settingValue) {
        SharedPreferences sharedPref =
ctx.getSharedPreferences(PREFERENCES_FILE,  Context.MODE_PRIVATE);
        SharedPreferences.Editor  editor = sharedPref.edit();
        editor.putString(settingName,  settingValue);
        editor.apply();
    }

    public static String readSharedSetting(Context  ctx, String settingName, String
defaultValue) {
        SharedPreferences sharedPref =
ctx.getSharedPreferences(PREFERENCES_FILE,  Context.MODE_PRIVATE);
        return sharedPref.getString(settingName,  defaultValue);
    }

}



package com.msu.myscribbler.navigationdrawer;

import android.graphics.drawable.Drawable;


public class NavigationItem {
    private String mText;
    private Drawable mDrawable;
    private boolean isProfile;
    private String title;
    private boolean isItem;
    private boolean isTitle;
```

```
public boolean isTitle() {
    return isTitle;
}

public void setTitle(boolean isTitle) {
    this.isTitle = isTitle;
}

public NavigationItem(String  text, Drawable drawable) {
   mText = text;
   mDrawable = drawable;
   isItem=true;
   isProfile=false;
   isTitle=false;
}

public NavigationItem(boolean  isProfile) {
    this(null, null);
    this.isProfile = isProfile;
    isItem=false;
    isTitle=false;
}

public NavigationItem(String  title) {
    this(null, null);
    this.title = title;
    this.isProfile=false;
    this.isItem=false;
    this.isTitle=true;
}

public boolean isItem() {
    return isItem;
}

public void setItem(boolean  isItem) {
    this.isItem = isItem;
}

public boolean isProfile() {
    return isProfile;
}

public void setProfile(boolean isProfile) {
    this.isProfile = isProfile;
}


public String getText() {
```

```
        return mText;
    }

    public void setText(String text) {
        mText = text;
    }

    public Drawable getDrawable() {
        return mDrawable;
    }

    public void setDrawable(Drawable drawable) {
        mDrawable = drawable;
    }

    public String getTitle() {
         return title;
     }

     public void setTitle(String title) {
         this.title = title;
     }

}



package com.msu.myscribbler.utils;

public class Constants {

    /**
     * Constants to represent Commands set to Scribbler
     */
     public static final String TURN_LEFT="Turn Left";
     public static final String TURN_RIGHT="Turn Right";
     public static final String GO_FORWARD="Go Forward";
     public static final String GO_BACKWARD="Go Backward";
     public static final String TAKE_PHOTO="Take Photo";
     public static final String STOP="Stop";
     public static final String BEEP="Beep";
     public static final String DANCE="Dance";

     // Tabs
    public static final String TAB_HOME="Home";
    public static final String TAB_INFO="Info";
    public static final String TAB_GALLERY="Gallery";

    /**
     * Strings for the drawer items
```

```
    */
   public static final String SCAN="Scan Devices";
   public static final String GUI="Commands GUI";
   public static final String DISCONNECT="Disconnect";
   /**
    * Home Fragment Controller Options
    */
   public static final String BUTTONS="Buttons";
   public static final String PHONE_TILT="Phone Tilt";
   public static final String VOICE="Voice";
}


package com.msu.myscribbler.utils;

import java.io.File;
import java.io.FilenameFilter;
import java.util.ArrayList;
import java.util.List;

import com.msu.myscribbler.utils.ListenerList.FireHandler;

import android.app.Activity;
import android.app.AlertDialog;
import android.app.Dialog;
import android.content.DialogInterface;
import android.content.DialogInterface.OnClickListener;
import android.os.Environment;

public class FileDialog {
            private static final String PARENT_DIR = "..";
            private final String TAG = getClass().getName();
            private String[] fileList;
            private File currentPath;
            public interface FileSelectedListener {
               void fileSelected(File file);
            }
            public interface DirectorySelectedListener {
               void directorySelected(File directory);
            }
            private ListenerList<FileSelectedListener> fileListenerList = new
ListenerList<FileDialog.FileSelectedListener>();
            private ListenerList<DirectorySelectedListener> dirListenerList = new
ListenerList<FileDialog.DirectorySelectedListener>();
            private final Activity activity;
            private boolean selectDirectoryOption;
            private String fileEndsWith;

            /**
             * @param activity
```

```
     * @param initialPath
     */
    public FileDialog(Activity activity, File path) {
        this.activity = activity;
        if (!path.exists()) path = Environment.getExternalStorageDirectory();
        loadFileList(path);
    }

    /**
     * @return file dialog
     */
    public Dialog createFileDialog() {
        Dialog dialog = null;
        AlertDialog.Builder builder = new AlertDialog.Builder(activity);

        builder.setTitle(currentPath.getPath());
        if (selectDirectoryOption) {
            builder.setPositiveButton("Select directory", new
OnClickListener() {
                public void onClick(DialogInterface dialog, int which) {
                    Log.d(TAG, currentPath.getPath());
                    fireDirectorySelectedEvent(currentPath);
                }
            });
        }

        builder.setItems(fileList, new DialogInterface.OnClickListener() {
            public void onClick(DialogInterface dialog, int which) {
                String fileChosen = fileList[which];
                File chosenFile = getChosenFile(fileChosen);
                if (chosenFile.isDirectory()) {
                    loadFileList(chosenFile);
                    dialog.cancel();
                    dialog.dismiss();
                    showDialog();
                } else fireFileSelectedEvent(chosenFile);
            }
        });

        dialog = builder.show();
        return dialog;
    }


    public void addFileListener(FileSelectedListener listener) {
        fileListenerList.add(listener);
    }

    public void removeFileListener(FileSelectedListener listener) {
        fileListenerList.remove(listener);
```

```
        }

        public void setSelectDirectoryOption(boolean selectDirectoryOption) {
            this.selectDirectoryOption = selectDirectoryOption;
        }

        public void addDirectoryListener(DirectorySelectedListener listener) {
            dirListenerList.add(listener);
        }

        public void removeDirectoryListener(DirectorySelectedListener
listener) {
            dirListenerList.remove(listener);
        }

        /**
         * Show file dialog
         */
        public void showDialog() {
            createFileDialog().show();
        }

        private void fireFileSelectedEvent(final File file) {
            fileListenerList.fireEvent(new
FireHandler<FileDialog.FileSelectedListener>() {
                public void fireEvent(FileSelectedListener listener) {
                    listener.fileSelected(file);
                }
            });
        }

        private void fireDirectorySelectedEvent(final File directory) {
            dirListenerList.fireEvent(new
FireHandler<FileDialog.DirectorySelectedListener>() {
                public void fireEvent(DirectorySelectedListener listener) {
                    listener.directorySelected(directory);
                }
            });
        }

        private void loadFileList(File path) {
            this.currentPath = path;
            List<String> r = new ArrayList<String>();
            if (path.exists()) {
                if (path.getParentFile() != null) r.add(PARENT_DIR);
                FilenameFilter filter = new FilenameFilter() {
                    public boolean accept(File dir, String filename) {
                        File sel = new File(dir, filename);
                        if (!sel.canRead()) return false;
                        if (selectDirectoryOption) return sel.isDirectory();
```

```
                    else {
                        boolean endsWith = fileEndsWith != null ?
filename.toLowerCase().endsWith(fileEndsWith) : true;
                        return endsWith || sel.isDirectory();
                    }
                }
            };
            String[] fileList1 = path.list(filter);
            for (String file : fileList1) {
                r.add(file);
            }
        }
        fileList = (String[]) r.toArray(new String[]{});
    }

    private File getChosenFile(String fileChosen) {
        if (fileChosen.equals(PARENT_DIR)) return
currentPath.getParentFile();
        else return new File(currentPath, fileChosen);
    }

    public void setFileEndsWith(String fileEndsWith) {
        this.fileEndsWith = fileEndsWith != null ?
fileEndsWith.toLowerCase() : fileEndsWith;
    }
}

class ListenerList<L> {
    private List<L> listenerList = new ArrayList<L>();

    public interface FireHandler<L> {
        void fireEvent(L listener);
    }

    public void add(L listener) {
        listenerList.add(listener);
    }

    public void fireEvent(FireHandler<L> fireHandler) {
        List<L> copy = new ArrayList<L>(listenerList);
        for (L l : copy) {
            fireHandler.fireEvent(l);
        }
    }

    public void remove(L listener) {
        listenerList.remove(listener);
    }

    public List<L> getListenerList() {
```

```
            return listenerList;
        }
    }
```

```
package com.msu.myscribbler.utils;

public class Log {
   static final boolean LOG = true;

    public static void i(String tag, String string) {
        if (LOG) android.util.Log.i(tag,  string);
    }
    public static void e(String tag, String string) {
        if (LOG) android.util.Log.e(tag,  string);
    }
    public static void d(String tag, String string) {
        if (LOG) android.util.Log.d(tag,  string);
    }
    public static void v(String tag, String string) {
        if (LOG) android.util.Log.v(tag,  string);
    }
    public static void w(String tag, String string) {
        if (LOG) android.util.Log.w(tag,  string);
    }
}
```

```
package com.msu.myscribbler.utils;

import android.annotation.TargetApi;
import android.content.Context;
import android.content.res.Configuration;
import android.graphics.Bitmap;
import android.graphics.Canvas;
import android.graphics.Matrix;
import android.graphics.PointF;
import android.graphics.RectF;
import android.graphics.drawable.Drawable;
import android.net.Uri;
import android.os.Build;
import android.os.Build.VERSION;
import android.os.Build.VERSION_CODES;
import android.os.Bundle;
import android.os.Parcelable;
import android.util.AttributeSet;
import android.util.Log;
import android.view.GestureDetector;
import android.view.MotionEvent;
import android.view.ScaleGestureDetector;
```

```
import android.view.View;
import android.view.animation.AccelerateDecelerateInterpolator;
import android.widget.ImageView;
import android.widget.OverScroller;
import android.widget.Scroller;

public class TouchImageView extends ImageView {

    private static final String DEBUG = "DEBUG";

    //
    // SuperMin and SuperMax multipliers. Determine how much the image can
be
    // zoomed below or above the zoom boundaries, before animating back to the
    // min/max zoom boundary.
    //
    private static final float SUPER_MIN_MULTIPLIER = .75f;
    private static final float SUPER_MAX_MULTIPLIER = 1.25f;

   //
   // Scale of image ranges from minScale to maxScale, where minScale == 1
   // when the image is stretched to fit view.
   //
   private float normalizedScale;

   //
   // Matrix applied to image. MSCALE_X and MSCALE_Y should always be
equal.
   // MTRANS_X and MTRANS_Y are the other values used. prevMatrix is the
matrix
   // saved prior to the screen rotating.
   //
    private Matrix matrix, prevMatrix;

   private static enum State { NONE, DRAG, ZOOM, FLING, ANIMATE_ZOOM };
   private State state;

   private float minScale;
   private float maxScale;
   private float superMinScale;
   private float superMaxScale;
   private float[] m;

   private Context context;
   private Fling fling;

   private ScaleType mScaleType;

   private boolean imageRenderedAtLeastOnce;
   private boolean onDrawReady;
```

```
    private ZoomVariables  delayedZoomVariables;

    //
    // Size of view and previous view size (ie before rotation)
    //
    private int viewWidth, viewHeight, prevViewWidth,  prevViewHeight;

    //
    // Size of image when it is stretched to fit view. Before and After rotation.
    //
    private float matchViewWidth,  matchViewHeight,  prevMatchViewWidth,
prevMatchViewHeight;

    private ScaleGestureDetector  mScaleDetector;
    private GestureDetector mGestureDetector;
    private GestureDetector.OnDoubleTapListener  doubleTapListener = null;
    private OnTouchListener userTouchListener = null;
    private OnTouchImageViewListener  touchImageViewListener  = null;

    public TouchImageView(Context  context) {
        super(context);
        sharedConstructing(context);
    }

    public TouchImageView(Context  context, AttributeSet attrs) {
        super(context,  attrs);
        sharedConstructing(context);
    }

    public TouchImageView(Context  context, AttributeSet attrs, int defStyle) {
        super(context,  attrs, defStyle);
        sharedConstructing(context);
    }

    private void sharedConstructing(Context  context) {
        super.setClickable(true);
        this.context = context;
        mScaleDetector = new ScaleGestureDetector(context,  new
ScaleListener());
        mGestureDetector = new GestureDetector(context,  new GestureListener());
        matrix = new Matrix();
        prevMatrix = new Matrix();
        m = new float[9];
        normalizedScale  = 1;
        if (mScaleType == null) {
            mScaleType = ScaleType.FIT_CENTER;
        }
        minScale = 1;
        maxScale = 3;
```

```
        superMinScale = SUPER_MIN_MULTIPLIER * minScale;
        superMaxScale = SUPER_MAX_MULTIPLIER * maxScale;
        setImageMatrix(matrix);
        setScaleType(ScaleType.MATRIX);
        setState(State.NONE);
        onDrawReady = false;
        super.setOnTouchListener(new PrivateOnTouchListener());
    }

    @Override
    public void setOnTouchListener(View.OnTouchListener l) {
        userTouchListener = l;
    }

    public void setOnTouchImageViewListener(OnTouchImageViewListener l) {
        touchImageViewListener = l;
    }

    public void setOnDoubleTapListener(GestureDetector.OnDoubleTapListener l)
{
        doubleTapListener = l;
    }

    @Override
    public void setImageResource(int resId) {
        super.setImageResource(resId);
        savePreviousImageValues();
        fitImageToView();
    }

    @Override
    public void setImageBitmap(Bitmap bm) {
        super.setImageBitmap(bm);
        savePreviousImageValues();
        fitImageToView();
    }

    @Override
    public void setImageDrawable(Drawable drawable) {
        super.setImageDrawable(drawable);
        savePreviousImageValues();
        fitImageToView();
    }

    @Override
    public void setImageURI(Uri uri) {
        super.setImageURI(uri);
        savePreviousImageValues();
        fitImageToView();
    }
```

```
  @Override
  public void setScaleType(ScaleType type) {
     if (type == ScaleType.FIT_START || type == ScaleType.FIT_END) {
        throw new UnsupportedOperationException("TouchImageView does not
support FIT_START or FIT_END");
     }
     if (type == ScaleType.MATRIX) {
        super.setScaleType(ScaleType.MATRIX);

     } else {
        mScaleType = type;
        if (onDrawReady) {
           //
           // If the image is already rendered, scaleType has been called
programmatically
           // and the TouchImageView should be updated with the new
scaleType.
           //
           setZoom(this);
        }
     }
  }

  @Override
  public ScaleType getScaleType() {
     return mScaleType;
  }

  /**
   * Returns false if image is in initial, unzoomed state. False, otherwise.
   * @return true if image is zoomed
   */
  public boolean isZoomed() {
     return normalizedScale != 1;
  }

  /**
   * Return a Rect representing the zoomed image.
   * @return rect representing zoomed image
   */
  public RectF getZoomedRect() {
     if (mScaleType == ScaleType.FIT_XY) {
        throw new UnsupportedOperationException("getZoomedRect() not
supported with FIT_XY");
     }
     PointF topLeft = transformCoordTouchToBitmap(0, 0, true);
     PointF bottomRight = transformCoordTouchToBitmap(viewWidth,
viewHeight, true);
```

```
        float w = getDrawable().getIntrinsicWidth();
        float h = getDrawable().getIntrinsicHeight();
        return new RectF(topLeft.x / w, topLeft.y / h, bottomRight.x / w,
bottomRight.y / h);
    }

    /**
     * Save the current matrix and view dimensions
     * in the prevMatrix and prevView variables.
     */
    private void savePreviousImageValues() {
        if (matrix != null && viewHeight != 0 && viewWidth != 0) {
            matrix.getValues(m);
            prevMatrix.setValues(m);
            prevMatchViewHeight = matchViewHeight;
            prevMatchViewWidth = matchViewWidth;
            prevViewHeight = viewHeight;
            prevViewWidth = viewWidth;
        }
    }

    @Override
    public Parcelable onSaveInstanceState() {
        Bundle bundle = new Bundle();
        bundle.putParcelable("instanceState", super.onSaveInstanceState());
        bundle.putFloat("saveScale", normalizedScale);
        bundle.putFloat("matchViewHeight", matchViewHeight);
        bundle.putFloat("matchViewWidth", matchViewWidth);
        bundle.putInt("viewWidth", viewWidth);
        bundle.putInt("viewHeight", viewHeight);
        matrix.getValues(m);
        bundle.putFloatArray("matrix", m);
        bundle.putBoolean("imageRendered", imageRenderedAtLeastOnce);
        return bundle;
    }

    @Override
    public void onRestoreInstanceState(Parcelable state) {
        if (state instanceof Bundle) {
            Bundle bundle = (Bundle) state;
            normalizedScale = bundle.getFloat("saveScale");
            m = bundle.getFloatArray("matrix");
            prevMatrix.setValues(m);
            prevMatchViewHeight = bundle.getFloat("matchViewHeight");
            prevMatchViewWidth = bundle.getFloat("matchViewWidth");
            prevViewHeight = bundle.getInt("viewHeight");
            prevViewWidth = bundle.getInt("viewWidth");
            imageRenderedAtLeastOnce = bundle.getBoolean("imageRendered");
            super.onRestoreInstanceState(bundle.getParcelable("instanceState"));
            return;
```

```
        }

        super.onRestoreInstanceState(state);
    }

    @Override
    protected void onDraw(Canvas canvas) {
        onDrawReady = true;
        imageRenderedAtLeastOnce = true;
        if (delayedZoomVariables != null) {
            setZoom(delayedZoomVariables.scale, delayedZoomVariables.focusX,
delayedZoomVariables.focusY, delayedZoomVariables.scaleType);
            delayedZoomVariables = null;
        }
        super.onDraw(canvas);
    }

    @Override
    public void onConfigurationChanged(Configuration newConfig) {
        super.onConfigurationChanged(newConfig);
        savePreviousImageValues();
    }

    /**
     * Get the max zoom multiplier.
     * @return max zoom multiplier.
     */
    public float getMaxZoom() {
        return maxScale;
    }

    /**
     * Set the max zoom multiplier. Default value: 3.
     * @param max max zoom multiplier.
     */
    public void setMaxZoom(float max) {
        maxScale = max;
        superMaxScale = SUPER_MAX_MULTIPLIER * maxScale;
    }

    /**
     * Get the min zoom multiplier.
     * @return min zoom multiplier.
     */
    public float getMinZoom() {
        return minScale;
    }

    /**
     * Get the current zoom. This is the zoom relative to the initial
```

```
     * scale, not the original resource.
     * @return current zoom multiplier.
     */
    public float getCurrentZoom() {
        return normalizedScale;
    }

    /**
     * Set the min zoom multiplier. Default value: 1.
     * @param min min zoom multiplier.
     */
    public void setMinZoom(float min) {
        minScale = min;
        superMinScale = SUPER_MIN_MULTIPLIER * minScale;
    }

    /**
     * Reset zoom and translation to initial state.
     */
    public void resetZoom() {
        normalizedScale = 1;
        fitImageToView();
    }

    /**
     * Set zoom to the specified scale. Image will be centered by default.
     * @param scale
     */
    public void setZoom(float scale) {
        setZoom(scale, 0.5f, 0.5f);
    }

    /**
     * Set zoom to the specified scale. Image will be centered around the point
     * (focusX, focusY). These floats range from 0 to 1 and denote the focus point
     * as a fraction from the left and top of the view. For example, the top left
     * corner of the image would be (0, 0). And the bottom right corner would be
(1, 1).
     * @param scale
     * @param focusX
     * @param focusY
     */
    public void setZoom(float scale, float focusX, float focusY) {
        setZoom(scale, focusX, focusY, mScaleType);
    }

    /**
     * Set zoom to the specified scale. Image will be centered around the point
     * (focusX, focusY). These floats range from 0 to 1 and denote the focus point
     * as a fraction from the left and top of the view. For example, the top left
```

```
     * corner of the image would be (0, 0). And the bottom right corner would be
(1, 1).
     * @param scale
     * @param focusX
     * @param focusY
     * @param scaleType
     */
    public void setZoom(float scale, float focusX, float focusY, ScaleType
scaleType) {
        //
        // setZoom can be called before the image is on the screen, but at this point,
        // image and view sizes have not yet been calculated in onMeasure. Thus,
we should
        // delay calling setZoom until the view has been measured.
        //
        if (!onDrawReady) {
            delayedZoomVariables = new ZoomVariables(scale, focusX, focusY,
scaleType);
            return;
        }

        if (scaleType != mScaleType) {
            setScaleType(scaleType);
        }
        resetZoom();
        scaleImage(scale, viewWidth / 2, viewHeight / 2, true);
        matrix.getValues(m);
        m[Matrix.MTRANS_X] = -((focusX * getImageWidth()) - (viewWidth * 0.5f));
        m[Matrix.MTRANS_Y] = -((focusY * getImageHeight()) - (viewHeight *
0.5f));
        matrix.setValues(m);
        fixTrans();
        setImageMatrix(matrix);
    }

    /**
     * Set zoom parameters equal to another TouchImageView. Including scale,
position,
     * and ScaleType.
     * @param TouchImageView
     */
    public void setZoom(TouchImageView img) {
        PointF center = img.getScrollPosition();
        setZoom(img.getCurrentZoom(), center.x, center.y, img.getScaleType());
    }

    /**
     * Return the point at the center of the zoomed image. The PointF coordinates
range
```

```
     * in value between 0 and 1 and the focus point is denoted as a fraction from
the left
     * and top of the view. For example, the top left corner of the image would be
(0, 0).
     * And the bottom right corner would be (1, 1).
     * @return PointF representing the scroll position of the zoomed image.
     */
    public PointF getScrollPosition() {
        Drawable drawable = getDrawable();
        if (drawable == null) {
            return null;
        }
        int drawableWidth = drawable.getIntrinsicWidth();
        int drawableHeight = drawable.getIntrinsicHeight();

        PointF point = transformCoordTouchToBitmap(viewWidth / 2, viewHeight /
2, true);
        point.x /= drawableWidth;
        point.y /= drawableHeight;
        return point;
    }

    /**
     * Set the focus point of the zoomed image. The focus points are denoted as a
fraction from the
     * left and top of the view. The focus points can range in value between 0 and
1.
     * @param focusX
     * @param focusY
     */
    public void setScrollPosition(float focusX, float focusY) {
        setZoom(normalizedScale, focusX, focusY);
    }

    /**
     * Performs boundary checking and fixes the image matrix if it
     * is out of bounds.
     */
    private void fixTrans() {
        matrix.getValues(m);
        float transX = m[Matrix.MTRANS_X];
        float transY = m[Matrix.MTRANS_Y];

        float fixTransX = getFixTrans(transX, viewWidth, getImageWidth());
        float fixTransY = getFixTrans(transY, viewHeight, getImageHeight());

        if (fixTransX != 0 || fixTransY != 0) {
            matrix.postTranslate(fixTransX, fixTransY);
        }
    }
```

```
    /**
     * When transitioning from zooming from focus to zoom from center (or vice
versa)
     * the image can become unaligned within the view. This is apparent when
zooming
     * quickly. When the content size is less than the view size, the content will
often
     * be centered incorrectly within the view. fixScaleTrans first calls fixTrans()
and
     * then makes sure the image is centered correctly within the view.
     */
    private void fixScaleTrans() {
        fixTrans();
        matrix.getValues(m);
        if (getImageWidth() < viewWidth) {
            m[Matrix.MTRANS_X] = (viewWidth - getImageWidth()) / 2;
        }

        if (getImageHeight() < viewHeight) {
            m[Matrix.MTRANS_Y] = (viewHeight - getImageHeight()) / 2;
        }
        matrix.setValues(m);
    }

    private float getFixTrans(float trans, float viewSize, float contentSize) {
        float minTrans, maxTrans;

        if (contentSize <= viewSize) {
            minTrans = 0;
            maxTrans = viewSize - contentSize;

        } else {
            minTrans = viewSize - contentSize;
            maxTrans = 0;
        }

        if (trans < minTrans)
            return -trans + minTrans;
        if (trans > maxTrans)
            return -trans + maxTrans;
        return 0;
    }

    private float getFixDragTrans(float delta, float viewSize, float contentSize) {
        if (contentSize <= viewSize) {
            return 0;
        }
        return delta;
    }
```

```
  private float getImageWidth() {
     return matchViewWidth * normalizedScale;
  }

  private float getImageHeight() {
     return matchViewHeight * normalizedScale;
  }

  @Override
  protected void onMeasure(int widthMeasureSpec, int heightMeasureSpec) {
     Drawable drawable = getDrawable();
     if (drawable == null || drawable.getIntrinsicWidth() == 0 ||
drawable.getIntrinsicHeight() == 0) {
        setMeasuredDimension(0, 0);
        return;
     }

     int drawableWidth = drawable.getIntrinsicWidth();
     int drawableHeight = drawable.getIntrinsicHeight();
     int widthSize = MeasureSpec.getSize(widthMeasureSpec);
     int widthMode = MeasureSpec.getMode(widthMeasureSpec);
     int heightSize = MeasureSpec.getSize(heightMeasureSpec);
     int heightMode = MeasureSpec.getMode(heightMeasureSpec);
     viewWidth = setViewSize(widthMode, widthSize, drawableWidth);
     viewHeight = setViewSize(heightMode, heightSize, drawableHeight);

     //
     // Set view dimensions
     //
     setMeasuredDimension(viewWidth, viewHeight);

     //
     // Fit content within view
     //
     fitImageToView();
  }

  /**
   * If the normalizedScale is equal to 1, then the image is made to fit the
screen. Otherwise,
   * it is made to fit the screen according to the dimensions of the previous
image matrix. This
   * allows the image to maintain its zoom after rotation.
   */
  private void fitImageToView() {
     Drawable drawable = getDrawable();
     if (drawable == null || drawable.getIntrinsicWidth() == 0 ||
drawable.getIntrinsicHeight() == 0) {
        return;
```

```
        }
        if (matrix == null || prevMatrix == null) {
            return;
        }

        int drawableWidth = drawable.getIntrinsicWidth();
        int drawableHeight = drawable.getIntrinsicHeight();

        //
        // Scale image for view
        //
        float scaleX = (float) viewWidth / drawableWidth;
        float scaleY = (float) viewHeight / drawableHeight;

        switch (mScaleType) {
        case CENTER:
            scaleX = scaleY = 1;
            break;

        case CENTER_CROP:
            scaleX = scaleY = Math.max(scaleX, scaleY);
            break;

        case CENTER_INSIDE:
            scaleX = scaleY = Math.min(1, Math.min(scaleX, scaleY));

        case FIT_CENTER:
            scaleX = scaleY = Math.min(scaleX, scaleY);
            break;

        case FIT_XY:
            break;

        default:
            //
            // FIT_START and FIT_END not supported
            //
            throw new UnsupportedOperationException("TouchImageView does not
support FIT_START or FIT_END");

        }

        //
        // Center the image
        //
        float redundantXSpace = viewWidth - (scaleX * drawableWidth);
        float redundantYSpace = viewHeight - (scaleY * drawableHeight);
        matchViewWidth = viewWidth - redundantXSpace;
        matchViewHeight = viewHeight - redundantYSpace;
        if (!isZoomed() && !imageRenderedAtLeastOnce) {
```

```
            //
            // Stretch and center image to fit view
            //
            matrix.setScale(scaleX, scaleY);
            matrix.postTranslate(redundantXSpace / 2, redundantYSpace / 2);
            normalizedScale = 1;

        } else {
            //
            // These values should never be 0 or we will set viewWidth and
viewHeight
            // to NaN in translateMatrixAfterRotate. To avoid this, call
savePreviousImageValues
            // to set them equal to the current values.
            //
            if (prevMatchViewWidth == 0 || prevMatchViewHeight == 0) {
                savePreviousImageValues();
            }

            prevMatrix.getValues(m);

            //
            // Rescale Matrix after rotation
            //
            m[Matrix.MSCALE_X] = matchViewWidth / drawableWidth *
normalizedScale;
            m[Matrix.MSCALE_Y] = matchViewHeight / drawableHeight *
normalizedScale;

            //
            // TransX and TransY from previous matrix
            //
            float transX = m[Matrix.MTRANS_X];
            float transY = m[Matrix.MTRANS_Y];

            //
            // Width
            //
            float prevActualWidth = prevMatchViewWidth * normalizedScale;
            float actualWidth = getImageWidth();
            translateMatrixAfterRotate(Matrix.MTRANS_X, transX, prevActualWidth,
actualWidth, prevViewWidth, viewWidth, drawableWidth);

            //
            // Height
            //
            float prevActualHeight = prevMatchViewHeight * normalizedScale;
            float actualHeight = getImageHeight();
            translateMatrixAfterRotate(Matrix.MTRANS_Y, transY, prevActualHeight,
actualHeight, prevViewHeight, viewHeight, drawableHeight);
```

```
            //
            // Set the matrix to the adjusted scale and translate values.
            //
            matrix.setValues(m);
        }
        fixTrans();
        setImageMatrix(matrix);
    }

    /**
     * Set view dimensions based on layout params
     *
     * @param mode
     * @param size
     * @param drawableWidth
     * @return
     */
    private int setViewSize(int mode, int size, int drawableWidth) {
        int viewSize;
        switch (mode) {
        case MeasureSpec.EXACTLY:
            viewSize = size;
            break;

        case MeasureSpec.AT_MOST:
            viewSize = Math.min(drawableWidth, size);
            break;

        case MeasureSpec.UNSPECIFIED:
            viewSize = drawableWidth;
            break;

        default:
            viewSize = size;
            break;
        }
        return viewSize;
    }

    /**
     * After rotating, the matrix needs to be translated. This function finds the area
of image
     * which was previously centered and adjusts translations so that is again the
center, post-rotation.
     *
     * @param axis Matrix.MTRANS_X or Matrix.MTRANS_Y
     * @param trans the value of trans in that axis before the rotation
     * @param prevImageSize the width/height of the image before the rotation
     * @param imageSize width/height of the image after rotation
```

```
     * @param prevViewSize width/height of view before rotation
     * @param viewSize width/height of view after rotation
     * @param drawableSize width/height of drawable
     */
    private void translateMatrixAfterRotate(int axis, float trans, float
prevImageSize, float imageSize, int prevViewSize, int viewSize, int
drawableSize) {
        if (imageSize < viewSize) {
            //
            // The width/height of image is less than the view's width/height. Center it.
            //
            m[axis] = (viewSize - (drawableSize * m[Matrix.MSCALE_X])) * 0.5f;

        } else if (trans > 0) {
            //
            // The image is larger than the view, but was not before rotation. Center
it.
            //
            m[axis] = -((imageSize - viewSize) * 0.5f);

        } else {
            //
            // Find the area of the image which was previously centered in the view.
Determine its distance
            // from the left/top side of the view as a fraction of the entire image's
width/height. Use that percentage
            // to calculate the trans in the new view width/height.
            //
            float percentage = (Math.abs(trans) + (0.5f * prevViewSize)) /
prevImageSize;
            m[axis] = -((percentage * imageSize) - (viewSize * 0.5f));
        }
    }

    private void setState(State state) {
        this.state = state;
    }

    public boolean canScrollHorizontallyFroyo(int direction) {
        return canScrollHorizontally(direction);
    }

    @Override
    public boolean canScrollHorizontally(int direction) {
        matrix.getValues(m);
        float x = m[Matrix.MTRANS_X];

        if (getImageWidth() < viewWidth) {
            return false;
```

```
      } else if (x >= -1 && direction < 0) {
         return false;

      } else if (Math.abs(x) + viewWidth + 1 >= getImageWidth() && direction > 0)
{
         return false;
      }

      return true;
   }

   /**
    * Gesture Listener detects a single click or long click and passes that on
    * to the view's listener.
    * @author Ortiz
    *
    */
   private class GestureListener extends
GestureDetector.SimpleOnGestureListener {

      @Override
      public boolean onSingleTapConfirmed(MotionEvent e)
      {
         if(doubleTapListener != null) {
            return doubleTapListener.onSingleTapConfirmed(e);
         }
         return performClick();
      }

      @Override
      public void onLongPress(MotionEvent e)
      {
         performLongClick();
      }

      @Override
      public boolean onFling(MotionEvent e1, MotionEvent e2, float velocityX,
float velocityY)
      {
         if (fling != null) {
            //
            // If a previous fling is still active, it should be cancelled so that two
flings
            // are not run simultaenously.
            //
            fling.cancelFling();
         }
         fling = new Fling((int) velocityX, (int) velocityY);
         compatPostOnAnimation(fling);
         return super.onFling(e1, e2, velocityX, velocityY);
```

```
        }

        @Override
        public boolean onDoubleTap(MotionEvent e) {
            boolean consumed = false;
            if(doubleTapListener != null) {
                consumed = doubleTapListener.onDoubleTap(e);
            }
            if (state == State.NONE) {
                float targetZoom = (normalizedScale == minScale) ? maxScale :
minScale;
                DoubleTapZoom doubleTap = new DoubleTapZoom(targetZoom,
e.getX(), e.getY(), false);
                compatPostOnAnimation(doubleTap);
                consumed = true;
            }
            return consumed;
        }

        @Override
        public boolean onDoubleTapEvent(MotionEvent e) {
            if(doubleTapListener != null) {
                return doubleTapListener.onDoubleTapEvent(e);
            }
            return false;
        }
    }

    public interface OnTouchImageViewListener {
        public void onMove();
    }

    /**
     * Responsible for all touch events. Handles the heavy lifting of drag and also
sends
     * touch events to Scale Detector and Gesture Detector.
     * @author Ortiz
     *
     */
    private class PrivateOnTouchListener implements OnTouchListener {

        //
        // Remember last point position for dragging
        //
        private PointF last = new PointF();

        @Override
        public boolean onTouch(View v, MotionEvent event) {
            mScaleDetector.onTouchEvent(event);
            mGestureDetector.onTouchEvent(event);
```

```
        PointF curr = new PointF(event.getX(), event.getY());

        if (state == State.NONE || state == State.DRAG || state == State.FLING) {
            switch (event.getAction()) {
                case MotionEvent.ACTION_DOWN:
                    last.set(curr);
                    if (fling != null)
                        fling.cancelFling();
                    setState(State.DRAG);
                    break;

                case MotionEvent.ACTION_MOVE:
                    if (state == State.DRAG) {
                        float deltaX = curr.x - last.x;
                        float deltaY = curr.y - last.y;
                        float fixTransX = getFixDragTrans(deltaX, viewWidth,
getImageWidth());
                        float fixTransY = getFixDragTrans(deltaY, viewHeight,
getImageHeight());
                        matrix.postTranslate(fixTransX, fixTransY);
                        fixTrans();
                        last.set(curr.x, curr.y);
                    }
                    break;

                case MotionEvent.ACTION_UP:
                case MotionEvent.ACTION_POINTER_UP:
                    setState(State.NONE);
                    break;
            }
        }

        setImageMatrix(matrix);

        //
        // User-defined OnTouchListener
        //
        if(userTouchListener != null) {
            userTouchListener.onTouch(v, event);
        }

        //
        // OnTouchImageViewListener is set: TouchImageView dragged by user.
        //
        if (touchImageViewListener != null) {
            touchImageViewListener.onMove();
        }

        //
        // indicate event was handled
```

```
            //
            return true;
        }
    }

    /**
     * ScaleListener detects user two finger scaling and scales image.
     * @author Ortiz
     *
     */
    private class ScaleListener extends
ScaleGestureDetector.SimpleOnScaleGestureListener {
        @Override
        public boolean onScaleBegin(ScaleGestureDetector detector) {
            setState(State.ZOOM);
            return true;
        }

        @Override
        public boolean onScale(ScaleGestureDetector detector) {
            scaleImage(detector.getScaleFactor(), detector.getFocusX(),
detector.getFocusY(), true);

            //
            // OnTouchImageViewListener is set: TouchImageView pinch zoomed by
user.
            //
            if (touchImageViewListener != null) {
                touchImageViewListener.onMove();
            }
            return true;
        }

        @Override
        public void onScaleEnd(ScaleGestureDetector detector) {
            super.onScaleEnd(detector);
            setState(State.NONE);
            boolean animateToZoomBoundary = false;
            float targetZoom = normalizedScale;
            if (normalizedScale > maxScale) {
                targetZoom = maxScale;
                animateToZoomBoundary = true;

            } else if (normalizedScale < minScale) {
                targetZoom = minScale;
                animateToZoomBoundary = true;
            }

            if (animateToZoomBoundary) {
```

```
            DoubleTapZoom doubleTap = new DoubleTapZoom(targetZoom,
viewWidth / 2, viewHeight / 2, true);
            compatPostOnAnimation(doubleTap);
          }
        }
    }

    private void scaleImage(double deltaScale, float focusX, float focusY, boolean
stretchImageToSuper) {

        float lowerScale, upperScale;
        if (stretchImageToSuper) {
            lowerScale = superMinScale;
            upperScale = superMaxScale;

        } else {
            lowerScale = minScale;
            upperScale = maxScale;
        }

        float origScale = normalizedScale;
        normalizedScale *= deltaScale;
        if (normalizedScale > upperScale) {
            normalizedScale = upperScale;
            deltaScale = upperScale / origScale;
        } else if (normalizedScale < lowerScale) {
            normalizedScale = lowerScale;
            deltaScale = lowerScale / origScale;
        }

        matrix.postScale((float) deltaScale, (float) deltaScale, focusX, focusY);
        fixScaleTrans();
    }

    /**
     * DoubleTapZoom calls a series of runnables which apply
     * an animated zoom in/out graphic to the image.
     * @author Ortiz
     *
     */
    private class DoubleTapZoom implements Runnable {

        private long startTime;
        private static final float ZOOM_TIME = 500;
        private float startZoom, targetZoom;
        private float bitmapX, bitmapY;
        private boolean stretchImageToSuper;
        private AccelerateDecelerateInterpolator interpolator = new
AccelerateDecelerateInterpolator();
        private PointF startTouch;
```

```
    private PointF endTouch;

    DoubleTapZoom(float targetZoom, float focusX, float focusY, boolean
stretchImageToSuper) {
        setState(State.ANIMATE_ZOOM);
        startTime = System.currentTimeMillis();
        this.startZoom = normalizedScale;
        this.targetZoom = targetZoom;
        this.stretchImageToSuper = stretchImageToSuper;
        PointF bitmapPoint = transformCoordTouchToBitmap(focusX, focusY,
false);
        this.bitmapX = bitmapPoint.x;
        this.bitmapY = bitmapPoint.y;

        //
        // Used for translating image during scaling
        //
        startTouch = transformCoordBitmapToTouch(bitmapX, bitmapY);
        endTouch = new PointF(viewWidth / 2, viewHeight / 2);
    }

    @Override
    public void run() {
        float t = interpolate();
        double deltaScale = calculateDeltaScale(t);
        scaleImage(deltaScale, bitmapX, bitmapY, stretchImageToSuper);
        translateImageToCenterTouchPosition(t);
        fixScaleTrans();
        setImageMatrix(matrix);

        //
        // OnTouchImageViewListener is set: double tap runnable updates
listener
        // with every frame.
        //
        if (touchImageViewListener != null) {
            touchImageViewListener.onMove();
        }

        if (t < 1f) {
            //
            // We haven't finished zooming
            //
            compatPostOnAnimation(this);

        } else {
            //
            // Finished zooming
            //
            setState(State.NONE);
```

```
            }
        }

        /**
        * Interpolate between where the image should start and end in order to
translate
        * the image so that the point that is touched is what ends up centered at
the end
        * of the zoom.
        * @param t
        */
        private void translateImageToCenterTouchPosition(float t) {
            float targetX = startTouch.x + t * (endTouch.x - startTouch.x);
            float targetY = startTouch.y + t * (endTouch.y - startTouch.y);
            PointF curr = transformCoordBitmapToTouch(bitmapX, bitmapY);
            matrix.postTranslate(targetX - curr.x, targetY - curr.y);
        }

        /**
        * Use interpolator to get t
        * @return
        */
        private float interpolate() {
            long currTime = System.currentTimeMillis();
            float elapsed = (currTime - startTime) / ZOOM_TIME;
            elapsed = Math.min(1f, elapsed);
            return interpolator.getInterpolation(elapsed);
        }

        /**
        * Interpolate the current targeted zoom and get the delta
        * from the current zoom.
        * @param t
        * @return
        */
        private double calculateDeltaScale(float t) {
            double zoom = startZoom + t * (targetZoom - startZoom);
            return zoom / normalizedScale;
        }
    }

    /**
     * This function will transform the coordinates in the touch event to the
coordinate
     * system of the drawable that the imageview contain
     * @param x x-coordinate of touch event
     * @param y y-coordinate of touch event
     * @param clipToBitmap Touch event may occur within view, but outside
image content. True, to clip return value
     *            to the bounds of the bitmap size.
```

```
     * @return Coordinates of the point touched, in the coordinate system of the
original drawable.
     */
    private PointF transformCoordTouchToBitmap(float x, float y, boolean
clipToBitmap) {
        matrix.getValues(m);
        float origW = getDrawable().getIntrinsicWidth();
        float origH = getDrawable().getIntrinsicHeight();
        float transX = m[Matrix.MTRANS_X];
        float transY = m[Matrix.MTRANS_Y];
        float finalX = ((x - transX) * origW) / getImageWidth();
        float finalY = ((y - transY) * origH) / getImageHeight();

        if (clipToBitmap) {
          finalX = Math.min(Math.max(finalX, 0), origW);
          finalY = Math.min(Math.max(finalY, 0), origH);
        }

        return new PointF(finalX , finalY);
    }

    /**
     * Inverse of transformCoordTouchToBitmap. This function will transform the
coordinates in the
     * drawable's coordinate system to the view's coordinate system.
     * @param bx x-coordinate in original bitmap coordinate system
     * @param by y-coordinate in original bitmap coordinate system
     * @return Coordinates of the point in the view's coordinate system.
     */
    private PointF transformCoordBitmapToTouch(float bx, float by) {
        matrix.getValues(m);
        float origW = getDrawable().getIntrinsicWidth();
        float origH = getDrawable().getIntrinsicHeight();
        float px = bx / origW;
        float py = by / origH;
        float finalX = m[Matrix.MTRANS_X] + getImageWidth() * px;
        float finalY = m[Matrix.MTRANS_Y] + getImageHeight() * py;
        return new PointF(finalX , finalY);
    }

    /**
     * Fling launches sequential runnables which apply
     * the fling graphic to the image. The values for the translation
     * are interpolated by the Scroller.
     * @author Ortiz
     *
     */
    private class Fling implements Runnable {

        CompatScroller scroller;
```

```
    int currX, currY;

    Fling(int velocityX, int velocityY) {
        setState(State.FLING);
        scroller = new CompatScroller(context);
        matrix.getValues(m);

        int startX = (int) m[Matrix.MTRANS_X];
        int startY = (int) m[Matrix.MTRANS_Y];
        int minX, maxX, minY, maxY;

        if (getImageWidth() > viewWidth) {
            minX = viewWidth - (int) getImageWidth();
            maxX = 0;

        } else {
            minX = maxX = startX;
        }

        if (getImageHeight() > viewHeight) {
            minY = viewHeight - (int) getImageHeight();
            maxY = 0;

        } else {
            minY = maxY = startY;
        }

        scroller.fling(startX, startY, (int) velocityX, (int) velocityY, minX,
                maxX, minY, maxY);
        currX = startX;
        currY = startY;
    }

    public void cancelFling() {
        if (scroller != null) {
            setState(State.NONE);
            scroller.forceFinished(true);
        }
    }

     @Override
     public void run() {

         //
         // OnTouchImageViewListener is set: TouchImageView listener has been
flung by user.
         // Listener runnable updated with each frame of fling animation.
         //
         if (touchImageViewListener != null) {
             touchImageViewListener.onMove();
```

```
            }

            if (scroller.isFinished()) {
                scroller = null;
                return;
            }

            if (scroller.computeScrollOffset()) {
                int newX = scroller.getCurrX();
                int newY = scroller.getCurrY();
                int transX = newX - currX;
                int transY = newY - currY;
                currX = newX;
                currY = newY;
                matrix.postTranslate(transX, transY);
                fixTrans();
                setImageMatrix(matrix);
                compatPostOnAnimation(this);
            }
        }
    }

    @TargetApi(Build.VERSION_CODES.GINGERBREAD)
    private class CompatScroller {
        Scroller scroller;
        OverScroller overScroller;
        boolean isPreGingerbread;

        public CompatScroller(Context context) {
            if (VERSION.SDK_INT < VERSION_CODES.GINGERBREAD) {
                isPreGingerbread = true;
                scroller = new Scroller(context);

            } else {
                isPreGingerbread = false;
                overScroller = new OverScroller(context);
            }
        }

        public void fling(int startX, int startY, int velocityX, int velocityY, int minX, int
maxX, int minY, int maxY) {
            if (isPreGingerbread) {
                scroller.fling(startX, startY, velocityX, velocityY, minX, maxX, minY,
maxY);
            } else {
                overScroller.fling(startX, startY, velocityX, velocityY, minX, maxX,
minY, maxY);
            }
        }
```

```
    public void forceFinished(boolean  finished) {
        if (isPreGingerbread) {
            scroller.forceFinished(finished);
        } else {
            overScroller.forceFinished(finished);
        }
    }

    public boolean isFinished() {
        if (isPreGingerbread) {
            return scroller.isFinished();
        } else {
            return overScroller.isFinished();
        }
    }

    public boolean computeScrollOffset() {
        if (isPreGingerbread) {
            return scroller.computeScrollOffset();
        } else {
            overScroller.computeScrollOffset();
            return overScroller.computeScrollOffset();
        }
    }

    public int getCurrX() {
        if (isPreGingerbread) {
            return scroller.getCurrX();
        } else {
            return overScroller.getCurrX();
        }
    }

    public int getCurrY() {
        if (isPreGingerbread) {
            return scroller.getCurrY();
        } else {
            return overScroller.getCurrY();
        }
    }
}

@TargetApi(Build.VERSION_CODES.JELLY_BEAN)
 private void compatPostOnAnimation(Runnable  runnable) {
    if (VERSION.SDK_INT  >= VERSION_CODES.JELLY_BEAN)  {
        postOnAnimation(runnable);

    } else {
        postDelayed(runnable,  1000/60);
    }
```

```
  }

  private class ZoomVariables {
     public float scale;
     public float focusX;
     public float focusY;
     public ScaleType scaleType;

     public ZoomVariables(float scale, float focusX, float focusY, ScaleType
scaleType) {
        this.scale = scale;
        this.focusX = focusX;
        this.focusY = focusY;
        this.scaleType = scaleType;
     }
  }

  private void printMatrixInfo() {
     float[] n = new float[9];
     matrix.getValues(n);
     Log.d(DEBUG, "Scale: " + n[Matrix.MSCALE_X] + " TransX: " +
n[Matrix.MTRANS_X] + " TransY: " + n[Matrix.MTRANS_Y]);
  }
}



package common.googlecode;


/*
* Copyright 2013 The Android Open Source Project
*
* Licensed under the Apache License, Version 2.0 (the "License");
* you may not use this file except in compliance with the License.
* You may obtain a copy of the License at
*
*      http://www.apache.org/licenses/LICENSE-2.0
*
* Unless required by applicable law or agreed to in writing, software
* distributed under the License is distributed on an "AS IS" BASIS,
* WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or
implied.
* See the License for the specific language governing permissions and
* limitations under the License.
*/

import android.content.Context;
import android.graphics.Color;
```

```
import android.graphics.Typeface;
import android.support.v4.view.PagerAdapter;
import android.support.v4.view.ViewPager;
import android.util.AttributeSet;
import android.util.SparseArray;
import android.util.TypedValue;
import android.view.Gravity;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.HorizontalScrollView;
import android.widget.LinearLayout;
import android.widget.TextView;

/**
* To be used with ViewPager to provide a tab indicator component which give
constant feedback as to
* the user's scroll progress.
* <p>
* To use the component, simply add it to your view hierarchy. Then in your
* {@link android.app.Activity} or {@link android.support.v4.app.Fragment} call
* {@link #setViewPager(ViewPager)} providing it the ViewPager this layout is
being used for.
* <p>
* The colors can be customized in two ways. The first and simplest is to provide
an array of colors
* via {@link #setSelectedIndicatorColors(int...)}. The
* alternative is via the {@link TabColorizer} interface which provides you
complete control over
* which color is used for any individual position.
* <p>
* The views used as tabs can be customized by calling {@link
#setCustomTabView(int, int)},
* providing the layout ID of your custom layout.
*/
public class SlidingTabLayout extends HorizontalScrollView {
  /**
   * Allows complete control over the colors drawn in the tab layout. Set with
   * {@link #setCustomTabColorizer(TabColorizer)}.
   */
  public interface TabColorizer {

      /**
       * @return return the color of the indicator used when {@code position} is
selected.
       */
      int getIndicatorColor(int position);

  }
```

```
  private static final int TITLE_OFFSET_DIPS = 24;
  private static final int TAB_VIEW_PADDING_DIPS = 16;
  private static final int TAB_VIEW_TEXT_SIZE_SP = 15;

  private int mTitleOffset;

  private int mTabViewLayoutId;
  private int mTabViewTextViewId;
  private boolean mDistributeEvenly;

  private ViewPager mViewPager;
  private SparseArray<String> mContentDescriptions = new
SparseArray<String>();
  private ViewPager.OnPageChangeListener mViewPagerPageChangeListener;

  private final SlidingTabStrip mTabStrip;

  public SlidingTabLayout(Context context) {
    this(context, null);
  }

  public SlidingTabLayout(Context context, AttributeSet attrs) {
    this(context, attrs, 0);
  }

  public SlidingTabLayout(Context context, AttributeSet attrs, int defStyle) {
    super(context, attrs, defStyle);

    // Disable the Scroll Bar
    setHorizontalScrollBarEnabled(false);
    // Make sure that the Tab Strips fills this View
    setFillViewport(true);

    mTitleOffset = (int) (TITLE_OFFSET_DIPS *
getResources().getDisplayMetrics().density);

    mTabStrip = new SlidingTabStrip(context);
    addView(mTabStrip, LayoutParams.MATCH_PARENT,
LayoutParams.WRAP_CONTENT);
  }

  /**
   * Set the custom {@link TabColorizer} to be used.
   *
   * If you only require simple custmisation then you can use
   * {@link #setSelectedIndicatorColors(int...)} to achieve
   * similar effects.
   */
  public void setCustomTabColorizer(TabColorizer tabColorizer) {
    mTabStrip.setCustomTabColorizer(tabColorizer);
```

```
    }

    public void setDistributeEvenly(boolean distributeEvenly) {
        mDistributeEvenly = distributeEvenly;
    }

    /**
     * Sets the colors to be used for indicating the selected tab. These colors are
treated as a
     * circular array. Providing one color will mean that all tabs are indicated with
the same color.
     */
    public void setSelectedIndicatorColors(int... colors) {
        mTabStrip.setSelectedIndicatorColors(colors);
    }

    /**
     * Set the {@link ViewPager.OnPageChangeListener}. When using {@link
SlidingTabLayout} you are
     * required to set any {@link ViewPager.OnPageChangeListener} through this
method. This is so
     * that the layout can update it's scroll position correctly.
     *
     * @see
ViewPager#setOnPageChangeListener(ViewPager.OnPageChangeListener)
     */
    public void setOnPageChangeListener(ViewPager.OnPageChangeListener
listener) {
        mViewPagerPageChangeListener = listener;
    }

    /**
     * Set the custom layout to be inflated for the tab views.
     *
     * @param layoutResId Layout id to be inflated
     * @param textViewId id of the {@link TextView} in the inflated view
     */
    public void setCustomTabView(int layoutResId, int textViewId) {
        mTabViewLayoutId = layoutResId;
        mTabViewTextViewId = textViewId;
    }

    /**
     * Sets the associated view pager. Note that the assumption here is that the
pager content
     * (number of tabs and tab titles) does not change after this call has been
made.
     */
    public void setViewPager(ViewPager viewPager) {
        mTabStrip.removeAllViews();
```

```
        mViewPager = viewPager;
        if (viewPager != null) {
            viewPager.setOnPageChangeListener(new InternalViewPagerListener());
            populateTabStrip();
        }
    }

    /**
     * Create a default view to be used for tabs. This is called if a custom tab view
is not set via
     * {@link #setCustomTabView(int, int)}.
     */
    protected TextView createDefaultTabView(Context context) {
        TextView textView = new TextView(context);
        textView.setGravity(Gravity.CENTER);
        textView.setTextSize(TypedValue.COMPLEX_UNIT_SP,
TAB_VIEW_TEXT_SIZE_SP);
        textView.setTypeface(Typeface.DEFAULT_BOLD);
        textView.setLayoutParams(new LinearLayout.LayoutParams(
                ViewGroup.LayoutParams.WRAP_CONTENT,
ViewGroup.LayoutParams.WRAP_CONTENT));

        TypedValue outValue = new TypedValue();
        getContext().getTheme().resolveAttribute(android.R.attr.selectableItemBack
ground,
                outValue, true);
        textView.setBackgroundResource(outValue.resourceId);
        textView.setTextColor(Color.parseColor("#000000"));
        textView.setAllCaps(true);

        int padding = (int) (TAB_VIEW_PADDING_DIPS *
getResources().getDisplayMetrics().density);
        textView.setPadding(padding, padding, padding, padding);

        return textView;
    }

    private void populateTabStrip() {
        final PagerAdapter adapter = mViewPager.getAdapter();
        final View.OnClickListener tabClickListener = new TabClickListener();

        for (int i = 0; i < adapter.getCount(); i++) {
            View tabView = null;
            TextView tabTitleView = null;

            if (mTabViewLayoutId != 0) {
                // If there is a custom tab view layout id set, try and inflate it
                tabView = LayoutInflater.from(getContext()).inflate(mTabViewLayoutId,
mTabStrip,
```

```
                    false);
            tabTitleView = (TextView)
tabView.findViewById(mTabViewTextViewId);
        }

        if (tabView == null) {
            tabView = createDefaultTabView(getContext());
        }

        if (tabTitleView == null && TextView.class.isInstance(tabView)) {
            tabTitleView = (TextView) tabView;
        }

        if (mDistributeEvenly) {
            LinearLayout.LayoutParams lp = (LinearLayout.LayoutParams)
tabView.getLayoutParams();
            lp.width = 0;
            lp.weight = 1;
        }

        tabTitleView.setText(adapter.getPageTitle(i));
        tabView.setOnClickListener(tabClickListener);
        String desc = mContentDescriptions.get(i, null);
        if (desc != null) {
            tabView.setContentDescription(desc);
        }

        mTabStrip.addView(tabView);
        if (i == mViewPager.getCurrentItem()) {
            tabView.setSelected(true);
        }
    }
  }

  public void setContentDescription(int i, String desc) {
    mContentDescriptions.put(i, desc);
  }

  @Override
  protected void onAttachedToWindow() {
    super.onAttachedToWindow();

    if (mViewPager != null) {
      scrollToTab(mViewPager.getCurrentItem(), 0);
    }
  }

  private void scrollToTab(int tabIndex, int positionOffset) {
    final int tabStripChildCount = mTabStrip.getChildCount();
```

```
        if (tabStripChildCount == 0 || tabIndex < 0 || tabIndex >=
tabStripChildCount) {
            return;
        }

        View selectedChild = mTabStrip.getChildAt(tabIndex);
        if (selectedChild != null) {
            int targetScrollX = selectedChild.getLeft() + positionOffset;

            if (tabIndex > 0 || positionOffset > 0) {
                // If we're not at the first child and are mid-scroll, make sure we obey
the offset
                targetScrollX -= mTitleOffset;
            }

            scrollTo(targetScrollX, 0);
        }
    }

    private class InternalViewPagerListener implements
ViewPager.OnPageChangeListener {
        private int mScrollState;

        @Override
        public void onPageScrolled(int position, float positionOffset, int
positionOffsetPixels) {
            int tabStripChildCount = mTabStrip.getChildCount();
            if ((tabStripChildCount == 0) || (position < 0) || (position >=
tabStripChildCount)) {
                return;
            }

            mTabStrip.onViewPagerPageChanged(position, positionOffset);

            View selectedTitle = mTabStrip.getChildAt(position);
            int extraOffset = (selectedTitle != null)
                    ? (int) (positionOffset * selectedTitle.getWidth())
                    : 0;
            scrollToTab(position, extraOffset);

            if (mViewPagerPageChangeListener != null) {
                mViewPagerPageChangeListener.onPageScrolled(position,
positionOffset,
                        positionOffsetPixels);
            }
        }

        @Override
        public void onPageScrollStateChanged(int state) {
            mScrollState = state;
```

```
            if (mViewPagerPageChangeListener != null) {
                mViewPagerPageChangeListener.onPageScrollStateChanged(state);
            }
        }

        @Override
        public void onPageSelected(int position) {
            if (mScrollState == ViewPager.SCROLL_STATE_IDLE) {
                mTabStrip.onViewPagerPageChanged(position, 0f);
                scrollToTab(position, 0);
            }
            for (int i = 0; i < mTabStrip.getChildCount(); i++) {
                mTabStrip.getChildAt(i).setSelected(position == i);
            }
            if (mViewPagerPageChangeListener != null) {
                mViewPagerPageChangeListener.onPageSelected(position);
            }
        }

    }

    private class TabClickListener implements View.OnClickListener {
        @Override
        public void onClick(View v) {
            for (int i = 0; i < mTabStrip.getChildCount(); i++) {
                if (v == mTabStrip.getChildAt(i)) {
                    mViewPager.setCurrentItem(i);
                    return;
                }
            }
        }
    }

}


package common.googlecode;


/*
* Copyright 2013 The Android Open Source Project
*
* Licensed under the Apache License, Version 2.0 (the "License");
* you may not use this file except in compliance with the License.
* You may obtain a copy of the License at
*
*      http://www.apache.org/licenses/LICENSE-2.0
*
* Unless required by applicable law or agreed to in writing, software
```

```
* distributed under the License is distributed on an "AS IS" BASIS,
* WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or
implied.
* See the License for the specific language governing permissions and
* limitations under the License.
*/

import android.R;
import android.content.Context;
import android.graphics.Canvas;
import android.graphics.Color;
import android.graphics.Paint;
import android.util.AttributeSet;
import android.util.TypedValue;
import android.view.View;
import android.widget.LinearLayout;

class SlidingTabStrip extends LinearLayout {

  private static final int DEFAULT_BOTTOM_BORDER_THICKNESS_DIPS = 0;
  private static final byte DEFAULT_BOTTOM_BORDER_COLOR_ALPHA =
0x26;
  private static final int SELECTED_INDICATOR_THICKNESS_DIPS = 3;
  private static final int DEFAULT_SELECTED_INDICATOR_COLOR =
0xFF33B5E5;

  private static final int DEFAULT_DIVIDER_THICKNESS_DIPS = 1;
  private static final byte DEFAULT_DIVIDER_COLOR_ALPHA = 0x20;
  private static final float DEFAULT_DIVIDER_HEIGHT = 0.5f;

  private final int mBottomBorderThickness;
  private final Paint mBottomBorderPaint;

  private final int mSelectedIndicatorThickness;
  private final Paint mSelectedIndicatorPaint;

  private final int mDefaultBottomBorderColor;

  private final Paint mDividerPaint;
  private final float mDividerHeight;

  private int mSelectedPosition;
  private float mSelectionOffset;

  private SlidingTabLayout.TabColorizer mCustomTabColorizer;
  private final SimpleTabColorizer mDefaultTabColorizer;

  SlidingTabStrip(Context context) {
    this(context, null);
  }
```

```
  SlidingTabStrip(Context context, AttributeSet attrs) {
      super(context, attrs);
      setWillNotDraw(false);

      final float density = getResources().getDisplayMetrics().density;

      TypedValue outValue = new TypedValue();
      context.getTheme().resolveAttribute(R.attr.colorForeground, outValue,
true);
      final int themeForegroundColor = outValue.data;

      mDefaultBottomBorderColor = setColorAlpha(themeForegroundColor,
            DEFAULT_BOTTOM_BORDER_COLOR_ALPHA);

      mDefaultTabColorizer = new SimpleTabColorizer();
      mDefaultTabColorizer.setIndicatorColors(DEFAULT_SELECTED_INDICAT
OR_COLOR);

      mBottomBorderThickness = (int)
(DEFAULT_BOTTOM_BORDER_THICKNESS_DIPS * density);
      mBottomBorderPaint = new Paint();
      mBottomBorderPaint.setColor(mDefaultBottomBorderColor);

      mSelectedIndicatorThickness = (int)
(SELECTED_INDICATOR_THICKNESS_DIPS * density);
      mSelectedIndicatorPaint = new Paint();

      mDividerHeight = DEFAULT_DIVIDER_HEIGHT;
      mDividerPaint = new Paint();
      mDividerPaint.setStrokeWidth((int)
(DEFAULT_DIVIDER_THICKNESS_DIPS * density));
  }

  void setCustomTabColorizer(SlidingTabLayout.TabColorizer
customTabColorizer) {
      mCustomTabColorizer = customTabColorizer;
      invalidate();
  }

  void setSelectedIndicatorColors(int... colors) {
      // Make sure that the custom colorizer is removed
      mCustomTabColorizer = null;
      mDefaultTabColorizer.setIndicatorColors(colors);
      invalidate();
  }

  void onViewPagerPageChanged(int position, float positionOffset) {
      mSelectedPosition = position;
      mSelectionOffset = positionOffset;
```

```
        invalidate();
    }

    @Override
    protected void onDraw(Canvas canvas) {
        final int height = getHeight();
        final int childCount = getChildCount();
        final int dividerHeightPx = (int) (Math.min(Math.max(0f, mDividerHeight), 1f)
* height);
        final SlidingTabLayout.TabColorizer tabColorizer = mCustomTabColorizer
!= null
                ? mCustomTabColorizer
                : mDefaultTabColorizer;

        // Thick colored underline below the current selection
        if (childCount > 0) {
            View selectedTitle = getChildAt(mSelectedPosition);
            int left = selectedTitle.getLeft();
            int right = selectedTitle.getRight();
            int color = tabColorizer.getIndicatorColor(mSelectedPosition);

            if (mSelectionOffset > 0f && mSelectedPosition < (getChildCount() - 1)) {
                int nextColor = tabColorizer.getIndicatorColor(mSelectedPosition + 1);
                if (color != nextColor) {
                    color = blendColors(nextColor, color, mSelectionOffset);
                }

                // Draw the selection partway between the tabs
                View nextTitle = getChildAt(mSelectedPosition + 1);
                left = (int) (mSelectionOffset * nextTitle.getLeft() +
                        (1.0f - mSelectionOffset) * left);
                right = (int) (mSelectionOffset * nextTitle.getRight() +
                        (1.0f - mSelectionOffset) * right);
            }

            mSelectedIndicatorPaint.setColor(color);

            canvas.drawRect(left, height - mSelectedIndicatorThickness, right,
                    height, mSelectedIndicatorPaint);
        }

        // Thin underline along the entire bottom edge
        canvas.drawRect(0, height - mBottomBorderThickness, getWidth(), height,
mBottomBorderPaint);

        // Vertical separators between the titles
        int separatorTop = (height - dividerHeightPx) / 2;
        for (int i = 0; i < childCount - 1; i++) {
            View child = getChildAt(i);
            mDividerPaint.setColor(Color.parseColor("#000000"));
```

```
            canvas.drawLine(child.getRight(), separatorTop, child.getRight(),
                    separatorTop + dividerHeightPx, mDividerPaint);
        }
    }

    /**
     * Set the alpha value of the {@code color} to be the given {@code alpha}
value.
     */
    private static int setColorAlpha(int color, byte alpha) {
        return Color.argb(alpha, Color.red(color), Color.green(color),
Color.blue(color));
    }

    /**
     * Blend {@code color1} and {@code color2} using the given ratio.
     *
     * @param ratio of which to blend. 1.0 will return {@code color1}, 0.5 will give
an even blend,
     *              0.0 will return {@code color2}.
     */
    private static int blendColors(int color1, int color2, float ratio) {
        final float inverseRation = 1f - ratio;
        float r = (Color.red(color1) * ratio) + (Color.red(color2) * inverseRation);
        float g = (Color.green(color1) * ratio) + (Color.green(color2) *
inverseRation);
        float b = (Color.blue(color1) * ratio) + (Color.blue(color2) * inverseRation);
        return Color.rgb((int) r, (int) g, (int) b);
    }

    private static class SimpleTabColorizer implements
SlidingTabLayout.TabColorizer {
        private int[] mIndicatorColors;

        @Override
        public final int getIndicatorColor(int position) {
            return mIndicatorColors[position % mIndicatorColors.length];
        }

        void setIndicatorColors(int... colors) {
            mIndicatorColors = colors;
        }
    }
}
```