IMPROVED GESTURE API FOR ANDROID PLATFORM

A Project Report

submitted by

YENNETI GOPI VENKATA DINESH

in partial fulfilment of the requirements
for the award of the degree of

BACHELOR OF TECHNOLOGY

DEPARTMENT OF ELECTRICAL ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY MADRAS.

MAY 2013

THESIS CERTIFICATE

This is to certify that the report titled **IMPROVED GESTURE API FOR ANDROID PLATFORM**, submitted by **YENNETI GOPI VENKATA DINESH**, to the Indian Institute of Technology, Madras, for the award of the degree of **Bachelor of Technology**, is a bona fide record of the project work done by him under our supervision. The contents of this report, in full or in parts, have not been submitted to any other Institute or University for the award of any degree or diploma.

Dr.T.G.Venkatesh
Project Guide
Assistant Professor
Dept. of Electrical Engineering
IIT-Madras, 600 036

Place: Chennai

Date: 23th May 2013

ACKNOWLEDGEMENTS

I wish to express my deep sense of gratitude to my project guide Dr.T.G.Vekatesh, Assistant Professor in Department of Electrical Engineering, for his most valuable guidance, discussions, suggestions and encouragement, from the conception to the completion of this project.

I would like to thank MS and PhD students under the guidance of Dr.T.G.Vekatesh for their help and support throughout the project.

I would also like to thank my professors and friends from my undergraduate studies while at Indian Institute of Technology, Madras. The preparation and experience I gained were invaluable.

Finally, I would to like to thank my family for their unconditional love and moral support, which will allow me to achieve what I have and will.

ABSTRACT

KEYWORDS: Android, Gestures, Character Recognition

Interacting with devices using multi-touch gestures is gaining popularity after the era of smart phones has started. Among various smart phone Operating systems, Google owned Android has the major market share and is known for its open source policy. This project aims at improving the Gesture API for Android to include features that allows user to define their own gestures.

In this project, an API that uses touch input to detect gestures has been developed. Later, the line gesture detection API provided by Android has been improved to allow users to define their own gestures. This API has been extended to detect characters drawn on the screen which is configurable according to the user's handwriting. Finally, this API has been used to develop an application for blind users which can be used to store contacts, search for contacts, make calls and send messages using Gesture Input on Screen.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABBREVIATIONS

TTS	Text To Speech
IR	Infra Red
SMS	Short Message Service
SD Card	Secure Digital Card
SDK	Software Development Kit

CHAPTER 1

Introduction

1.1 Smartphones

The term ’Smartphone’ is defined as a mobile phone built on a mobile operating system with advanced computing system. In the year 1974, the first basic smart phone was patented by Greek scientist Theodore Paraskevakos. In the year 2000, the first smartphone operating system, Symbian was released. The first smartphones combined the functions of a personal digital assistant. In the year 2008, the Android OS released by and quickly became the most used smart phone Operating system in the world, with the market share of 70.1%. The second most used OS is the iOS by Apple Inc. which has a market share of 21.0%. Both of these combined covers 91.1% of market share.

1.2 Programming in Android Platform

The Android Platform is built on a Linux Operating System with Android applications running on it. Android applications are written in the Java programming language. The Android SDK tools compile the code along with any data and resource files into an Android package, an archive file with an .apk suffix. All the code in a single .apk file is considered to be one application and is the file that Android powered devices use to install the application.

1.2.1 Activities in Android

Each android application consists of various Activities. An activity represents a single screen with a user interface (1). For example, an email application might have one activity that shows a list of new emails, another activity to compose an email, and another activity for reading emails.

1.2.2 Life cycle of an Activity

Each activity has standard methods that the system can call to change the state of the activity. Some of these methods are:

onCreate()

Called by the system to begin an activity. This is similar to a constructor on a normal java application. This is used to initialize all the variables and set the layout of the application.

onPause()

The system calls this method as the first indication that the user is leaving the activity. This doesn't mean that the activity is getting destroyed. This is usually where any changes should be committed that should be persisted beyond the current user session.

OnDestroy()

This method is called when the android system is destroying the current application. This is the final call that the activity will receive. An activity can be destroyed either by the system or by the application by calling the 'finish()' method. This is where all the temporary data should be stored.

The life-cycle of an activity is called the time-span between onCreate and onDestroy are called.

1.2.3 Manifest file

Before the Android system can start an application component, the system must know that the component exists by reading the application's AndroidManifest.xml file (the "manifest" file). An application must declare all its components in this file, which must be at the root of the application project directory.

1.2.4 User-interface of an Android application

All user interface elements in an Android app are built using View and ViewGroup objects. A View is an object that draws something on the screen that the user can interact with. A ViewGroup is an object that holds other View (and ViewGroup) objects in order to define the layout of the interface.

User interface in Android can be programmed in two ways:

- Declare UI elements in XML. Android provides a straightforward XML vocabulary that corresponds to the View classes and subclasses, such as those for widgets and layouts.
- Instantiate layout elements at runtime. The application can create View and ViewGroup objects (and manipulate their properties) programmatically.

1.3 Gesture API on Android

Android has defined API for gestures which are commonly used on smart phones such as double tap, single tap, fling, long press and scroll in a class called GestureDetector(android.view.GestureDetector) which is discussed in (2). This API provides the following methods:

onDoubleTap(MotionEvent e)

Notified when a double-tap occurs.

onFling(MotionEvent e1, MotionEvent e2, float velocityX, float velocityY)

Notified of a fling event when it occurs with the initial on down MotionEvent and the matching up MotionEvent. A fling event is said to be occurred when the user moves pointer from one point to another point on screen with velocity greater than a limit.

onLongPress(MotionEvent e)

Notified when a long press occurs with the initial on down MotionEvent that trigged it.

onScroll(MotionEvent e1, MotionEvent e2, float distanceX, float distanceY)

Notified when a scroll occurs with the initial on down MotionEvent and the current move MotionEvent.

1.4 Line gesture API on Android

Android provides API to detect line gestures using android.gesture package as shown in (3). This API has the following components:

Gesture

A gesture is a hand-drawn shape on a touch screen. It can have one or multiple strokes. Each stroke is a sequence of timed points. A user-defined gesture can be recognized by a GestureLibrary.

GestueOverlayView

A transparent overlay for gesture input that can be placed on top of other widgets or contain other widgets.

GestureStore

This class maintains gesture data and makes predictions on a new gesture.

GestureUtils

This class contains static methods which are utilities for sampling a given gesture.

OnGesturePerformed

This is an interface whose method 'onGesturePerformed' is called when the gesture is performed by the user.

1.5 Limitations of Google API for Gestures

Although the API provided by Google contains methods for detecting Gestures, all these methods depend on hard-coded Gesture data that is stored in the resource file in the application. There is no API for allowing the users to add their own gestures while using the application. Such an API is required because the gestures added by the users will offer better detection rate and better convenience than those gestures already defined and hard-coded by the developer in the application.

1.6 The solution provided by this project

The above problem in the Google API is solved in this project by extending the API to allow users to define their own gestures. Initially in this project, detecting basic finger made gestures is discussed. Then, an API is developed that can detect line gestures as well as allow user to define their own gestures. Later, this was extended to detect hand written characters, where users can tune detection data according to their handwriting. These APIs are finally used to develop a gesture based contacts application for blind users.

1.7 Platform and Version

The device used to test this project is Spice MI 350 Android smart phone. The Android Platform version is Gingerbread Android 2.3. Any program working for this version will work on 90.3 % of Android devices that are currently in use (4).

Table 1.1: Table showing the distribution of Android versions

Version	Percentage
Donut	0.001
Eclair	0.017
Froyo	0.037
GingerBread	38.5
HoneyComb	0.1
IceCreamSandwich	27.5
JellyBean	28.4

1.8 Organization of this project

This project is organized into four chapters.

The first chapter discusses about the API that detects the finger gestures like fling, scroll, double tap, etc. and how new such gestures can be added to the developed API in the future.

The second chapter discusses about the improved line gesture detection API that allows user to define own gestures.

The third chapter discusses about the character recognition and user training API that detects characters and lets users define gestures for characters.

In the fourth chapter, the contacts application designed for blind users is described. The program is explained and details about the usage are given.

In all the above chapters, each chapter begins with an introduction. Then, it explains the classes present in the API and their functions. Later, the usage of the API is explained through various use-cases. Finally, the results are shown and a conclusion is given.

In the final chapter, the conclusion and the scope for possible future work is discussed.

CHAPTER 2

Simple gesture detector

2.1 Introduction

This chapter discusses about the API that allows the developer to define his own gestures and add them to their applications. This API takes the touch input from the user and delivers it to the respective gesture detection methods that the developer defines. Later in this chapter, a case study of demonstration of usage of this API is shown in this chapter.

2.2 Functioning of the API

The API has a main class and an interface.The gestures are detected from the touch input given by the touch library of Android. Any class implementing the `OnTouchListener` is delivered touch events continuously by the android API. These events are analyzed to detect whether a particular gesture has been given as input by user.

For example, the angle of rotation is detected from the `MotionEvent` as follows:

```
double delta_x = (event.getX(0) - event.getX(1));
double delta_y = (event.getY(0) - event.getY(1));
double radians = Math.atan2(delta_y, delta_x);
double degrees = Math.toDegrees(radians);
```

The above code is used to deliver events to the main activity, whenever the angle of the two fingers placed on the screen changes.

Whenever touch events occur, the android API notifies the `GestureMAnager`. These events are delivered to the detectors which are specified in the main class. These detectors perform calculations to detect if that event has occurred. If the event has occurred, they compute the parameters related to the Gesture performed. The computed parameters are stored in ’GestureManagerEvent’ object. This object is used to notify the registered `OnGestureManagerListener`. This flow of events is as shown in the figure below:

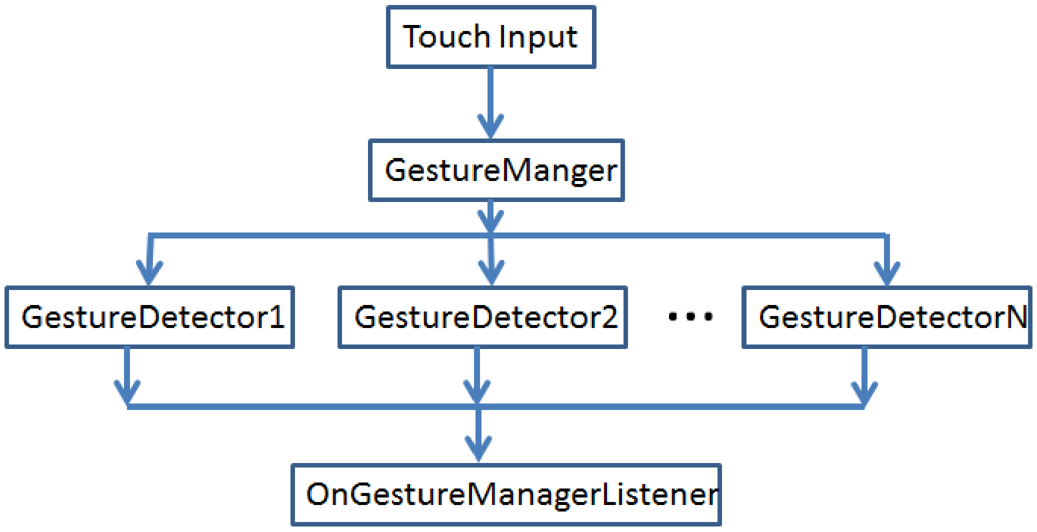
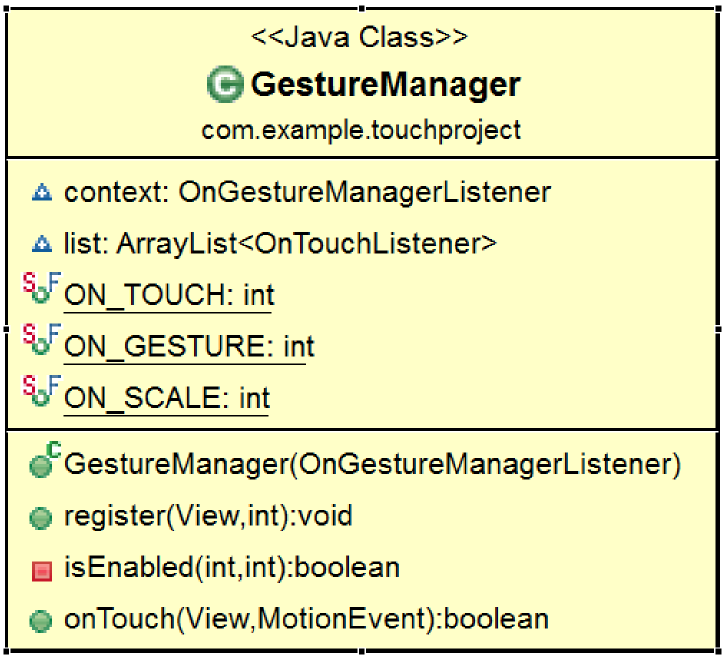


Figure 2.1: A flow chart showing the structure of Gesture API

2.3 Architecture

2.3.1 GestureManager



<<Java Class>>
GestureManager
com.example.touchproject
context: OnGestureManagerListener
list: ArrayList<OnTouchListener>
ON_TOUCH: int
ON_GESTURE: int
ON_SCALE: int
GestureManager(OnGestureManagerListener)
register(View,int):void
isEnabled(int,int):boolean
onTouch(View,MotionEvent):boolean

This class is the main class which takes the touch inputs from the user. This class implements 'OnTouchListener' interface, which allows it to take the touch inputs. This class also manages the list of gesture detectors that are registered by the developer. Each of the detector should implement 'OnTouchListener' interface so that whenever an event occurs, all the detector objects present in the list will be notified by passing the 'android.view.MotionEvent' to the 'OnTouch' of the detector.

GestureManager(OnGestureManagerListener mContext)

This is the constructor which takes the object which implements 'OnGestureManagerListener' interface.

register(View mView,int options)

This method is used by the developer to describe on which view the gesture needs to be detected. By using the 'options' parameter, the user selects which detectors are used. This parameter 'options' is an integer in which each bit corresponds to each gesture detector.

For example: Scale gesture is defined as

```
public static final int ON_SCALE = 1<<2.
```

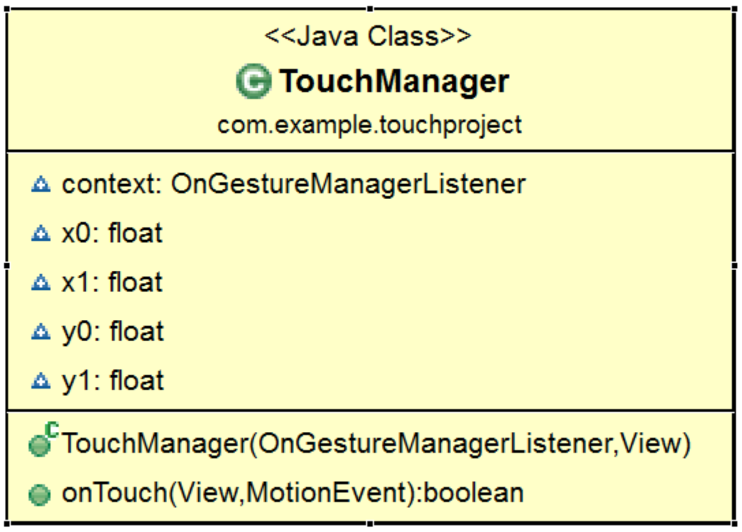
Rotate gesture is defined as

```
public static final int ON_ROTATE = 1<<3.
```

Hence, to select both scale and rotate gesture detectors, options parameter should be set as `ON_SCALE` + `ON_ROTATE`.

The register method checks each bit of the 'options' parameter and creates the objects of the detectors for each bit set in it.

2.3.2 TouchManager



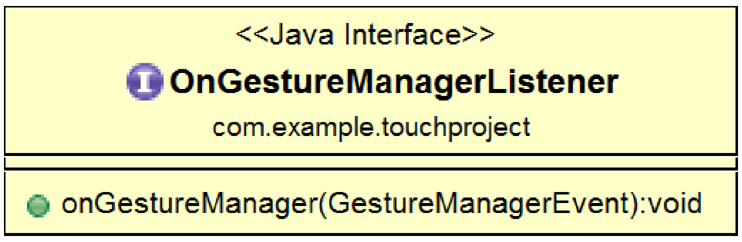
<<Java Class>>
TouchManager
com.example.touchproject
context: OnGestureManagerListener
x0: float
x1: float
y0: float
y1: float
TouchManager(OnGestureManagerListener,View)
onTouch(View,MotionEvent):boolean

This class is instantiated by the GestureManager. This class takes the MotionEvents from the GestureManager and interprets the events. If it finds that the locations of pointers has changed, it notifies the corresponding detector events. This class implements 'OnTouchListener' interface. Whenever a touch event is delivered to it, it compares this event with the previous touch event that has occurred. If there is any change in the event, it notifies the registered gesture detectors.

TouchManager(OnGestureManagerListener mContext,View mView)

The constructor for this class. This class takes an 'OnGestureManagerListener' as a parameter so that whenever an event has occurred, it will notify it. The 'mView' parameter tells on which 'View'(android.view.View) object the event has occurred.

2.3.3 OnGestureManagerListener

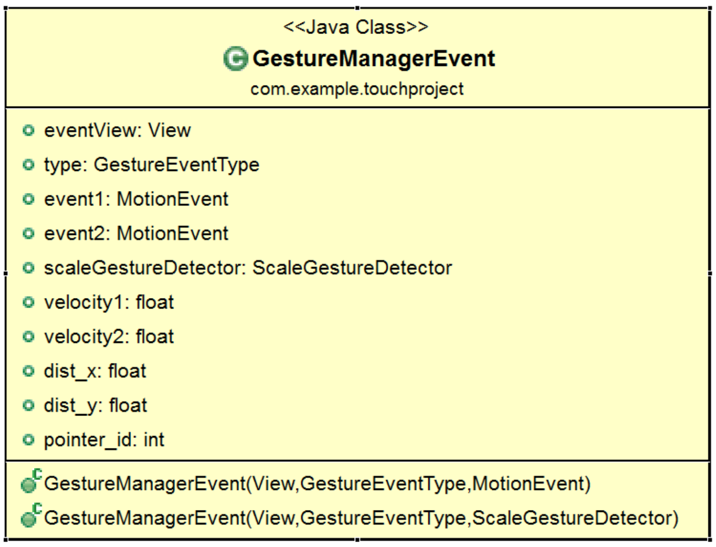


This is an interface which needs to be implemented by the class which wants to be notified when a gesture is detected.

onGestureManager(GestureManagerEvent event)

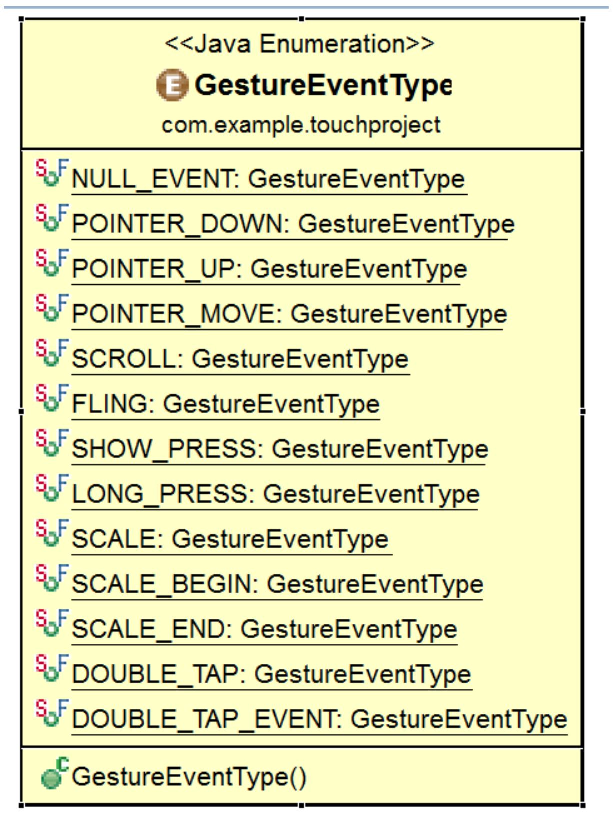
This method is called by the gesture detector when a gesture has been detected. An object of 'GestureManagerEvent' will be passed to it, which contains the data of the gesture that has been detected.

2.3.4 GestureManagerEvent



An object of this class holds the data of the gesture that has been detected by a gesture detector. When a detector detects a gesture, it creates an instance of this class and stores the gesture information in it. This object is passed as an argument to the 'onGestureManager' method of a class that implements 'OnGestureManager' interface.

2.3.5 GestureEventType



<<Java Enumeration>>
GestureEventType
com.example.touchproject
NULL_EVENT: GestureEventType
POINTER_DOWN: GestureEventType
POINTER_UP: GestureEventType
POINTER_MOVE: GestureEventType
SCROLL: GestureEventType
FLING: GestureEventType
SHOW_PRESS: GestureEventType
LONG_PRESS: GestureEventType
SCALE: GestureEventType
SCALE_BEGIN: GestureEventType
SCALE_END: GestureEventType
DOUBLE_TAP: GestureEventType
DOUBLE_TAP_EVENT: GestureEventType
GestureEventType()

This is an enumeration which contains the type of the gestur that has been detected. This is a variable present in the 'GestureManagerEvent' class.

Types of events

- NULL_EVENT: Default value if no event is set
- POINTER_DOWN: Pointer is just down on screen
- POINTER_UP: Pointer is lifted up from screen
- POINTER_MOVE: Pointer moved on screen
- FLING: Pointer moved at high speed on the screen
- ROTATE: Rotate gesture event
- SHOW_PRESS: Pointer is down at single point for more than 0.5 seconds
- LONG_PRESS: Pointer is down at single point for more than 0.5 seconds
- SCALE: Scale gesture event
- DOUBLE_TAP: Pointer is down twice with short gap in between

2.4 Use cases

2.4.1 Case 1: Using existing Gestures

- Implement an `OnGestureManagerListener` class and create a method called `onGestureManagerListener` in it.
- Make an object of the `GestureManager` class by passing the `OnGesture ManagerListener` created above as its argument.

  ```
  GestureManager gManager = new GestureManager(listener);
  ```

- Call the `register` method on the `GestureManager` by passing 2 arguments. The first argument is the View on which the user inputs gesture which needs to be detected. The second is the set of gestures for which the manager notifies the listener.

  ```
  gManager.register(image,GestureManager.ON_TOUCH +
   GestureManager.ON_GESTURE);
  ```

- The `onGestureManagerListener` method is called every time the user inputs a gesture and is detected by the detector.

2.4.2 Case 2: Adding New Gestures

In this case, a new class has to be created, which implements ’OnTouchListener’ interface. The constructor of this class should follow the following structure:

```
ClassName(OnGestureManagerListener, View)
```

In the ’OnTouch’ method of the class, the routine that computes the parameters of the detected gesture should be calculated. Then, a new ’GestureManagerEvent’ object should be created and the data related to this event should be stored in it. This object should be used as a parameter to call the ’onGestureManagerListener’ method on the ’ OnGestureManagerListener’ class. This completes the cylce of gesture detection.

This gesture detector should be added to the ’GestureManager’ class by creating a new static variable for it and adding it to the ’register’ method.

Thus, whenever an event occurs, the GestureManager passes the event to the detector. The detector computes the parameters, creates an ’GestureManagerEvent’ and passes it to the ’OnGestureManagerListener’.

2.4.3 Example methods for detecting gestures

As shown in (8), various techniques have evolved to detect gestures given touch input. Two of them are described below.

Scale

When the user places two fingers on the screen and pinches or spreads the fingers, this event is said to be occurred. To detect this gesture, the following conditions needs to be met: a) There should be only two pointers on the screen. b) One or both the pointers should change positions.

When these conditions are met, the scale factor is calculated as the ratio of new distance between the pointers to the previous distance between the pointers.

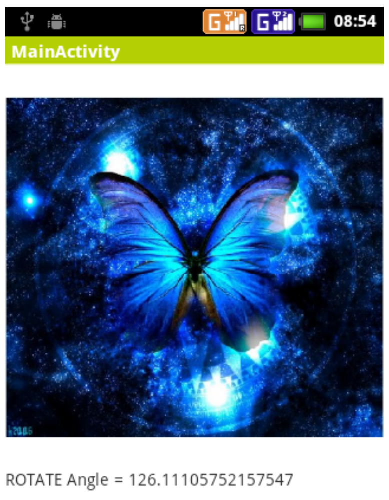
For example: previous positions are: (100, 50) and (130, 90) new positions are: (101,50) and (130,88), the scale factor is 0.956, which indicates that the pointers moved closer and the pinching gesture has been performed.

Rotate

When one of the pointers remained static and the other pointer moves, this gesture is said to be performed. This can be simplified as when the angle that the line joining the two pointer makes with the horizantal changes, this gesture is said to be performed. To compute this, the angle made by the two pointers with the horizantal is computed and when a change is detected in this angle, the gesture is said to be performed.

For example, If the pointer positions are (x1,y1) and (x2,y2), the angle made by them is arcTan((y2-y1)/(x2-x1)).

2.5 Results



ROTATE Angle = 126.11105752157547

Figure 2.2: A screenshot of an application that uses this API

The figure 2.2 shows the screenshot of an application that uses this API. The application has an image on which the gestures are performed. These feedback for these gestures are displayed on screen and are added to the log file.

The following text is the sample log file that has been recorded by a class that implements this API.

```
05-20 09:29:12.570: Registered for ON_GESTURE
05-20 09:29:12.571: Registered for ON_SCALE
05-20 09:29:58.074: POINTER_DOWN at 110.0, 169.0
05-20 09:29:58.140: SHOW_PRESS at 110.0, 169.0
05-20 09:29:58.641: LONG_PRESS at 110.0, 169.0
05-20 09:30:01.049: POINTER_DOWN at 90.0, 216.0
05-20 09:30:01.117: SCROLL from 90.0, 216.0 to 111.0, 199.0
05-20 09:30:01.134: SCROLL from 90.0, 216.0 to 126.0, 188.0
05-20 09:30:01.186: SCROLL from 90.0, 216.0 to 168.0, 159.0
05-20 09:30:01.204: SCROLL from 90.0, 216.0 to 174.0, 156.0
05-20 09:30:01.240: FLING from 90.0, 216.0 to 174.0, 156.0
```

```
05-20 09:32:50.196: SCALE Scale Factor = 0.9693622
05-20 09:32:50.213: SCALE Scale Factor = 0.97668684
05-20 09:32:50.231: SCALE Scale Factor = 0.9708189
05-20 09:32:50.266: SCALE Scale Factor = 0.9765371
05-20 09:32:50.284: SCALE Scale Factor = 0.97422564
```

2.6 Conclusion

This chapter explains how using the API structure that has been developed, new gestures can be defined and added to it. Altough this API can be used to define basic finger made gestures, an advanced API is required to detect the line gestures that are drawn on screen, since they need to be compated to the set of line drawings that are defined. This is discussed in the next chapter.

CHAPTER 3

Line gesture Detector

3.1 Introduction

This chapter talks about the detection of user defined line gestures. The google API provides methods for recognizing a given gesture from a set of previously defined gesture data. But, this API doesn't provide methods that developer can use to let user define his own gestures. Instead, the gestures are hard-coded into the application and are stored as an application resource. To overcome this, an API is developed that developer can use to allow user to define his own line gestures in the application. In this chapter, the description of API is discussed. Later in this chapter, the usage of this API is also described with an example use-case.

3.2 A brief history of Gestures on Touch Screen

The first pen-based hypertext browser for input device, the Rand tablet, was funded by ARPA Sketchpad used light-pen gestures(1963)(5). Ever since then, both academia and corporate researchers did research on improving the human-computer interaction experience through gestures on touch screen devices. Teitelman developed the first trainable gesture detector in the year 1964. Since 1970s, the gestures are used in CAD systems, but these came to public notice only through Apple Newton in the year 1992. Although various other methods of interaction have been developed, the touch screen gestures again became an important area of research in the 2000s with the advent of smart phone era.

3.3 Functioning of the API

The recorder activity displays a `GestureOverlayView` on the screen on which gestures are drawn. When the user finishes drawing the gesture, `onGesturePerformed` method is called, with the recoded gesture passed as an argument to this gesture. This gesture is stored in gesture array. At the same time, the string name entered in the text-box by the user is stored in the string array. When this activity exits, these arrays are parceled in a `Bundle` and are passed to the main activity.

When the detector activity is called, this bundle is passed to the detector activity. This the data from this bundle is extracted by the detector activity and is stored in a `GestureLibrary`. When the user enters a gesture, the `GestureLibrary` has a recognizer which compares the user input gesture with the gesture data and returs the name of the gesture with the best possible match.

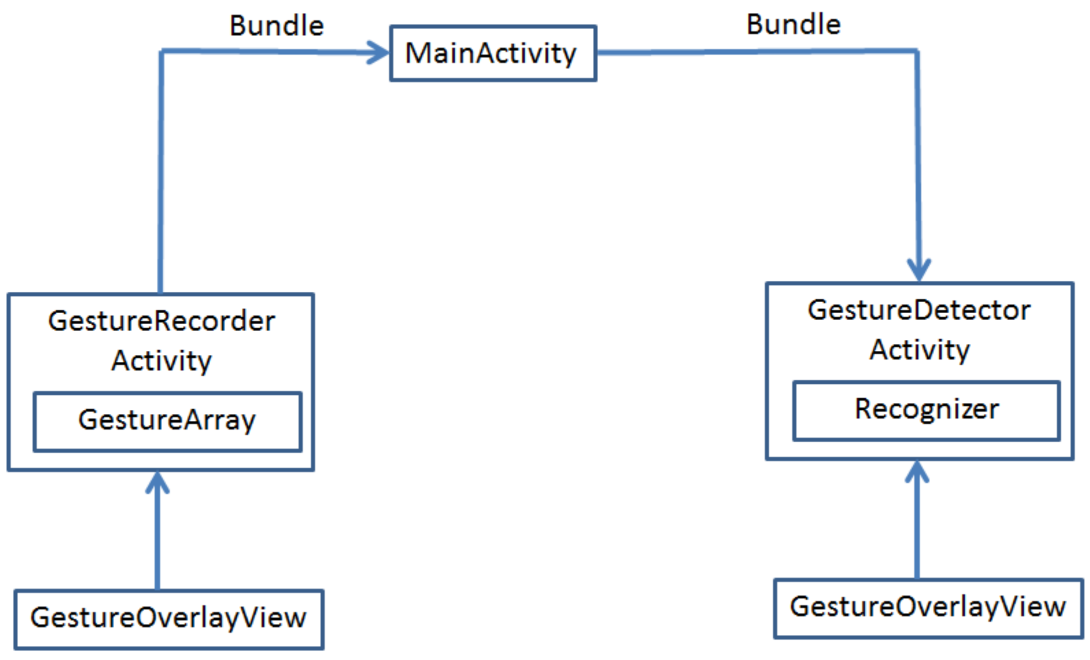
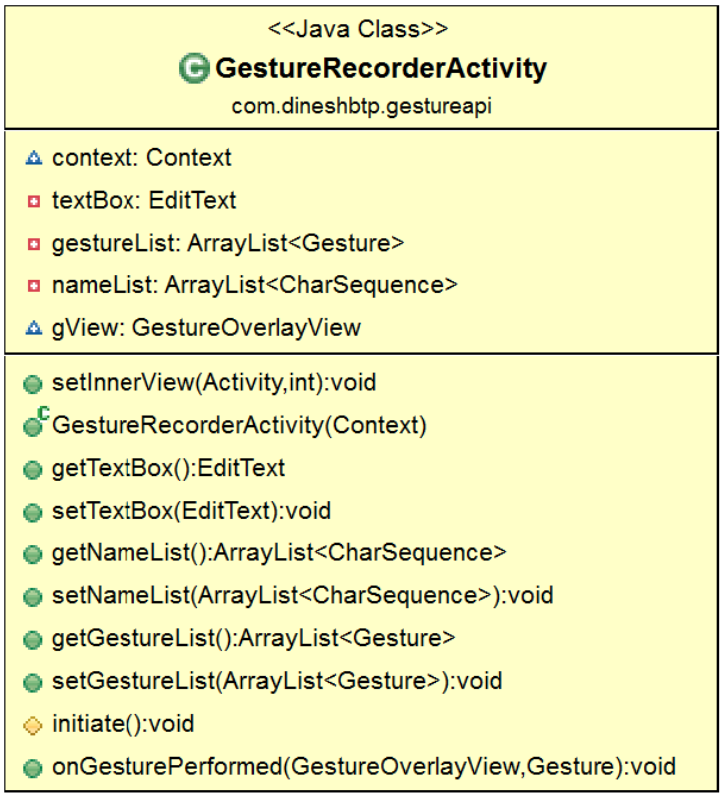


Figure 3.1: A flow chart showing the functioning of GestureRecorderActivity and GestureDetectorActivity

3.4 Architecture

3.4.1 GestureRecorderActivity


<<Java Class>>
GestureRecorderActivity
com.dineshbtp.gestureapi
context: Context
textBox: EditText
gestureList: ArrayList<Gesture>
nameList: ArrayList<CharSequence>
gView: GestureOverlayView
setInnerView(Activity,int):void
GestureRecorderActivity(Context)
getTextBox():EditText
setTextBox(EditText):void
getNameList():ArrayList<CharSequence>
setNameList(ArrayList<CharSequence>):void
getGestureList():ArrayList<Gesture>
setGestureList(ArrayList<Gesture>):void
initiate():void
onGesturePerformed(GestureOverlayView,Gesture):void

This class extends the class 'android.gesture.GestureOverlayView'. This class lets the user define his own gestures. The 'GestureOverlayView' takes the gesture data and handles it to 'OnGesturePerformed' method in this class. This data is stored in an Arraylist(java.util.ArrayList) of gestures. This class needs an EditText(android.widget.EditText) widget into which the user has to type the name of the gesture that he wants to define. This can be set using 'setTextBox' method.

When the name is entered and a gesture is drawn, the name is added to the an ArrayList of Strings and the Gesture is added to an ArrayList of Gestures, both with the same index.

When the Activity exits, these two ArrayList objects are stored in Bundle(android.os.Bundle). This Bundle is added as an extra to an Intent(android.content.intent). This intent calls back the Activity from which this activity has been called. Thus, it passes the Gesture data it has recorded from the user back to the main activity which needs to data to recognize new gestures.

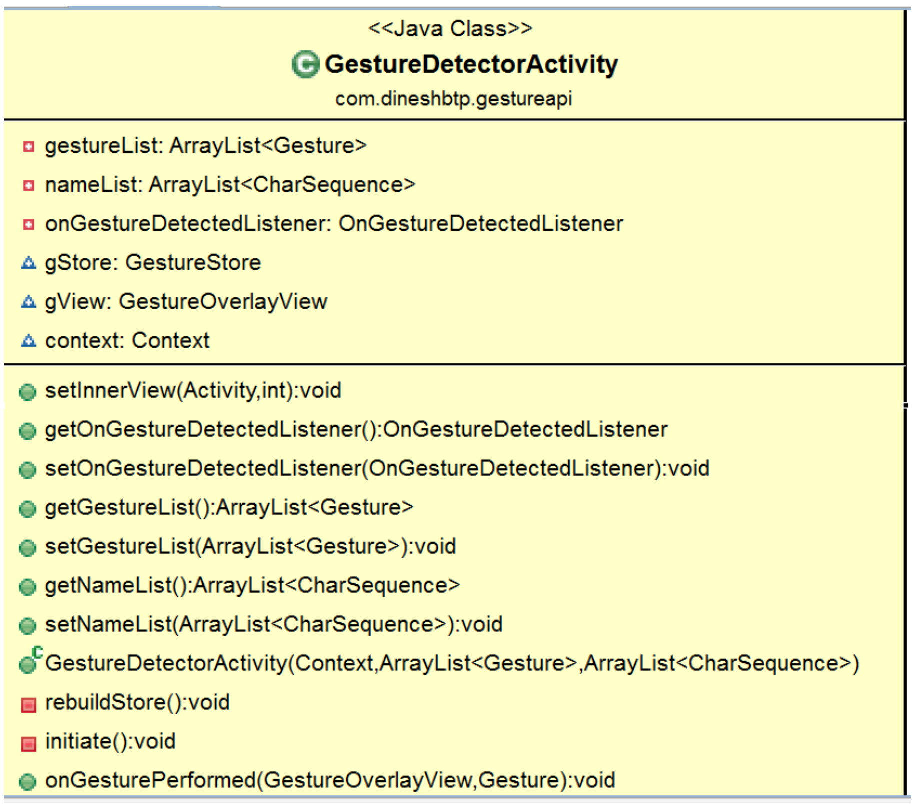
initiate()

This method is called inside the constructor. This method builds the layout and initializes the variables. This is analogous to the constructor on normal java applications.

onGesturePerformed(GestureOverlayView overlay, Gesture gesture)

This method is called by the GestureOverlayView when the user completes drawing a gesture. This method stores the gesture data passed to it as an argument in Arraylists. When the user didn't enter any name for the gesture, this method displays a toast saying that the name field is empty.

3.4.2 GestureDetectorActivity


<<Java Class>>
GestureDetectorActivity
com.dineshbtp.gestureapi
gestureList: ArrayList<Gesture>
nameList: ArrayList<CharSequence>
onGestureDetectedListener: OnGestureDetectedListener
gStore: GestureStore
gView: GestureOverlayView
context: Context
setInnerView(Activity,int):void
getOnGestureDetectedListener():OnGestureDetectedListener
setOnGestureDetectedListener(OnGestureDetectedListener):void
getGestureList():ArrayList<Gesture>
setGestureList(ArrayList<Gesture>):void
getNameList():ArrayList<CharSequence>
setNameList(ArrayList<CharSequence>):void
GestureDetectorActivity(Context,ArrayList<Gesture>,ArrayList<CharSequence>)
rebuildStore():void
initiate():void
onGesturePerformed(GestureOverlayView,Gesture):void

This class extends 'GestureOverlayView' class. This class detects the gestures drawn by user on the screen. For this purpose, it uses the gesture data given by the GestureRecorderActivity. This data is set using the arguments in the constructor. This data is stored in a GestureStore(android.gesture.GestureStore) object.

This GestureOverlayView takes the gesture drawn by the user and delivers it to this activity. This activity uses the 'recognize' method present in 'GestureStore' class to recognize the gesture. The recognize method gives an ArrayList of names of the gestures that match with this gesture. Lower the index of the gesture in the list, better the match. The activity displays the name of the first name present in the list using a Toast(android.widget.Toast).

setInnerView(Context, int)

This method can be used to add inner views to the GestureOverlayView. This method also sets the layout of the current activity as the current GestureOverlayView.

onGesturePerformed(GestureOverlayView overlay, Gesture gesture)

This method is called by the GestureOverlayView when the user completes drawing a gesture. This method uses the ’gesture’ argument in the ’recognize’ method of ’GestureStore’ to get the list of predicted names and displays the name of the first gesture using Toast.

3.5 Use Cases

For recording gestures, the recorder activity needs to be invoked first. This is done by calling the `GestureRecorderActivity` from the main activity.

```
Intent intent = new Intent(thisContext,
    GestureRecorderActivity.class);
startActivity(intent);
```

When the recorder activity finishes recording gestures, it returns to the main activity. While returning, it transfers a `extras` with the name two items. An arraylist of gestures with the name `glist` and an arraylist of strings which contain the names for these gestures in the same order with the name `namelist`. These can be extracted from the bundle as follows:

```
gList = extras.getParcelableArrayList("gList");
nameList = extras.getCharSequenceArrayList("nameList");
```

For using the detector, the `GestureDetectorActivity` should be invoked with gesture array included in the extras in the same format as returned by the recorder.

```
Intent intent = new Intent(thisContext,
    GestureDetectorActivity.class);
bundle.putParcelableArrayList("gList", gList);
bundle.putCharSequenceArrayList("nameList",nameList);
intent.putExtra("bundle", bundle);
startActivity(intent);
```

This calls the `GestureDetectorActivity` which uses the gesture list provided to it to detect the names of the gestures input by the user on the screen.

3.6 Results

The follwing screenshots shows the application which uses this API:

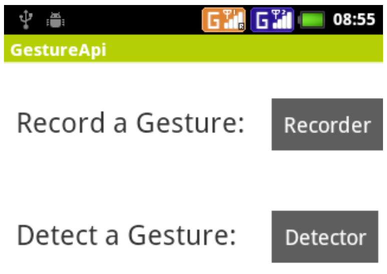


Figure 3.2: A screenshot the initial screen of the Application

In figure 3.2, the initial screen of the application is shown. It has two buttons. Selecting the 'Recorder' opens 'GestureRecorderActivity' while selecting the 'Detector' opens 'GestureDetectorActivity'.

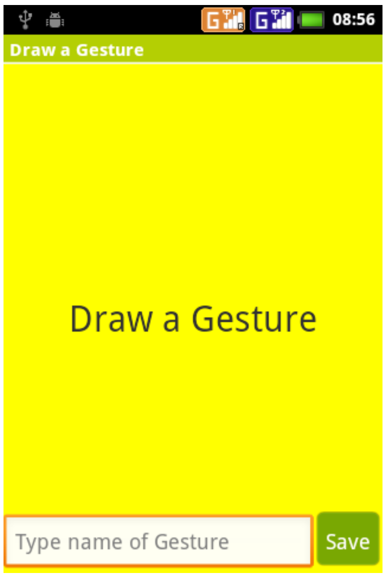


Figure 3.3: A screenshot of the GestureRecorderActivity

In the figure 3.3, the GestureRecorderActivity is shown.

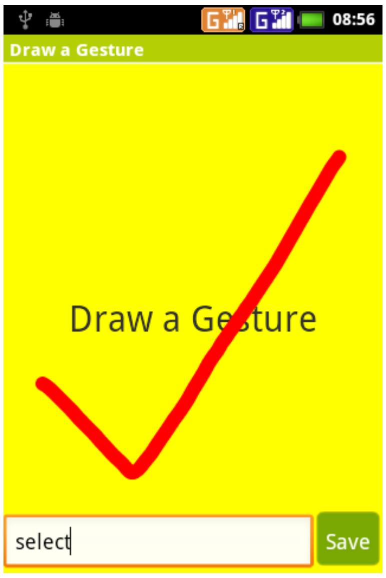


Figure 3.4: A screenshot of GestureRecorderActivity after entering the name and the gesture

In the figure 3.4, the name of the gesture is entered and the gesture is drawn on screen.

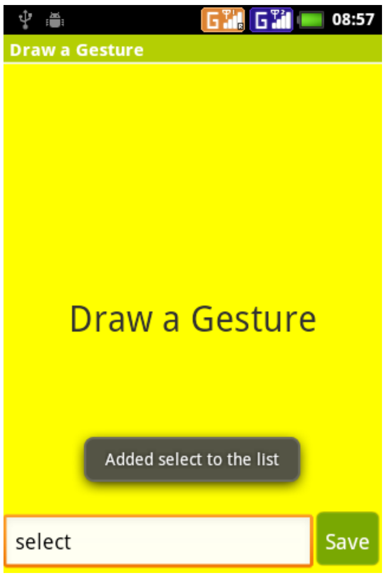


Figure 3.5: A screenshot of GestureRecorderActivity with a toast message showing that the gesture has been saved

In the figure 3.5, a toast message is shown saying that the gesture is saved.

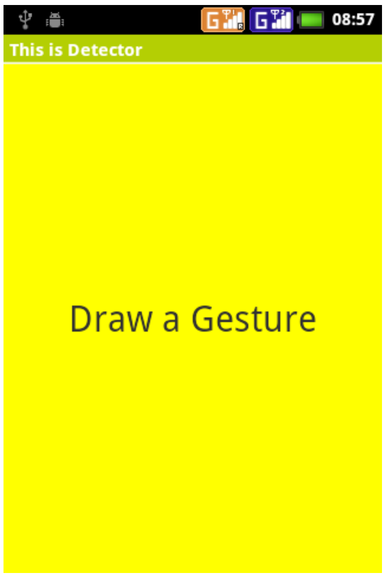


Figure 3.6: A screenshot of the GestureDetectorActivity

In the figure 3.6, the GestureDetectorActivity is shown after its initialization.

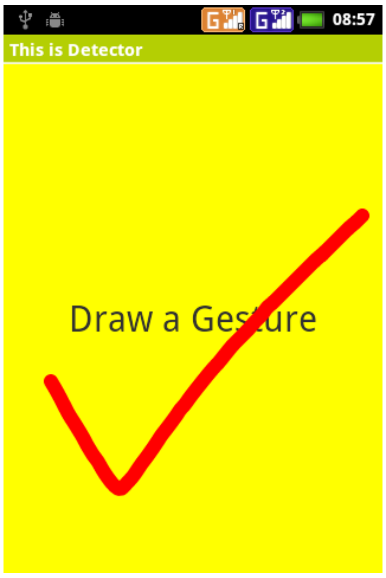


Figure 3.7: A screenshot of GestureDetectorActivity showing a gesture drawn on screen

In the figure 3.7, a gesture is drawn on screen.

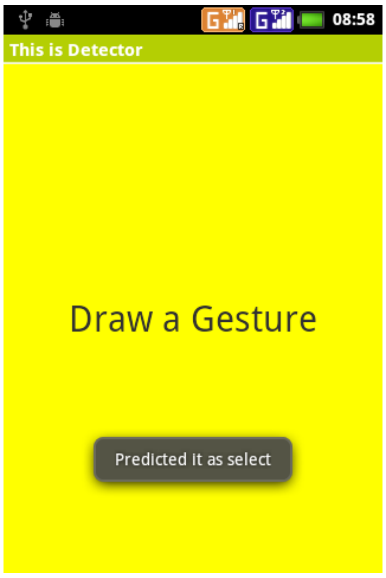


Figure 3.8: A screenshot of GestureDetectorActivity showing the prediction it made in the form of a toast message

In the figure 3.8, the system predicts the name of the gesture drawn on screen and displays it as the toast message.

3.7 Conclusion

Thus the problem of letting the user define his own line gestures is solved by using this API. This concept in used in the next chapter to build an API that lets the user type characters into a ’TextBox’ using line gestures and lets the user define his own gestures to represent each character, so that the accuracy of character detection will be improved.

CHAPTER 4

Character Recognizer and Trainer

4.1 Introduction

In the previous chapter, an API has been developed that the developer can use to let the user define his own gestures. This concept of detecting line gestures can be extended to detecting characters drawn on screen. This is undertaken in this chapter. This chapter describes an API which uses the line gesture detection method to detect characters entered on screen. This idea is inspired from (6).

The API that can be used to let users define their own gestures is extended in this chapter to let users define their own gestures for each character. Thus, the gesture data used to detect characters is trained according to the user's hand-writing, which improves the character detection efficiency.

4.2 Functioning of the API

4.2.1 Character Recognizer

The character recognizer takes in, the data from the file or raw source and stores the data in `GestureLibrary`. Whenever the user inputs a character, the data is taken by the `GestureOerlayView` and is passed on the the recognizer. This recognizer uses this data and compares it with the data present int the `GestureLibrary`. The character with the best match is appended to the `TextView` or is passed as an argument to the `OnCharacterRecognizedListener`.

4.2.2 Character Recognizer Trainer

The trainer promtps the user too enter a particular character. When the user enters a character, the `GestureOerlayView` takes the data and passes it to the trainer. This data is stored in a `GestureLibrary`. When all the characters are entered, this data is stored into a file on the SD card.

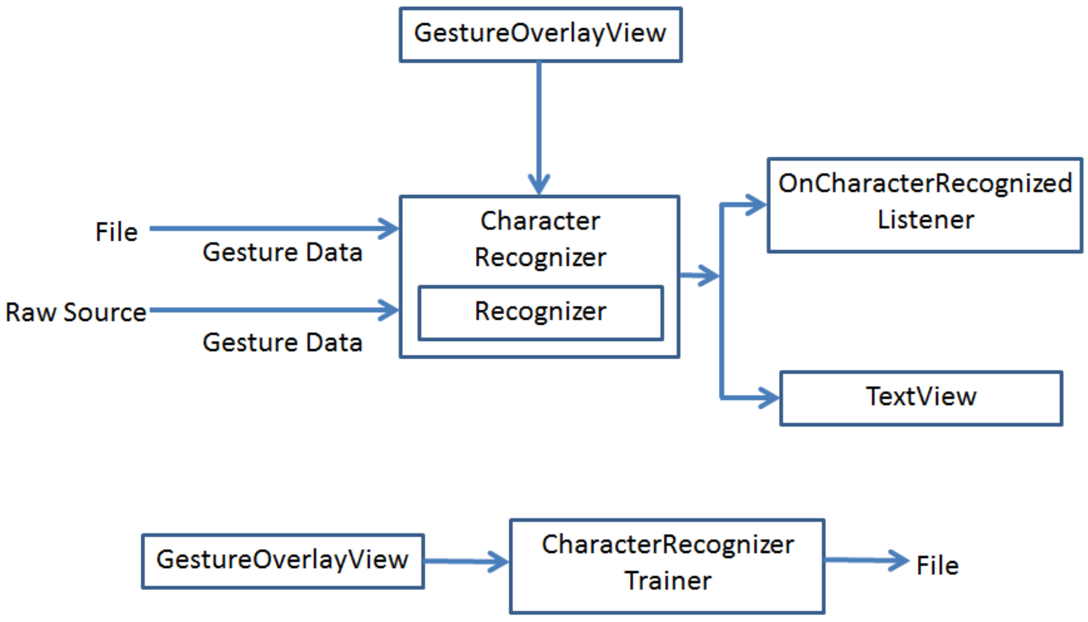


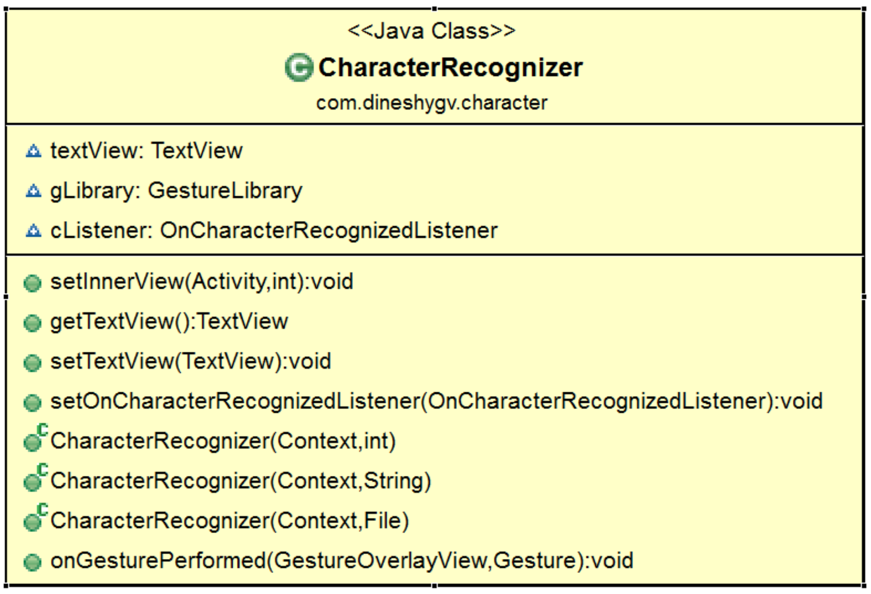
Figure 4.1: A flow chart showing the functioning of the Character Recognizer and CharacterRecognizerTester

4.3 Architecture

4.3.1 CharacterRecognizer

This class extends 'GestureOverlayView'(android.view.GestureOverlayView) class. Hence, this class can be directly used as a View in the layout. This class contains methods to load gesture data present in a file on SD-Card or from a resource file present in the application. This class can be used in two ways.

The first way is to let the class append the detected character to the string present



```
textView: TextView
gLibrary: GestureLibrary
cListener: OnCharacterRecognizedListener
setInnerView(Activity,int):void
getTextView():TextView
setTextView(TextView):void
setOnCharacterRecognizedListener(OnCharacterRecognizedListener):void
CharacterRecognizer(Context,int)
CharacterRecognizer(Context,String)
CharacterRecognizer(Context,File)
onGesturePerformed(GestureOverlayView,Gesture):void
```

in a TextView. This is done by setting the 'textView' varible present in this class. The second way is to let this class notify another class when a character has been recognized. This is done by defining a class which implements 'OnCharacterRecognized' interface and setting the ' cListener' variable present in this class.

CharacterRecognizer(Context context, int mRawSource)

When the gesture data is present in the resources folder as a raw file, this constructor needs to be used. The resource id of the raw file is passed as the second argument.

CharacterRecognizer(Context context, String path)

The 'path' argument represents the path of the file on the SD-Card which contains the gesture data. The file present in this path is used to extract the gesture data.

CharacterRecognizer(Context context, File file)

The 'file' argument represents the file object which contains the gesture data. This file is used to extract the gesture data.

setInnerView(Activity mContext, int layoutId)

This method is used to add an inner view to the ’GestureOverlayView’, which is a part of this class. This method takes the resouce id of the inner view, inflates it and adds it to as an inner view.

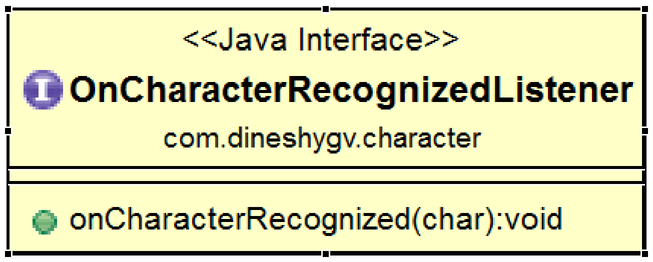
setTextView(TextView textView)

This method is used to set the TextView(android.widget.TextView) to whose string the new character that is detected needs to be appended.

setOnCharacterRecognizedListener(OnCharacterRecognizedListener mCListener)

This method is used to set the listener which needs to be notified once a character is detected.

4.3.2 OnCharacterRecognizedListener

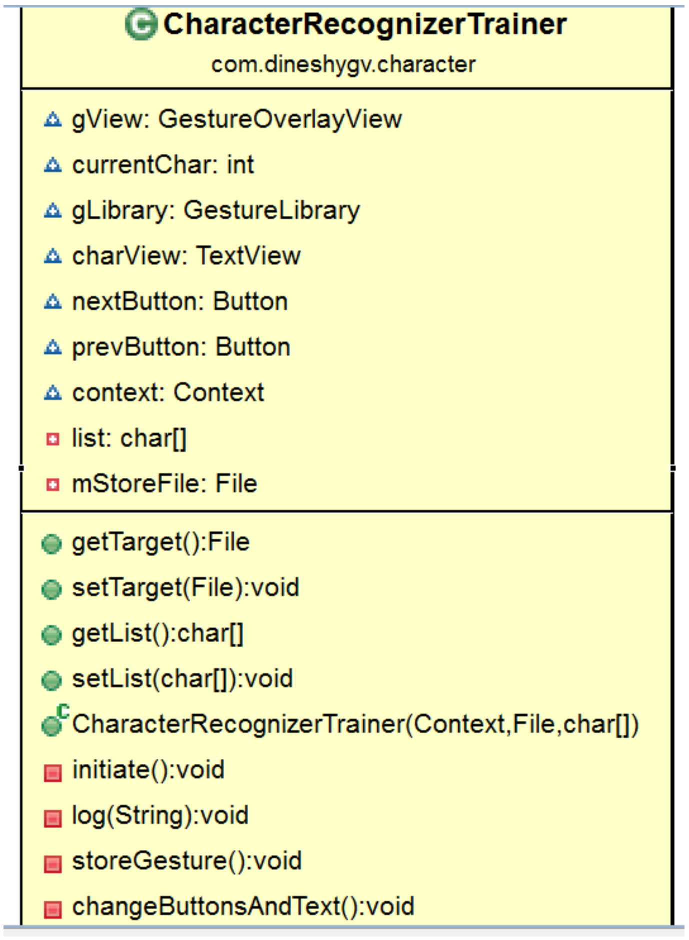

<<Java Interface>>
OnCharacterRecognizedListener
com.dineshygv.character
onCharacterRecognized(char):void

This is an interface that needs to be implemented by a class that needs to be notified when a character has been detected by the ’CharacterRecognizer’ class. An object of the class that implements this interface is passed as an argument to the ’setOnCharacterRecognizedListener’ method of ’CharacterRecognizer’ class.

onCharacterRecognized(char recognizedChar)

This method is called when the character is recognized. The argument 'recognizedChar' represents the character that has been recognized by the 'CharacterRecognizer' class.

CharacterRecognizerTrainer



This is an activity that lets the user define his own gestures for individual characters. This is a standalone activity and doesn't depend upon any other resources. Its layout is built by itself from the java code written in it. Its layout contains TextView(android.view.TextView) which prompts the user to input a character. At the bottom, it has 'Next' and 'Previous' buttons which lets the user switch to next and previous characters. All the remaining portion is filled with 'GestureOverlayView' which takes the gesture input from the user and delivers it to the activity.

buildLayout()

This method builds the layout of the application. This happens by defining the required TextViews and Buttons, setting their parameters and adding them to the GestureOverlayView. All these are then, finally added to a relative layout. The position parameters of widgets are given as relative layout parameters.

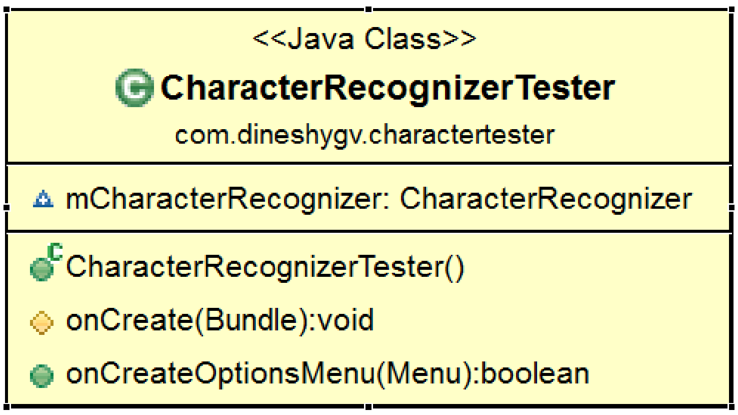
ChangeButtonsAndText()

This method is called when one of the buttons are pressed. This method changes the text at the top of the layout from previous character to the current character. It also disables the buttons if the end of the character set is reached.

storeGesture()

This method first checks if the the length of the gesture drawn is more than the tolerable length. This is to avoid storing the gesture if the user didn't draw any gesture. If the length is above tolerable level, the gesture is added to a GestureStore(android.gesture.GestureStore) object and saves it to a file on SD-Card.

4.3.3 CharacterRecognizerTester



```
<<Java Class>>
CharacterRecognizerTester
com.dineshygv.charactertester

mCharacterRecognizer: CharacterRecognizer

CharacterRecognizerTester()
onCreate(Bundle):void
onCreateOptionsMenu(Menu):boolean
```

This is an activity that demonstrates the usage of the ’CharacterRecognizer’ API. Its layout has a TextView and a GestureOverlayView. Whenever a character is drawn on screen, it is detected by the CharacterRecognizer API and is appended to the text view’s string.

onCreate

This method makes an instance of ’CharacterRecognizer’. It uses its ’addInnerView’ method to add its layout as inner view of the GestureOverlayView of the CharacterRecognizer. Then, it uses ’setTextView’ of the CharacterRecognizer to set the recognizer’s target text view, so that newly detected characters are appended to its string.

4.4 Use Cases

4.4.1 CharacterRecognizer

To use the detector, the following steps needs to be followed:

- An instance of `CharacterRecognizer` class needs to be created.
- The constructor of the above class takes two arguments. The first argument is the context of the application. The second argument is the location of the data file.
- Instead of file input, a `rawSource` can also be given by including the data file in the resources folder of the application.
- The interface `OnCharacterRecognizedListener` needs to be implemented, which requires implementing the method `onCharacterRecognized` by the class. An instance of such class is set in the `CharacterRecognizer` using the `setOnCharacterRecognizedListener` method.
- Whenever a character is recognized, the method `onCharacterRecognized` is called with the recognized character as the argument.
- Instead of implementing the interface, the `TextView` which should be filled on recognizing the character can be set using the `setTextView` method. When a character has been recognized, the recognizer automatically appends the character at the end of the text view.

4.4.2 CharacterRecognizerTrainer

To use the API to re-configure the data file, the following steps needs to be followed by the developer. Following these steps allows user to input their own gestures in place of the gestures predefined in the application.

- An option should be created for the user to initialize the trainer activity in the form of a button or a menu item.
- When this option has been executed, the `CharacterRecognizerTrainer` should be called as follows:

```
Intent intent = new Intent(context,
    CharacterRecognizerTrainer.class);
startActivity(intent);
```

The trainer is a stand-alone activity which doesn't use any resource files to build its layout. Instead, its layout is built from the java code present in the activity.

4.5 Results

The following screenshots are of the CharacterRecognizerTester and CharacterRecognizerTrainer applications:

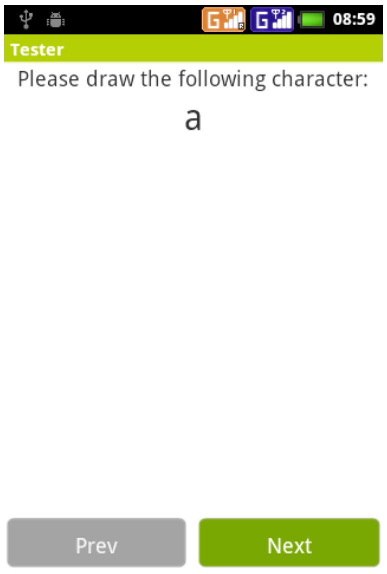


Figure 4.2: A screenshot of the trainer application

In the figure 4.2, a screenshot of the CharacterRecognizerTrainer has been shown.

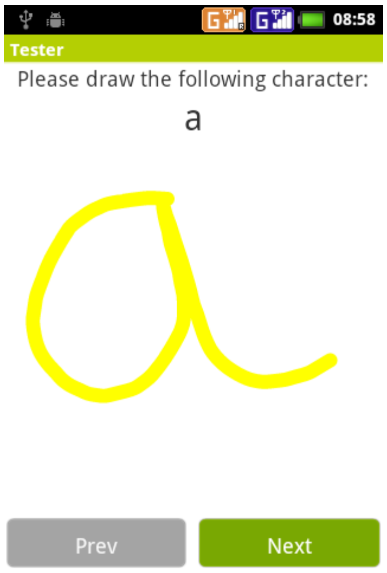


Figure 4.3: A screenshot of the trainer application with a character drawn on screen

In the figure 4.3, a screenshot of the CharacterRecognizerTrainer has been shown with a character drawn on screen.

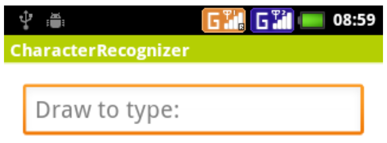


Figure 4.4: A screenshot of the tester application

In the figure 4.4, a screenshot of the CharacterRecognizerTrester has been shown.

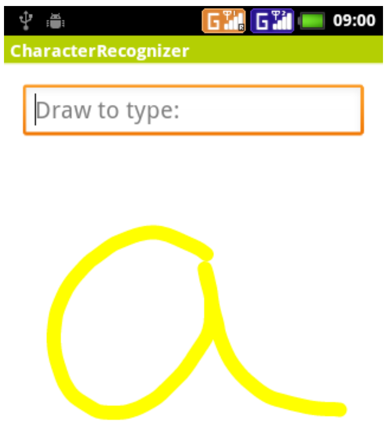


Figure 4.5: A screenshot of the tester application with a charcter drawn on screen

In the figure 4.5, a screenshot of the CharacterRecognizerTrester has been shown with a charcter drawn on screen.

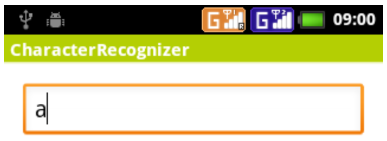


Figure 4.6: A screenshot of the trainer application with the predicted character entered into the text box

In the figure 4.6, A screenshot of the trainer application with the predicted character entered into the text box is shown.

4.6 Conclusion

Thus, the API for user configurable line gesture recognition has been extended to user configurable character recognition API. These API are used in the next chapter to develop a demonstration application, which allows blind users to use the basic features of a phone like searching for contacts, making calls and sending messages. The input for this application is through gestures which are configurable to their hand-writing by using the API developed in this chapter.

CHAPTER 5

Contacts Application based on Gestures for people without vision

5.1 Introduction

Recent advances in touch screen technology have increased the prevalence of touch screens and have prompted a wave of new touch screen-based devices. However, touch screens are still largely inaccessible to blind users due to interaction techniques that require the user to visually locate objects on the screen.

To address this problem, in this chapter, the API developed in the previous chapters is used to build an application for blind users to use the basic features of a phone like making calls, searching for contacts and sending messages. This application takes input from the blind user in the form of line drawing gestures on the screen. The output is given using Text-To-Speech engine. In this chapter, we look into some of the previous and current work that is being done that lets blind users use smart phones. Then, the description of the application is given, which is followed by guidelines to use the application. Finally, results and conclusion are discussed.

5.2 Smart phone applications for blind people

According to the World Health Organization, 285 million people are visually impaired worldwide: 39 million are blind and 246 have low vision (9). Smart phones have become incredibly powerful tools for these people. A phone's camera can identify money and read text, and GPS navigation tells blind users where they are and what's nearby. Most of these applications depend upon speech recognition, text-to-speech, sound and vibration feedbacks.

Trinetra (10) is a set of three applications that solves the problem of finding groceries in stores, using an RFID tags and smart phones.

Talkback is an application provided by Google that reads back the contents on the screen for helping the visually impaired.

5.2.1 Slide Rule

Slide rule (11) is a device designed with the similar purpose as this chapter. It takes the input from the blind users in the form of a gesture. In a user study conducted by its designers, 7 out of 10 people preferred this application to a button based application. However, users made more errors when using Slide Rule than when using the more familiar button-based system.

5.3 About this application

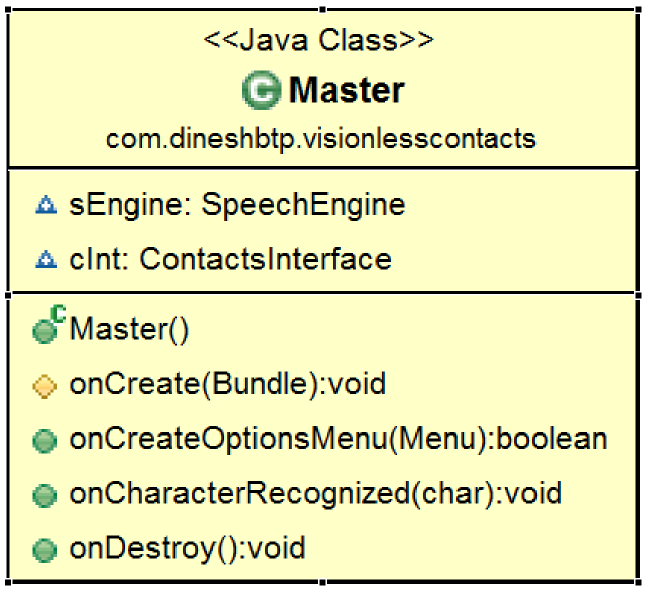
This application uses the line drawing gestures for taking the input from the blind user. Some of the standard operations performed on an application are predefined in this application like

- left button or ok
- right button or back or cancel
- alphabets
- numbers
- special characters
- backspace
- scroll keys

These can be re-configured with the help of a menu present in the application. The output is given with the help of a Text-to-speech engine. This engine takes a string or a character as an input and speaks it through the phone's speaker.

5.4 Class definitions

5.4.1 Master



This is the main class which extends the android activity(android.app.Activity). This class makes an instance of 'CharacterRecognizer' present in the previous chapter. It implements the 'OnCharacterRecognizerLisener' interface discussed in the previous chapter. This interface is set as as the target of the CharacterRecognizer, so that the method 'onCharacterRecognized' will be called, whenever a character has been recognized. In this method, the 'process' method on 'ContactsInterface' which processes the input and performs the corresponding actions.

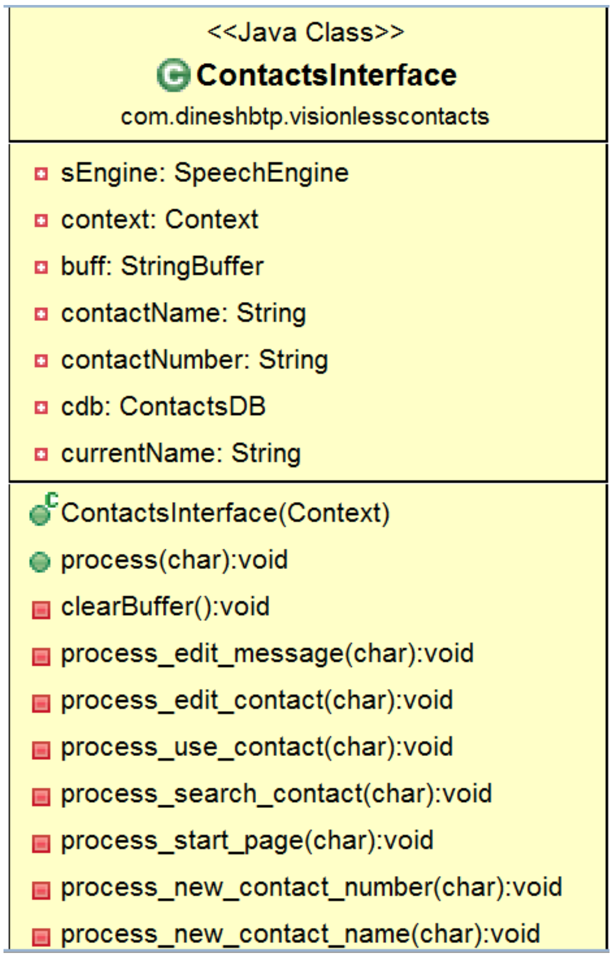
onCreate

This method makes an instance of CharacterRecognizer and sets its target. It also initializes the 'SpeechEngine' obejct which is required to deliver the Text-to-speech output.

OnDestroy

This method shuts down the speech engine to prevent memory leaks and exits the application.

5.4.2 ContactsInterface



<<Java Class>>
ContactsInterface
com.dineshbtp.visionlesscontacts
sEngine: SpeechEngine
context: Context
buff: StringBuffer
contactName: String
contactNumber: String
cdb: ContactsDB
currentName: String
ContactsInterface(Context)
process(char):void
clearBuffer():void
process_edit_message(char):void
process_edit_contact(char):void
process_use_contact(char):void
process_search_contact(char):void
process_start_page(char):void
process_new_contact_number(char):void
process_new_contact_name(char):void

This class processes the input given by the user. The state machine present in this application proceeds to the next stage according to the given character input. It has an enumeration 'State' defined in it which tells in which state, the application currently is in.

process(char c)

This method is the main method which is called by the 'Master' class of this application. This method has a switch case which calls the corresponding processing function for the given state and passes the character input to that processing function.

Enum State

This is an enumeration for the states in which the current the state machine in the application can exist. These states are explained as below:

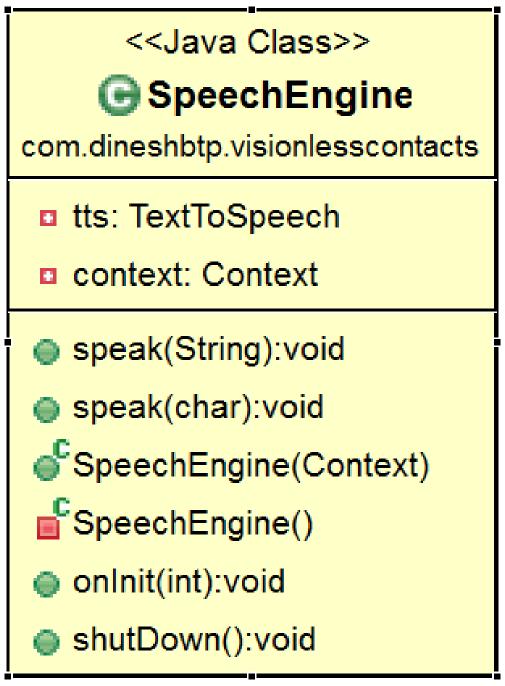
- START_PAGE: This is the starting page of the application. When the application starts, the state machine is going to enter this state. Also, if a particular operation is complete, the application again returns to this state.
- NEW_CONTACT_NAME: In this state, the user is prompted to enter the name of the new contact. This state is preceeded by the start page and is succeded by the new contact number state.
- NEW_CONTACT_NUMBER: In this state, the user is prompted to enter the number of a new contact he wishes to save in his phone. After saving the contact, the state of the application returns to state page.
- SEARCH_CONTACT: In this state, the user enters the first few characters of the name of the contact he wishes to search. The speech engine reads out the names of the contacts that start with these characters. The user can also scroll up and down to go to next or previous contact in a list of contacts that start with these characters.
- USE_CONTACT: When a contact is selected from a list using search feature, this state is entered. In this state, the user is prompted to enter a gesture that suggests whether the users wants to call the selected contact, send a message, edit or delete the contact.
- EDIT_CONTACT: This state in entered when the user enters gesture to edit the contact in the use contact state. In this state, the user is prompted to enter the new number of the old contact. After this, the contact is saved with the new contact and the state returns to the start page.
- EDIT_MESSAGE: When the user enters gesture to send a message in the use contact state, this state is entered. In this state, the user enters the text message that he wants to send to the contact. After entering 'ok' gesture, the message is sent to the contact and the state returns to the start page.

Processing functions

Each of the processing function has the following basic properties: It has a switch case which decides which operations to perform on receiving which input.

- ?: At any time inside the application, drawing a '?' leads to help option, which tells the user the possible gestures he can draw and the outcomes of these gestures. For example, if the application is in the state 'USE_CONTACT', drawing a '?' will result in speech output 'Draw C for call, M for message, E for edit contact and D for delete contact'.
- Back: When this gesture is performed, the application returns to its previous state. This gesture is represented by the character '&'. This is similar to the 'back' key or on the phone.
- Ok: This is used to go to next state in the application. This is similar to the 'Select' key on the phone. This is represented by the character '*'.
- characters a to z , 0 to 9: In the states which require user to enter name or number, the processing method takes the input and stores it in a string buffer. When 'ok' gesture is performed, the string buffer is extracted into a string and is used for storing or searching.
- Backspace: When this character is received as argument, the last character from the string buffer is removed. This character is represented by the character '#'.

5.4.3 SpeechEngine



This class has methods for interfacing with the text-to-speech engine present in the android API. These methods can be used to initialize the engine, deliver string to the engine to speak and shutdown the engine.

SpeechEngine(Context)

This method is used to initialize the speech engine.

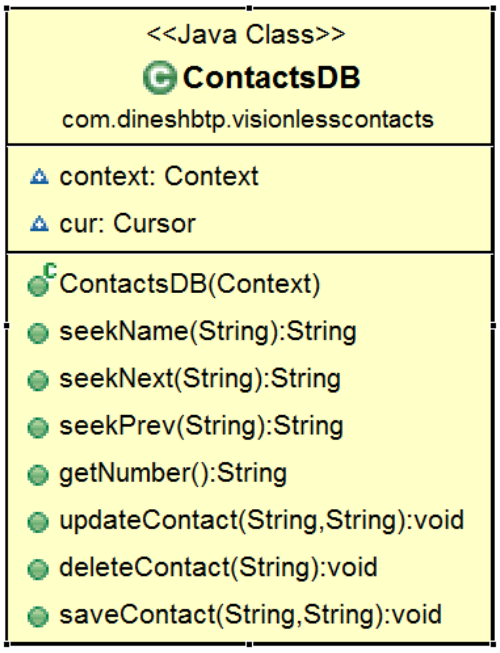
speak

The string delivered to this method as an argument is delivered to the engine to speak.

shutdown

This method shuts down the speech engine. This method has to be performed before the application exits.

5.4.4 ContactsDB



This class is used as an API to interact with the contacts database of the Android system. The android system stores the contacts in the form of a SQLite database. Each contact contain the following fields:

- First Name
- Last Name
- Home
- Mobile
- e-mail
- Description

The methods present in this class can be used to search for a contact, edit a contact and delete a contact.

saveContact(String name, String number)

This method stores a contact with the given name and number in the database.

seekName(String key)

This method gets the name of the first contact in a list of contacts whose names start with the given key.

SeekNext()

This method gives the next name in the list of contacts whose names start with the key set previously using seekName method. It returns null if the list is empty or the list has reached its end.

SeekPrev()

This method gives the next name in the list of contacts whose names start with the key set previously using seekName method. It returns null if the list is empty or the list has reached its beginning.

GetNumber()

This method gives the number of the contact that was previously returned by any of the three methods given above.

saveContact(String name, String number)

This method updates a contact with the give name and number in the contacts database.

deleteContact(String name)

This method deletes the contact with the give name.

5.5 Use Cases

5.5.1 Configuring the gestures

- After starting the application, enter 'C' to configure. The speech engine prompts the user to enter a certain character starting from a to z.
- Once the user enters the gesture, double tap to proceed to the next character.
- When the user wants to skip a character, double tap to proceed to the next gesture. The gesture that is already there in the database remains unchanged.

5.5.2 Searching for a contact

- In the start page, enter 'S' to enter search mode.
- In the search mode, enter the first letter of the required contact. The application reads out the name of the first contact with that letter as its first letter in the name.
- Use 'scroll up' and 'scroll down' gestures to move up or down among the contacts named with the given letter.
- Enter more letters to form a string. The application reads out the name of the contact whose name starts with the string.
- When the required contact has been reached, enter 'ok' gesture to select the given contact.

5.5.3 Editing and Deleting a Contact

- Once a contact has been selected, enter ’E’ to edit the contact.
- Enter the new number for the contact.
- Once the number has been entered, draw ’ok’ gesture.
- In the same way, entering ’delete’ gesture after selecting the contact deletes the contact.

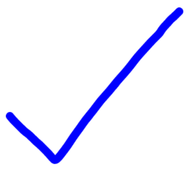
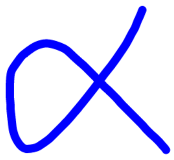
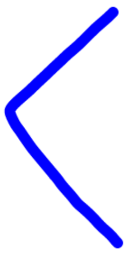
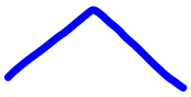
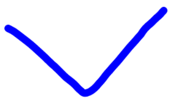
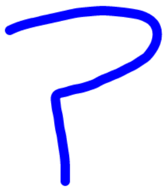
5.5.4 Calling and sending an SMS

- Once a contact has been selected, enter ’C’ to call the contact.
- Enter ’M’ to enter message mode.
- After entering message mode, enter the message text and draw ’ok’ gesture to send message to the contact.

5.6 Standard Gestures defined in the Application

Some of the standard gestures that are pre-defined in the application are shown in the table 5.1.

Table 5.1: Table showing some of the gesture defined in the application by default

OK	
BACK	
BACKSPACE	
UP	
DOWN	
HELP	

5.7 Functioning of the Application

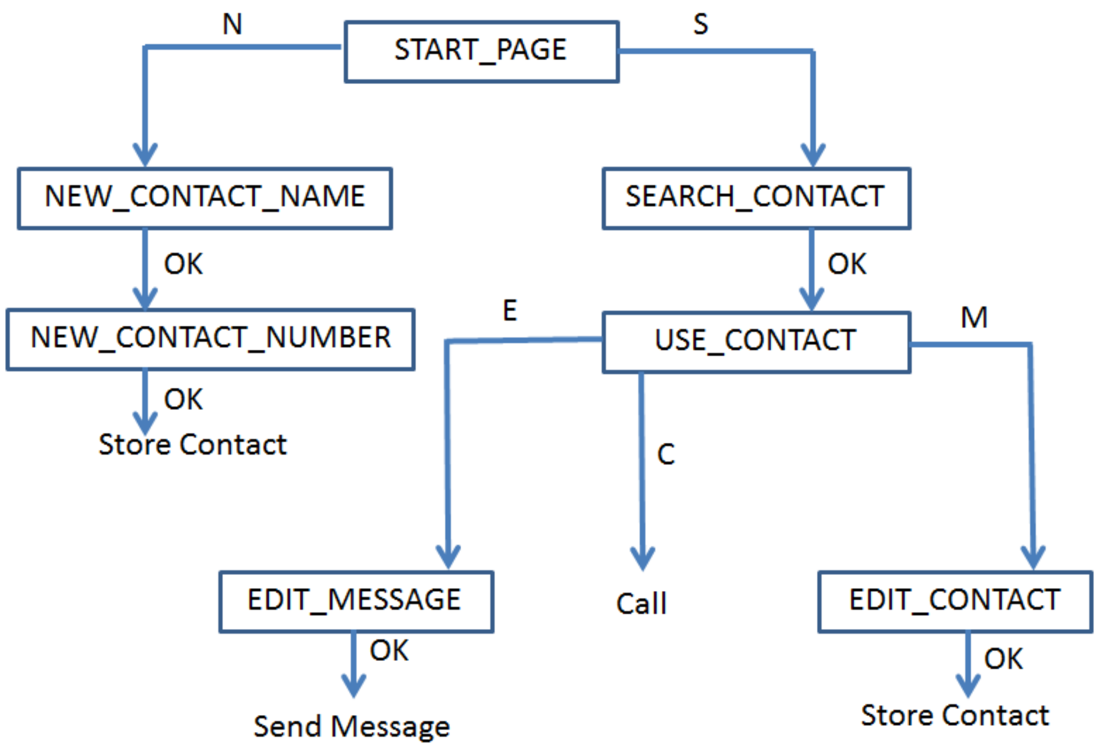


Figure 5.1: A flow chart showing the state-machine of the Application

A state-machine is designed which advances to a particular state based on the character input recognized by the API. For example, this state machine initially is in `START_PAGE` state advances to `SEARCH_CONTACT` state on receiving an input of 'S' from the user. The flow of the state machine is shown in the figure 5.7.

5.8 Conclusion

This the API developed in the previous chapters has been used to develop an application that lets the blind people use basic features of the phone using gestures and also re-configure the gestures according to their handwriting. In the next chapter, we describe some of the improvements that can be done along with the scope for the future work.

CHAPTER 6

Conclusion and Scope for Future work

6.1 Conclusion

In this project, an API that uses touch input to detect gestures was developed. Later, the line gesture detection API provided by Android was improved to allow users to define their own gestures. This API has been extended to detect characters drawn on the screen which is configurable according to the user's handwriting. Finally, this API was used to develop an application for blind users which can be used to store contacts, search for contacts, make calls and send messages using Gesture Input on Screen.

Thus, the problem of Google API which is the lack of user configurability has been rectified by extending the API and the developed API has been put into use through an application.

6.2 Scope for Future Work

This project can be further extended by adding new gestures and developing more applications using the API. Some of the ideas are stated as follows:

6.2.1 New Gestures

Some of the other forms of gestures can be included in the framework developed in first part of the project, like gestures based on accelerometer sensors, gestures based on IR sensor present on the front face of the phone, etc. For example, some of the accelerometer gestures are shaking the phone, flipping the phone, etc.

6.2.2 New Applications

More new applications can be developed based on gestures using gesture API developed in this project. One example of such application is to control a PC using gestures on phone or tablet, where the phone serves as a remote and a substitute for keyboard and mouse.

REFERENCES

[1] Android Activity `http://developer.android.com/reference/android/app/Activity.html`

[2] Android Gesture Detector API `http://developer.android.com/reference/android/view/GestureDetector.html`.

[3] Android Line Gesture Detection Package `http://developer.android.com/reference/android/gesture/package-summary.html`.

[4] Android Versions `http://developer.android.com/about/dashboards/index.html`.

[5] Brad A. Myers *A brief history of human-computer interaction technology* 1998.

[6] Emilio Raymond Reeves & Alan Ryan Wick *Associating gestures on a touch screen with characters* 2010.

[8] Daniel W. Hillis *Bounding box gesture recognition on a touch detecting interactive display* 2006.

[8] Jakub Plichta *System and method for developing and classifying touch gestures* 2009.

[9] World Health Organization Report on Blindness `http://www.who.int/blindness/Vision2020_report.pdf`

[10] Priya Narasimhan, Rajeev Gandhi & Dan Rossi *Smartphone-based assistive technologies for the blind* 2009.

[11] Shaun K. Kane, Jeffrey P. Bigham & Jacob O. Wobbrock *Slide rule: making mobile touch screens accessible to blind people using multi-touch interaction techniques* 2008.