RUTGERS
School of Engineering

Department of Electrical and Computer Engineering

332:428 Senior Capstone Design Final Report Spring 2012

Gesture Controlled Smartphone

Advisors:
Dr. Christopher Rose
Dr. Kristin Dana

Team Members:
Anton Krivosheyev
Richard Romanowski
Michelle Greenfarb

Table of Contents

Abstract

The progression of technology is fast-moving, and there is no sign of this pace slowing down in the near future. Society has attempted to make personal computers even more personal, with the invention of the desktop computer, the portable notebook PC, and the cellphone. The smartphone is an innovation that has allowed users to have everything they need at their fingertips. This project, continuing in the trend of personalization, is set out to design a portable head-mounted personal display that would eliminate the need for a physical screen on portable Android devices, replacing it with a virtual screen projected onto arbitrary surfaces or directly into the eye.

In a time where cell phones become obsolete in just a few months, this project provides the user a way to expand the utility of their current Android device. With the download of an Android application and a wearable physical portion of the project, a user has all the advantages of their smartphone without the need for tactile interaction. The external wearable projector armed with its supporting circuitry will allow the user to read messages and other information sent from the device. Different gestures will translate to different commands that the phone will act upon to provide information to the user of the device. In addition, the ability to control a mobile device using hand gestures will allow a user to control a mobile Android device while it is still in a pocket.

This paper examines the design process necessary to create this human-computer interface accessory for Android.

Design Goals

The initial goal of this project was to design a head-mounted heads-up display that would communicate wirelessly with an Android device via hand gestures. Very early in the project, the laser heads-up display portion was moved to the list of long-term goals due to time restraints and safety regulations. The goal for the semester then changed to a system that would display information onto either the wrist or a wall. Then, this could later be extended into the heads-up display. After this modification, the project goals included an Android application and a physical component to be worn around the neck.

After the construction of the wearable portion, the project was divided into two areas. In order for the entire project to be successful, each subproject necessarily had to function independently.

The first of these subsystems includes the path of information from the camera to the phone. This includes the procedure to recognize hand gestures with a homogeneous background and then send a command to the phone once a gesture is recognized.

The second project includes the path from the phone to the projector. The goals of this task essentially included everything else necessary for the project to work including working on the user interface, the coding of the Android application, the coding of sending commands to display information, and the optimization of the speed of the system.

Related Work

The demand is increasing for wearable computers and personal displays. These displays are typically characterized by having a unique function of displaying desirable digital information, at all times, without the need to ever physically interact with the device that is serving the information to the user. These devices are commonly known as heads-up displays, or HUDs. The interest in HUDs is currently rising as can be observed using the Google search request increase for "heads up display", as shown in Fig 1.

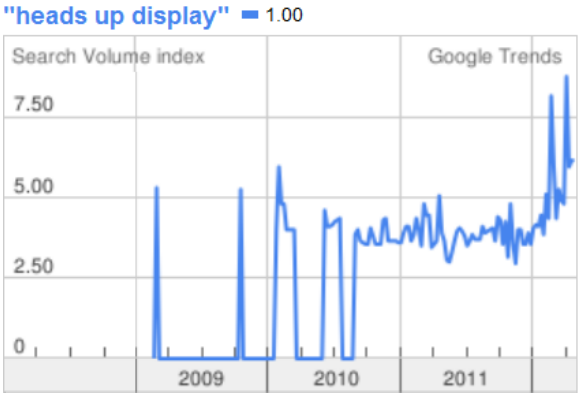

Fig. 1: Google search trends for "heads up display".

There were numerous previous attempts to design a system with similar functions. However, many of them were neither compact nor cheap to manufacture, and those that did make it off the drawing boards to the working prototypes were either too expensive or had extremely limited functionality, such as absence of real-time visual data analysis and a much-useful augmentation of physical and virtual realities.

Fig.2: STAR™ 1200, see-thru augmented reality display.

Fig.3: MOD Live GPS (HUD) for ski and snowboard goggles.

The first of these devices, shown in Fig. 2, appeared on the market in summer 2011. Although a great device, the price tag of nearly $5000 does not make this an ideal product for the average consumer. Another device currently on the market, MOD Live, shown in Fig. 3 carries a much more reasonable price tag of $400, and with a few of modifications would be the perfect solution to the problem--an affordable Android HUD.

System Design

The system design proved to be very important in the success of the project. Power consumption, data transfer rates, the physical size and shape of the system, as well as its use, are all dependent on the hardware components chosen. Each of the hardware components are modular and designed to work with other components which communicate on the same protocol. This modularity is taken advantage of by the system to either command or extract needed information from each hardware component as required by the design.

The current working system has the functionality shown in Fig. 4. There are three devices receiving input for processing: a camera built into the laptop, an ultrasonic distance detector, and the light sensor on the Android phone. The data transferred throughout the circuit from the distance detector is shown by green, and data transferred from the laptop camera and light sensor is shown in blue. The orange indicates feedback sent from the µVGA board to the phone, and the red arrow indicates all outgoing data from the phone necessary for projecting information.

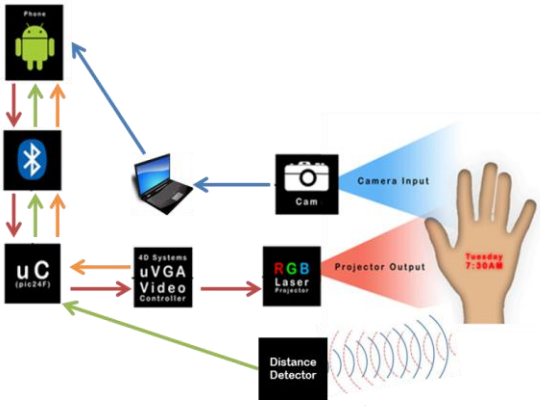

Fig. 4: Current system diagram.

The ultrasonic distance detector checks whether there are any objects present in front of the user (to detect a hand or a surface to project onto).

The laptop camera takes live images of the hand, which are analyzed using the gesture recognition algorithm on the laptop. Once a gesture is recognized, the laptop screen displays a screen color corresponding the a specific gesture. The phone lays against the laptop screen and based on the light intensity from the color on the screen sends a command to the phone.

All of the code for the Android application runs on the phone. Once the phone receives a command from the light sensor, it processes the information and forwards a command to the rest of the circuit to display relevant information.

The IOIO board, shown in the figure as a Bluetooth module and microcontroller, is the connector between the phone and the circuit. The connection can be either wired or wireless. The Bluetooth module is the main gateway for the data transferred between the device and the phone's internal Bluetooth transceiver. The microcontroller is responsible for forwarding the commands to the µVGA graphics controller. The µVGA graphics controller displays data on the pico projector through a direct connection. The portable laser projector contains a CPU used to display graphics, video, and custom graphical user interface (GUI), based on the commands that it receives from the main application running on the Android device.

Software

There are two software components: the software used for hand recognition, and the software used for the Android application. The hand recognition code is run and processed on the laptop. The application code is run on the phone. Each system can run independently of one another.

Gesture Recognition

For a machine to recognize gestures, several steps need to occur beforehand. A sequence of images needs to be acquired for analysis, in this case, a webcam. Then, the hand needs to segmented in each frame from the other objects that are not of interest. Finally, features of the hand, such as number of fingers, and orientation need to be extracted. These numbers will then be translated into a gesture.

When approaching object segmentation via color identification, there are several color spaces to be considered. The main three experimented with in this project were RGB, HSV, and YCbCr. Red-Green-Blue (RGB) color space is the most intuitive and convenient color space process in since this is how humans see the world. All colors are made up of a mixture of red, green, and blue, however, since white is the sum of all color primitives, computers have problems with illumination changes. Color consistency, is a skill the human brain can perform subconsciously, but a computer needs to be told explicitly that different shades of the same color are equivalent in different images, if the lighting changes. Because of this, RGB is not a good color space in which to perform color segmentation.

Hue-Saturation-Value (HSV) color space is arranged in a cylinder, with hue the degrees of rotation from a starting point, saturation is interpreted as the radius, and value, or intensity, as the height. This breakdown has hue as a better definition of the color of a pixel, more invariant to illumination since the brightness is translated into the saturation and values portions.

OpenCV calls the YUV color space YCbCr, and it is defined as a color space of one luma (Y) and two chrominance components (UV). The conversion from RGB to YUV contains three linear combinations. YUV also separates intensity and color information effectively like HSV. The distribution of skin-tone is most favorable in this color space than the former two, so this was the color space chosen for the algorithm.

There are several ways to segment a hand from a background. First, is to look at color segmentation, and choose all pixels of a certain color. A second method is to use two cameras in stereo, and create a depth map from the disparities between pixels. The hand would be the big object that is close to the camera. A third option is to use an infrared camera so that the temperature of human skin (30 to 34 degrees C) is easily differentiated from the cooler background.

To segment the hand from the background, all of the pixels that are skin need to be found. When the module starts up, it performs k-means clustering on random subset of pixels from the image, with k=2. Resulting is two coordinates in

YUV space that represent the color of the hand and the background. Each pixel is then compared to these two points and labeled as the centroid to which it is closer. Re-clustering occurs periodically after the first training to compensate for changes in illumination. Just in case, there are stray pixels that are labeled skin, or there are small, bright patches on the background, only the largest shape is considered for analysis.

Once the hand is segmented from the background, analysis to determine the gesture can begin. Contours will define the border of the hand. Several questions need to be answered to classify the gesture. The location of the center of the hand is the main piece of information that needs to be extracted so the fingertips can be found relative to this center. The palm of the hand can be approximated with a circle, and conveniently the largest approximating circle of any part of the entire hand/arm object. So, the center of the circle inside the contour of the hand that is of the largest area will be a very good estimation of the center of the hand. Locating the center of this circle, is the same as finding the point whose minimum distance to any point on the contour is the largest. This description of the problem is one that is easily solved, with the following algorithm:

1. Loop over all points that are considered part of the hand.
2. For each point p, loop over all points of the contour.
3. Save the minimum distance among p to all points of the contour.
4. Save the maximum of these minimums.

Since we are looping over a rectangular image, and a long contour, this is a polynomial algorithm. Some optimizations can be made, by adding conditional breaks, but it is still polynomial. Then, there are two ways to determine where the ends of the fingers are located, so that we can count how many fingers are extended. The first is to use the convexity hulls and defects. The hull is the shape that surrounds the contour as a rubber band would. The defects are where this hull is far away from the rubber band, the knuckles, and the points of contact are where the fingers are, or the edge of the image. A defect is defined as a set of three points: a start, middle, and end. The start and end are two points along the contour that touch the hull. The middle point is the point on the contour in between the start and end that is furthest away from the hull. The problem with this method is that ring and middle fingers would have trouble showing up since the start and end of their defects weren't well defined.

A second way of counting fingers is to find all the points of high curvature on the contour since the end of fingers, and the valleys where they meet are the places of highest curvature on the hand. These valleys, knuckles, can be differentiated from fingertips in two ways: if the method of determining curvature is signed, then the two will have opposite signs. The other way of separating knuckles from fingers is to calculate the euclidean distance from the center of the hand. For this project, any potential candidate for a fingertip that was further than twice the radius of the palm is confidently labeled as a fingertip. Anything else is assumed to be a knuckle or a finger that is not fully extended. Another facet of gestures is which fingers are extended. The angles between fingers can be used to discern from different gestures.

Android Application

The code starts with importation of libraries that allow Android to communicate with the IOIO board.

One of the microcontroller's pins (#46) is then assigned, output viewport is created, and two UART pairs of communication ports are initiated.

Next, the function onCreate initiates objects required for the app to run. Then, IOIOThread class is defined, which hosts functions that run either only once, loop continuously, or run conditionally (for example, if connection is terminated unexpectedly).

Within the IOIOThread class, the function setup() runs once when the app is launched on the phone. It opens an analog input port on the microcontroller; opens the UART ports with RX pin 4, TX pin 5, baud rate of 9600, with no parity bits, and a stop bit of one; it then proceeds to assign buffer variables for data coming out of the phone and data coming into the phone. After all initiations are completed, the control data is then is being sent.

While still inside the setup() function, three commands are sent. First, is the autobaud command; it allows the microcontroller to automatically set the baud rate. Next, the projector's screen is cleared by sending a hexidecimal byte 0x45. This clears anything remaining on the screen from previous time the app was run. And last, the projector's resolution is set to be 640x480 by sending a string of bytes corresponding to the uVGA board's communication protocol.

Now that the setup is complete, a loop function is run which loops indefinitely until the app is terminated. The functions contained in this loop continuously receive gesture data from the gesture recognition part, reads the output of the ultrasonic rangefinder, extract data from the Android's system (time, text messages, GPS position, etc), and then send appropriate commands to the microcontroller's UART ports to command the projector connected to the uVGA.

The loop function first reads the range from the ultrasonic rangefinder. This value is then converted into an integer for convenience sake. The screen is cleared again at this point, and the value at the UART's input port is read to make sure the data is successfully received. Then if the distance of any object is within approximately 3 feet of the device, the Android phone's internal functions are accessed to read the light sensor, text messages, and current date and time. Then these data are translated into something the uVGA can understand (e.g. individual bytes of ASCII characters), and it is sent byte-by-byte to the uVGA in accordance with its communication protocol.

Both software components can run independently from one another with the same results. For the system to work as a HUD, both sections are necessary. These components share data that is used to make correct decisions as expected by the user.

Design Problems Encountered and Solutions

The system design had to be modified until it was ready to demonstrate the capabilities of this electonic tool.

The initial system design is shown in Fig. 5. There were many issues with this system. The camera used first was chosen because the pictures taken were bmps. Early on, the camera had to be changed because the data sheet was incomplete and so some commands needed were not available.

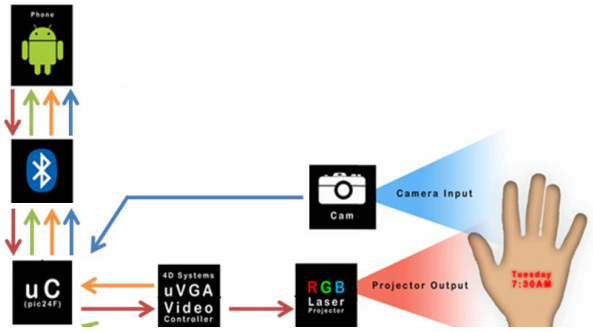

Fig. 5: Original system attempted.

The camera used was switched to another external camera, which takes images as jpegs. This was an issue because each component in the circuit uses bmps, so the images could not be sent to the phone for processing. Also, if the tranfer of the photos was possible, the processing time to save, convert, and process the photo would have been too long.

Next, an ultrasonic distance detector was put in place of the camera. The issue with this, however, was that it eliminated the ability to use the hand gesture algorithms with the system. It was used, nontheless, to show proof of concept to be able to send data from the phone to the projector via some kind of external command.

There were three ranges of distance utilized to shown three different displays

of information from the phone. After this was complete, a camera was put back into the system.

The system currently used utilizes both the ultrasonic distance detector and the camera from the laptop. This is the system decribed above in Fig. 4.

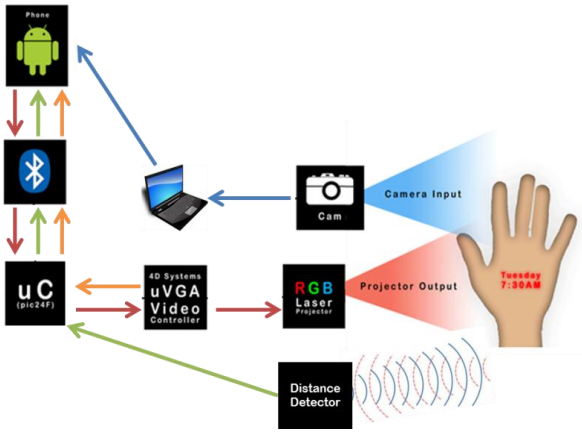

Fig. 6:Current system diagram.

Throughout the duration of the project, there were also many issues encountered with the software. The majority of these issues were encountered early on in the project because of the lack of knowledge in Android development. Once the communication protocol was understood, the issues with software were resolved fairly quickly.

The only ongoing software issue is the inability to access the Android camera while simultaneously running the application.

Future Considerations

The next step in completing this project is to implement the system shown below in Fig. 7.

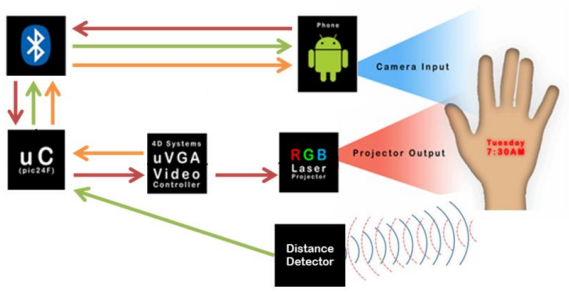

Fig. 7: System diagram replacing webcam with Android camera.

The OpenCV gesture recognition code can be tranferred to an application utilizing current OpenCV libraries which are made specifically for Android.

The above system still does not conform to the initial goal of seperating the hardware entirely from Android. Since the goal is to design a standalone device (and because by then the image analysis gesture-extracting algorithms would be transferred to Android OpenCV), the system design would be modified to the one shown below.

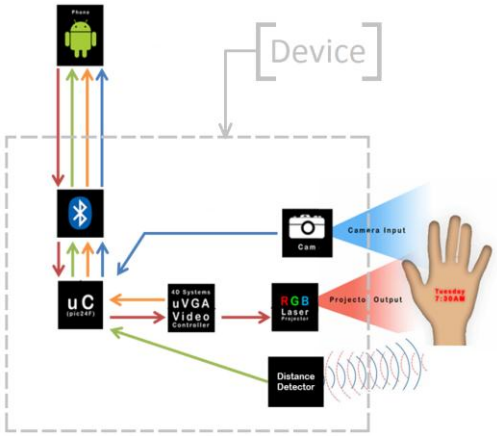

Fig. 8: Future system diagram using external camera separating Android phone from the device.

Conclusion

This project has a lot of potential for expansion. Once the phone is separated and the rest of the system is a standalone device, the application can be expanded to incorporate anything from Android. Ideally, the data rates will eventually be high enough to be able to send entire screenshots from the phone to the projector. Currently, the system only extracts information from the phone and displays it using a GUI developed during the duration of this project.

Another future point of optimization comes with expansion of gestures. With the light sensor currently being used, there is a maximum of three recognized gestures that can be sent to the phone.

Finally, the gesture recognition currently works only with a black background in a well-lit area. This is something that will have to be addressed if this product is going to be used widely; the convenience aspect of the project is nonexistent if it can only be used in a very specific setting.

The most important part for this project moving forward is the camera. The proof of concept for hand recognition has been shown. The proof of the ability to manipulate the phone through some external device has been shown. With a camera that can work with each component in the system in a timely manner, this project will be complete and ready for expansion.

Appendix A

Gesture Recognition

handed in as hard copy

Android Application

```
package romanowski.main;

/*
 * Head-mounted gesture-controlled smartphone with projected-onto-the-wrist visual feedback.
 * Upon approval on human testing, functions extend to a head-mounted heads-up display
 * to directly transmit images onto the retina (instead of first reflecting off the wrist).
 *
 * This project is a combination of computer vision, android app development,
 * embedded electronics, and communications fields of electrical and computer engineering.
 *
 * Project Authors: Anton Krivosheyev, Richard Romanowski, Michelle Greenfarb
 * Date First Created: February 1, 2012
 * Date Last Modified: 5/2/2012
 *
 */

import ioio.lib.api.AnalogInput;
import ioio.lib.api.Uart;
import ioio.lib.api.exception.ConnectionLostException;
import ioio.lib.util.AbstractIOIOActivity;

import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
import java.text.DateFormat;
import java.util.Date;

import android.content.Context;
import android.database.Cursor;
import android.graphics.Bitmap;
import android.graphics.Canvas;
import android.graphics.Paint;
import android.graphics.Rect;
import android.hardware.Sensor;
import android.hardware.SensorEvent;
import android.hardware.SensorEventListener;
import android.hardware.SensorManager;
import android.net.Uri;
import android.os.Bundle;
import android.view.View;
import android.view.WindowManager;
import android.widget.Button;
import android.widget.LinearLayout;
import android.widget.TextView;

//Modified Activity to AbstractIOIOActivity:
public class ScreenCaptureActivity extends AbstractIOIOActivity implements
SensorEventListener{
```

```
        private static final int ULTRASONIC_PIN = 46;
        Bitmap bm = null;
        private TextView output;
        private Uart uart_uVGA;
        private InputStream in_uVGA;
        private OutputStream out_uVGA;
        private int lightQuantity;


         private  SensorManager sensorManager = null;
         private  Sensor currentSensor = null;

         @Override
         public void onResume(){
          super.onResume();
          if(currentSensor != null)sensorManager.registerListener(this, currentSensor,
SensorManager.SENSOR_DELAY_FASTEST);
         }

         @Override
         public void onPause(){
          super.onPause();
          sensorManager.unregisterListener(this);
         }



        LinearLayout view;
        @Override
        protected void onCreate(Bundle savedInstanceState) {
                // TODO Auto-generated method stub
                super.onCreate(savedInstanceState);
                setContentView(R.layout.main);

                getWindow().addFlags(WindowManager.LayoutParams.FLAG_KEEP_SCREEN_ON); //keep
screen of phone on while application is open

                sensorManager = (SensorManager) this.getSystemService(SENSOR_SERVICE);
                  currentSensor = sensorManager.getDefaultSensor(Sensor.TYPE_LIGHT );
                  if(currentSensor != null){
                   sensorManager.registerListener(this, currentSensor,
SensorManager.SENSOR_DELAY_FASTEST);
                  }else{
                   //do nothing
                  }

                output = (TextView)findViewById(R.id.textview);
                view = (LinearLayout)findViewById(R.id.main);
                Button myBtn = (Button)findViewById(R.id.myBtn);

                myBtn.setOnClickListener(new View.OnClickListener(){
                        public void onClick(View v) {
                                click();
                        }
                });
        }

        @Override
         public void onAccuracyChanged(Sensor sensor, int accuracy) {
          // TODO Auto-generated method stub
```

```
        }

        @Override
        public void onSensorChanged(SensorEvent event) {
         // TODO Auto-generated method stub

         if (event.sensor.getType() == Sensor.TYPE_LIGHT){
                //output = (TextView)findViewById(R.id.textview);
                //output.setText( output.getText()+ "value: " +event.values[0] + " lux ");
                lightQuantity = (int) event.values[0];
         }
        }




       public void click(){
              // TODO Auto-generated method stub

              View v1 = view.getRootView();
              System.out.println("Root View : "+v1);
              v1.setDrawingCacheEnabled(true);
              bm = v1.getDrawingCache();
              setContentView(new BitMapView(this, bm));

              System.out.println("Bitmap : bm.getPixel(0,0) =?= "+bm.getPixel(0,0));
       }
       class BitMapView extends View {
              Bitmap mBitmap = null;
              public BitMapView(Context context, Bitmap bm) {
              super(context);
              mBitmap = bm;
              }

              @Override
              protected void onDraw(Canvas canvas) {
              // called when view is drawn
              Paint paint = new Paint();
              paint.setFilterBitmap(true);
              // The image will be scaled so it will fill the width, and the
              // height will preserve the image's aspect ratio
              double aspectRatio = ((double) mBitmap.getWidth()) / mBitmap.getHeight();
              Rect dest = new Rect(0, 0, this.getWidth(),(int) (this.getHeight() /
aspectRatio));
              canvas.drawBitmap(mBitmap, null, dest, paint);
              }
       }


       class IOIOThread extends AbstractIOIOActivity.IOIOThread {
              private AnalogInput sonicInput;
              private int sonicVal;
              private int sonicValOld;
              private int sonicVal2;
              private int sonicValOld2;

              public void setup() throws ConnectionLostException {
                     try {
```

```
                        sonicInput = ioio_.openAnalogInput(ULTRASONIC_PIN);
                        uart_uVGA =
ioio_.openUart(4,5,9600,Uart.Parity.NONE,Uart.StopBits.ONE);

                        int value1 = 1, value2 = 2, value3 = 3; //variables for testing
bytes returned
                        in_uVGA = uart_uVGA.getInputStream();
                        out_uVGA = uart_uVGA.getOutputStream();


                        out_uVGA.write(0x55); //autobaud command (must be sent to uVGAII
board first)
                        value1=(byte) in_uVGA.read(); //device replies with 0x06 if
command successfully received

                        out_uVGA.write(0x45); //clear screen command
                        value2=(byte) in_uVGA.read(); //device replies with 0x06 if
command successfully received

                        out_uVGA.write(0x59); //display control functions command
                        out_uVGA.write(0x0C); //change VGA resolution sub-command
                        out_uVGA.write(0x01); //01:640x480 02:800x480 sub-command
                    value3=(byte) in_uVGA.read(); //replies with 0x06 if command
successfully received

                //drawAndroid();


                } catch (ConnectionLostException e) {
                        throw e;
                } catch (IOException e) {
                        // TODO Auto-generated catch block
                        e.printStackTrace();
                }
        }

        public void loop() throws ConnectionLostException {
                try {
                        int val = 0;
                        final float sonicValue = sonicInput.read();//read ultrasonic
analog
                        //final float sonicVal11=sonicInput.read();
                        //final float sonicVal22=sonicInput.read();
                        //final float sonicVal33=sonicInput.read();
                        //sonicVal = (int) (((sonicVal11+sonicVal22+sonicVal33)/3)*1000);
                        final int sonicVal = (int) (sonicValue*1000);
                        //sleep(100);


                        //Scanner st = new Scanner(new
File("/sys/devices/virtual/lightsensor/switch_cmd/lightsensor_file_state"));
                        //int lux = st.nextInt();
                        //st.close();


                        /*
                out_uVGA.write(0x43); //Draw Circle- 43hex
                out_uVGA.write(0x01); //x
                out_uVGA.write(0x40); //x
```

```
                        out_uVGA.write(0x00); //y
                        out_uVGA.write(0x50); //y
                        out_uVGA.write(0x00); //radius
                        out_uVGA.write((byte) lightQuantity); //radius
                        out_uVGA.write(0xFF); //color
                        out_uVGA.write(0xFF); //color
                        val=(byte) in_uVGA.read(); // replies with 0x06 if command successfully
received
                        */

                        //if(sonicVal > (sonicValOld+7) || sonicVal < (sonicValOld-7))
                        //{
                                //sonicValOld = sonicVal;
                                sleep(100);
                                out_uVGA.write(0x45); //clear screen command
                                val=(byte) in_uVGA.read(); //device replies with 0x06 if command
successfully received
                                if(sonicVal < 20)
                                {
                                        drawAndroid();
                                        /*
                                        out_uVGA.write(0x43); //Draw Circle- 43hex
                                        out_uVGA.write(0x01); //x
                                        out_uVGA.write(0x40); //x
                                        out_uVGA.write(0x00); //y
                                        out_uVGA.write(0x50); //y
                                        out_uVGA.write(0x00); //radius
                                        out_uVGA.write(0x1F); //radius
                                        out_uVGA.write(0x00); //color blue
                                        out_uVGA.write(0x1F); //color blue
                                        val=(byte) in_uVGA.read(); // replies with 0x06 if command
successfully received
                                        */

                                }
                                else if((sonicVal > 20 && sonicVal <30) && (lightQuantity >0 &&
lightQuantity <30 ))
                                {
                        Cursor cursor =
getContentResolver().query(Uri.parse("content://sms/inbox"), null, null, null, null);
                        cursor.moveToFirst();
                           String msgData = "";
                           msgData += " Last TXT:" + cursor.getString(13);
                           char[] msgDataChar = msgData.toCharArray();

                            out_uVGA.write(0x73); //Draw “String” of ASCII Text (text format) -
73hex
                            out_uVGA.write(0x01); //column
                            out_uVGA.write(0x01); //row
                            out_uVGA.write(0x03); // 3: 12x16 font
                            out_uVGA.write(0xFF); // ffff white
                            out_uVGA.write(0xFF); // ffff white
                            out_uVGA.write(msgDataChar[0]); // ASCII
                            out_uVGA.write(msgDataChar[1]); // ASCII
                            out_uVGA.write(msgDataChar[2]); // ASCII
                            out_uVGA.write(msgDataChar[3]); // ASCII
                            out_uVGA.write(msgDataChar[4]); // ASCII
                            out_uVGA.write(msgDataChar[5]); // ASCII
                            out_uVGA.write(msgDataChar[6]); // ASCII
                            out_uVGA.write(msgDataChar[7]); // ASCII
```

```
                        out_uVGA.write(msgDataChar[8]); // ASCII
                        out_uVGA.write(msgDataChar[9]); // ASCII
                        out_uVGA.write(msgDataChar[10]); // ASCII
                        out_uVGA.write(msgDataChar[11]); // ASCII
                        out_uVGA.write(msgDataChar[12]); // ASCII
                        out_uVGA.write(msgDataChar[13]); // ASCII
                        out_uVGA.write(msgDataChar[14]); // ASCII
                        out_uVGA.write(msgDataChar[15]); // ASCII
                        out_uVGA.write(msgDataChar[16]); // ASCII
                        out_uVGA.write(msgDataChar[17]); // ASCII
                        out_uVGA.write(msgDataChar[18]); // ASCII
                        out_uVGA.write(msgDataChar[19]); // ASCII
                        out_uVGA.write(msgDataChar[20]); // ASCII
                        out_uVGA.write(msgDataChar[21]); // ASCII
                        out_uVGA.write(msgDataChar[22]); // ASCII
                        out_uVGA.write(msgDataChar[23]); // ASCII
                        out_uVGA.write(msgDataChar[24]); // ASCII
                        out_uVGA.write(msgDataChar[25]); // ASCII
                        out_uVGA.write(msgDataChar[26]); // ASCII
                        out_uVGA.write(msgDataChar[27]); // ASCII
                        out_uVGA.write(msgDataChar[28]); // ASCII

                        out_uVGA.write(0x00); // TERMINATOR
                        val=(byte) in_uVGA.read(); //replies with 0x06 if command
successfully received

                                /*out_uVGA.write(0x43); //Draw Circle- 43hex
                                out_uVGA.write(0x01); //x
                                out_uVGA.write(0x40); //x
                                out_uVGA.write(0x00); //y
                                out_uVGA.write(0x50); //y
                                out_uVGA.write(0x00); //radius
                                out_uVGA.write(0x1F); //radius
                                out_uVGA.write(0xF8); //color red
                                out_uVGA.write(0x00); //color red
                                val=(byte) in_uVGA.read(); // replies with 0x06 if command
successfully received
                                */
                        }
                        else if((sonicVal > 20 && sonicVal <30) && (lightQuantity >30 &&
lightQuantity <500 ))
                        {
                                String currentDateTimeString =
DateFormat.getDateTimeInstance().format(new Date());

                                        char[] currentDateChar =
currentDateTimeString.toCharArray();

                          out_uVGA.write(0x73); //Draw “String” of ASCII Text (text
format) - 73hex
                          out_uVGA.write(0x01); //column
                          out_uVGA.write(0x01); //row
                          out_uVGA.write(0x03); // 3: 12x16 font
                          out_uVGA.write(0xFF); // ffff white
                          out_uVGA.write(0xFF); // ffff white
                          out_uVGA.write(currentDateChar[0]); // ASCII
                          out_uVGA.write(currentDateChar[1]); // ASCII
                          out_uVGA.write(currentDateChar[2]); // ASCII
                          out_uVGA.write(currentDateChar[3]); // ASCII
                          out_uVGA.write(currentDateChar[4]); // ASCII
```

```
                                out_uVGA.write(currentDateChar[5]); // ASCII
                                out_uVGA.write(currentDateChar[6]); // ASCII
                                out_uVGA.write(currentDateChar[7]); // ASCII
                                out_uVGA.write(currentDateChar[8]); // ASCII
                                out_uVGA.write(currentDateChar[9]); // ASCII
                                out_uVGA.write(currentDateChar[10]); // ASCII
                                out_uVGA.write(currentDateChar[11]); // ASCII
                                out_uVGA.write(currentDateChar[12]); // ASCII
                                out_uVGA.write(currentDateChar[13]); // ASCII
                                out_uVGA.write(currentDateChar[14]); // ASCII
                                out_uVGA.write(currentDateChar[15]); // ASCII
                                out_uVGA.write(currentDateChar[16]); // ASCII
                                out_uVGA.write(0x00); // TERMINATOR
                                val=(byte) in_uVGA.read(); //replies with 0x06 if command
successfully received
                                      /*out_uVGA.write(0x43); //Draw Circle- 43hex
                                      out_uVGA.write(0x01); //x
                                      out_uVGA.write(0x40); //x
                                      out_uVGA.write(0x00); //y
                                      out_uVGA.write(0x50); //y
                                      out_uVGA.write(0x00); //radius
                                      out_uVGA.write(0x1F); //radius
                                      out_uVGA.write(0x07); //color green
                                      out_uVGA.write(0xE0); //color green
                                      val=(byte) in_uVGA.read(); // replies with 0x06 if command
successfully received
                                      */
                               }


                        //}


                               //int val=0;
                               //out_uVGA.write(0x45);
                               //val=(byte) in_uVGA.read();

                               //read read read read until x=val1; if x=val1: read again, if x =
val2 read again if x = val3 wait 2 seconds send command


                                      //out_uVGA.write(0x45); //clear screen command
                               //val=(byte) in_uVGA.read(); //device replies with 0x06 if
command successfully received
                                      //out_uVGA.write(0x70);//Set Pen Size-70hex
                               //out_uVGA.write(0x00);//object will be drawn solid


                               /*
                               out_uVGA.write(0x70);//Set Pen Size-70hex
                               out_uVGA.write(0x01);//object will be drawn with wire frame

                               out_uVGA.write(0x72); //Draw Rectangle-72hex
                               out_uVGA.write(0x00); //x1
                               out_uVGA.write(0x01); //x1
                               out_uVGA.write(0x00); //y1
                               out_uVGA.write(0x01); //y1
                               out_uVGA.write(0x02); //x2 0x0280
                               out_uVGA.write(0x80); //x2
                               out_uVGA.write(0x01); //y2 0x01E0
```

```
                        out_uVGA.write(0xE0); //y2
                        out_uVGA.write(0xFF); //color white
                        out_uVGA.write(0xFF); //color
                        val=(byte) in_uVGA.read(); // replies with 0x06 if command
successfully received

                        out_uVGA.write(0x70);//Set Pen Size-70hex
                        out_uVGA.write(0x00);//object will be drawn solid
                        */
                        /*
                if (sonicVal>30)
                      {
                      //out_uVGA.write(0x45); //clear screen command
                      //val=(byte) in_uVGA.read(); //device replies with 0x06 if command
successfully received


                      out_uVGA.write(0x47); //Draw Triangle- 47hex
                             out_uVGA.write(0x00); //x1
                             out_uVGA.write(0x0A); //x1
                             out_uVGA.write(0x00); //y1
                             out_uVGA.write(0x0A); //y1
                             out_uVGA.write(0x00); //x2
                             out_uVGA.write(0x2A); //x2
                             out_uVGA.write(0x00); //y2
                             out_uVGA.write(0x2A); //y2
                             out_uVGA.write(0x00); //x3
                             out_uVGA.write(0x03); //x3
                             out_uVGA.write(0x00); //y3
                             out_uVGA.write(0x2A); //y3
                             out_uVGA.write(0xFF); //color
                             out_uVGA.write(0xFF); //color

                             val=(byte) in_uVGA.read(); // replies with 0x06 if command
successfully received
                      }
                      else if (sonicVal>20 && sonicVal<=30)
                      {
                             //out_uVGA.write(0x45); //clear screen command
                             //val=(byte) in_uVGA.read(); //device replies with 0x06 if
command successfully received


                             out_uVGA.write(0x43); //Draw Circle- 43hex
                             out_uVGA.write(0x01); //x
                             out_uVGA.write(0x40); //x
                             out_uVGA.write(0x00); //y
                             out_uVGA.write(0x50); //y
                             out_uVGA.write(0x00); //radius
                             out_uVGA.write((byte)sonicVal); //radius
                             out_uVGA.write(0xFF); //color
                             out_uVGA.write(0xFF); //color
                             val=(byte) in_uVGA.read(); // replies with 0x06 if command
successfully received
                      }
                      else if (sonicVal > 10 && sonicVal<=20)
                      {
                             //out_uVGA.write(0x45); //clear screen command
                             //val=(byte) in_uVGA.read(); //device replies with 0x06 if
command successfully received
```

```
                        out_uVGA.write(0x72); //Draw Rectangle-72hex
                        out_uVGA.write(0x00); //x1
                        out_uVGA.write(0x00); //x1
                        out_uVGA.write(0x00); //y1
                        out_uVGA.write(0x00); //y1
                        out_uVGA.write(0x02); //x2
                        out_uVGA.write(0x80); //x2
                        out_uVGA.write(0x01); //y2
                        out_uVGA.write(0xE0); //y2
                        out_uVGA.write(0xAA); //color
                        out_uVGA.write(0x66); //color
                        val=(byte) in_uVGA.read(); // replies with 0x06 if command
successfully received


                    }
                    else if (sonicVal > 0 && sonicVal<=10)
                    {
                        //
                    }
                        */
                        /*
                        else if (sonicVal>0 && sonicVal<25)
                        {
                        String currentDateTimeString =
DateFormat.getDateTimeInstance().format(new Date());

                           char[] currentDateChar = currentDateTimeString.toCharArray();

                     out_uVGA.write(0x73); //Draw “String” of ASCII Text (text format) -
73hex
                     out_uVGA.write(0x01); //column
                     out_uVGA.write(0x01); //row
                     out_uVGA.write(0x03); // 3: 12x16 font
                     out_uVGA.write(0xFF); // ffff white
                     out_uVGA.write(0xFF); // ffff white
                     out_uVGA.write(currentDateChar[0]); // ASCII
                     out_uVGA.write(currentDateChar[1]); // ASCII
                     out_uVGA.write(currentDateChar[2]); // ASCII
                     out_uVGA.write(currentDateChar[3]); // ASCII
                     out_uVGA.write(currentDateChar[4]); // ASCII
                     out_uVGA.write(currentDateChar[5]); // ASCII
                     out_uVGA.write(currentDateChar[6]); // ASCII
                     out_uVGA.write(currentDateChar[7]); // ASCII
                     out_uVGA.write(currentDateChar[8]); // ASCII
                     out_uVGA.write(currentDateChar[9]); // ASCII
                     out_uVGA.write(currentDateChar[10]); // ASCII
                     out_uVGA.write(currentDateChar[11]); // ASCII
                     out_uVGA.write(currentDateChar[12]); // ASCII
                     out_uVGA.write(currentDateChar[13]); // ASCII
                     out_uVGA.write(currentDateChar[14]); // ASCII
                     out_uVGA.write(currentDateChar[15]); // ASCII
                     out_uVGA.write(currentDateChar[16]); // ASCII
                     out_uVGA.write(0x00); // TERMINATOR
                     val=(byte) in_uVGA.read(); //replies with 0x06 if command successfully
received
                        }
                        */

                        //sleep(1);
```

```
                        } catch (InterruptedException e) {
                                // TODO Auto-generated catch block
                                e.printStackTrace();
                        } catch (IOException e) {
                                // TODO Auto-generated catch block
                                e.printStackTrace();
                        }
                }
public void drawAndroid() throws IOException{
    int val=0;


        out_uVGA.write(0x73); //Draw ascii string 73hex
        out_uVGA.write(0x0F); //column
        out_uVGA.write(0x01);// row
        out_uVGA.write(0x03);//font size
        out_uVGA.write(0xFF);// string color
        out_uVGA.write(0xFF);//sting color
        out_uVGA.write(0x57);//w
        out_uVGA.write(0x65);//e
        out_uVGA.write(0x6C);//l
        out_uVGA.write(0x63);//c
        out_uVGA.write(0x6F);//o
        out_uVGA.write(0x6D);//m
        out_uVGA.write(0x65);//e
        out_uVGA.write(0x20);//space
        out_uVGA.write(0x74);//t
        out_uVGA.write(0x6F);//o
        out_uVGA.write(0x20);//space
        out_uVGA.write(0x56);//V
        out_uVGA.write(0x69);//i
        out_uVGA.write(0x72);//r
        out_uVGA.write(0x74);//t
        out_uVGA.write(0x75);//u
        out_uVGA.write(0x61);//a
        out_uVGA.write(0x6C);//l
        out_uVGA.write(0x20);//space
        out_uVGA.write(0x41); //A
        out_uVGA.write(0x6E);//n
        out_uVGA.write(0x64);//d
        out_uVGA.write(0x72);//r
        out_uVGA.write(0x6F);//o
        out_uVGA.write(0x69);//i
        out_uVGA.write(0x64);//d
        out_uVGA.write(0x21);//!
        out_uVGA.write(0x00);//terminator
        val=(byte) in_uVGA.read(); // replies with 0x06 if command successfully received

        out_uVGA.write(0x43); //Draw Circle- 43hex
        out_uVGA.write(0x02); //x
        out_uVGA.write(0x0F); //x
        out_uVGA.write(0x00); //y
        out_uVGA.write(0x0F); //y
        out_uVGA.write(0x00); //radius
        out_uVGA.write(0x05); //radius
        out_uVGA.write(0x0F); //color
        out_uVGA.write(0x0F); //color
        val=(byte) in_uVGA.read(); // replies with 0x06 if command successfully received

        out_uVGA.write(0x72); //Draw Rectangle-72hex
```

```
out_uVGA.write(0x02); //x1
out_uVGA.write(0x09); //x1
out_uVGA.write(0x00); //y1
out_uVGA.write(0x10); //y1
out_uVGA.write(0x02); //x2
out_uVGA.write(0x15); //x2
out_uVGA.write(0x00); //y2
out_uVGA.write(0x1C); //y2
out_uVGA.write(0x0F); //color
out_uVGA.write(0x0F); //color
val=(byte) in_uVGA.read(); // replies with 0x06 if command successfully received

out_uVGA.write(0x4C); // draw line 4C-hex
out_uVGA.write(0x02); //x1
out_uVGA.write(0x09); //x1
out_uVGA.write(0x00); //y1
out_uVGA.write(0x10); //y1
out_uVGA.write(0x02); //x2
out_uVGA.write(0x15); //x2
out_uVGA.write(0x00); //y2
out_uVGA.write(0x10); //y2
out_uVGA.write(0x00); //color black
out_uVGA.write(0x00); //color black
val=(byte) in_uVGA.read(); // replies with 0x06 if command successfully received

out_uVGA.write(0x4C); // draw line 4C-hex
out_uVGA.write(0x02); //x1
out_uVGA.write(0x10); //x1
out_uVGA.write(0x00); //y1
out_uVGA.write(0x0F); //y1
out_uVGA.write(0x02); //x2
out_uVGA.write(0x12); //x2
out_uVGA.write(0x00); //y2
out_uVGA.write(0x06); //y2
out_uVGA.write(0x0F); //color green
out_uVGA.write(0x0F); //color green
val=(byte) in_uVGA.read(); // replies with 0x06 if command successfully received
out_uVGA.write(0x4C); // draw line 4C-hex
out_uVGA.write(0x02); //x1
out_uVGA.write(0x0E); //x1
out_uVGA.write(0x00); //y1
out_uVGA.write(0x0F); //y1
out_uVGA.write(0x02); //x2
out_uVGA.write(0x0C); //x2
out_uVGA.write(0x00); //y2
out_uVGA.write(0x06); //y2
out_uVGA.write(0x0F); //color green
out_uVGA.write(0x0F); //color green
val=(byte) in_uVGA.read(); // replies with 0x06 if command successfully received
//right foot
out_uVGA.write(0x43); //Draw Circle- 43hex
out_uVGA.write(0x02); //x
out_uVGA.write(0x12); //x
out_uVGA.write(0x00); //y
out_uVGA.write(0x22); //y
out_uVGA.write(0x00); //radius
out_uVGA.write(0x01); //radius
out_uVGA.write(0x0F); //color
out_uVGA.write(0x0F); //color
val=(byte) in_uVGA.read(); // replies with 0x06 if command successfully received
```

```
//left foot
out_uVGA.write(0x43); //Draw Circle- 43hex
out_uVGA.write(0x02); //x
out_uVGA.write(0x0C); //x
out_uVGA.write(0x00); //y
out_uVGA.write(0x22); //y
out_uVGA.write(0x00); //radius
out_uVGA.write(0x01); //radius
out_uVGA.write(0x0F); //color
out_uVGA.write(0x0F); //color
val=(byte) in_uVGA.read(); // replies with 0x06 if command successfully received
//left leg
out_uVGA.write(0x72); //Draw Rectangle-72hex
out_uVGA.write(0x02); //x1
out_uVGA.write(0x0B); //x1
out_uVGA.write(0x00); //y1
out_uVGA.write(0x17); //y1
out_uVGA.write(0x02); //x2
out_uVGA.write(0x0D); //x2
out_uVGA.write(0x00); //y2
out_uVGA.write(0x22); //y2
out_uVGA.write(0x0F); //color
out_uVGA.write(0x0F); //color
val=(byte) in_uVGA.read(); // replies with 0x06 if command successfully received

//right leg
out_uVGA.write(0x72); //Draw Rectangle-72hex
out_uVGA.write(0x02); //x1
out_uVGA.write(0x11); //x1
out_uVGA.write(0x00); //y1
out_uVGA.write(0x17); //y1
out_uVGA.write(0x02); //x2
out_uVGA.write(0x13); //x2
out_uVGA.write(0x00); //y2
out_uVGA.write(0x22); //y2
out_uVGA.write(0x0F); //color
out_uVGA.write(0x0F); //color
val=(byte) in_uVGA.read(); // replies with 0x06 if command successfully received


//right hand
out_uVGA.write(0x43); //Draw Circle- 43hex
out_uVGA.write(0x02); //x
out_uVGA.write(0x18); //x  +7
out_uVGA.write(0x00); //y
out_uVGA.write(0x1B); //y -8
out_uVGA.write(0x00); //radius
out_uVGA.write(0x01); //radius
out_uVGA.write(0x0F); //color
out_uVGA.write(0x0F); //color
val=(byte) in_uVGA.read(); // replies with 0x06 if command successfully received
//left hand
out_uVGA.write(0x43); //Draw Circle- 43hex
out_uVGA.write(0x02); //x
out_uVGA.write(0x06); //x -7
out_uVGA.write(0x00); //y
out_uVGA.write(0x1B); //y  -8
out_uVGA.write(0x00); //radius
out_uVGA.write(0x01); //radius
out_uVGA.write(0x0F); //color
```

```
		out_uVGA.write(0x0F); //color
		val=(byte) in_uVGA.read(); // replies with 0x06 if command successfully received
		//left arm
		out_uVGA.write(0x72); //Draw Rectangle-72hex
		out_uVGA.write(0x02); //x1
		out_uVGA.write(0x05); //x1 -7
		out_uVGA.write(0x00); //y1
		out_uVGA.write(0x13); //y1
		out_uVGA.write(0x02); //x2
		out_uVGA.write(0x07); //x2 -7
		out_uVGA.write(0x00); //y2
		out_uVGA.write(0x1B); //y2 -8
		out_uVGA.write(0x0F); //color
		out_uVGA.write(0x0F); //color
		val=(byte) in_uVGA.read(); // replies with 0x06 if command successfully received
		//right arm
		out_uVGA.write(0x72); //Draw Rectangle-72hex
		out_uVGA.write(0x02); //x1
		out_uVGA.write(0x17); //x1 +7
		out_uVGA.write(0x00); //y1
		out_uVGA.write(0x13); //y1 was 0f
		out_uVGA.write(0x02); //x2
		out_uVGA.write(0x19); //x2 +7
		out_uVGA.write(0x00); //y2
		out_uVGA.write(0x1B); //y2 -8
		out_uVGA.write(0x0F); //color
		out_uVGA.write(0x0F); //color
		val=(byte) in_uVGA.read(); // replies with 0x06 if command successfully received
}
}



		@Override
		protected AbstractIOIOActivity.IOIOThread createIOIOThread() {
			return new IOIOThread();
		}

}
```