Computer Vision with Google Glass

New Mexico Supercomputing Challenge

Final Report

April 1, 2015

Team 51
Los Alamos High School

Team Members:
Sudeep Dasari
Colin Redman

Teachers:
Adam Drew

Executive Summary

We believe that given its unique form factor, Glass has great potential in uses for communication that had not been previously explored. With that in mind, the goal of our project was to integrate the portability and convenience of Google Glass with the substantially more powerful computational capabilities of modern Android smartphones in order to allow for the analysis and recognition of human gestures in real space. The dual purposes of this project were accomplished using Bluetooth communication and the OpenCV library.

Overall, the project was a partial success. Our image analysis was able to reliably and efficiently detect certain gestures, which makes it a promising solution to our problem. In addition, we were able to complete the Bluetooth communication software that we needed in order to transmit image data between devices.

Unfortunately, Google Glass was also commercially discontinued during the course of our project, which severely limited the potential applications of our work. In addition, unexpected difficulties with OpenCV on our android phone prevented us from completing the integration of image analysis on our phone side of the application, leaving us with a partially incomplete system.

Problem Statement

Over the past years, innovations in computing have been continuously shrinking the size and increasing the power of portable computing. Most people now carry incredible computing power in their pockets, and even, now, sometimes on their heads. Google Glass is a relatively recent creation that, when it was first announced, pushed the worlds imagination to a new level. Unfortunately, as the initial awe began to fade, the lack of actual concrete applications for hardware such as Glass became more apparent, and eventually contributed to it being discontinued. Beyond image/video capture and very specific instances, Glass, despite its fantastically complex and interesting engineering, lacked a real purpose to fill.

With this project, we wanted to provide, at the very least, a proof of concept that proved a general use and therefore justification for Glass as a product. Our goal was to use Glass to portably capture hand gestures and eventually translate those gestures (for example in sign language) into something that the wearer would be able to understand so that we could contribute to greater ease in communication. The other part of the problem is that Glass itself has insufficient computation onboard to analyze the images on its own, so we wanted to create a way to send the images to a paired phone (which would be nearby) so that they could be processed there.

Problem Solution

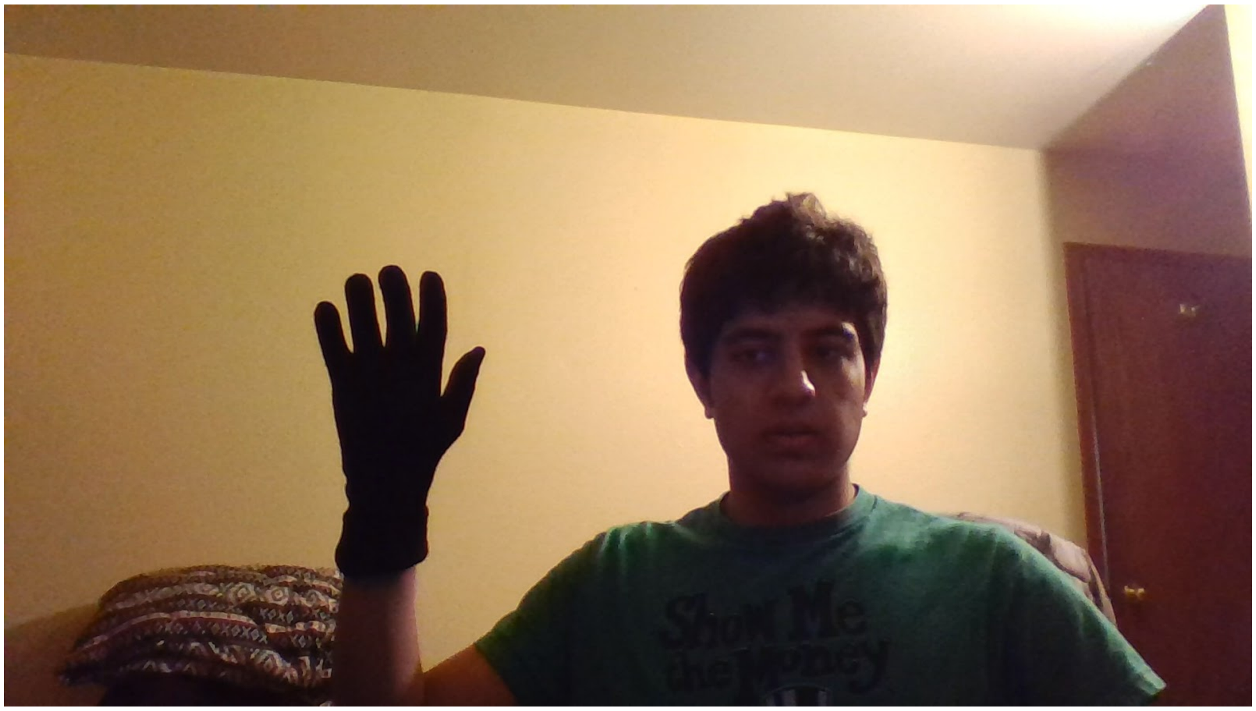
The image processing segment of code had rather simple, but extraordinarily hard to execute, objectives. Given a picture a code must be written that can extract useful information from that picture. In this case the problem was two fold. The code must extract hands from within images and then detect the gesture the hands are presently holding. In order to simplify the task of finding hands from complex backgrounds, we imposed upon the user the burden of wearing black gloves. While this isn’t as convenient as originally intended it allows us to significantly simplify the task of finding the correct object from a scene. Rather than teach a computer to recognize hands from other objects, say buses, the computer can use a simple threshold test to find black objects within a picture. Thanks to the OpenCV library methods already exist that can threshold images, and then extract shapes - or contours - from those images. The processing code would take in each image, threshold them to separate the black gloved hand from other parts of the image scene, and then extract the shape of the black hand from the image.

Once the contour data has been sucessfully extracted from the image, shape recognition methods can be used to determine the specific shape made by the contour. Images were collected of clasped hands and open hands (all gloved of course), and those images had contours extracted from them, before any other images were given to the program. These original images act as training data to teach the computer what closed and open hands look like. Once a new image was fed to the program, shape and contour data was extracted from it as well. The largest contour (theoretically the hand) of the new image, was then compared with the largest contour of the training images. Hu moments - a scale and translation invariant method of shape comparison in OpenCV - were used to conduct the contour comparison. Once the contours were compared the new image was determined to be either “close” or “open” depending on its closest match. For example, if the new image matched closest with the training image containing a closed hand it was predicted by the program to have a closed hand in it. Thus, the code is able to make predictions of the state of the hand detected in each image (open/closed) which could then be extended to other more numerous gestures. Admittedly, there is much to desire in terms of accuracy and robustness of the Hu/contour method of shape comparison, but it is very easy and computationally inexpensive to implement. This is an important feature because the computations are intended to be performed on phone hardware, which is not reasonably capable of complex analysis.

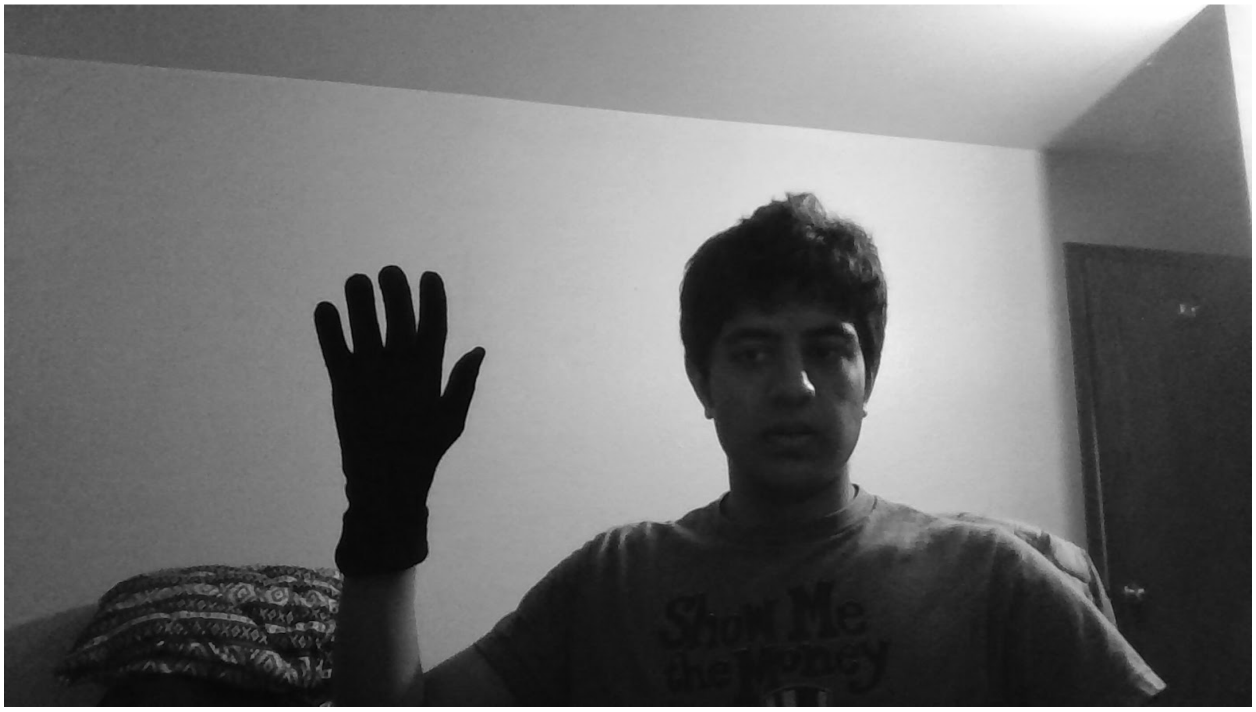
Before the images can be analyzed, however, they must be first captured by Glass, and then communicated to a phone. This is performed by a Bluetooth connection, where the headset runs a simple server application that the phone app connects to. While a connection is formed or being formed, an Android app running on the headset listens for a user tap on the Glass controls. When this happens, a pictures is captured which is then confirmed or denied by the user. Images are then taken as bitmaps, and streamed from Glass to the phone, where it is reconstructed from the byte stream back into an image.

Results

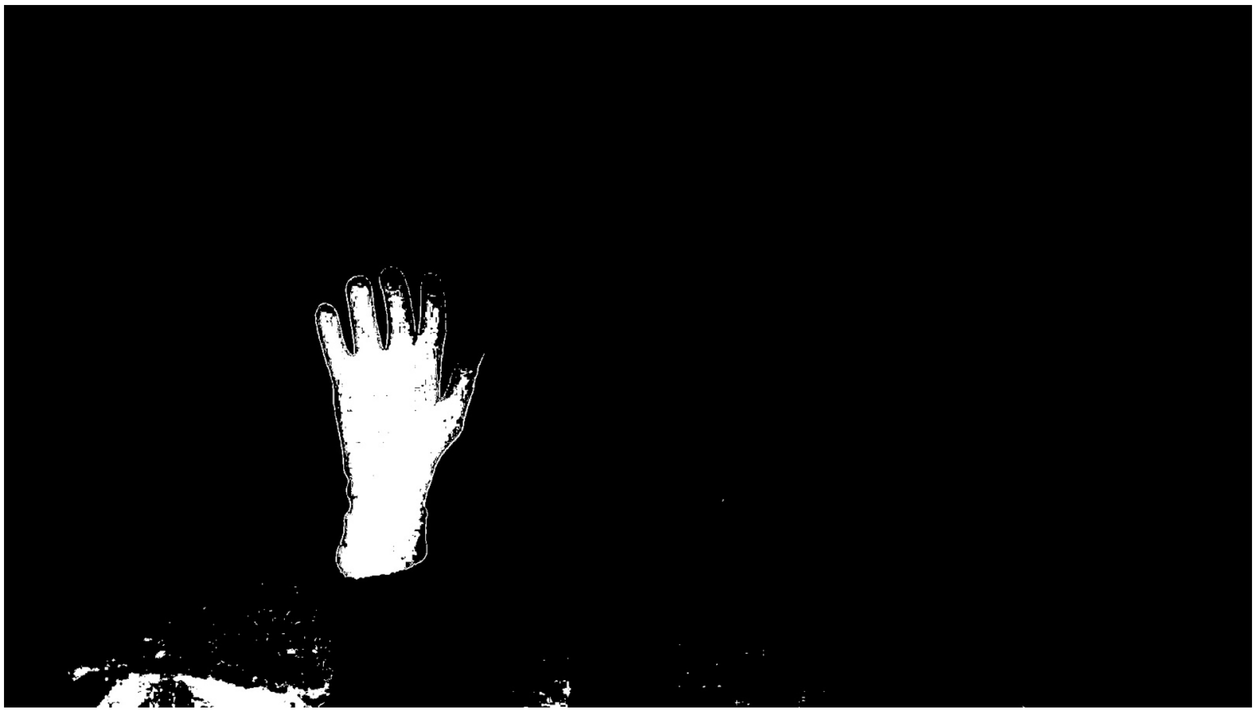
The image processing code has visual results that show the process that an image takes from raw form to extracted hand. Take for example the below image of Sudeep holding up his gloved hand..



The image is first converted to greyscale.

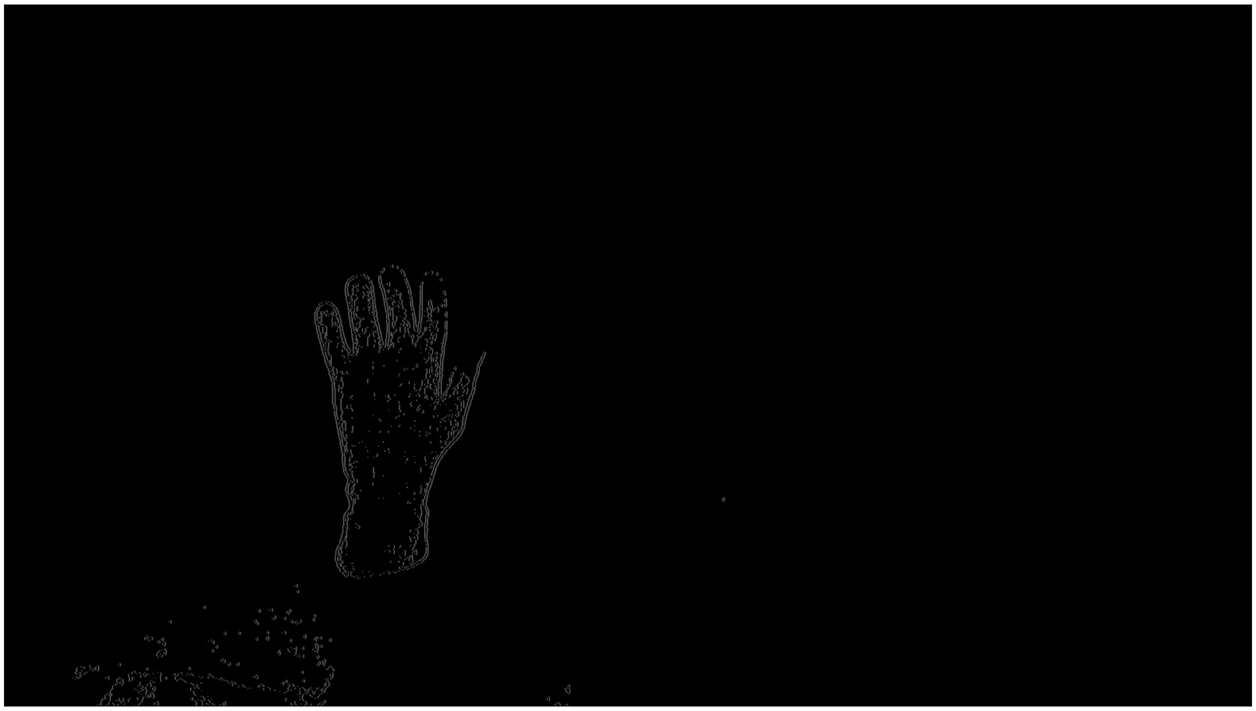


The grayscale image is then thresholded so that only the darkest parts of the image (and thus the glove) are kept as white dots and the rest of the image is discarded.



Once the threshold is complete the vague filling of the hand is the only notable object in the picture as the glove should ensure. It is notable that black background “noise” can interfere with this process.

Once the threshold is complete the image is run through a Canny edge filter to convert it from a filled hand to a shaped outline, which can easily be turned into a shape contour.



These images offer a nice visual representation of what an image goes through during its journey from full scene to hand outline. Once in hand outline form the code will compare the outline to outlines of known closed and known open hands given to it. The code will then try to predict if outlines it extracted are open or close. Thus far it has managed to correctly predict about 70% of the hands fed to it, although much more testing is needed to get a useful measure of accuracy(we only were able to run on the scale of six trials). It is notable that errors in extracting the outline can compound with statistical errors in the Hu comparison method to make the accuracy of the prediction system decrease.

Conclusion

Our project was split into two separate segments that we hope to unify before the expo. One branch of the project is a framework to allow images to be transmitted from Glass to the phone. The other part can process images fed to it to extract hands from the images, and then make predictions as to the gesture of the hand. The two branches must be unified before this project can truly be demoed, but since OpenCV exists on Android unification should be a trivial step. Much less trivial is the necessary work to reach true computer proficiency in vision. Currently image processing requires the users to wear gloves, does not have a very robust method to determine hand from surrounding even with gloves (black background can confuse it), and the gesture system could be vastly improved using current machine learning methods. The reasoning for using more simplistic - albeit rudimentary - methods of analysis is their ease of implementation and low computational cost. As our framework matures, and as phones increase in power both concerns will diminish over time. In order to truly live up to the goal of true computer vision, not only must more advanced analytical tools be deployed, but better and more supported hardware must be deployed as well. Unfortunately, Google discontinued work on the device during the span of our project. It makes sense, because ultimately the Glass was an immature piece of hardware. It mostly functioned as a head mounted camera and display system with little to no real world application, and bad engineering overall. This does not mean augmented reality devices have no promise. This means that in order for computer vision to truly succeed, better and more thought through hardware must also exist.

Acknowledgements

First and foremost we'd like to thank the New Mexico Supercomputing Challenge for making this opportunity possible. Without the help of consult, the interim report reviewers, and the March evaluation judges among many others this forum for problem solving would simply not exist. Next we would like to thank Noah Caulfield and Dmitri Malygin - Voorhees. Their project from last year *ASL Detection for Practical Inspiration,* served as the original inspiration for this project, and they've been supportive of us attempting to continue their work.

Bibliography

Chen, Feng-Sheng, Chih-Ming Fu, and Chung-Lin Huang. "Hand gesture recognition using a real-time tracking method and hidden Markov models."Image and vision computing 21.8 (2003): 745-758.

Caulfield, Noah, and Dmitri Malygin-Voorhees. ASL Detection for Practical Implementation. Tech. NM: Supercomputing Challenge, 2014. Web.

Freeman, William T., and Michal Roth. "Orientation histograms for hand gesture recognition." International workshop on automatic face and gesture recognition. Vol. 12. 1995.

Google Developer's Guide. Google, nd. Web.

Waggoner, Nathaniel. GoogleGlassBlutooth. Github, 2014. Web.

Code

Hand Detection

HandDetect.java

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

import org.opencv.core.Core;
import org.opencv.core.Mat;
import org.opencv.core.MatOfRect;
import org.opencv.core.Point;
import org.opencv.core.Rect;
import org.opencv.core.Scalar;
import org.opencv.imgcodecs.Imgcodecs;
import org.opencv.imgproc.Imgproc;
import org.opencv.objdetect.CascadeClassifier;
import org.opencv.core.MatOfPoint;

//feed code an image, get out hand picture
public class HandDetect {
	public static void main(String args[]){

		System.loadLibrary(Core.NATIVE_LIBRARY_NAME);

		Mat image = Imgcodecs.imread("data/open3.jpg");
		Mat output = new Mat();


		MatOfPoint contourInsert = handDetect(image, output);


		Mat fill  = new Mat();
		//open
		Mat open = Imgcodecs.imread("data/open.jpg");
		MatOfPoint contourOpen = handDetect(open, fill);
		//close
		Mat close = Imgcodecs.imread("data/close.jpg");
		MatOfPoint contourClose = handDetect(close, fill);

		//compare
		double openCheck = Imgproc.matchShapes(contourInsert, contourOpen,
Imgproc.CV_CONTOURS_MATCH_I1, 0);
		double closeCheck = Imgproc.matchShapes(contourInsert, contourClose,
Imgproc.CV_CONTOURS_MATCH_I1, 0);

		System.out.println(openCheck +" "+closeCheck);
```

```
                    if(openCheck<closeCheck)
                              System.out.println("Open");
                    else if (openCheck == closeCheck)
                              System.out.println("broken comparison");
                    else
                              System.out.println("Close");


                    Imgcodecs.imwrite("output.jpg", output);
          }
          public static MatOfPoint handDetect (Mat input, Mat output){
                    Imgproc.cvtColor(input, output, Imgproc.COLOR_RGB2GRAY);
                    Imgproc.threshold(output, output, 12, 255, Imgproc.THRESH_BINARY_INV);
                    Imgproc.adaptiveThreshold(output, output, 200,
Imgproc.ADAPTIVE_THRESH_GAUSSIAN_C, Imgproc.THRESH_BINARY_INV, 5, 2);

                    //Imgproc.Canny(output, output, 0, 255);


                    List<MatOfPoint> contours = new ArrayList<>();
                    Imgproc.findContours(output, contours, new Mat(), Imgproc.RETR_LIST,
Imgproc.CHAIN_APPROX_SIMPLE);
                    output = new Mat();
                    int indexLargest = 0;
                    double areaLargest=0;
                    for(int i=0;i<contours.size();i++){
                              double d = Imgproc.contourArea(contours.get(i));
                              if(d>areaLargest){
                                        areaLargest = d;
                                        indexLargest = i;
                              }
                    }
                    System.out.println(indexLargest+ " "+ areaLargest);


      Imgproc.drawContours(output, contours, indexLargest, new Scalar(255,255,255));
      return contours.get(indexLargest);

          }
}
```

Phone Code

AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
   package="com.glassbluetooth.phonebluetooth" >
   <uses-permission android:name="android.permission.BLUETOOTH" />
```

```
    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >
        <activity
            android:name=".BlueToothActivity"
            android:label="@string/app_name" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>

</manifest>
```

BluetoothActivity.java

```
package com.glassbluetooth.phonebluetooth;

import android.bluetooth.BluetoothAdapter;
import android.bluetooth.BluetoothDevice;
import android.bluetooth.BluetoothSocket;
import android.graphics.Bitmap;
import android.graphics.BitmapFactory;
import android.os.ParcelUuid;
import android.support.v7.app.ActionBarActivity;
import android.os.Bundle;
import android.util.Log;
import android.view.Menu;
import android.view.MenuItem;

import java.io.ByteArrayInputStream;
import java.io.ByteArrayOutputStream;
import java.io.IOException;
import java.io.InputStream;
import java.nio.ByteBuffer;
import java.util.ArrayList;
import java.util.Set;


public class BlueToothActivity extends ActionBarActivity {

    protected BluetoothAdapter bluetoothAdapter;
protected BluetoothSocket mSocket;

protected BluetoothDevice pairedGlass;

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

    bluetoothAdapter = BluetoothAdapter.getDefaultAdapter();

    Set<BluetoothDevice> pairedDevices = bluetoothAdapter.getBondedDevices();

    for (BluetoothDevice pairedDevice:pairedDevices){
        pairedGlass = pairedDevice;
    }


    ParcelUuid[] uuids = pairedGlass.getUuids();
    try {
        mSocket = pairedGlass.createInsecureRfcommSocketToServiceRecord(uuids[0].getUuid());
        mSocket.connect();
    } catch (IOException e) {
        e.printStackTrace();
    }
    if (mSocket.isConnected()){
        Log.e("ClientConnector", "WORKED");
        InputStream inputStream = null;
        try {
            inputStream = mSocket.getInputStream();
        } catch (IOException e) {
            e.printStackTrace();
        }
        ArrayList<byte[]> buffers = new ArrayList<byte[]>();
        while (true){
            byte[] byteBuffer = null;
            try {
                ByteArrayOutputStream buffer = new ByteArrayOutputStream();

                int nRead;
                byte[] data = new byte[16384];

                while ((nRead = inputStream.read(data, 0, data.length)) != -1) {
                    buffer.write(data, 0, nRead);
                }

                buffer.flush();
                byteBuffer = buffer.toByteArray();
            } catch (IOException e) {
                e.printStackTrace();
            }

            Bitmap bitmap = BitmapFactory.decodeByteArray(byteBuffer,0,byteBuffer.length);
        }
    }
}


@Override
public boolean onCreateOptionsMenu(Menu menu) {
    // Inflate the menu; this adds items to the action bar if it is present.
    getMenuInflater().inflate(R.menu.menu_blue_tooth, menu);
    return true;
}

@Override
public boolean onOptionsItemSelected(MenuItem item) {
    // Handle action bar item clicks here. The action bar will
    // automatically handle clicks on the Home/Up button, so long
    // as you specify a parent activity in AndroidManifest.xml.
    int id = item.getItemId();

    //noinspection SimplifiableIfStatement
    if (id == R.id.action_settings) {
        return true;
    }

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

Glass Code

AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.glassbluetooth.glassbluetooth" >

    <uses-permission android:name="android.permission.BLUETOOTH" />
    <uses-permission android:name="android.permission.CAMERA" />
    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
    <uses-feature android:name="android.hardware.camera" />
    <uses-feature android:name="android.hardware.camera.autofocus" />
    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >
        <service
            android:name=".HandDetectionBluetooth"
            android:icon="@drawable/ic_glass_logo"
            android:label="@string/title_activity_hand_detection_bluetooth" >
            <intent-filter>
                <action android:name="com.google.android.glass.action.VOICE_TRIGGER" />
            </intent-filter>

            <meta-data
                android:name="com.google.android.glass.VoiceTrigger"
                android:resource="@xml/voice_trigger" />
        </service>

        <activity
            android:name=".BluetoothActivity"
            android:configChanges="orientation|keyboardHidden|screenSize"
            android:label="@string/title_activity_bluetooth_display"
            android:theme="@style/FullscreenTheme" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>

    </application>

</manifest>
```

HandDetectionBluetooth.java

```
package com.glassbluetooth.glassbluetooth;

import android.app.PendingIntent;
import android.app.Service;
import android.content.Intent;
import android.graphics.Bitmap;
import android.os.IBinder;

import com.google.android.glass.timeline.LiveCard;
import com.google.android.glass.timeline.LiveCard.PublishMode;

/**
 * A {@link Service} that publishes a {@link LiveCard} in the timeline.
 */
public class HandDetectionBluetooth extends Service {
```

```
private static final String LIVE_CARD_TAG = "HandDetectionBluetooth";


// there can only be one
private static LiveCard mLiveCard;

@Override
public IBinder onBind(Intent intent) {
    return null;
}


public static String currentLetter = "a";

public static Bitmap bitmap;

public static LiveCard getLiveCard (){
    return mLiveCard;
}

@Override
public int onStartCommand(Intent intent, int flags, int startId) {
    if (mLiveCard == null) {
        mLiveCard = new LiveCard(this, LIVE_CARD_TAG);

        LiveCardRenderer renderer = new LiveCardRenderer(this);
        mLiveCard.setDirectRenderingEnabled(true).getSurfaceHolder().addCallback(renderer);

        // Display the options menu when the live card is tapped.
        Intent menuIntent = new Intent(this, LiveCardMenuActivity.class);
        mLiveCard.setAction(PendingIntent.getActivity(this, 0, menuIntent, 0));
        mLiveCard.attach(this);
        mLiveCard.publish(PublishMode.REVEAL);
    } else {
        mLiveCard.navigate();
    }



    return START_STICKY;
}



@Override
public void onDestroy() {
    if (mLiveCard != null && mLiveCard.isPublished()) {
            mLiveCard.unpublish();
            mLiveCard = null;
        }
        super.onDestroy();
    }
}
```

BluetoothActivity.java

```
package com.glassbluetooth.glassbluetooth;

import android.annotation.TargetApi;
import android.app.Activity;
import android.app.Service;
import android.bluetooth.BluetoothAdapter;
import android.bluetooth.BluetoothDevice;
import android.bluetooth.BluetoothServerSocket;
import android.bluetooth.BluetoothSocket;
import android.content.Context;
import android.content.Intent;
import android.graphics.Bitmap;
import android.graphics.Canvas;
import android.graphics.Rect;
import android.hardware.Camera;
import android.os.Build;
import android.os.Bundle;
import android.os.Environment;
import android.os.Handler;
import android.os.ParcelUuid;
import android.provider.MediaStore;
import android.util.Log;
import android.view.MotionEvent;
import android.view.Surface;
import android.view.SurfaceHolder;
import android.view.View;

import com.glassbluetooth.glassbluetooth.util.SystemUiHider;
import com.google.android.glass.touchpad.Gesture;
import com.google.android.glass.touchpad.GestureDetector;

import java.io.File;
import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
import java.lang.reflect.InvocationTargetException;
import java.lang.reflect.Method;
import java.net.URI;
import java.net.URISyntaxException;
import java.nio.Buffer;
import java.nio.ByteBuffer;
import java.util.Set;
import java.util.UUID;


/**
 * An example full-screen activity that shows and hides the system UI (i.e.
 * status bar and navigation/system bar) with user interaction.
 *
 * @see SystemUiHider
 */
public class BluetoothActivity extends Activity {
    /**
     * Whether or not the system UI should be auto-hidden after
     * {@link #AUTO_HIDE_DELAY_MILLIS} milliseconds.
     */
    private static final boolean AUTO_HIDE = true;

    /**
     * If {@link #AUTO_HIDE} is set, the number of milliseconds to wait after
     * user interaction before hiding the system UI.
     */
    private static final int AUTO_HIDE_DELAY_MILLIS = 3000;

    /**
     * If set, will toggle the system UI visibility upon interaction. Otherwise,
     * will show the system UI visibility upon interaction.
     */
    private static final boolean TOGGLE_ON_CLICK = true;

    /**
     * The flags to pass to {@link SystemUiHider#getInstance}.
     */
    private static final int HIDER_FLAGS = SystemUiHider.FLAG_HIDE_NAVIGATION;

    /**
     * The instance of the {@link SystemUiHider} for this activity.
     */
    private SystemUiHider mSystemUiHider;

    private HandDetectionBluetooth mService;

    protected BluetoothAdapter mBluetooth;

    protected BluetoothDevice pairedPhone;

    protected BluetoothServerSocket mServerSocket;
    protected BluetoothSocket mSocket;

    protected GestureDetector mTouchDetector;

    protected OutputStream outputStream;
    protected InputStream inputStream;

    // public static final String UUID = "da8cf13c-cd18-4d4e-a683-1b1c2f325b50";

    @Override
    protected void onResume(){
        super.onResume();
        if (HandDetectionBluetooth.getLiveCard()!=null) {
            HandDetectionBluetooth.getLiveCard().navigate();
        }
    }

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
//        setContentView(R.layout.activity_bluetooth_display);
//
//
//        final View controlsView = findViewById(R.id.fullscreen_content_controls);
//        final View contentView = findViewById(R.id.fullscreen_content);
//        // Set up an instance of SystemUiHider to control the system UI for
//        // this activity.
//        mSystemUiHider = SystemUiHider.getInstance(this, contentView, HIDER_FLAGS);
//        mSystemUiHider.setup();
//        mSystemUiHider
//                .setOnVisibilityChangeListener(new SystemUiHider.OnVisibilityChangeListener() {
//                    // Cached values.
//                    int mControlsHeight;
//                    int mShortAnimTime;
//
//                    @Override
//                    @TargetApi(Build.VERSION_CODES.HONEYCOMB_MR2)
//                    public void onVisibilityChange(boolean visible) {
//                        if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.HONEYCOMB_MR2) {
//                            // If the ViewPropertyAnimator API is available
//                            // (Honeycomb MR2 and later), use it to animate the
//                            // in-layout UI controls at the bottom of the
//                            // screen.
//                            if (mControlsHeight == 0) {
//                                mControlsHeight = controlsView.getHeight();
//                            }
//                            if (mShortAnimTime == 0) {
```

```
//                              mShortAnimTime = getResources().getInteger(
//                                      android.R.integer.config_shortAnimTime);
//                          }
//                          controlsView.animate()
//                                  .translationY(visible ? 0 : mControlsHeight)
//                                  .setDuration(mShortAnimTime);
//                      } else {
//                          // If the ViewPropertyAnimator APIs aren't
//                          // available, simply show or hide the in-layout UI
//                          // controls.
//                          controlsView.setVisibility(visible ? View.VISIBLE : View.GONE);
//                      }
//
//                      if (visible && AUTO_HIDE) {
//                          // Schedule a hide().
//                          delayedHide(AUTO_HIDE_DELAY_MILLIS);
//                      }
//                  }
//              });
//
//      // Set up the user interaction to manually show or hide the system UI.
//      contentView.setOnClickListener(new View.OnClickListener() {
//          @Override
//          public void onClick(View view) {
//              if (TOGGLE_ON_CLICK) {
//                  mSystemUiHider.toggle();
//              } else {
//                  mSystemUiHider.show();
//              }
//          }
//      });
//
//      // Upon interacting with UI controls, delay any scheduled hide()
//      // operations to prevent the jarring behavior of controls going away
//      // while interacting with the UI.
//      findViewById(R.id.dummy_button).setOnTouchListener(mDelayHideTouchListener);

        Intent serviceIntent = new Intent(this, HandDetectionBluetooth.class);
        startService(serviceIntent);

        mBluetooth = BluetoothAdapter.getDefaultAdapter();

        Set<BluetoothDevice> pairedDevices = mBluetooth.getBondedDevices();
        // If there are paired devices
        if (pairedDevices.size() > 0) {
            // Loop through paired devices
            for (BluetoothDevice device : pairedDevices) {
                pairedPhone = device;
                // Add the name and address to an array adapter to show in a ListView
                //              btArray.add(device.getName() + "\n" + device.getAddress());
                //              update();
            }
        }

        try {
            ParcelUuid[] uuids = pairedPhone.getUuids();
            mServerSocket = mBluetooth.listenUsingInsecureRfcommWithServiceRecord("Glass 0009",
uuids[0].getUuid());
        } catch (IOException e) {
            e.printStackTrace();
        }


        ConnectThread connect = new ConnectThread();
        connect.start();
        HandDetectionBluetooth.currentLetter = "success";
    }


    protected class ConnectThread extends Thread {

        Camera camera;

        public void run() {
            try {
                Thread.sleep(1000);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
            while (true){
                Intent intent = new Intent(MediaStore.ACTION_IMAGE_CAPTURE);
                File photo = new File(new File("/DCIM/Camera/"),  "Pic.jpg");
                URI fileURI = null;
                try {
                    fileURI = new URI("test.jpg");
                } catch (URISyntaxException e) {
                    e.printStackTrace();
                }
                Log.e("Path", photo.getPath());
                intent.putExtra(MediaStore.EXTRA_OUTPUT, photo.toURI()); // set the image file name
                // start the image capture Intent
                startActivityForResult(intent, 1);

            while (!photo.exists()){
                try {
                    Thread.sleep(15);
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
            }


                //

//          camera = Camera.open();
//          Camera.Parameters parameters = camera.getParameters();
//          try {
//              SurfaceHolder holder = new SurfaceHolder() {
//                  @Override
//                  public void addCallback(Callback callback) {
//
//                  }
//
//                  @Override
//                  public void removeCallback(Callback callback) {
//
//                  }
//
//                  @Override
//                  public boolean isCreating() {
//                      return false;
//                  }
//
//                  @Override
//                  public void setType(int type) {
//
//                  }
//
//                  @Override
//                  public void setFixedSize(int width, int height) {
//
//                  }
//
//                  @Override
//                  public void setSizeFromLayout() {
//
//                  }
//
//                  @Override
//                  public void setFormat(int format) {
//
//                  }
//
```

```
//                  @Override
//                  public void setKeepScreenOn(boolean screenOn) {
//
//                  }
//
//                  @Override
//                  public Canvas lockCanvas() {
//                      return null;
//                  }
//
//                  @Override
//                  public Canvas lockCanvas(Rect dirty) {
//                      return null;
//                  }
//
//                  @Override
//                  public void unlockCanvasAndPost(Canvas canvas) {
//
//                  }
//
//                  @Override
//                  public Rect getSurfaceFrame() {
//                      return null;
//                  }
//
//                  @Override
//                  public Surface getSurface() {
//                      return null;
//                  }
//              };
//              camera.setPreviewDisplay(holder);
//              camera.startPreview();
//              HandDetectionBluetooth.currentLetter = "preview?";
//              Camera.PictureCallback jpegCallback = new Camera.PictureCallback() {
//                  public boolean pictureTaken = false;
//                  @Override
//                  public void onPictureTaken(byte[] data, Camera camera) {
//                      pictureTaken= true;
//                  }
//              };
//
//          } catch (IOException e) {
//              e.printStackTrace();
//          }

            BluetoothSocket tmp;
            Log.e("Connect Thread", "Trying to connect");
            if (mSocket == null) {
                try {
//                      tmp = 
pairedPhone.createInsecureRfcommSocketToServiceRecord(java.util.UUID.fromString(UUID));
//                      try {
//                          Thread.sleep(200);
//                      } catch (InterruptedException e) {
//                          e.printStackTrace();
//                      }
//                      Class<?> clazz = tmp.getRemoteDevice().getClass();
//                      Class<?>[] paramTypes = new Class<?>[]{Integer.TYPE};
//                      Method m = clazz.getMethod("createRfcommSocket", paramTypes);
//                      Object[] params = new Object[]{Integer.valueOf(1)};
//                      mSocket = (BluetoothSocket) m.invoke(tmp.getRemoteDevice(), params);
//                      try {
//                          Thread.sleep(200);
//                      } catch (InterruptedException e) {
//                          e.printStackTrace();
//                      }
//
//                      if (mSocket.isConnected()) {
//
//                      }
//                      HandDetectionBluetooth.currentLetter = mSocket.getRemoteDevice().getName();
//                      mSocket = tmp;
//                      outputStream = mSocket.getOutputStream();
//                      inputStream = mSocket.getInputStream();

                    mSocket = mServerSocket.accept();
                    outputStream = mSocket.getOutputStream();
                } catch (IOException e) {
                    e.printStackTrace();
                }
            }else{
                Log.e("Connect Thread", "Connection found");
            }
                Bitmap bitmap = (Bitmap) intent.getExtras().get("data");
                ByteBuffer buffer = ByteBuffer.allocate(bitmap.getByteCount());
                bitmap.copyPixelsToBuffer(buffer);
            try {
                outputStream.write(buffer.array());
                outputStream.flush();
            } catch (IOException e) {
                e.printStackTrace();
                try {
                    mSocket.close();
                    outputStream.close();
                } catch (IOException e1) {
                    e1.printStackTrace();
                }
            }
                LiveCardRenderer.tapped = false;
                while (!LiveCardRenderer.tapped) {
                    try {
                        Thread.sleep(30);
                    } catch (InterruptedException e) {
                        e.printStackTrace();
                    }
                }
            }
        }

    }


//    Runnable mHideRunnable = new Runnable() {
//        @Override
//        public void run() {
//            mSystemUiHider.hide();
//        }
//    };


}
```

LiveCardMenuActivity.java

```
package com.glassbluetooth.glassbluetooth;

import android.app.Activity;
import android.content.Intent;
import android.view.Menu;
import android.view.MenuItem;

/**
 * A transparent {@link Activity} displaying a "Stop" options menu to remove the {@link LiveCard}.
 */
public class LiveCardMenuActivity extends Activity {

    @Override
    public void onAttachedToWindow() {
        super.onAttachedToWindow();
        // Open the options menu right away.
        openOptionsMenu();
    }

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        getMenuInflater().inflate(R.menu.hand_detection_bluetooth, menu);
        return true;
    }

    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
        switch (item.getItemId()) {
            case R.id.action_stop:
                // Stop the service which will unpublish the live card.
                stopService(new Intent(this, HandDetectionBluetooth.class));
                return true;
            default:
                return super.onOptionsItemSelected(item);
        }
    }

    @Override
    public void onOptionsMenuClosed(Menu menu) {
        super.onOptionsMenuClosed(menu);
        // Nothing else to do, finish the Activity.
        finish();
    }
}
```

LiveCardRenderer.java

```
package com.glassbluetooth.glassbluetooth;

import com.google.android.glass.timeline.DirectRenderingCallback;

import android.app.Activity;
import android.content.Context;
import android.graphics.Bitmap;
import android.graphics.Canvas;
import android.graphics.Color;
import android.graphics.Paint;
import android.graphics.PorterDuff;
import android.graphics.Rect;
import android.graphics.Typeface;
import android.os.SystemClock;
import android.util.Log;
import android.view.GestureDetector;
import android.view.KeyEvent;
import android.view.SurfaceHolder;
import android.view.View;

/**
```

```
 * Renders a fading "Hello world!" in a {@link LiveCard}.
 */
public class LiveCardRenderer extends Activity implements DirectRenderingCallback {

    /**
     * The duration, in millisconds, of one frame.
     */
    private static final long FRAME_TIME_MILLIS = 40;

    /**
     * "Hello world" text size.
     */
    private static final float TEXT_SIZE = 70f;

    /**
     * Alpha variation per frame.
     */
    private static final int ALPHA_INCREMENT = 5;

    /**
     * Max alpha value.
     */
    private static final int MAX_ALPHA = 256;

    private final Paint mPaint;
    private final String mText;

    private int mCenterX;
    private int mCenterY;

    private SurfaceHolder mHolder;
    private boolean mRenderingPaused;


    public static boolean tapped = false;

    private RenderThread mRenderThread;

    protected GestureDetector mGestureDetector;

    public LiveCardRenderer(Context context) {
        mPaint = new Paint();
        mPaint.setStyle(Paint.Style.FILL);
        mPaint.setColor(Color.WHITE);
        mPaint.setAntiAlias(true);
        mPaint.setTextSize(TEXT_SIZE);
        mPaint.setTextAlign(Paint.Align.CENTER);
        mPaint.setTypeface(Typeface.create("sans-serif-thin", Typeface.NORMAL));
        mPaint.setAlpha(0);

        mText = context.getResources().getString(R.string.hello_world);


    }

    @Override
    public boolean onKeyDown(int keyCode, KeyEvent event) {
        if(keyCode == KeyEvent.KEYCODE_DPAD_CENTER){
            tapped = true;
            return true;
        }

        return false;
    }

    @Override
    public void surfaceChanged(SurfaceHolder holder, int format, int width, int height) {
        mCenterX = width / 2;
        mCenterY = height / 2;
    }

    @Override
    public void surfaceCreated(SurfaceHolder holder) {
        mHolder = holder;
        mRenderingPaused = false;
        updateRenderingState();
    }

    @Override
    public void surfaceDestroyed(SurfaceHolder holder) {
        mHolder = null;
        updateRenderingState();
    }

    @Override
    public void renderingPaused(SurfaceHolder holder, boolean paused) {
        mRenderingPaused = paused;
        updateRenderingState();
    }

    /**
     * Starts or stops rendering according to the {@link LiveCard}'s state.
     */
    private void updateRenderingState() {
        boolean shouldRender = (mHolder != null) && !mRenderingPaused;
        boolean isRendering = (mRenderThread != null);
    if (shouldRender != isRendering) {
        if (shouldRender) {
            mRenderThread = new RenderThread();
            mRenderThread.start();
        } else {
            mRenderThread.quit();
            mRenderThread = null;
        }
    }
}

/**
 * Draws the view in the SurfaceHolder's canvas.
 */
private void draw() {
    Canvas canvas;
    try {
        canvas = mHolder.lockCanvas();
    } catch (Exception e) {
        return;
    }
    if (canvas != null) {
        // Clear the canvas.
        canvas.drawColor(Color.TRANSPARENT, PorterDuff.Mode.CLEAR);



        // Update the text alpha and draw the text on the canvas.
        mPaint.setARGB(255,255,255,255);
        if (HandDetectionBluetooth.bitmap!=null){
            Rect destinationRect = new Rect();
            destinationRect.set(0,0,100,100);
            canvas.drawBitmap(HandDetectionBluetooth.bitmap,0,0,mPaint);
            Log.e("Renderer", "Drwaing bitmap");
        }
       // canvas.drawText(HandDetectionBluetooth.currentLetter, mCenterX, mCenterY, mPaint);


        // Unlock the canvas and post the updates.
        mHolder.unlockCanvasAndPost(canvas);
    }
}

/**
 * Redraws the {@link View} in the background.
 */
private class RenderThread extends Thread {
        private boolean mShouldRun;

        /**
         * Initializes the background rendering thread.
         */
        public RenderThread() {
            mShouldRun = true;
        }

        /**
         * Returns true if the rendering thread should continue to run.
         *
         * @return true if the rendering thread should continue to run
         */
        private synchronized boolean shouldRun() {
            return mShouldRun;
        }

        /**
         * Requests that the rendering thread exit at the next opportunity.
         */
        public synchronized void quit() {
            mShouldRun = false;
        }

        @Override
        public void run() {
            while (shouldRun()) {
                long frameStart = SystemClock.elapsedRealtime();
                draw();
                long frameLength = SystemClock.elapsedRealtime() - frameStart;

                long sleepTime = FRAME_TIME_MILLIS - frameLength;
                if (sleepTime > 0) {
                    SystemClock.sleep(sleepTime);
                }
            }
        }
    }

}
```