Development of Interactive Applications for Google Glass

Caio Novaes Lins
João Marcelo Teixeira
Rafael Alves Roberto
Veronica Teichrieb

Abstract

Wearable technology is just recently being made available for the masses and with this come endless possibilities of problem solving on a futuristic level. One example is the Google Glass, a wearable computer with an optical head mounted display. This tutorial aims to guide the audience on how to create interactive applications for this device. We will explain in detail its hardware components and the first steps to begin application development for it. Finally, examples of interactive applications using Google Glass will be demonstrated. Among these examples are an augmented reality solution with face recognition and teleoperation of drones.

1.1 Introduction

The Head Mounted Display, or simply HMD, is a device that is highly associated with Virtual and Augmented Reality, also known as VR and AR, respectively. It is due the fact that these gadgets provide an immersive way to display the virtual content used to the user. Moreover, the HMD allows users to be with their hands free to manipulate interaction devices. Many of them are composed of screens made of different technologies, such as LCD or OLED, in front of the users' eyes. Some models have one canvas in front of one or both eyes, others have two monitors one in front of each eye.

One kind of HMD is the Optical Head Mounted Display, or OHMD. The dif- ference between it and the HMD is that the OHMD exhibits the virtual content on the screen while it allows the user to see the real world through it. One of the drawbacks of this approach is that virtual content does not appear very bright and with high resolu- tion. On the other hand, the semi-transparent screen is more pleasant for the user. Most of them report that the OHMD does not induce motion sickness as much as the HMD with monitors. Another benefit of using optical devices regards safety on critical applica- tions. In case of failure of the display, the user with an OHMD can still see the real world [Rolland and Fuchs 2000].

One popular example of an OHMD is the Google Glass [Google 2014b]. Released in February of 2013, this gadget caught the attention when Google promised a display lighter and cheaper than the other OHMDs available in the market without compromising its quality.

This tutorial aims to: 1) introduce the Google Glass; 2) show its characteristics and hardware configuration; 3) present a complete guide to the audience on how to develop applications for this device, covering from environment setup to an augmented reality example; and finally 4) discuss the possibilities that it brings with examples of some interactive applications.

1.2 Google Glass

1.2.1 Google Glass Hardware

Google's most recent gadget, Glass, works as a 50g Android-based wearable computer. Figure 1.1 explains how it works.

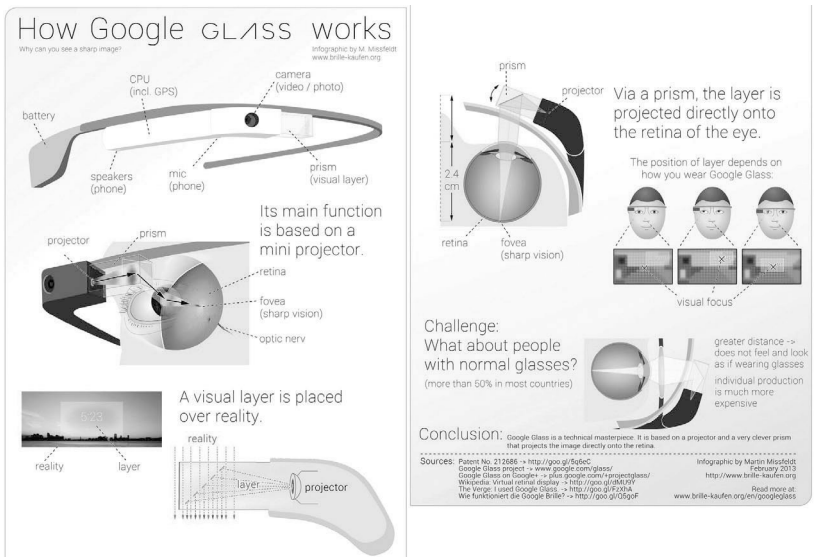

Figure 1.1. How google glass works [Missfeldt 2014].

Tests performed by the authors show that the Glass battery has a maximum life- time of one hour, when used in full load situations, such as recording a 720p resolution video or running computer vision algorithms. Therefore, Google Glass was not meant to be an active device. It was originally conceived to give punctual information (notifica- tions) to the user. Working that way, its battery can last a day or more.

The touchpad, located on the external part of the CPU case, on the lateral side of the device, supports multitouch interaction. It is possible to identify single or double tap gestures, and also when the user swipes fingers in a specific direction.

All content on Glass can be displayed on its 640x360 display, located on the top right view field of the user as a 25-inch screen and 2.4m away. Since the prism is not placed right in front of user's view, it does not interfere significantly with his/her vision. Due to the fact that the prism is located slightly above user's eye of sight, the creation of see-through augmented reality applications is compromised. In order to implement them, developers must first solve the alignment problem between virtual screen (projected by the prism) and real world content, which means that a small portion of the image captured by the Glass camera matches the virtual display area. This mapping may vary according to the distance between Glass and real object being viewed.

The Glass platform was designed so that the existing Android SDK just works fine. This enables developers to code in a familiar environment, but for a uniquely novel device [Google 2014a]. In addition, all of the existing Android development tools can be used, and the developed software for Glass, called Glassware, is delivered as a standard Android Package (APK).

Google also provides the Glass Development Kit (GDK), which works as an add- on to the Android SDK that lets developers build Glassware that runs directly on Glass. In order to make sure that a project is compatible with the Glass platform, the developer must simply set the Android target to version 4.0.3 (which corresponds to API version 15). Another detail that must be taken into consideration is that the application input must be mapped to the Glass input peripherals, because of the fact that interaction is performed through the touchpad and other sensors (camera, gyroscope, accelerometer, GPS, etc.). Table 1.1 lists some of Glass features.

1.3 Developing for Google Glass

As seen on the previous section, Google Glass' foundation is very similar to the smart- phone. It runs on Android, a very popular platform for mobile devices. The essence of developing Android applications is clung to the Java programming language with open- ing to native development kits coded in C/C++ all on top of a Linux kernel. This section will teach how to setup the workspace environment on Windows. After setting up the environment, we will provide a guide through the implementation of a basic application and then an interactive one.

1.3.1 Environment Setup

This short course focuses on development for Google Glass and assumes the learner has a basic knowledge of Android. With this in mind we will be covering the Android envi- ronment setup, then we'll make the necessary preparations for Google Glass integration and leave the workspace ready for development.

Table 1.1. Google Glass features.

Feature	Description
Operating system	Android (4.0.4)
Power	Lithium Polymer battery (2.1 Wh)
CPU	OMAP 4430 SoC, dual-core
Memory	1GB RAM (682MB available to developers)
Storage	16 GB Flash total (12 GB of usable memory)
Display	Prism projector, 640x360 pixels (equivalent of a 25 inch/64 cm screen from 8 ft/2.4 m away)
Sound	Bone conduction transducer
Input	Voice command through microphone, accelero- meter, gyroscope, magnetometer, ambient light sensor, prox- imity sensor
Controller input	Touchpad, MyGlass phone app
Camera	Photos: 5 MP, videos: 720p
Connectivity	Wi-Fi 802.11b/g, Bluetooth, micro USB
Weight	50g
Backward compatibility	Any Bluetooth-capable phone; MyGlass com- panion app requires Android 4.0.3 (Ice Cream Sandwich) or higher or any iOS 7.0 or higher

1.3.1.1 Android Setup

There are a few components needed for programming on Android. An Integrated De- velopment Environment (IDE) is needed along with the Android Developer Tools (ADT) plugin. The ADT plugin serves to streamline the application development giving the necessary tools to build, test and debug the apps. By default, Google uses Eclipse as IDE. We will also need the API libraries. All these components are encapsulated into a package called ADT Bundle and can be downloaded from the Google Android site at http://developer.android.com/sdk/index.html.

Before downloading the ADT Bundle, make sure the computer has the Java De- velopment Kit (JDK) installed. In case it doesn't, the latest version is available at Oracle's website. With the JDK in- stalled we're good to go. After downloading the bundle, simply extract the packages and we're good to develop applications for Android.

1.3.1.2 Google Glass Integration

To program for Google Glass your computer must make a connection with the device. We must provide its drivers just like any other Android-using hardware. Also we need to manipulate the Glass' hardware components and use some Glass-specific features, gen- erating the need for the tools to do so. For this we must use an add-on to the Android

SDK called Glass Development Kit, also known as GDK. Google provides a Mirror API that allows you to build web-based services that interact with Glass. It provides a cloud based API that does not require running code on Glass. But we won't be detailing it here for the fact that for our interactive scenario, which consists of augmented reality appli- cations, real-time user interaction is crucial and that is where the GDK comes into play. This section will explain how to install the device drivers and the GDK with details.

We'll need to modify a few files so we can update the Google Glass via windows device manager. Follow the step-by-step instructions labeled below:

1. Locate the folder containing the Google USB Driver; it should be something similar to "<ADTBundlepath>\sdk\extras\google\usb_driver\".
2. Now we must add the vendor and product identification, or VID and PID, for Google Glass on the file "android_winusb.inf" under the [Google.NTamd64] and [Google.NTx86] sections. In the end we should have something like the following:

```
[Google.NTx86]
;Google Glass
%SingleAdbInterface% = USB_Install, USB\VID_18D1&PID_4E11&REV_0216
%CompositeAdbInterface% = USB_Install, USB\VID_18D1&PID_4E11&MI_01

%SingleAdbInterface% = USB_Install, USB\VID_18D1&PID_9001&REV_0216
%CompositeAdbInterface% = USB_Install, USB\VID_18D1&PID_9001&MI_01

;Other devices and IDs...

[Google.NTamd64]
;Google Glass
%SingleAdbInterface% = USB_Install, USB\VID_18D1&PID_4E11&REV_0216
%CompositeAdbInterface% = USB_Install, USB\VID_18D1&PID_4E11&MI_01

%SingleAdbInterface% = USB_Install, USB\VID_18D1&PID_9001&REV_0216
%CompositeAdbInterface% = USB_Install, USB\VID_18D1&PID_9001&MI_01

;Other devices and IDs...
```

Figure 1.2. Snippet to add to the "android_winusb.inf" file.

3. Next we'll disable Driver Signature Verification since Google Glass' driver is not a signed driver. For this we'll create a separate set of instructions based on [Gibb 2013]:
   (a) Click the "Change PC settings" option in the settings button found on the Charms Bar;
   (b) Now that we're on the Modern Control Panel, switch to the Update & Recov- ery section. Figure 1.3 shows (a) and (b);

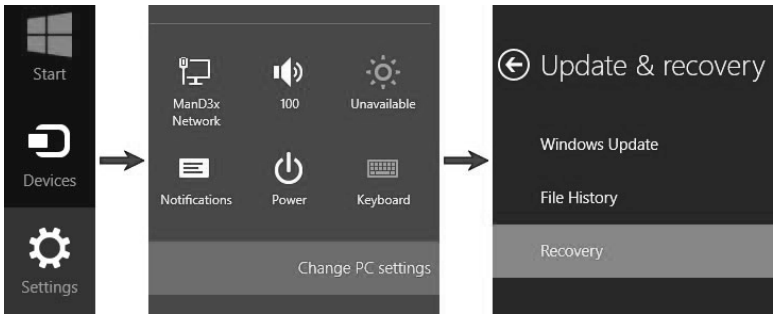

Figure 1.3. First steps for disabling the driver signature verification in Windows 8.1.

(c) Next you should see an advanced startup section to the right, so go ahead and restart in order to proceed to the advanced startup;
(d) Once your computer is rebooted you will be given a few options. Choose them in the following order: Troubleshoot, Advanced Options, Startup Settings, Restart;
(e) Finally, after rebooting, disable driver signature enforcement by pressing F7.
Your PC will reboot once again and you can now install the Google Glass driver.

4. Plug Google Glass into your computer, open the Device Manager and locate the Google Glass device. It should have a warning because it is not installed. Right click it and select Update Driver Software.
5. Choose "Browse my computer for driver software" and enter the folder where the "android_ winusb.inf" file is located.
6. After confirming, a security dialog will be shown. Choose "Install this driver soft- ware anyway" to install and complete the driver installation.

Now that we have installed the drivers, let's acquire the GDK.

Since Google Glass uses Android 4.0.3 (API 15) let's install its SDK Platform along with the Google SDK Sneak Peak from the SDK Manager. It is important to men- tion that the GDK is currently in a preview version, hence the name Sneak Peak and therefore some important parts are missing. You can open SDK manager from Eclipse or from the executable located on the ADT Bundle folder. Once opened, go to the API 15 section as shown on Figure 1.4, and mark the SDK Platform, the GDK Sneak Peak, and then click install. After the installation our environment is ready for programming. To finish, turn on the debug option of your Google Glass device. Do this by going (inside Google Glass menu) into Settings > Device Info > Turn on debug. This will allow your development system to communicate with Glass.

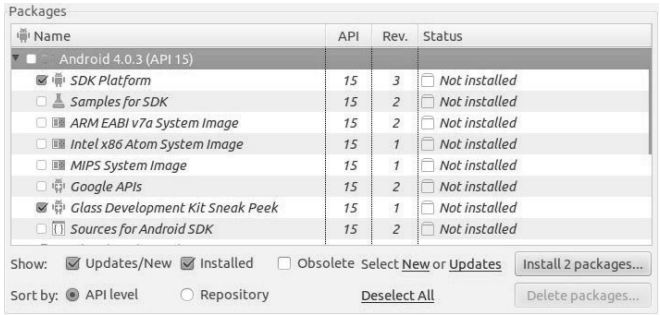

Figure 1.4. Android SDK Manager with the components to program for Google Glass marked.

1.3.2 Basic Project Implementation

This section will guide you through a Google Glass application implementation. There are two ways of showing your applications on Glass using the GDK. One is Live Cards and the other is Immersion [Kaspari 2013]. Figure 1.5 illustrates the main idea of both. Immersion applications take full control of the UI and are essentially regular Android activities. To run any other application or access a card you must leave the immersion. On the other hand Live Card applications show up as a card in the timeline and can run independently from other cards. One example is the Compass application that comes with the GDK.

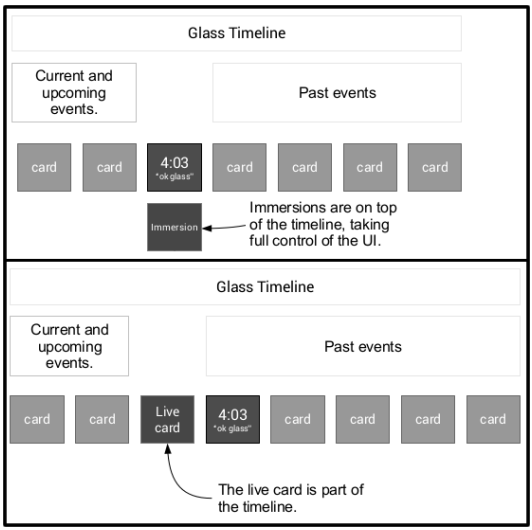

Figure 1.5. Scheme comparing the two different ways of making applications for Google Glass: Immersion (top) and Live Card (bottom).

1.3.2.1 Project Setup

Now we'll begin implementing our first Google Glass activity using Immersion. First, create an Android application, as shown on Figure 1.6. Set minimum and target SDK to API 15, compile with GDK Sneak Peak and set the Theme to none.

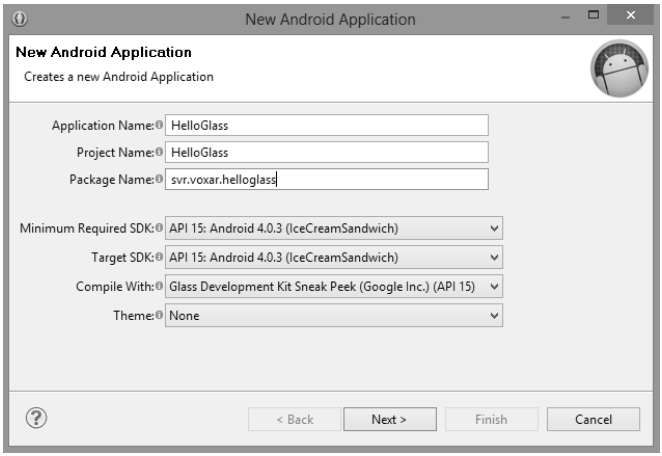

Figure 1.6. Initial setup of the Google Glass basic project.

1.3.2.2 Create Immersion

On the MainActivity class go ahead and create this simple activity, as seen in Figure 1.7.

```
public class MainActivity extends Activity
{
    @Override
    protected void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        Card card = new Card(this);
        card.setText("Hello SVR!");
        card.setFootnote("cin.ufpe.br/voxarlabs");
        setContentView(card.toView());
    }
}
```

Figure 1.7. Simple main activity of the Google Glass basic project.

The Card object from the GDK library gives us a simple layout to be displayed on Google Glass' interface. We set a title, a footnote and then convert it into a view to be displayed using the setContentView() method.

1.3.2.3 Voice Commands

A common way to launch Google Glass applications is by using voice triggers. To do this we must implement three things. First we declare a string resource for our voice command, as can be seen in Figure 1.8.

```
<?xml version="1.0" encoding="utf-8"?>
<resources>

    <string name="app_name">HelloGlass</string>
    <string name="show_basic_example">show basic example</string>

</resources>
```

Figure 1.8. Adding the string value for the voice launch command in the strings xml file.

Then create an XML resource file for the voice trigger using the previously created string, as shown in Figure 1.9.

```
<?xml version="1.0" encoding="utf-8"?>
<trigger keyword="@string/show_basic_example"/>
```

Figure 1.9. XML file created for the voice command trigger.

Finally on our manifest we add an intent filter for the VOICE_TRIGGER action to our activity and link the previously created XML to the intent using a meta-data tag. The manifest should then look like the one on Figure 1.10.

```
<application
    android:allowBackup="true"
    android:icon="@drawable/ic_launcher"
    android:label="@string/app_name">
    <activity
        android:name="svr.voxar.helloglass.MainActivity"
        android:label="@string/app_name"
        android:icon="@drawable/hello_icon">
        <intent-filter>
            <action android:name="android.intent.action.MAIN" />

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

Figure 1.10. Final version of the Android manifest of the basic Google Glass application.

Now say "ok glass" followed by "show basic example" and you should get the result shown on Figure 1.11. Now let's proceed to a more advanced, interactive project.

Figure 1.11. Running the Google Glass basic application.

1.3.3 Interactive Project Implementation

This section will cover a step-by-step implementation of an interactive application for Glass. It consists of visualizing 3D models using an augmented reality library called Metaio SDK [metaio 2014]. The application will use gesture commands to switch be- tween geometries, which are virtually projected on top of a real marker. Below is a step-by-step walkthrough of the implementation.

1. First, download and run the Metaio SDK installer from this site: https://metaio. app.box.com/SDK-PC.
2. Import the metaio SDK library to your Eclipse workspace; it is found on <MetaioLocation>\metaioSDK5.3_Android\.
3. Create a new application as done on the basic project but changing the name and package for something like "Interactive Example".
4. Now that you have your Glass interactive project and the metaio SDK library on your workspace you need to set the metaio SDK as one of your app's library. Right- click the Interactive-Example project, go to Properties > Android, add the metaio SDK library to the project and then click OK.
5. The ARViewActivity holds the necessary methods to make the augmented experi- ence possible but Metaio advises not to change such class, so we will create another class, name it ARActivity, inside our InteractiveExample project and let it extend the ARViewActivity class and implement its methods.
6. So now we have an empty MainActivity, an ARActivity with empty methods and and the ARViewActivity. Let's go ahead and modify our manifest to look like the one on Figure 1.12. Note

that we put some permissions that the Metaio SDK needs to function properly and the launch by voice trigger information from the previous example. Remember that we have to add the string and XML resources as well.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="svr.voxar.interactiveexample"
    android:installLocation="preferExternal"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk
        android:minSdkVersion="15"
        android:targetSdkVersion="15" />

    <!-- Metaio SDK permissions -->
    <uses-permission android:name="android.permission.CAMERA" />
    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
    <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
    <uses-permission android:name="android.permission.MODIFY_AUDIO_SETTINGS" />

    <application
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher">
        <activity
            android:name="svr.voxar.interactiveexample.MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
            <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"/>
        </activity>
        <activity android:name="svr.voxar.interactiveexample.ARActivity"/>
        <activity android:name="svr.voxar.interactiveexample.ARViewActivity"/>
    </application>
</manifest>
```

Figure 1.12. Final version of the Android manifest of the interactive Google Glass application with the Metaio SDK permissions.

7. For the layout we'll have a simple Relative Layout called ar_layout.xml as shown on Figure 1.13.

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent" >
</RelativeLayout>
```

Figure 1.13. xml layout to be passed to the ARActivity's getGUILayout() method.

8. We also need to get our assets which consist of a tracking configuration xml, the pattern image to track and the 3D models. Download the .zip with all the assets from https://www.dropbox.com/s/69n37hizoi7pc03/Assets.rar. Ex- tract and copy the whole Assets folder to the assets folder inside the InteractiveEx- ample directory.
9. Now implement the MainActivity as shown on Figure 1.14. This activity runs an AsyncTask to extract all the necessary assets at startup and then starts the ARAc- tivity after a few seconds.

```
public class MainActivity extends Activity
{
    AssetsExtracter mTask;

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

        Card card = new Card(this);
        card.setText("Interactive Example");
        card.setFootnote("cin.ufpe.br/voxarlabs");
        setContentView(card.toView());

        mTask = new AssetsExtracter();
        mTask.execute(0);

        Handler handle = new Handler();

        Runnable run = new Runnable() {
            @Override
            public void run() {
                Intent mainIntent = new Intent(MainActivity.this, ARActivity.class);
                startActivity(mainIntent);
                finish();
            }
        };
        handle.postDelayed(run, 2000);
    }

    private class AssetsExtracter extends AsyncTask<Integer, Integer, Boolean> {
        @Override
        protected Boolean doInBackground(Integer... params) {
            try {
                AssetsManager.extractAllAssets(getApplicationContext(), true);
            } catch (IOException e) {
                MetaioDebug.printStackTrace(Log.ERROR, e);
                return false;
            }
            return true;
        }
    }
}
```

Figure 1.14. Main activity of the Google Glass interactive project.

10. Switching to the ARActivity, we are going to edit the getGUILayout() and set its return to "R.layout.ar_layout" and the loadContents() method will look like Fig- ure 1.15. This is where we load the tracking configuration file and create the 3D models for rendering. The onGeometryTouch() and the getMetaioSDKCallback- Handler() won't be covered in this tutorial, but

if you're curious to see how they work there are examples inside the metaio SDK folder that you can use.

```
@Override
protected void loadContents()
{
    // Getting a file path for tracking configuration XML file
    String trackingConfigFile =
            AssetsManager.getAssetPath(getApplicationContext(),
            "Assets/TrackingData_MarkerlessFast.xml");
    // Assigning tracking configuration
    boolean result = metaioSDK.setTrackingConfiguration(trackingConfigFile);
    MetaioDebug.log("Tracking data loaded: " + result);
    // Getting a file path for a 3D geometry
    String manModel = AssetsManager.getAssetPath(getApplicationContext(),
            "Assets/metaioman.md2");
    if (manModel != null)
    {
        // Loading 3D geometry
        mMan = metaioSDK.createGeometry(manModel);
        if (mMan != null)
        {
            // Set geometry properties
            mMan.setScale(new Vector3d(4.0f, 4.0f, 4.0f));
        }
        else
            MetaioDebug.log(Log.ERROR,"Error loading geometry: " + manModel);
    }
    String tigerModel =AssetsManager.getAssetPath(getApplicationContext(),
            "Assets/tiger.md2");
    if (tigerModel != null)
    {
        mTiger = metaioSDK.createGeometry(tigerModel);
        if (mTiger != null)
        {
            mTiger.setScale(new Vector3d(10.0f, 10.0f, 10.0f));
            mTiger.setVisible(false);

        }
        else
            MetaioDebug.log(Log.ERROR, "Error loading geometry: " + tigerModel);
    }
}
```

Figure 1.15. ARActivity's loadContents() method.

11. We'll also add the methods that will be called by gestures and these will control the switch between the two geometries. We'll create a global GestureDetector, an object class from the GDK, and listen specifically for the TAP gesture and each tap will switch between geometries. The snippet below shows how this is done. Note that we have overridden the onCreate() method to initialize the GestureDetector.

```
@Override
public void onCreate(Bundle savedInstanceState)
{
    super.onCreate(savedInstanceState);
    mGestureDetector = createGestureDetector(this);
}

private GestureDetector createGestureDetector(Context context)
{
    GestureDetector auxGestureDetector = new GestureDetector(context);

  auxGestureDetector.setBaseListener( new BaseListener()
  {
      @Override
      public boolean onGesture(Gesture gesture)
      {
          if (gesture == Gesture.TAP)
          {
              if(mMan != null && mTiger != null)
              {
                  if(mMan.isVisible())
                  {
                      mMan.setVisible(false);
                      mTiger.setVisible(true);
                  } else
                  {
                      mTiger.setVisible(false);
                      mMan.setVisible(true);
                  }
              }
              return true;
          }
          return false;
      }
  });

  auxGestureDetector.setFingerListener(new FingerListener()
  {
      @Override
      public void onFingerCountChanged(int previousCount, int currentCount) { }
  });

  auxGestureDetector.setScrollListener(new ScrollListener()
  {
      @Override
      public boolean onScroll(float arg0, float arg1, float arg2) { return false; }
  });

  return auxGestureDetector;
}

@Override
public boolean onGenericMotionEvent(MotionEvent event)
{
    if(mGestureDetector != null)
    {
        return mGestureDetector.onMotionEvent(event);
    }
    return false;
}
```

Figure 1.16. ARActivity's GestureDetector setup.

12. Before we run the application we need to provide a metaio application signature. To do this login at http://dev.metaio.com/, navigate to "my apps" and add your Application ID and name to generate the new signature. Then copy and add the signature as a value to a new string resource with the name "metaioSDKSignature" and make sure ARViewActivity is using the "R.string.metaioSDKSignature" key (this video shows a step by step tutorial on how to setup Metaio SDK for an Android development environment: http://youtu.be/KVtCi-WwmFU).
13. Now go ahead and run the application and you should see the results as shown on Figure 1.17. The source code of the basic and interactive applications shown in this

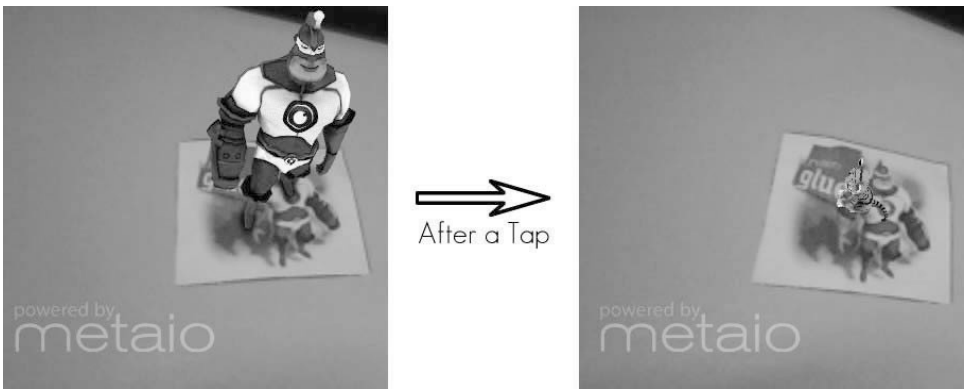

Figure 1.17. ARActivity's loadContents() method.

With that we conclude our tutorial on implementing an interactive application. Next section shows some advanced use case scenarios and solutions.

1.4 Use Cases

It is clear that the range of ideas and solutions Google Glass explores is enormous. This section will show two use cases designed and developed for this platform.

1.4.1 Face Recognition using OpenCV

For this we will introduce Glassist, which consists of a teacher assistant application that allows the creation of profiles for each child (student), entering and viewing information about them [da Silva et al. 2014]. In a seamless and non-intrusive way the application uses augmented reality to help day-to-day activities of teachers.

For implementation we used the Open Source Computer Vision Library (OpenCV) [Bradski 2000] for the image processing techniques. Although the library has many image handling examples it lacked a face recognition algorithm. To overcome this obstacle we used its face detection algorithm and a sim-

ple yet sufficient comparison algorithm. The OpenCV library gives us the position and boundary of the faces in screen coordinates, next we extract the face image from the current frame and then fondly compare it to a set of face images stored in the application's database. The implementation of the comparison algorithm was done by comparing byte by byte and checking if the sum of the squared differences of each byte pair is small enough to be a valid candidate. To use the OpenCV Android library, the developer can download it from its homepage and import the library the same way that was explained on last section with the Metaio SDK. To setup permissions on the manifest and create basic projects, we recommend studying the example applications that come with the library.

The application flow is the following: the system starts with a full screen visual- ization of the camera image. When a student is detected, his/her name and status icons appear right above him/her, indicating that the face recognition algorithm has recognized him/her. When the user taps the touchpad, the system enters the profile screen of the stu- dent that is in the center of the camera view. In this level, the teacher can swipe between student profiles or tap to view more information about one of them and add new notes, photos, videos, or other information to it. The user can also add data to the portfolios by centering a student on the camera view and clicking on the shutter button, adding a photo or video directly to his/her portfolio. On Figure 1.18 we can see two screenshots of the core functions of the Glassist application.

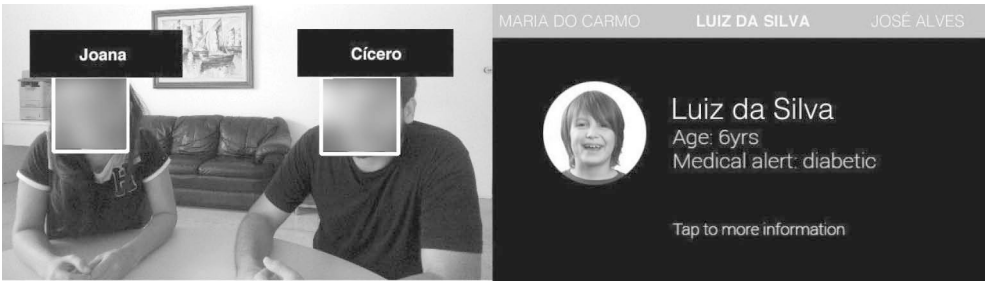

Figure 1.18. Glassist recognizing faces (left) and visualizing a portfolio (right).

1.4.2 Teleoperating Drones with Glass

In this case study we will analyze a scenario where the user controls an Unmanned Aerial Vehicle (UAV) through head positioning and gestures using Google Glass. This project for visualization and control was designed to be applied to the structural inspection of buildings. In order to do that, a connection is established between the Glass and an UAV in a form of a Drone and through this connection the Drone sends captured images to Glass while it controls the UAV through gesture inputs.

1.4.2.1 The Drone

An UAV is defined as an uninhabited motorized vehicle that can be controlled by a re- mote control, semi-autonomous, autonomous, or a combination of the capabilities above. It can be used for military

purposes, to solve specific tasks like traffic and meteorological surveillance, carrying specific payload, act where presence of a human pilot can be un- healthy or dangerous and spy the enemy areas. As compared to a manned vehicle, this one has some advantages, such as reduced cost, longer endurance and less risk to the crew [Fahlstrom and Gleason 2012]. The UAV used in this work is the Parrot AR.Drone 2.0 [Parrot 2014], and its components are shown on Figure 1.19.

1.4.2.2 Application Details

There are several SDKs available that allow the remote communication and controlling of a drone. They vary on features, programming language and platform. Table 1.2 compares some of the drone SDKs found in the literature. We have chosen the SDK called CVDrone because it integrates drone operation with OpenCV. This way, once the image is received from the drone, it can be easily encoded and transmitted to the Glass.

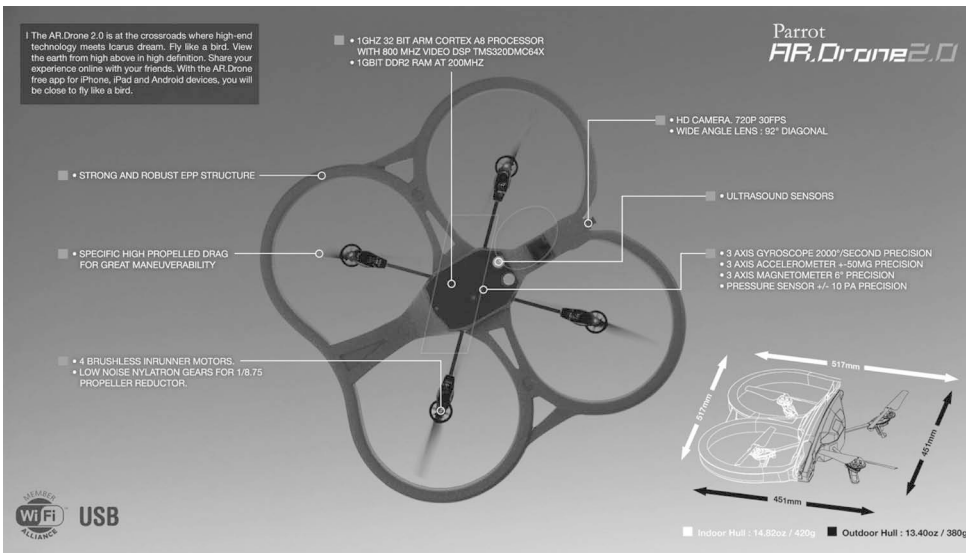

Figure 1.19. AR.Drone 2.0 and its components.

Data exchange between the Glass and the drone requires wireless network com- munication. Since the AR.Drone creates a local wireless network to which controller devices can connect, we decided to use it as communication channel. In fact, when con- structing a network architecture for such teleoperation application, two possibilities come to mind: directly connecting both Glass and drone, or using a PC as bridge between these devices. We adopted the second approach for simplicity reasons. Additionally, we were not able to find any working SDK capable of directly providing control and image information exchange between these two devices without the help of a computer.

The architecture designed for the application is basically composed by two major modules: an Android client built specifically for Google Glass (implemented using Java), responsible for managing image visualization and remote control, and a PC server (im- plemented using C++), responsible for intermediating communication between the drone and the Glass.

The Client module has three sub-modules: UI, Image Receiver and Command Sender, as shown in Figure 1.20. The UI module is responsible for displaying the images received from the Image Receiver module and for capturing interaction events (gestures and head motion), used for sending commands to the Command Sender module.

The Image Receiver module is responsible for receiving the images captured by the drone from the network and routing them to the UI module. The Command Sender module is responsible for sending the commands over the network for controlling the drone, which has nine possible options: take off, land, rotate right and left, go front, back, left and right, and stand (used to stop the current command being executed).

The Server module has, analogously to the Client module, three sub-modules: Drone Manager, Image Server and Command Server, as shown in Figure 1.20. The Drone

Table 1.2. Comparison between some AR.Drone SDKs.

SDK	Language	Features
CVDrone [puku0x 2014]	C/C++	1. Integrates drone operation and OpenCV 2. Allows full control and image access from both cameras
Javadrone [Codeminders 2014]	Java	1. Allows full control 2. Runs on multiple platforms 3. Has version compatibility problems
ARDrone.Net [Endres 2014]	C#	1. Allows full control and image access from both cameras
YADrone [Bade 2014]	Java	1. Allows full control 2. Image access does not work on Android platforms
EasyDrone [Kenai 2014]	Java	1. Provides easier movement commands and complex behaviours such as face detection/face following and tag

Manager takes care of the communication between PC and drone, acquiring the images captured from its camera and sending it commands for performing the desired actions.

The Image Server is the module responsible for the PC/Glass image streaming, retrieving the images captured from the Drone Manager, compressing them into JPEG format, and then sending the data through the network. The Command Server is respon- sible for receiving the commands sent from the Glass and routing them to the Drone Manager module.

If you would like to analyze the implementation in detail, the source code for the entire system (both client and server sides) is available for download and distributed as open-source at http://goo.gl/YoWA5q.

1.4.2.3 Case Study Analysis

The drone is controlled using the commands listed on Table 1.3 and can be used in two different scenarios: indoor and outdoor. The image captured by the AR.Drone is displayed in the Glass as a JPEG with 30% of compression to avoid transmission latency. Figure 1.21 shows an operator controlling the AR.Drone using the Google Glass. Even though, it is still possible to extract relevant information from it, such as the structural damage on the wall and close to the ceiling.

1.4.3. Tools and Tips

During the development of applications for Google Glass, we found some useful tools and functionalities that helped us in the process. One is the Droid@Screen application

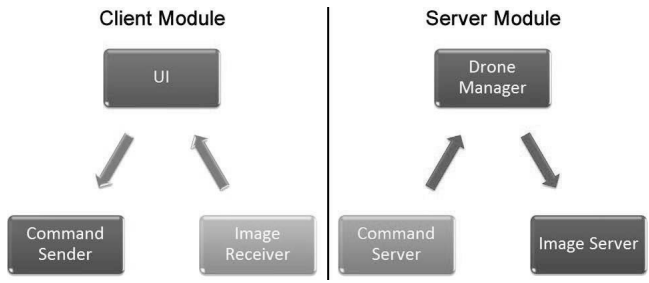

Figure 1.20. Client (left) and Server (right) modules architecture.

Table 1.3. Action map between Glass and AR.Drone.

Operator Gesture	Teleoperator Action
Head position to the front	Drone stands still
Rotate head to the left	Rotate drone to the left
Rotate head to the right	Rotate drone to the right
Swipe right	Drone goes forward
Swipe left	Drone stands still
Swipe up	Drone goes up
Swipe up with two fingers	Drone goes down
Two-tap	Tale off/land

[Riboe 2014]. This app was used to capture the images displayed on the device. It receives from a dedicated USB stream every image that is shown to the user. It is important to say that such tool is used for debugging purposes, so the image refresh rate is low (approximately 1 image every 200ms).

Another tip is concerning the fact that Glass goes into standby mode very easily, to save battery. For interactive applications we want the screen to stay on to produce the desired experience. This can be done by adding a single line of code to the onCreate() method of your Activity1. This will prevent your app from going into standby, simple and perfect for real-time applications.

Overall the examples that come with the GDK are very complete and studying them is a great way to get used to working with Google Glass development.

1.5 Final Remarks

This tutorial provided a guide for those who want to start developing or already develop for Google Glass and want to explore interactive solutions this wearable device provides.

1The line is "getWindow().addFlags(WindowManager.LayoutParams.FLAG_KEEP_SCREEN_ON);".

Figure 1.21. On top (a), we can see an operator interacting with Google Glass in an outdoor environment. On the bottom left (b) there is his vision while perform- ing a structural inspection task. On the bottom right (c), an open field navigation test.

It was explained in the work what is Glass and how it works from hardware infrastruc- ture to software details. We then made a walkthrough from setting up the workspace to creating, step by step, a basic and an interactive augmented reality application using the Metaio SDK. Finally we show some use cases of interactive Glass apps, focusing aug- mented reality aiding in classrooms and operation of a drone for structural inspection using intercommunication between Glass, computer and the AR.Drone.

Google Glass gave us a whole new way of thinking, because we can develop unique solutions, such as the teleoperation of drones showed, and also adapt an already implemented application in such a way that it will also be more interesting. Just by having an HMD with all the main components of a smartphone augments our range of possible outcomes. Focusing on augmented reality, there are a few limitations like the need to optimize codes to the extreme because Google Glass does not last long with active apps, and also because of its limited computing power.

Google Glass can do much more things that were not covered by this course, such as lifelike streaming of videos and extracting important information from real world scenarios. There is an infinity of branches to be explored. Soon it will be available to the public so everybody can create stunning applications. Overall we think Glass has expanded the pool of ways to solve problems and it was an amazing opportunity to think and develop for it.

References

[Bade 2014] Bade, D. (2014). Yadrone : Yet another open framework for controlling the ar.drone 2.0. http://bit.ly/1dIX0gd.

[Bradski 2000] Bradski, G. (2000). The opencv library. Dr. Dobb's Journal of Software Tools.

[Codeminders 2014] Codeminders (2014). Javadrone. http://bit.ly/1dkMu3I.

[da Silva et al. 2014] da Silva, M. M. O., Freitas, D., Neto, E., Lins, C., ao Marcelo Teixeira, J., and Teichrieb, V. (2014). Glassist: Using google glass as an aid to classroom management. In Under Review.

[Endres 2014] Endres, T. (2014). Ardrone.net. http://bit.ly/1dIX0gd. [Fahlstrom and Gleason 2012] Fahlstrom, P. and Gleason, T. (2012). Introduction to UAV Systems. Aerospace Series. Wiley.

[Gibb 2013] Gibb, T. (2013). How to disable driver signature verification on 64-bit win- dows 8.1 (so that you can install unsigned drivers). http://bit.ly/JsBH9W.

[Google 2014a] Google (2014a). Glass development kit. http://bit.ly/ 1gTNxm5.

[Google 2014b] Google, I. (2014b). Google glass. http://bit.ly/1gfLM7E. [Kaspari 2013] Kaspari, S. (2013). Mirror api and gdk - developing for google glass #1. http://bit.ly/1eHxITl.

[Kenai 2014] Kenai, P. (2014). Easydrone - the javadrone composer. http://bit. ly/1gvXFqW.

[metaio 2014] metaio, G. (2014). metaio home augmented reality products & solutions. http://bit.ly/1bszicy.

[Missfeldt 2014] Missfeldt, M. (2014). Google glass (infographic) - how it works. http://bit.ly/1dnG58d.

[Parrot 2014] Parrot, S. (2014). Ar.drone 2.0. parrot new wi-fi quadricopter - ar.drone.com - hd camera - civil drone - parrot. http://bit.ly/1gPDuP9.

[puku0x 2014] puku0x (2014). Cv drone (= opencv + ar.drone). http://bit.ly/ 1eHwUxx.

[Riboe 2014] Riboe, J. (2014). Droid at screen | show your android device for your audience. http://bit.ly/1lilMeX.

[Rolland and Fuchs 2000] Rolland, J. P. and Fuchs, H. (2000). Optical versus video see- through head-mounted displays in medical visualization. Presence: Teleoper. Virtual Environ., 9(3):287–309.