Chapter 6

A Simple Location-Based App Using Augmented Reality and the Maps API

This chapter outlines how to make a very simple real world augmented reality (AR) application. By the end of this chapter, you will have a fully functional example app.

The app will have the following basic features:

- The app will start and display a live camera preview on the screen.
- The camera preview will be overlayed with the sensor and location data, as in the Chapter 3 widget overlay example app.

R. Sood, *Pro Android Augmented Reality*

- When the phone is held parallel to the ground, the app will switch over to display a map. We will add a margin of ±7 because it is unlikely that the user will be able to hold the device perfectly parallel to the ground. The user's current location will be marked on the app. The map will have the option to switch among satellite view, street view, and both. The map will be provided using the Google maps application programming interface (API).
- When the device is moved into an orientation that is not parallel to the ground, the app will switch back to a camera view.

This app can act as a standalone application or be extended to provide an augmented reality navigation system, which we will do in the next chapter.

To start, create a new Android project. The project should target the Google APIs (API level 7, as we are targeting 2.1 and above) so that we can use the map functionality of Android. The project used throughout this chapter has the package name `com.paar.ch06`, with the project name Pro Android AR 6: A Simple App Using AR. You can use any other package and project name you want, as long as you remember to change any references in the example code to match your changes.

After creating the project, add another class to your project by right-clicking the package name in the left bar of eclipse and selecting Class from the New menu (see Figure 6-1):

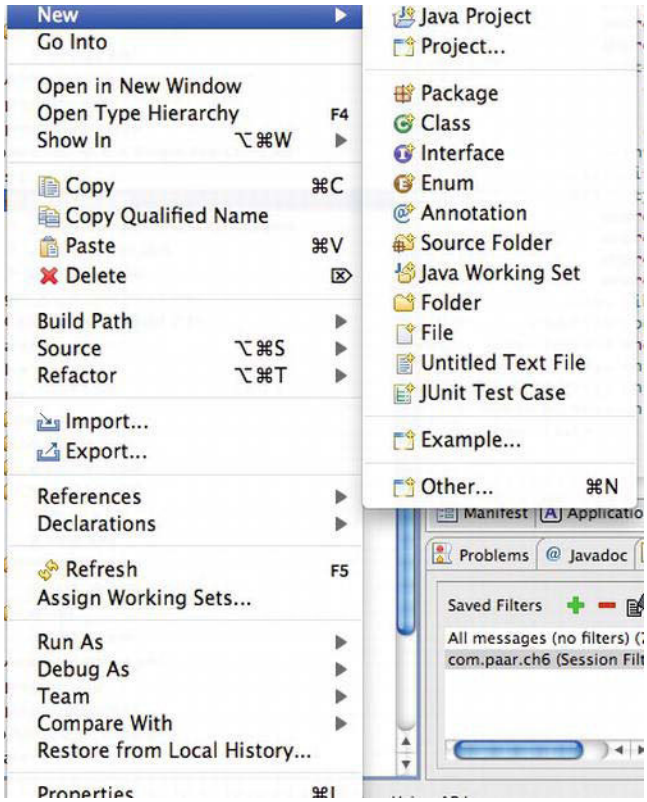

Figure 6-1. *The menu to create a new class.*

Name this class `FlatBack`. It will hold the `MapView` and related location APIs. Then create another class called `FixLocation`. You'll learn more about this class later in the chapter.

Editing the XML

After the necessary classes are created, we can start the coding work. First of all, edit `AndroidManifest.xml` to declare the new `activity` and ask for the necessary features, libraries, and permissions. Update the `AndroidManifest.xml` as follows:

Listing 6-1. *Updated AndroidManifest.xml*

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

```
    package="com.paar.ch6"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk android:minSdkVersion="7" />

    <application
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name" >
        <activity
            android:label="@string/app_name"
            android:name=".ASimpleAppUsingARActivity"
            android:screenOrientation = "landscape"
            android:theme="@android:style/Theme.NoTitleBar.Fullscreen"
            android:configChanges = "keyboardHidden|orientation">
            <intent-filter >
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <activity
            android:name=".FlatBack"
            android:screenOrientation="landscape"
            android:theme="@android:style/Theme.NoTitleBar.Fullscreen"
            android:configChanges="keyboardHidden|orientation"></activity>
        <uses-library android:name="com.google.android.maps" />
    </application>
<uses-feature android:name="android.hardware.camera" />
<uses-permission android:name="android.permission.CAMERA" />
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
<uses-permission android:name="android.permission.INTERNET"/>
</manifest>
```

Make sure that the `FlatBack Activity` is declared exactly as previously, that the `<uses-library>` tag is inside the `<application>` tag, and that all the permissions and feature requests are outside the `<application>` tag and inside the `<manifest>` tag. This is pretty much everything that needs to be done in the `AndroidManifest` for now.

We will need to add some strings now, which will be used in the overlays and in the Help dialog box for the app. Modify your `strings.xml` to the following:

Listing 6-2. *Updated strings.xml*

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
        <string name="hello">Hello World, ASimpleAppUsingARActivity!</string>
        <string name="app_name">A Simple App Using AR</string>
```

```
        <string name="xAxis">X Axis:</string>
        <string name="yAxis">Y Axis:</string>
        <string name="zAxis">Z Axis:</string>
        <string name="heading">Heading:</string>
        <string name="pitch">Pitch:</string>
        <string name="roll">Roll:</string>
        <string name="altitude">Altitude:</string>
        <string name="longitude">Longitude:</string>
        <string name="latitude">Latitude:</string>
        <string name="empty"></string>
        <string name="help">This is the example app from Chapter 6 of Pro
Android Augmented Reality. This app outlines some of the basic features of
Augmented Reality and how to implement them in real world applications.</string>
        <string name="helpLabel">Help</string>

</resources>
```

Creating Menu Resources

You will create two menu resources: one for the camera preview `Activity`, and one for the `MapActivity`. To do this, create a new subfolder in the `/res` directory of your project called `menu`. Create two XML files in that directory titled `main_menu` and `map_toggle`, respectively. In `main_menu`, add the following:

Listing 6-3. *main_menu.xml*

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android" >
    <item
        android:id="@+id/help"
        android:title="Help"></item>
</menu>
```

This is basically the help option in the main `Activity`. Now in `map_toggle`, we will be having three options so add the following to it:

Listing 6-4. *map_toggle.xml*

```
<?xml version="1.0" encoding="utf-8"?>
<menu
  xmlns:android="http://schemas.android.com/apk/res/android">
    <item
        android:id="@+id/map"
        android:title="Map View"></item>
    <item
        android:id="@+id/sat"
        android:title="Satellite View"></item>
```

```
    <item
        android:id="@+id/both"
        android:title="Map + Satellite View"></item>
</menu>
```

The first option allows users to set the kind of map displayed to the street view, as you see on a roadmap. The second option allows them to use satellite images on the map. The third option overlays a roadmap onto satellite images of that place. Of course, both these files only define parts of the user interface, and the actual work will be done in the Java files.

Layout Files

There are three layout files in this project. One is for the main camera preview and related overlays, one is for the Help dialog box, and one is for the map.

Camera Preview

The camera preview `Activity` layout file is the normal `main.xml`, with a few changes to its standard contents:

Listing 6-5. *The Camera Preview Layout File*

```
<RelativeLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
        android:id="@+id/relativeLayout1"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent" >
        <SurfaceView
                android:id="@+id/cameraPreview"
                android:layout_width="fill_parent"
                android:layout_height="fill_parent" />

        <TextView
            android:id="@+id/xAxisLabel"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_alignParentLeft="true"
            android:layout_alignParentTop="true"
            android:layout_marginLeft="18dp"
            android:layout_marginTop="15dp"
            android:text="@string/xAxis" />

        <TextView
            android:id="@+id/yAxisLabel"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
```

```
    android:layout_alignLeft="@+id/xAxisLabel"
    android:layout_below="@+id/xAxisLabel"
    android:text="@string/yAxis" />

<TextView
    android:id="@+id/zAxisLabel"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignLeft="@+id/yAxisLabel"
    android:layout_below="@+id/yAxisLabel"
    android:text="@string/zAxis" />

<TextView
    android:id="@+id/headingLabel"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignLeft="@+id/zAxisLabel"
    android:layout_below="@+id/zAxisLabel"
    android:layout_marginTop="19dp"
    android:text="@string/heading" />

<TextView
    android:id="@+id/pitchLabel"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignLeft="@+id/headingLabel"
    android:layout_below="@+id/headingLabel"
    android:text="@string/pitch" />

<TextView
    android:id="@+id/rollLabel"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignLeft="@+id/pitchLabel"
    android:layout_below="@+id/pitchLabel"
    android:text="@string/roll" />

<TextView
    android:id="@+id/latitudeLabel"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignLeft="@+id/rollLabel"
    android:layout_below="@+id/rollLabel"
    android:layout_marginTop="34dp"
    android:text="@string/latitude" />

<TextView
    android:id="@+id/longitudeLabel"
```

```
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignLeft="@+id/latitudeLabel"
        android:layout_below="@+id/latitudeLabel"
        android:text="@string/longitude" />

    <TextView
        android:id="@+id/altitudeLabel"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignLeft="@+id/longitudeLabel"
        android:layout_below="@+id/longitudeLabel"
        android:text="@string/altitude" />

    <TextView
        android:id="@+id/xAxisValue"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignBottom="@+id/xAxisLabel"
        android:layout_marginLeft="56dp"
        android:layout_toRightOf="@+id/longitudeLabel"
        android:text="@string/empty" />

    <TextView
        android:id="@+id/yAxisValue"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignBaseline="@+id/yAxisLabel"
        android:layout_alignBottom="@+id/yAxisLabel"
        android:layout_alignLeft="@+id/xAxisValue"
        android:text="@string/empty" />

    <TextView
        android:id="@+id/zAxisValue"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_above="@+id/headingLabel"
        android:layout_alignLeft="@+id/yAxisValue"
        android:text="@string/empty" />

    <TextView
        android:id="@+id/headingValue"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignBaseline="@+id/headingLabel"
        android:layout_alignBottom="@+id/headingLabel"
        android:layout_alignLeft="@+id/zAxisValue"
        android:text="@string/empty" />
```

```
<TextView
    android:id="@+id/pitchValue"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignBaseline="@+id/pitchLabel"
    android:layout_alignBottom="@+id/pitchLabel"
    android:layout_alignLeft="@+id/headingValue"
    android:text="@string/empty" />

<TextView
    android:id="@+id/rollValue"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_above="@+id/latitudeLabel"
    android:layout_alignLeft="@+id/pitchValue"
    android:text="@string/empty" />

<TextView
    android:id="@+id/latitudeValue"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignBottom="@+id/latitudeLabel"
    android:layout_alignLeft="@+id/rollValue"
    android:text="@string/empty" />

<TextView
    android:id="@+id/longitudeValue"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignBaseline="@+id/longitudeLabel"
    android:layout_alignBottom="@+id/longitudeLabel"
    android:layout_alignLeft="@+id/latitudeValue"
    android:text="@string/empty" />

<TextView
    android:id="@+id/altitudeValue"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignBaseline="@+id/altitudeLabel"
    android:layout_alignBottom="@+id/altitudeLabel"
    android:layout_alignLeft="@+id/longitudeValue"
    android:text="@string/empty" />

<Button
    android:id="@+id/helpButton"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignLeft="@+id/altitudeLabel"
    android:layout_below="@+id/altitudeValue"
    android:layout_marginTop="15dp"
```

```
            android:text="@string/helpLabel" />

</RelativeLayout>
```

Once more, you will need to make sure that all the IDs are in order and you haven't made any typos anywhere because it will affect the entire layout. The only major difference from the layout in the first part of Chapter 3 is the addition of a Help button that will launch the Help dialog box. The Help menu option will do the same thing, but it is good to have a more easily visible option available.

Help Dialog Box

Now create another XML file in the `/res/layout` directory called `help.xml`. This will contain the layout design for the Help dialog box, which has a scrollable `TextView` to display the actual help text and a button to close the dialog box. Add the following to the `help.xml` file:

Listing 6-6. *The Help Dialog Box Layout File*

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

 <ScrollView
     android:id="@+id/ScrollView01"
     android:layout_width="wrap_content"
     android:layout_height="200px">

 <TextView
     android:text="@+id/TextView01"
     android:id="@+id/TextView01"
     android:layout_width="wrap_content"
     android:layout_height="wrap_content" />

 </ScrollView>

 <Button
     android:id="@+id/Button01"
     android:layout_below="@id/ScrollView01"
     android:layout_width="wrap_content"
     android:layout_height="wrap_content"
     android:layout_centerHorizontal="true"
     android:text="Okay" />

</RelativeLayout>
```

As you can see, it is a relatively simple `RelativeLayout` that is used in the dialog box layout. There is a `ScrollView` with a `TextView` inside it to hold the Help dialog box content and a `Button` to close the dialog box has been placed right at the bottom of the file.

Map Layout

Now we need to create the final layout file: the map layout. Create a `map.xml` in your `/res/layout` folder and add the following to it:

Listing 6-7. *The Map Layout File*

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical" >
<com.google.android.maps.MapView
    android:id="@+id/mapView"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:clickable="true"
    android:apiKey="<your_key_here>" />
</LinearLayout>
```

Getting API Keys

You will get an error if your project is not set to build against a Google APIs target. The other important thing here is the API key. This is assigned to you by Google on a certificate basis. It is generated from your certificate's MD5 hash, which you must submit in an online form. Android uses digital certificates to validate the install files for applications. If the signing certificates are a mismatch across the installed and new version, Android will throw a security exception and not let you update the install. The map API key is unique to each certificate. Therefore, if you plan to publish your application, you have to generate two API keys: one for your debug certificate (the one Eclipse signs your app with during the development and testing process) and one for your release certificate (the one you'll sign your app with before uploading it to an online market such as the Android Market). The steps are different for getting the MD5 of any key on different operating systems.

Getting the MD5 of Your Keys

For the debug key:

The debug key is normally in the following locations:

- Mac/Linux: ~/.android/debug.keystore
- Windows Vista/7: C:\Users\<user>\.android\debug.keystore
- Windows XP: C:\Documents and Settings\<user>\.android\debug.keystore

You will need to run the following command to get the MD5 out. The command uses the Keytool utility:

```
keytool -list -alias androiddebugkey -keystore <path_to_debug_keystore>.keystore
-storepass android -keypass android
```

For the signing key:

The signing key has no fixed location on the system. It is saved wherever you saved it or moved it during or after its creation. Run the following command to get its MD5, replacing `alias_name` with the alias on the key and `my-release-key` with the location to your key:

```
keytool -list -alias alias_name -keystore my-release-key.keystore
```

After you have extracted whatever keys' MD5 you wanted, navigate to `http://code.google.com/android/maps-api-signup.html` using your favorite web browser. Enter the MD5 and complete anything else you are asked to do. After submitting the form, you will be presented with the API key you need for your app to work.

Java Code

Now the XML setup is ready to go. All that is needed is the marker image and the actual code. Let's start with the marker image. It is called `ic_maps_current_position_indicator.png` and can be found in the `drawable-mdpi` and `drawable-hdpi` folders of this project's source. Make sure to copy each folder's image to its counterpart in your project and not switch them over by mistake.

Main Activity

With the image out of the way, we can get down to the code. We will start with the main `Activity`.

Imports and Variable Declarations

First, we'll take a look at the imports and class declaration and variable declarations:

Listing 6-8. *Main Activity Imports and Declarations*

```
package com.paar.ch6;

import android.app.Activity;
import android.app.Dialog;
import android.content.Intent;
import android.hardware.Camera;
import android.hardware.Sensor;
import android.hardware.SensorEvent;
import android.hardware.SensorEventListener;
import android.hardware.SensorManager;
import android.location.Location;
import android.location.LocationListener;
import android.location.LocationManager;
import android.os.Bundle;
import android.util.Log;
import android.view.Menu;
import android.view.MenuInflater;
import android.view.MenuItem;
import android.view.SurfaceHolder;
import android.view.SurfaceView;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.TextView;

public class ASimpleAppUsingARActivity extends Activity {
        SurfaceView cameraPreview;
        SurfaceHolder previewHolder;
        Camera camera;
        boolean inPreview;

        final static String TAG = "PAAR";
        SensorManager sensorManager;

        int orientationSensor;
        float headingAngle;
```

```
    float pitchAngle;
    float rollAngle;

    int accelerometerSensor;
    float xAxis;
    float yAxis;
    float zAxis;

    LocationManager locationManager;
    double latitude;
    double longitude;
    double altitude;

    TextView xAxisValue;
    TextView yAxisValue;
    TextView zAxisValue;
    TextView headingValue;
    TextView pitchValue;
    TextView rollValue;
    TextView altitudeValue;
    TextView latitudeValue;
    TextView longitudeValue;

    Button button;
```

The import statements and class declaration are standard Java, and the variables have been named to describe their function. Let's move on to the different methods in the class now.

onCreate() Method

The first method of the app, `onCreate()`, does a lot of things. It sets the `main.xml` file as the `Activity` view. It then gets the location and sensor system services. It registers listeners for the accelerometer and orientation sensors and the global positioning system (GPS). It then does part of the camera initialization (the rest of it is done later on). Finally, it gets references to nine of the `TextViews` so that they can be updated later on in the app and gets a reference to the Help button and sets its `onClickListener`. The code for this method is as follows:

Listing 6-9. *Main Activity's onCreate()*

```
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);

    locationManager = (LocationManager) getSystemService(LOCATION_SERVICE);
```

```
    locationManager.requestLocationUpdates(LocationManager.GPS_PROVIDER, 2000,
2, locationListener);

    sensorManager = (SensorManager) getSystemService(SENSOR_SERVICE);
    orientationSensor = Sensor.TYPE_ORIENTATION;
    accelerometerSensor = Sensor.TYPE_ACCELEROMETER;
    sensorManager.registerListener(sensorEventListener, sensorManager
.getDefaultSensor(orientationSensor), SensorManager.SENSOR_DELAY_NORMAL);
    sensorManager.registerListener(sensorEventListener, sensorManager
.getDefaultSensor(accelerometerSensor), SensorManager.SENSOR_DELAY_NORMAL);

    inPreview = false;

    cameraPreview = (SurfaceView)findViewById(R.id.cameraPreview);
    previewHolder = cameraPreview.getHolder();
    previewHolder.addCallback(surfaceCallback);
    previewHolder.setType(SurfaceHolder.SURFACE_TYPE_PUSH_BUFFERS);

    xAxisValue = (TextView) findViewById(R.id.xAxisValue);
    yAxisValue = (TextView) findViewById(R.id.yAxisValue);
    zAxisValue = (TextView) findViewById(R.id.zAxisValue);
    headingValue = (TextView) findViewById(R.id.headingValue);
    pitchValue = (TextView) findViewById(R.id.pitchValue);
    rollValue = (TextView) findViewById(R.id.rollValue);
    altitudeValue = (TextView) findViewById(R.id.altitudeValue);
    longitudeValue = (TextView) findViewById(R.id.longitudeValue);
    latitudeValue = (TextView) findViewById(R.id.latitudeValue);
    button = (Button) findViewById(R.id.helpButton);
    button.setOnClickListener(new OnClickListener() {
            public void onClick(View v) {
          showHelp();
        }
    });
}
```

LocationListener

Next in the code is the `LocationListener`, which listens for location updates from the location services (the GPS, in this case). Upon receiving an update from the GPS, it updates the local variables with new information, prints out the new information to the `LogCat`, and updates three of the `TextViews` with the new information. It also contains autogenerated method stubs for methods that aren't used in the app.

Listing 6-10. *LocationListener*

```
LocationListener locationListener = new LocationListener() {
    public void onLocationChanged(Location location) {
            latitude = location.getLatitude();
            longitude = location.getLongitude();
            altitude = location.getAltitude();

            Log.d(TAG, "Latitude: " + String.valueOf(latitude));
            Log.d(TAG, "Longitude: " + String.valueOf(longitude));
            Log.d(TAG, "Altitude: " + String.valueOf(altitude));

            latitudeValue.setText(String.valueOf(latitude));
            longitudeValue.setText(String.valueOf(longitude));
            altitudeValue.setText(String.valueOf(altitude));
    }

            public void onProviderDisabled(String arg0) {
                    // TODO Auto-generated method stub

            }

            public void onProviderEnabled(String arg0) {
                    // TODO Auto-generated method stub

            }

            public void onStatusChanged(String arg0, int arg1, Bundle arg2) {
                    // TODO Auto-generated method stub

            }
};
```

Launching the Map

Up next for explanation is the `launchFlatBack()` method. This method is called by the `SensorEventListener` whenever the condition for the phone being more or less parallel to the ground is met. This method then launches the map.

Listing 6-11. *launchFlatBack()*

```
public void launchFlatBack() {
    Intent flatBackIntent = new Intent(this, FlatBack.class);
    startActivity(flatBackIntent);
}
```

Options Menu

The Options menu is created and used by overriding the `onCreateOptionsMenu()` and `onOptionsItemSelected()` methods. The first one creates it from the menu resource (`main_menu.xml`), and the second one listens for click events on the menu. If the Help item has been clicked, it calls the appropriate method that will show the Help dialog box.

Listing 6-12. *onCreateOptionsMenu() and onOptionsItemSelected()*

```
@Override
public boolean onCreateOptionsMenu(Menu menu) {
    MenuInflater inflater = getMenuInflater();
    inflater.inflate(R.menu.main_menu, menu);
    return true;
}

public boolean onOptionsItemSelected(MenuItem item) {
    // Handle item selection
    switch (item.getItemId()) {
    case R.id.help:
            showHelp();
    default:
        return super.onOptionsItemSelected(item);
    }
}
```

Showing the Help Dialog box

`showHelp()` is the appropriate method mentioned previously. It is called when the Help menu item is clicked.

Listing 6-13. *showHelp()*

```
public void showHelp() {
        final Dialog dialog = new Dialog(this);
        dialog.setContentView(R.layout.help);
        dialog.setTitle("Help");
        dialog.setCancelable(true);
        //there are a lot of settings, for dialog, check them all out!

        //set up text
        TextView text = (TextView) dialog.findViewById(R.id.TextView01);
        text.setText(R.string.help);
```

```
        //set up button
        Button button = (Button) dialog.findViewById(R.id.ButtonO1);
        button.setOnClickListener(new OnClickListener() {
        public void onClick(View v) {
                dialog.cancel();
            }
        });
        //now that the dialog is set up, it's time to show it
        dialog.show();
    }
```

Listening to the Sensors

Now we come to the `SensorEventListener`. There is an `if` statement that differentiates between the orientation sensor and accelerometer. Both of the sensor's updates are printing out to the `LogCat` and the appropriate `TextViews`. In addition, an `if` statement inside the orientation sensor part of the code decides whether the device is more or less parallel to the ground. There is a leeway of 14 degrees because it is unlikely that anyone will be able to hold the device perfectly parallel to the ground.

Listing 6-14. *SensorEventListener*

```
final SensorEventListener sensorEventListener = new SensorEventListener() {
    public void onSensorChanged(SensorEvent sensorEvent) {
            if (sensorEvent.sensor.getType() == Sensor.TYPE_ORIENTATION)
            {
                    headingAngle = sensorEvent.values[0];
                    pitchAngle = sensorEvent.values[1];
                    rollAngle = sensorEvent.values[2];

                    Log.d(TAG, "Heading: " + String.valueOf(headingAngle));
                    Log.d(TAG, "Pitch: " + String.valueOf(pitchAngle));
                    Log.d(TAG, "Roll: " + String.valueOf(rollAngle));

                    headingValue.setText(String.valueOf(headingAngle));
                    pitchValue.setText(String.valueOf(pitchAngle));
                    rollValue.setText(String.valueOf(rollAngle));

                    if (pitchAngle < 7 && pitchAngle > -7 && rollAngle < 7 &&
 rollAngle > -7)
                    {
                            launchFlatBack();
                    }
            }
```

```
            else if (sensorEvent.sensor.getType() == Sensor.TYPE_ACCELEROMETER)
            {
                    xAxis = sensorEvent.values[0];
                    yAxis = sensorEvent.values[1];
                    zAxis = sensorEvent.values[2];

                    Log.d(TAG, "X Axis: " + String.valueOf(xAxis));
                    Log.d(TAG, "Y Axis: " + String.valueOf(yAxis));
                    Log.d(TAG, "Z Axis: " + String.valueOf(zAxis));

                    xAxisValue.setText(String.valueOf(xAxis));
                    yAxisValue.setText(String.valueOf(yAxis));
                    zAxisValue.setText(String.valueOf(zAxis));
            }
    }

    public void onAccuracyChanged (Sensor senor, int accuracy) {
            //Not used
    }
};
```

onResume(), onPause(), and onDestroy() methods

We override the `onResume()`, `onPause()`, and `onDestroy()` methods so that we can release and reacquire the `SensorEventListener`, `LocationListener`, and `Camera`. We release them when the app is paused (the user switches to another app) or destroyed (Android terminates the process) to save the user's battery and use up fewer system resources. Also, only one app can use the `Camera` at a time, so by releasing it we make it available to the other applications.

Listing 6-15. *onResume(), onPause() and onDestroy()*

```
@Override
public void onResume() {
  super.onResume();
  locationManager.requestLocationUpdates(LocationManager.GPS_PROVIDER, 2000, 2,
locationListener);
  sensorManager.registerListener(sensorEventListener, sensorManager
.getDefaultSensor(orientationSensor), SensorManager.SENSOR_DELAY_NORMAL);
  sensorManager.registerListener(sensorEventListener, sensorManager
.getDefaultSensor(accelerometerSensor), SensorManager.SENSOR_DELAY_NORMAL);
  //Camera camera;
}

@Override
public void onPause() {
  if (inPreview) {
    camera.stopPreview();
```

```
  }
  locationManager.removeUpdates(locationListener);
  sensorManager.unregisterListener(sensorEventListener);
  if (camera != null)
  {
      camera.release();
      camera=null;
  }
  inPreview=false;

  super.onPause();
}

@Override
public void onDestroy() {
    camera.release();
    camera=null;
}
```

Managing the SurfaceView and Camera

These last four methods deal with managing the `SurfaceView`, its `SurfaceHolder`, and the `Camera`.

- The `getBestPreviewSize()` method gets a list of available preview sizes and chooses the best one.
- The `surfaceCallback` is called when the `SurfaceView` is ready. The camera is set up and opened there.
- The `surfaceChanged()` method is called if any changes are made by Android to the `SurfaceView` (after an orientation change, for example).
- The `surfaceDestroyed()` method is called when the `SurfaceView` is, well, destroyed.

Listing 6-16. *getBestPreviewSize(), surfaceCallback(), surfaceChanged() and surfaceDestroyed()*

```
    private Camera.Size getBestPreviewSize(int width, int height,
Camera.Parameters parameters) {
        Camera.Size result=null;

        for (Camera.Size size : parameters.getSupportedPreviewSizes()) {
                if (size.width<=width && size.height<=height) {
                        if (result==null) {
                                result=size;
                        }
```

```
                    else {
                            int resultArea=result.width*result.height;
                            int newArea=size.width*size.height;

                            if (newArea>resultArea) {
                                    result=size;
                            }
                    }
            }
        }

        return(result);
    }

    SurfaceHolder.Callback surfaceCallback=new SurfaceHolder.Callback() {
        public void surfaceCreated(SurfaceHolder holder) {
                if (camera == null) {
                camera = Camera.open();
                }
                try {
                        camera.setPreviewDisplay(previewHolder);
                }
                catch (Throwable t) {
                        Log.e(TAG, "Exception in setPreviewDisplay()", t);
                }
        }

        public void surfaceChanged(SurfaceHolder holder, int format, int width,
int height) {
                Camera.Parameters parameters=camera.getParameters();
                Camera.Size size=getBestPreviewSize(width, height, parameters);

                if (size!=null) {
                        parameters.setPreviewSize(size.width, size.height);
                        camera.setParameters(parameters);
                        camera.startPreview();
                        inPreview=true;
                }
        }

        public void surfaceDestroyed(SurfaceHolder holder) {
                if (camera != null) {
                camera.stopPreview();
                camera.setPreviewCallback(null);
                camera.release();
                camera = null;
            }
        }
```

```
    };
}
```

This is the end of the first Java file. This file works with the GPS and sensors to get updates and then displays them via `TextViews` and `LogCat` outputs.

FlatBack.java

Now we are going to work on `FlatBack.java`. This `Activity` is called whenever the phone is held parallel to the ground and displays your current location on a map. The class won't make much sense right now because part of the work is done in `FixLocation`.

Imports, Variable Declarations and onCreate() Method

In the `onCreate()` in this `Activity`, we repeat the `SensorManager` stuff as always in the beginning. We need the sensor inputs here because we want to switch back to the `CameraView` when the device is no longer parallel to the ground. After that, we get a reference to the `MapView` (the one in the XML layout), tell Android that we will not be implementing our own zoom controls, pass the `MapView` to `FixLocation`, add the location overlay to the `MapView`, tell it to update, and call the custom method that will zoom it to the user's location.

Listing 6-17. *Flatback.java's imports, declarations and onCreate()*

```
package com.paar.ch6;

import com.google.android.maps.GeoPoint;
import com.google.android.maps.MapActivity;
import com.google.android.maps.MapView;
import com.google.android.maps.MyLocationOverlay;

import android.hardware.Sensor;
import android.hardware.SensorEvent;
import android.hardware.SensorEventListener;
import android.hardware.SensorManager;
import android.os.Bundle;
import android.util.Log;
import android.view.Menu;
import android.view.MenuInflater;
import android.view.MenuItem;
```

```
public class FlatBack extends MapActivity{
        private MapView mapView;
    private MyLocationOverlay myLocationOverlay;
        final static String TAG = "PAAR";
        SensorManager sensorManager;

        int orientationSensor;
        float headingAngle;
        float pitchAngle;
        float rollAngle;
/** Called when the activity is first created. */
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);

    // main.xml contains a MapView
    setContentView(R.layout.map);

    sensorManager = (SensorManager) getSystemService(SENSOR_SERVICE);
    orientationSensor = Sensor.TYPE_ORIENTATION;
    sensorManager.registerListener(sensorEventListener, sensorManager
.getDefaultSensor(orientationSensor), SensorManager.SENSOR_DELAY_NORMAL);

    // extract MapView from layout
            mapView = (MapView) findViewById(R.id.mapView);
            mapView.setBuiltInZoomControls(true);

            // create an overlay that shows our current location
            myLocationOverlay = new FixLocation(this, mapView);

            // add this overlay to the MapView and refresh it
            mapView.getOverlays().add(myLocationOverlay);
            mapView.postInvalidate();

            // call convenience method that zooms map on our location
            zoomToMyLocation();
}
```

onCreateOptionsMenu() and onOptionsItemSelected() methods

Next are the two Options menu–related methods, which create the Options menu, watch for clicks, distinguish between which option was clicked, and execute the appropriate action on the map.

Listing 6-18. *onCreateOptionsMenu() and onOptionsItemSelected()*

```
@Override
public boolean onCreateOptionsMenu(Menu menu) {
    MenuInflater inflater = getMenuInflater();
    inflater.inflate(R.menu.map_toggle, menu);
    return true;
}

public boolean onOptionsItemSelected(MenuItem item) {
    // Handle item selection
    switch (item.getItemId()) {
    case R.id.map:
        if (mapView.isSatellite() == true) {
                mapView.setSatellite(false);
                mapView.setStreetView(true);
        }
        return true;
    case R.id.sat:
        if (mapView.isSatellite()==false){
                mapView.setSatellite(true);
                mapView.setStreetView(false);
        }
        return true;
    case R.id.both:
        mapView.setSatellite(true);
        mapView.setStreetView(true);
    default:
        return super.onOptionsItemSelected(item);
    }
}
```

SensorEventListener

Next is the `SensorEventListener`, which is similar to that in the previous class, except that it checks to see if the phone is no longer parallel to the ground and then calls the custom method that will take us back to the camera preview.

Listing 6-19. *SensorEventListener*

```
final SensorEventListener sensorEventListener = new SensorEventListener() {
        public void onSensorChanged(SensorEvent sensorEvent) {
                if (sensorEvent.sensor.getType() == Sensor.TYPE_ORIENTATION)
                {
                        headingAngle = sensorEvent.values[0];
                        pitchAngle = sensorEvent.values[1];
                        rollAngle = sensorEvent.values[2];
```

```
                        Log.d(TAG, "Heading: " + String.valueOf(headingAngle));
                        Log.d(TAG, "Pitch: " + String.valueOf(pitchAngle));
                        Log.d(TAG, "Roll: " + String.valueOf(rollAngle));

                        if (pitchAngle > 7 || pitchAngle < -7 || rollAngle > 7
|| rollAngle < -7)
                        {
                                launchCameraView();
                        }
                }
}

        public void onAccuracyChanged(Sensor arg0, int arg1) {
                // TODO Auto-generated method stub

        }
};
```

launchCameraView() Method

The `launchCameraView()` method finishes the current `activity` so that we can get to the camera preview without any problems. An `Intent` is commented out that seems to do the same thing. I have commented it out because although it does end up launching the camera preview, it does so by creating another instance of that `activity`, which will give an error because the camera will already be in use by the first instance of the activity. Therefore, it is best to return to the previous instance.

Listing 6-20. *launchCameraView()*

```
public void launchCameraView() {
        finish();
        //Intent cameraView = new Intent(this, ASimpleAppUsingARActivity.class);
        //startActivity(cameraView);
}
```

onResume() and onPause() Methods

Then are the `onResume()` and `onPause()` methods, which enable and disable location updates to save resources.

Listing 6-21. *onResume() and onPause()*

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

```
            myLocationOverlay.enableMyLocation();
    }

    @Override
    protected void onPause() {
            super.onPause();
            myLocationOverlay.disableMyLocation();
    }
```

zoomToMyLocation() Method

After this is the custom `zoomToMyLocation()` method. This method applies a zoom level of 10 to the current location on the map.

Listing 6-22. *zoomToMyLocation()*

```
private void zoomToMyLocation() {
        GeoPoint myLocationGeoPoint = myLocationOverlay.getMyLocation();
        if(myLocationGeoPoint != null) {
                mapView.getController().animateTo(myLocationGeoPoint);
                mapView.getController().setZoom(10);
        }
}
```

isRouteDisplayed() Method

Finally is the Boolean method `isRouteDisplayed()`. Because it is not used in the app, it is set to false.

Listing 6-23. *isRouteDisplayed()*

```
    protected boolean isRouteDisplayed() {
            return false;
    }
}
```

This brings us to the end of `FlatBack.java`. Notice that most of the actual location work seems to be done in `FixLocation.java`. Before you get tired of Eclipse giving you errors at its references, we'll move on and write that class.

FixLocation.java

Now is the time to get to know what `FixLocation` is for. The `MyLocationOverlay` class has severe bugs in some Android-powered devices, most notable of which is the Motorola DROID. `FixLocation` tries to use the standard

`MyLocationOverlay`, but if it fails to work properly, it implements its own version of the same that will produce the same result. Source code first, followed by the explanation:

Listing 6-24. *FixLocation.java*

```
package com.paar.ch6;

import android.content.Context;
import android.graphics.Canvas;
import android.graphics.Paint;
import android.graphics.Point;
import android.graphics.Paint.Style;
import android.graphics.drawable.Drawable;
import android.location.Location;

import com.google.android.maps.GeoPoint;
import com.google.android.maps.MapView;
import com.google.android.maps.MyLocationOverlay;
import com.google.android.maps.Projection;


public class FixLocation extends MyLocationOverlay {

        private boolean bugged = false;

        private Drawable drawable;
        private Paint accuracyPaint;
        private Point center;
        private Point left;
        private int width;
        private int height;

        public FixLocation(Context context, MapView mapView) {
                super(context, mapView);
        }

        @Override
        protected void drawMyLocation(Canvas canvas, MapView mapView,
                        Location lastFix, GeoPoint myLocation, long when) {
                if(!bugged) {
                        try {
                                super.drawMyLocation(canvas, mapView, lastFix,
 myLocation, when);
                        } catch (Exception e) {
                                // we found a buggy phone, draw the location
icons ourselves
                                bugged = true;
                        }
                }
```

```
                if(bugged) {
                        if(drawable == null) {

                                accuracyPaint = new Paint();
                                accuracyPaint.setAntiAlias(true);
                                accuracyPaint.setStrokeWidth(2.0f);

                                drawable = mapView.getContext()
.getResources().getDrawable(R.drawable.ic_maps_indicator_current_position);
                                width = drawable.getIntrinsicWidth();
                                height = drawable.getIntrinsicHeight();
                                center = new Point();
                                left = new Point();
                        }

                        Projection projection = mapView.getProjection();
                        double latitude = lastFix.getLatitude();
                        double longitude = lastFix.getLongitude();
                        float accuracy = lastFix.getAccuracy();

                        float[] result = new float[1];

                        Location.distanceBetween(latitude, longitude, latitude,
longitude + 1, result);
                        float longitudeLineDistance = result[0];

                        GeoPoint leftGeo = new GeoPoint((int)(latitude*1e6),
 (int)((longitude-accuracy/longitudeLineDistance)*1e6));
                        projection.toPixels(leftGeo, left);
                        projection.toPixels(myLocation, center);
                        int radius = center.x - left.x;

                        accuracyPaint.setColor(0xff6666ff);
                        accuracyPaint.setStyle(Style.STROKE);
                        canvas.drawCircle(center.x, center.y, radius,
accuracyPaint);

                        accuracyPaint.setColor(0x186666ff);
                        accuracyPaint.setStyle(Style.FILL);
                        canvas.drawCircle(center.x, center.y, radius,
accuracyPaint);

                        drawable.setBounds(center.x - width/2, center.y -
height/2, center.x + width/2, center.y + height/2);
                        drawable.draw(canvas);
                }
        }
}
```

To begin with, we have the method that receives the call from `FlatBack`. We then override the `drawMyLocation()` method. In the implementation, we check to see whether it is bugged or not. We try to let it run the normal course, but if we get an exception, we set bugged to true and then proceed to execute our own implementation of the work.

If it does turn out to be bugged, we set up the paints, get a reference to the drawable, get the location, calculate the accuracy, and then draw the marker, along with the accuracy circle onto the map. The accuracy circle means that the location is not 100% accurate and that you are somewhere inside that circle.

This brings us to the end of this sample application. Now take a quick look on how to run the application and see some screenshots to go with it.

Running the App

The app should compile without any errors or warnings. If you do happen to get an error, go through the common errors section that follows.

When debugging on a device, you might see an orange triangle like the one shown in Figure 6-2.

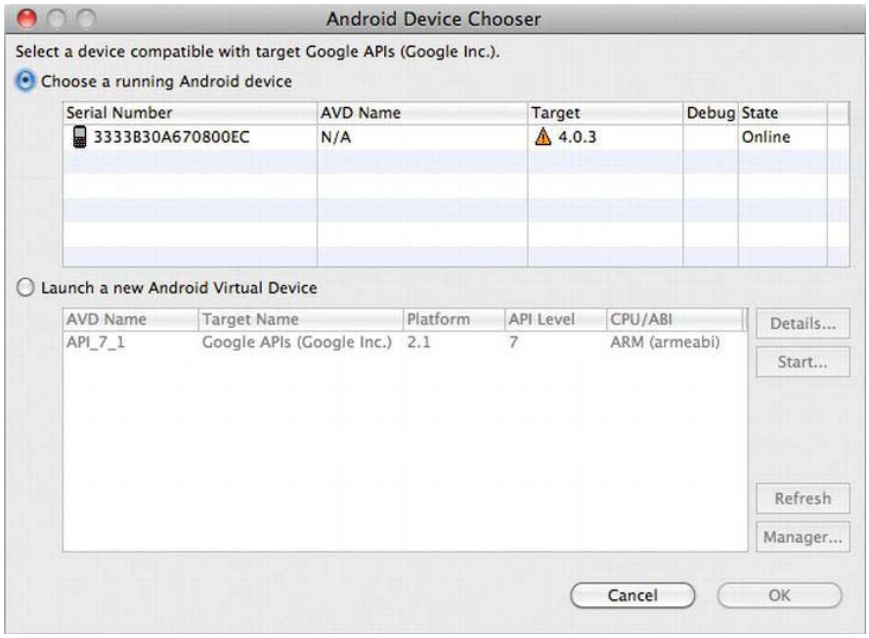

Figure 6-2. *Orange warning triangle*

This triangle simply means that Eclipse could not confirm that the Google APIs are installed on your device. If your Android device came preloaded with Android Market, you can be pretty sure that it has the Google APIs installed.

When you do run the app, you should see something like the screenshots in Figure 6-3 to Figure 6-5.

Figure 6-3. *Augmented reality view of the app*

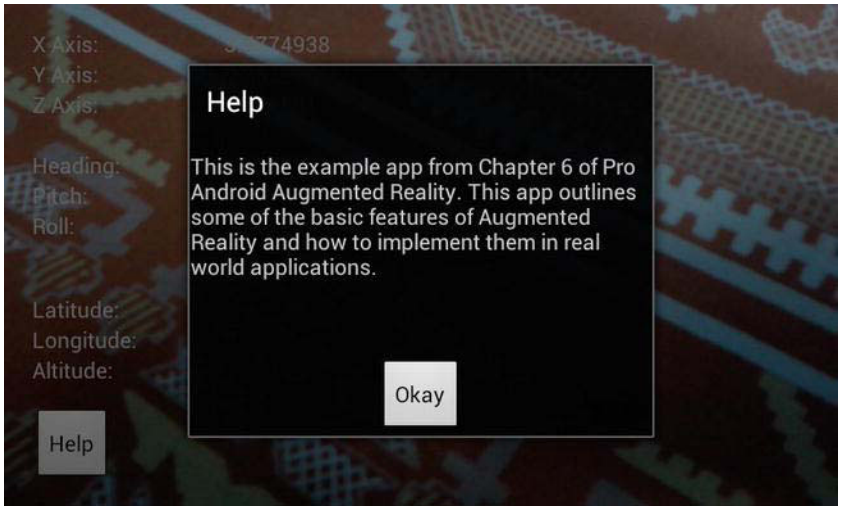

Figure 6-4. *Help dialog box for the app*

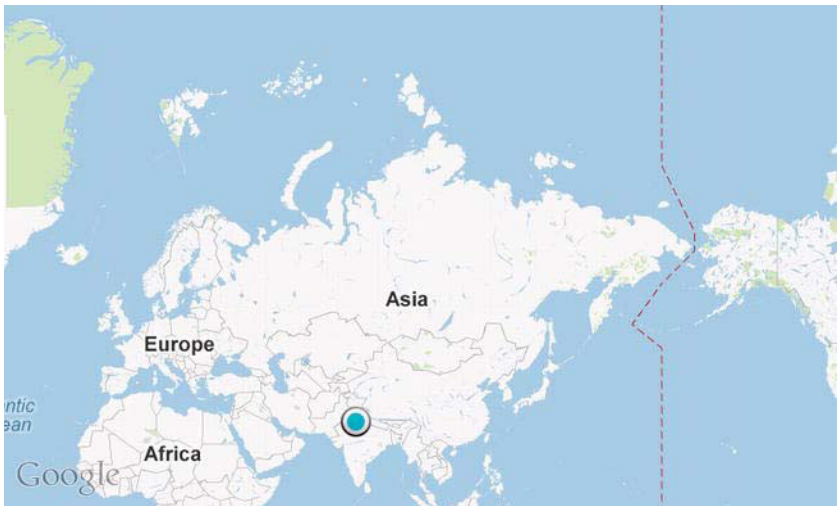

Figure 6-5. *Map shown when device is parallel to the ground*

The LogCat should look similar to Figure 6-6.

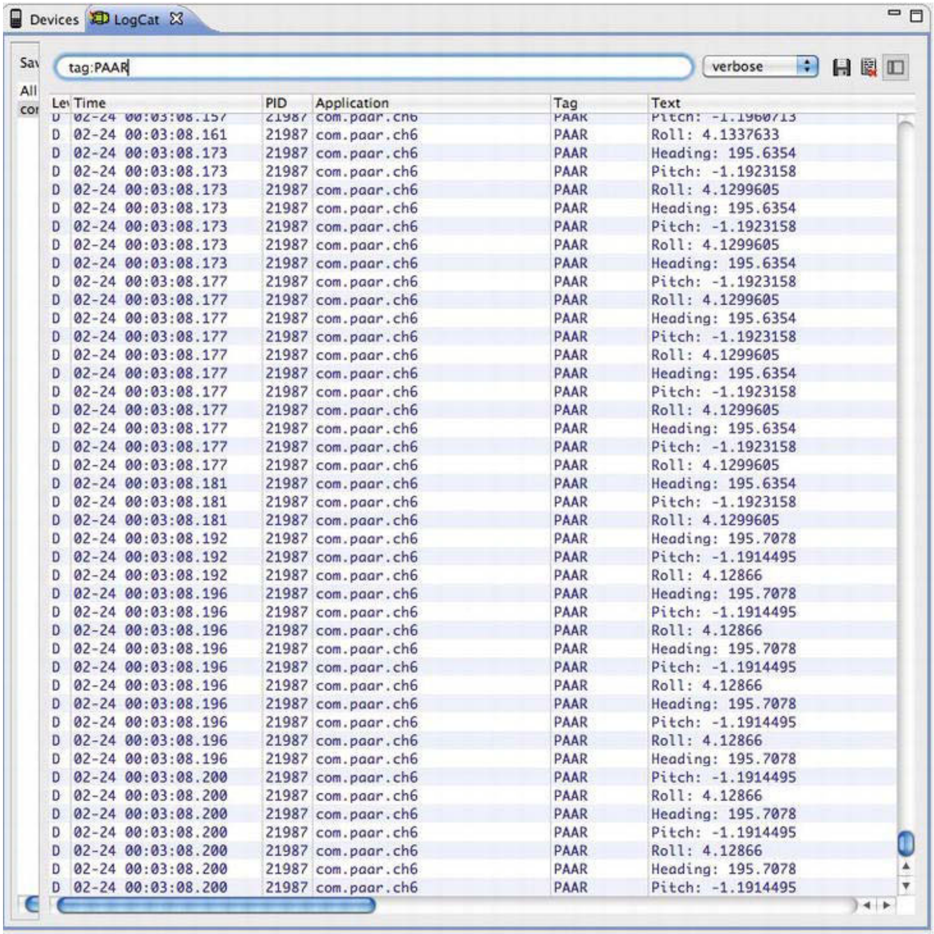

Figure 6-6. *Screenshot of the LogCat for the app*

Common errors

Following are four common errors from the app. For anything else, check with the android-developers Google group or stackoverflow.com for help.

- **Failed to connect to camera service:** The only time I have ever seen this error is when something else is already using the camera. This error can be resolved in several ways, and stackoverflow.com should be able to give you the answers.

- **Anything that looks related to the Map:** This is most likely because you are not building against Google APIs, or because you forgot to declare `<uses-library>` in the AndroidManifest or are using an incorrect API key.
- **Anything that looks related to R.something:** This error is most likely due to a mistake or mismatch in your XML files, or to a missing drawable. You can fix it by checking your XML files. If you are sure that they are correct and that your marker drawable is in place, try building from scratch by either compiling after deleting your /bin directory or using Project -> Clean.
- **Security exceptions:** These will most likely be due to a missing permission in your `AndroidManifest`.

Summary

This brings us to the end of the first example app in this book, which demonstrates how to do the following:

- Augment sensor information over a live camera preview, using the standard Android SDK
- Launch an `Activity` when the device is held in a particular manner, in this case parallel to the ground
- Display the user's current location on a map using the Google Maps APIs
- Implement a fix in case the Maps API is broken on a device

This app will be built upon in the next chapter to act as a simple navigational app with AR.