Chapter 7

A Basic Navigational App Using Augmented Reality, the GPS, and Maps

In Chapter 6, we designed a simple AR app that would display sensor data over a camera preview and display a location on a map if the device was held parallel to the ground. In this chapter, we will extend this app so that it can be used for basic navigational purposes.

The New App

The extended version of the app will have the following features:

- When held nonparallel to the ground, the app will display a camera preview with various data overlayed onto it.
- When held parallel to the ground, a map will be launched. The user can locate the desired location on the map and enable the Tap To Set mode. Once the mode is enabled, the user can tap the desired location. This location is saved.

- When the camera preview is displayed again, a bearing to the desired location is calculated from the GPS's data, along with the distance between the two locations.
- The bearing and distance are updated every time a new location fix is received.

This app gives you every calculation you'll need, in case you ever want to extend it to add a compass and do other things.

Now without further ado, let's get to the coding.

Create a new project to begin with. In the example, the package name is com.paar.ch7, and the build target is the Google APIs, for android 2.1. We must target the Google APIs SDK as we are using Google Maps.

First, duplicate the Chapter 6 project. Change the name of the main `Activity` (the one with the camera preview) to whatever you want, as long as you remember to update the manifest to go with it. Also, because this is a new project, you'll probably want another package name as well.

Updated XML files

First, we need to update some of our XML files. Let's start with `strings.xml`:

Listing 7-1. *Updated strings.xml*

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="hello">Hello World, ASimpleAppUsingARActivity!</string>
    <string name="app_name">A Slightly More Complex AR App</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 7 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. This app includes
a basic system that can be used for navigation. To make use of this system, put
the app in the map mode by holding the device flat. Then enable \"Enable tap to
set\" from the menu option after you have located the place you want to go to.
After that, switch back to camera mode. If a reliable GPS fix is available, you
```

```
will be given your current bearing to that location. The bearing will be updated
every time a new location fix is received.</string>
    <string name="helpLabel">Help</string>
    <string name="go">Go</string>
    <string name="bearingLabel">Bearing:</string>
    <string name="distanceLabel">Distance:</string>
</resources>
```

The “`Distance:`” over here will be used as the label where we tell the user the distance from his/her current location to the location selected, as the crow flies. The crows’s path is the distance in a direct line from Point A to Point B. It does not tell the distance via road or any other such path. If you remember high school physics, it’s pretty much like displacement. It is the shortest distance from Point A to Point B, regardless of whether that distance is actually traversable or not.

You’ll notice a few new strings and an increase in the size of the help string. Apart from that, our `strings.xml` is mainly the same. Next, we need to update our `map_toggle.xml` from the `/res/menu` folder. We need to add a new option to allow the user to set the location.

Listing 7-2. *Updated 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>
    <item
        android:id="@+id/toggleSetDestination"
        android:title="Enable Tap to Set">
    </item>
</menu>
```

Our new menu option is “Enable Tap to Set.” This option will be used to allow the user to enable and disable the tap to set the functionality of our app. If we do not add a check, every time the user moves the map around or tries to zoom, a new location will be set. To avoid this, we make an enable/disable option.

Now for the final change in our biggest XML file: main.xml. We need to add two `TextViews` and move our help button a little. The code that follows shows only the updated parts. Anything not given here is exactly the same as in the previous chapter.

Listing 7-3. *Updated main.xml*

```
// Cut here

        <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" />

        <TextView
            android:id="@+id/textView1"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_alignLeft="@+id/altitudeLabel"
            android:layout_below="@+id/altitudeLabel"
            android:text="@string/bearingLabel" />

        <TextView
            android:id="@+id/bearingValue"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_alignLeft="@+id/altitudeValue"
            android:layout_below="@+id/altitudeValue"
            android:text="@string/empty" />

        <Button
            android:id="@+id/helpButton"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_alignParentBottom="true"
            android:layout_alignParentRight="true"
            android:text="@string/helpLabel" />

        <TextView
            android:id="@+id/distanceLabel"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_alignLeft="@+id/textView1"
            android:layout_below="@+id/textView1"
            android:text="@string/distanceLabel" />

        <TextView
        android:id="@+id/distanceValue"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignBottom="@+id/distanceLabel"
        android:layout_alignLeft="@+id/bearingValue"
        android:text="@string/empty" />

</RelativeLayout>
```

Seeing as even what we added follows the pattern of the previous chapter, I hope that this code is pretty self-explanatory. The `TextViews` with IDs that have "`label`" in them are the labels for the actual values. These will not be referenced from our Java code. The `TextViews` with "`value`" in their IDs will be updated dynamically from our Java code to display the values.

Updated Java files

Now we can get to the main Java code. Two out of three of our Java files need to be updated with new code.

In `FixLocation.java`, you need to update the package declaration to match the new one. That's the one and only change in that file.

Updates to FlatBack.java

Now let's move on to the next file that we need to update: `FlatBack.java`:

Listing 7-4. *Updated FlatBack.java*

```
package com.paar.ch7;

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.content.SharedPreferences;
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;
import android.view.MotionEvent;
import android.view.View;
import android.view.View.OnTouchListener;

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

    SharedPreferences prefs;
    SharedPreferences.Editor editor;

    int orientationSensor;
    float headingAngle;
    float pitchAngle;
    float rollAngle;
    String enteredAddress;
    boolean tapToSet;
/** 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);
    prefs = getSharedPreferences("PAARCH7", 0);
    editor = prefs.edit();
    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();

            mapView.setOnTouchListener(new OnTouchListener() {
```

```
            public boolean onTouch(View arg0, MotionEvent arg1) {

                if(tapToSet == true)
                {
                GeoPoint p = mapView.getProjection().fromPixels((int)
arg1.getX(), (int) arg1.getY());

                Log.d(TAG,"Latitude:" + String.valueOf(p.getLatitudeE6()/1e6));
                Log.d(TAG,"Longitude:" +
String.valueOf(p.getLongitudeE6()/1e6));
                float lat =(float) ((float) p.getLatitudeE6()/1e6);
                float lon = (float) ((float) p.getLongitudeE6()/1e6);
                editor.putFloat("SetLatitude", lat);
                editor.putFloat("SetLongitude", lon);
                editor.commit();
                return true;
                }
                return false;

            }

            });

}

@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);
        case R.id.toggleSetDestination:
            if(tapToSet == false)
            {
                tapToSet = true;
                item.setTitle("Disable Tap to Set");
            }
            else if(tapToSet == true)
            {
                tapToSet = false;
                item.setTitle("Enable Tap to Set");
                mapView.invalidate();
            }
        default:
            return super.onOptionsItemSelected(item);
        }
    }

    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) {

        }
    };

    public void launchCameraView() {
        finish();
    }

    @Override
        protected void onResume() {
                super.onResume();
            myLocationOverlay.enableMyLocation();
    }

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


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

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

Let's look at what's changed. First, we have some new variables at the top:

```
boolean tapToSet;
SharedPreferences prefs;
SharedPreferences.Editor editor;
```

The `boolean tapToSet` will tell us if the Tap To Set mode is enabled or not. The other two are the `SharedPreferences` related variables. We will be using `SharedPreferences` to store the user's set value because we will be accessing it from both activities of our class. Sure, we could use `startActivityForResult()` when launching the `MapActivity` and get the user's set value that way, but by using `SharedPreferences`, we can also keep the user's last used location, in case the app is started later and a new location is not set.

Next, we have added some new stuff to our `onCreate()` method. These two lines are responsible for getting access to our `SharedPreferences` and allowing us to edit them later on:

```
prefs = getSharedPreferences("PAARCH7", 0);
editor = prefs.edit();
```

PAARCH7 is the name of our preferences file, standing for **P**ro **A**ndroid **A**ugmented **R**eality **Ch**apter **7**. If you extend this app on your own and use `SharedPreferences` from multiple places at once, keep in mind that when editing the same preference file, the changes are visible to everyone instantaneously. On the first run, the PAARCH7 file does not exist, so Android creates it. The little

0 right after the comma tells Android that this file is private. The next line assigns the editor to be able to edit our preferences.

Now we have some more changes in our onCreate() method. We assign an onTouchListener() to our MapView:

```
mapView.setOnTouchListener(new OnTouchListener() {

        public boolean onTouch(View arg0, MotionEvent arg1) {

                if(tapToSet == true)
                {
                GeoPoint p =
mapView.getProjection().fromPixels((int)arg1.getX(),↵
 (int) arg1.getY());

                Log.d(TAG,"Latitude:" +String.valueOf(p.getLatitudeE6()/1e6));
                Log.d(TAG,"Longitude:" +String.valueOf(p.getLongitudeE6()/1e6));
                float lat =(float) ((float) p.getLatitudeE6()/1e6);
                float lon = (float) ((float) p.getLongitudeE6()/1e6);
                editor.putFloat("SetLatitude", lat);
                editor.putFloat("SetLongitude", lon);
                editor.commit();
                return true;
                }
                return false;

                }

            });
```

In this onTouchListener(), we filter each touch. If Tap To Set mode is enabled, we capture the touch event and get the latitude and longitude. Then we convert the doubles we received from the touched GeoPoint into floats, so that we can write them to our preferences, which is exactly what we do. We put both the floats in our preferences file and then call editor.commit() to write them to the file. We return true if we capture the touch and false if we don't. By returning false, we allow the MapView to continue on its normal course of scrolling around and zooming in and out.

The last thing we need to do is alter our onOptionsItemSelected() method to allow for the Enable Tap To Set option.

```
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);
        case R.id.toggleSetDestination:
            if(tapToSet == false)
            {
                tapToSet = true;
                item.setTitle("Disable Tap to Set");
            }
            else if(tapToSet == true)
            {
                tapToSet = false;
                item.setTitle("Enable Tap to Set");
                mapView.invalidate();
            }
        default:
            return super.onOptionsItemSelected(item);
        }
}
```

We check to see whether `tapToSet` is `false` first. If so, we set it to `true` and change the title to "Disable Tap to Set." If it is `true`, we change it to `false` and change the title back to "Enable Tap to Set."

That wraps up this file.

The main Activity file

Now we are left only with our main file.

We'll begin by looking at the new variables.

Listing 7-5. *Package declaration, imports, and new variables*

```
package com.paar.ch7;

import android.app.Activity;
import android.app.Dialog;
import android.content.Intent;
import android.content.SharedPreferences;
import android.hardware.Camera;

…

    double bearing;
    double distance;

    float lat;
    float lon;

    Location setLoc;
    Location locationInUse;

    SharedPreferences prefs;

…

    TextView bearingValue;
    TextView distanceValue;
```

The two floats `lat` and `lon` will store the values that we saved into our `SharedPreferences` in the `MapActivity` when they are read from the file. The Location `setLoc` will be passed the aforementioned latitude and longitude to create a new `Location`. We will then use that location to get the user's bearing. `locationInUse` is a copy of our GPS's location fix. The two `TextViews` will display our results. The doubles `bearing` and `distance` will store our results.

Now we need to make some changes to our `onCreate()` method.

Listing 7-6. *Updated onCreate()*

```
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);
    setLoc = new Location("");

    prefs = getSharedPreferences("PAARCH7", 0);


    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);
    bearingValue = (TextView) findViewById(R.id.bearingValue);
    distanceValue = (TextView) findViewById(R.id.distanceValue);
    button = (Button) findViewById(R.id.helpButton);
    button.setOnClickListener(new OnClickListener() {
        public void onClick(View v) {
          showHelp();
        }
    });
}
```

The line `prefs = getSharedPreferences("PAARCH7", 0);` gets us access to our `SharedPreferences`. The next new lines (`bearingValue = (TextView) findViewById(R.id.bearingValue);` and `distanceValue = (TextView) findViewById(R.id.distanceValue);`) get a reference to our new `TextViews` and will allow us to update them later on.

Now we must update the `LocationListener` so that our calculations are updated as and when the location is updated. This is relatively simple.

Listing 7-7. *Updated LocationListener*

```
LocationListener locationListener = new LocationListener() {
    public void onLocationChanged(Location location) {
       locationInUse = 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));

          lat = prefs.getFloat("SetLatitude", 0.0f);
          lon = prefs.getFloat("SetLongitude", 0.0f);


          setLoc.setLatitude(lat);
          setLoc.setLongitude(lon);
          if(locationInUse != null)
          {
          bearing = locationInUse.bearingTo(setLoc);
          distance = locationInUse.distanceTo(setLoc);
          bearingValue.setText(String.valueOf(bearing));
          distanceValue.setText(String.valueOf(distance));
          }
    }
```

Our modifications include getting the values from the SharedPreferences and checking to see whether we have a valid location; if there is a valid location, we calculate and display the bearing and distance. If there isn't one, we do nothing.

We need to repeat somewhat the same thing in our onResume(). This is because when we switch to the MapActivity and set the location, we will come back to the camera preview. This means that the onResume() will be invoked, thus making it the perfect place to update our locations and calculations.

Listing 7-8. *Updated onResume*

```
@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;

  lat = prefs.getFloat("SetLatitude", 0.0f);
  lon = prefs.getFloat("SetLongitude", 0.0f);

  setLoc.setLatitude(lat);
  setLoc.setLongitude(lon);
  if(locationInUse != null)
  {
  bearing = locationInUse.bearingTo(setLoc);
  distance = locationInUse.distanceTo(setLoc);
  bearingValue.setText(String.valueOf(bearing));
  distanceValue.setText(String.valueOf(distance));
  }
  else
  {
      bearingValue.setText("Unable to get your location reliably.");
      distanceValue.setText("Unable to get your location reliably.");
  }
}
```

Pretty much the exact same thing, except that we also give a message if we can't get the location to calculate the distance and bearing.

Updated AndroidManifest

This pretty much wraps up this example app. All the files not given here are exactly the same as those in Chapter 6. The final update is to `AndroidManifest.xml`, in which the `Activity` declaration has been edited:

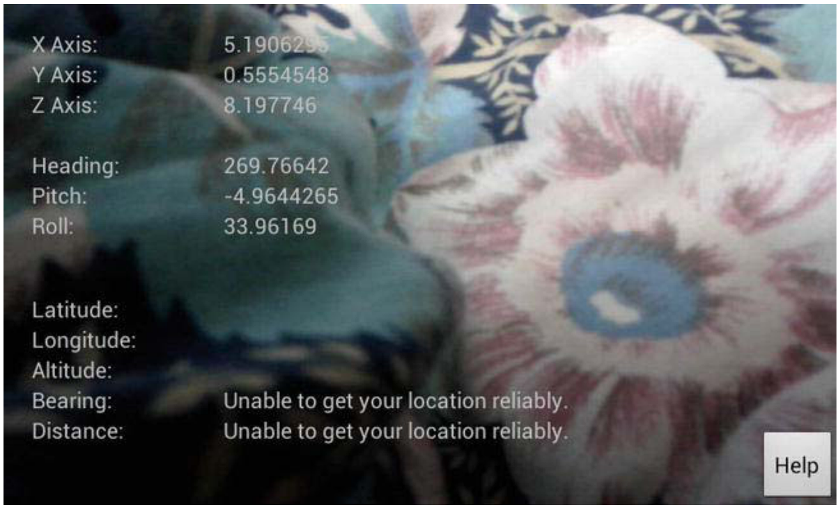
Listing 7-9. *Updated AndroidManifest.xml*

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.paar.ch7"
    android:versionCode="1"
    android:versionName="1.0" >

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

The Completed App

Figures 7-1–7-5 show the app in the augmented reality mode, with the Help dialog box open and with the map open.

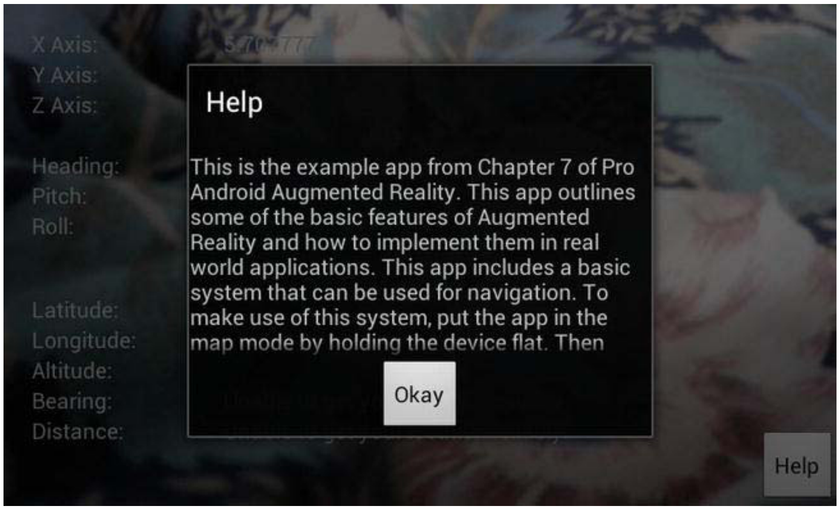


```
X Axis:      5.1906295
Y Axis:      0.5554548
Z Axis:      8.197746

Heading:     269.76642
Pitch:       -4.9644265
Roll:        33.96169


Latitude:
Longitude:
Altitude:
Bearing:     Unable to get your location reliably.
Distance:    Unable to get your location reliably.
```

Figure 7-1. *The app on startup, with no GPS fix*



Help

This is the example app from Chapter 7 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. This app includes a basic system that can be used for navigation. To make use of this system, put the app in the map mode by holding the device flat. Then

Figure 7-2. *The app with the Help dialog box open*

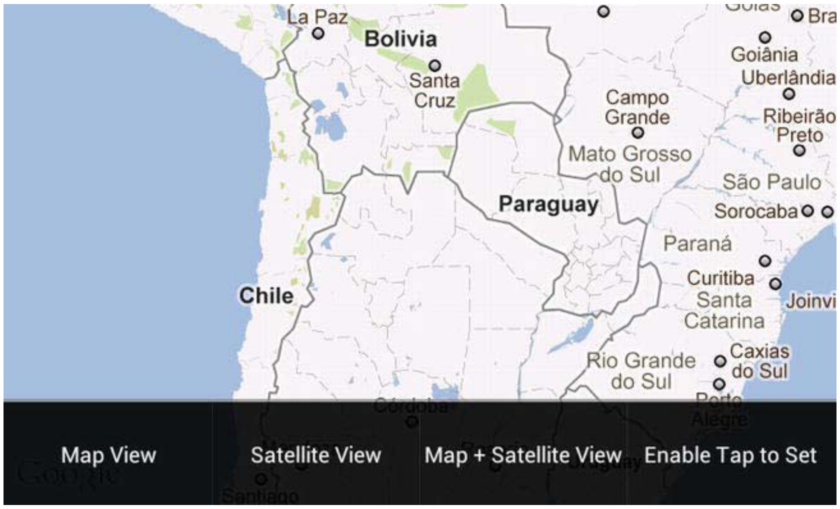


Figure 7-3. *The app with the map open, showing the options menu*

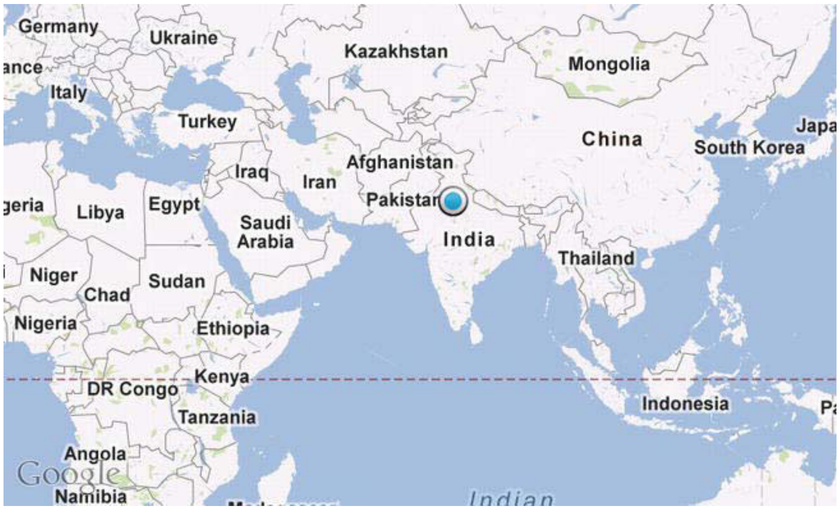
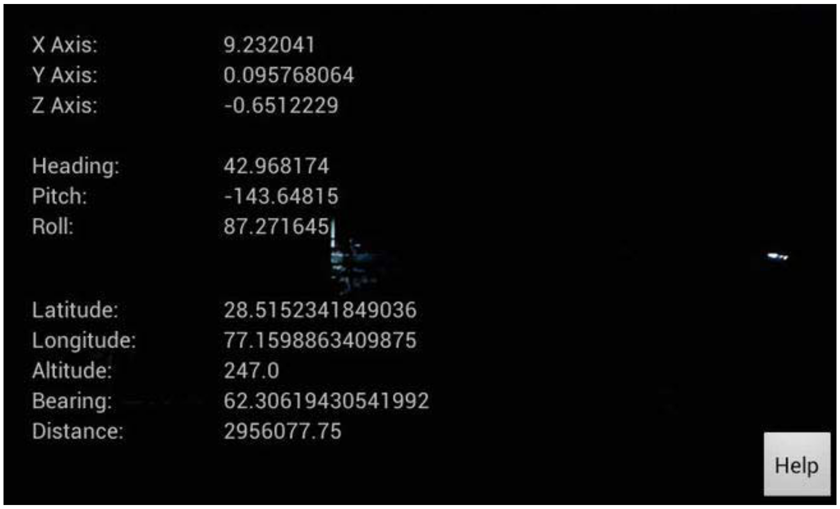


Figure 7-4. *The app with the user's current location displayed*



```
X Axis:       9.232041
Y Axis:       0.095768064
Z Axis:       -0.6512229

Heading:      42.968174
Pitch:        -143.64815
Roll:         87.271645


Latitude:     28.5152341849036
Longitude:    77.1598863409875
Altitude:     247.0
Bearing:      62.30619430541992
Distance:     2956077.75
```

Figure 7-5. *The app with the bearing and distance to the set location. The location was set to the middle of China, and I was facing north.*

You can get the complete source from this book's page on apress.com or from the GitHub repository.

Summary

This chapter discussed how to make the basic skeleton of a navigational app. We allow the user to select any point on a map to go to, and then we calculate the direction in which the user needs to move as the bearing. Converting this to a publishable app will only require you to draw an arrow pointing the user in the correct direction while in the augmented reality view. However, adding that in the example app will increase its complexity to beyond the scope of this chapter.

In the next chapter, you will learn how to design and implement a marker-based augmented reality viewer.