Sensors

Sensors

Hardware devices that measure the physical environment

Motion

Position

Environment

Some Example Sensors

Motion - 3-axis Accelerometer

Position - 3-axis Magnetic field

Environment - Pressure

SensorManager

System service that manages sensors

Get instance with

```
getSystemService(!
    Context.SENSOR_SERVICE )
```

Access a specific sensor with

```
SensorManager.!
   getDefaultSensor(int type)
```

Some Sensor Type Constants

Accelerometer -
    Sensor.TYPE_ACCELEROMETER

Magnetic field -
    Sensor.TYPE_MAGNETIC_FIELD

Pressure –
    Sensor.TYPE_PRESSURE

SensorEventListener

Interface for SensorEvent callbacks

SensorEventListener

Called when the accuracy of a sensor has changed

void onAccuracyChanged(!
        Sensor sensor, int accuracy)

SensorEventListener

Called when sensor values have changed

void onSensorChanged(!
                    SensorEvent event)

Registering for SensorEvents

Use the SensorManager to register/ unregister for SensorEvents

Registering for SensorEvents

To register a SensorEventListener for a given sensor

```
public boolean registerListener (!
            SensorEventListener listener,!
            Sensor sensor, int rate)
```

Registering for SensorEvents

Unregisters a listener for the sensors with which it is registered

```
public void unregisterListener (!
            SensorEventListener listener,!
            Sensor sensor)
```

SensorEvent

Represents a Sensor event

Data is sensor-specific

sensor type

time-stamp

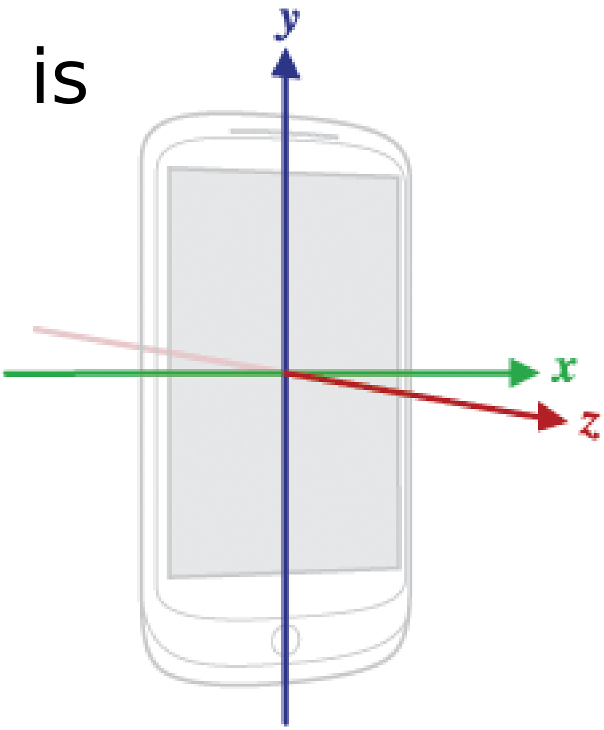
Accuracy

measurement data

Sensor Coordinate System

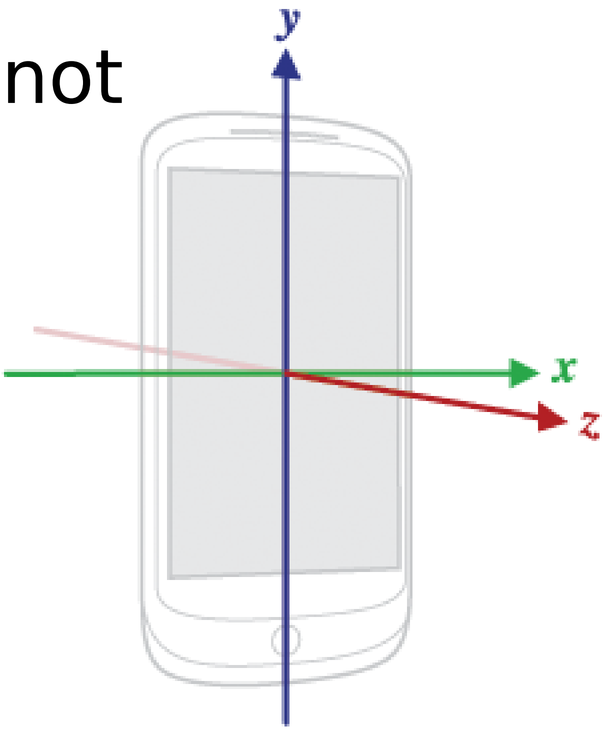
When default orientation is portrait & the device is lying flat, face-up on a table, axes run

- X – Right to left
- Y – Bottom to top
- Z – Down to up



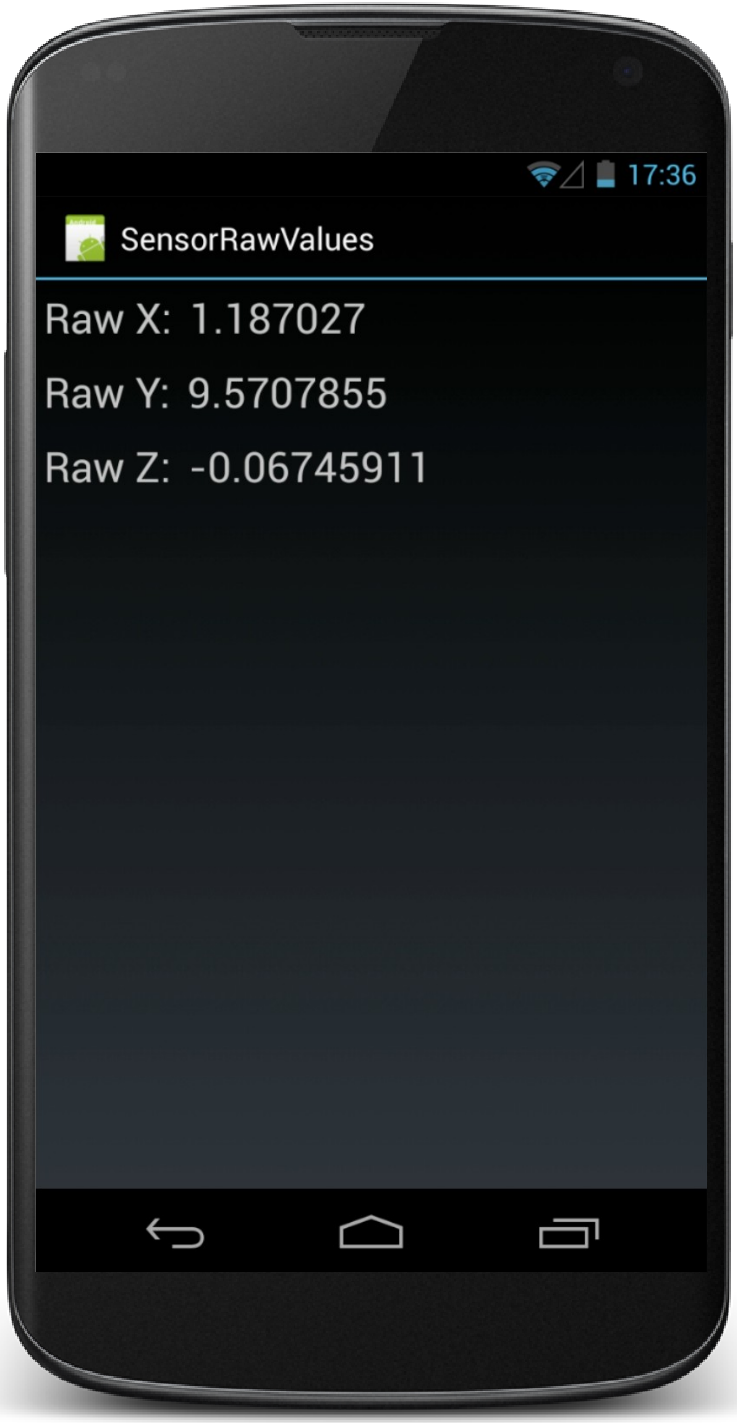
Sensor Coordinate System

Coordinate system does not change when device orientation changes



SensorRawAccelerometer

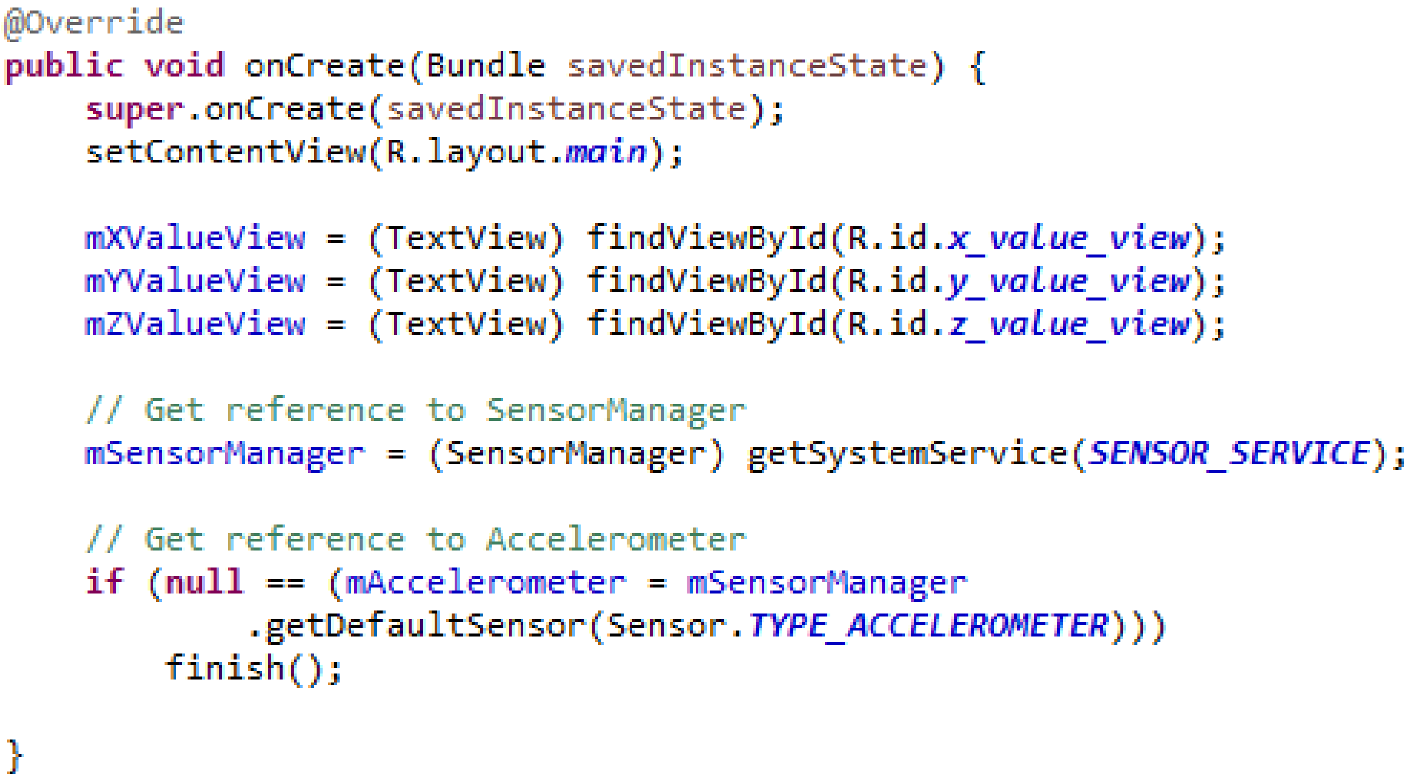
Displays the raw values read from the device’s accelerometer



SensorRawValues

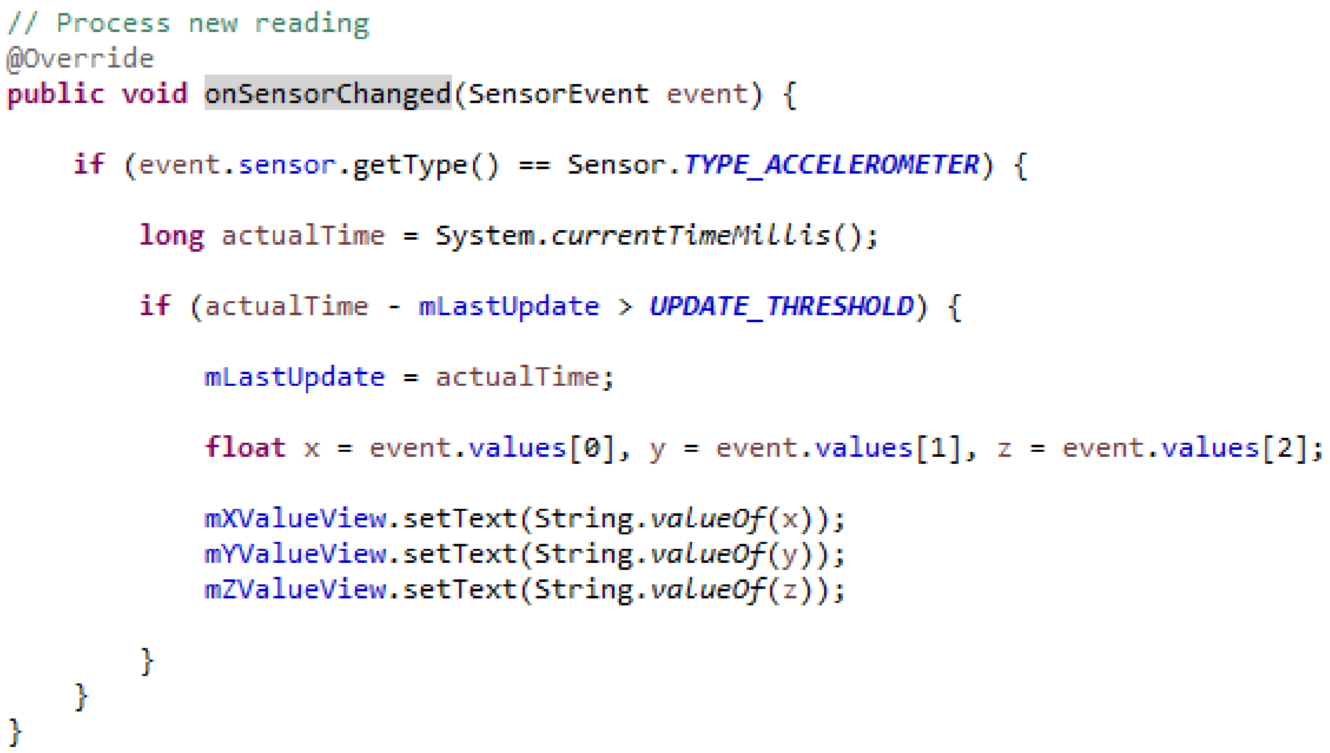
```
Raw X:  1.187027

Raw Y:  9.5707855

Raw Z:  -0.06745911
```

SensorRawAccelerometer



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

    mXValueView = (TextView) findViewById(R.id.x_value_view);
    mYValueView = (TextView) findViewById(R.id.y_value_view);
    mZValueView = (TextView) findViewById(R.id.z_value_view);

    // Get reference to SensorManager
    mSensorManager = (SensorManager) getSystemService(SENSOR_SERVICE);

    // Get reference to Accelerometer
    if (null == (mAccelerometer = mSensorManager
            .getDefaultSensor(Sensor.TYPE_ACCELEROMETER)))
        finish();

}
```

SensorRawAccelerometer



```
// Process new reading
@Override
public void onSensorChanged(SensorEvent event) {

    if (event.sensor.getType() == Sensor.TYPE_ACCELEROMETER) {

        long actualTime = System.currentTimeMillis();

        if (actualTime - mLastUpdate > UPDATE_THRESHOLD) {

            mLastUpdate = actualTime;

            float x = event.values[0], y = event.values[1], z = event.values[2];

            mXValueView.setText(String.valueOf(x));
            mYValueView.setText(String.valueOf(y));
            mZValueView.setText(String.valueOf(z));

        }
    }
}
```

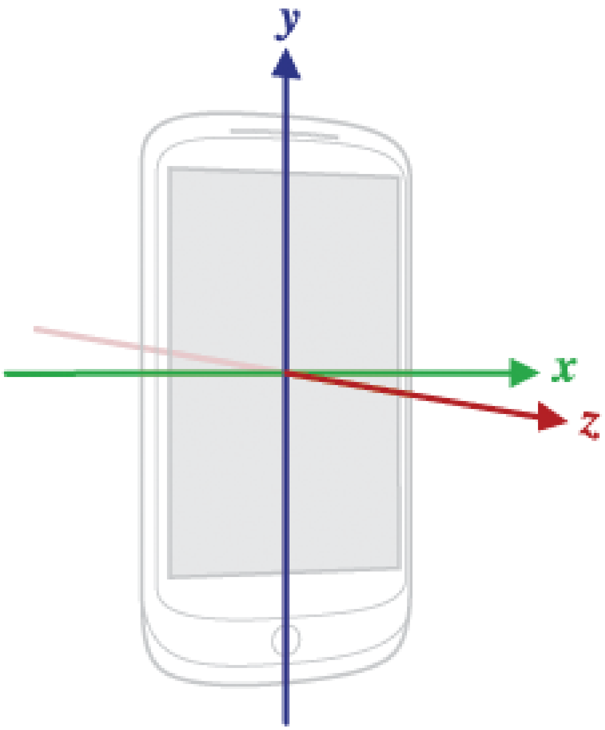
Accelerometer Values

If the device were standing straight up, the accelerometer would ideally report:

X ≈0 m/s^2

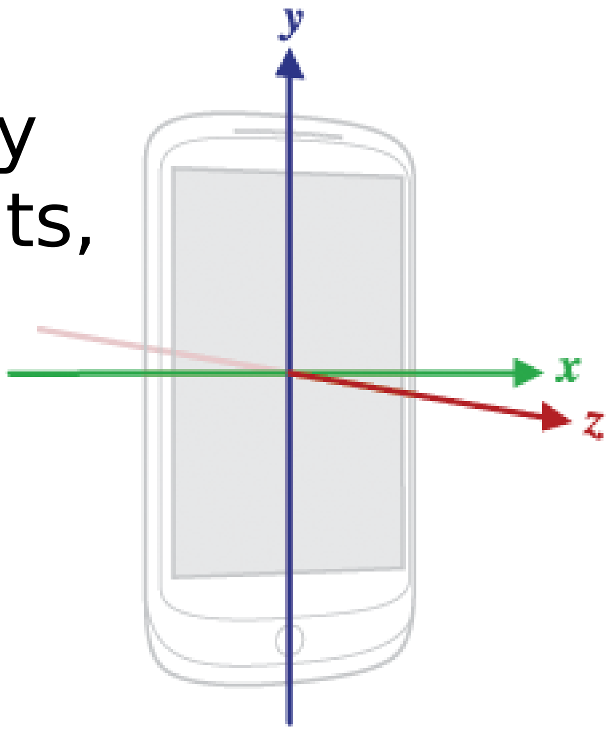
Y ≈9.81 m/s^2

Z ≈0 m/s^2



Accelerometer values

But these values will vary due to natural movements, non-flat surfaces, noise, etc.



Filtering Accelerometer values

Two common transforms

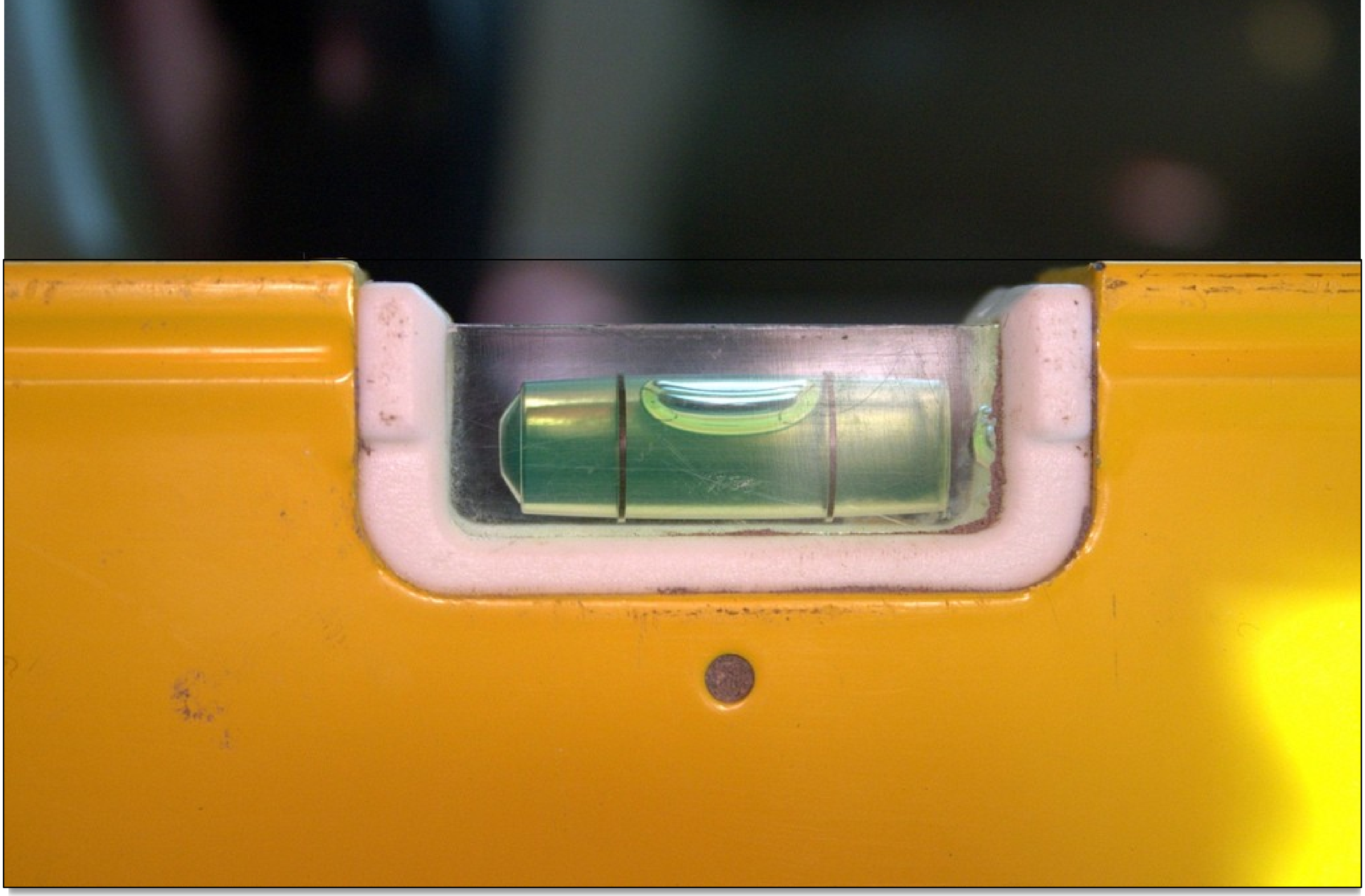
Low-pass filter

High-pass filter

Low-pass filters

Deemphasize transient force changes

Emphasize constant force components



Carpenter's Level

High-pass filters

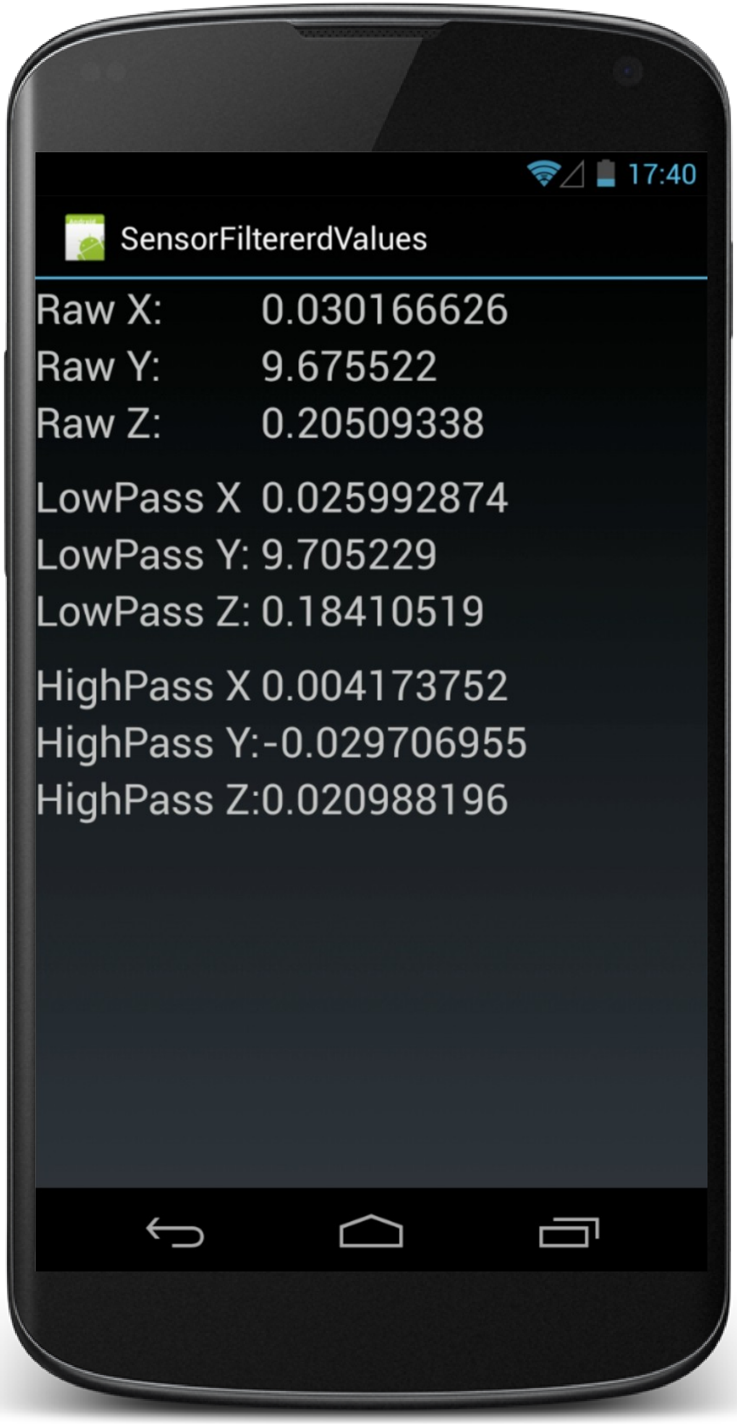
Emphasize transient force changes

Deemphasize constant force components

SensorFilteredAccelerometer

Applies both a low-pass and a high-pass filter to raw accelerometer values

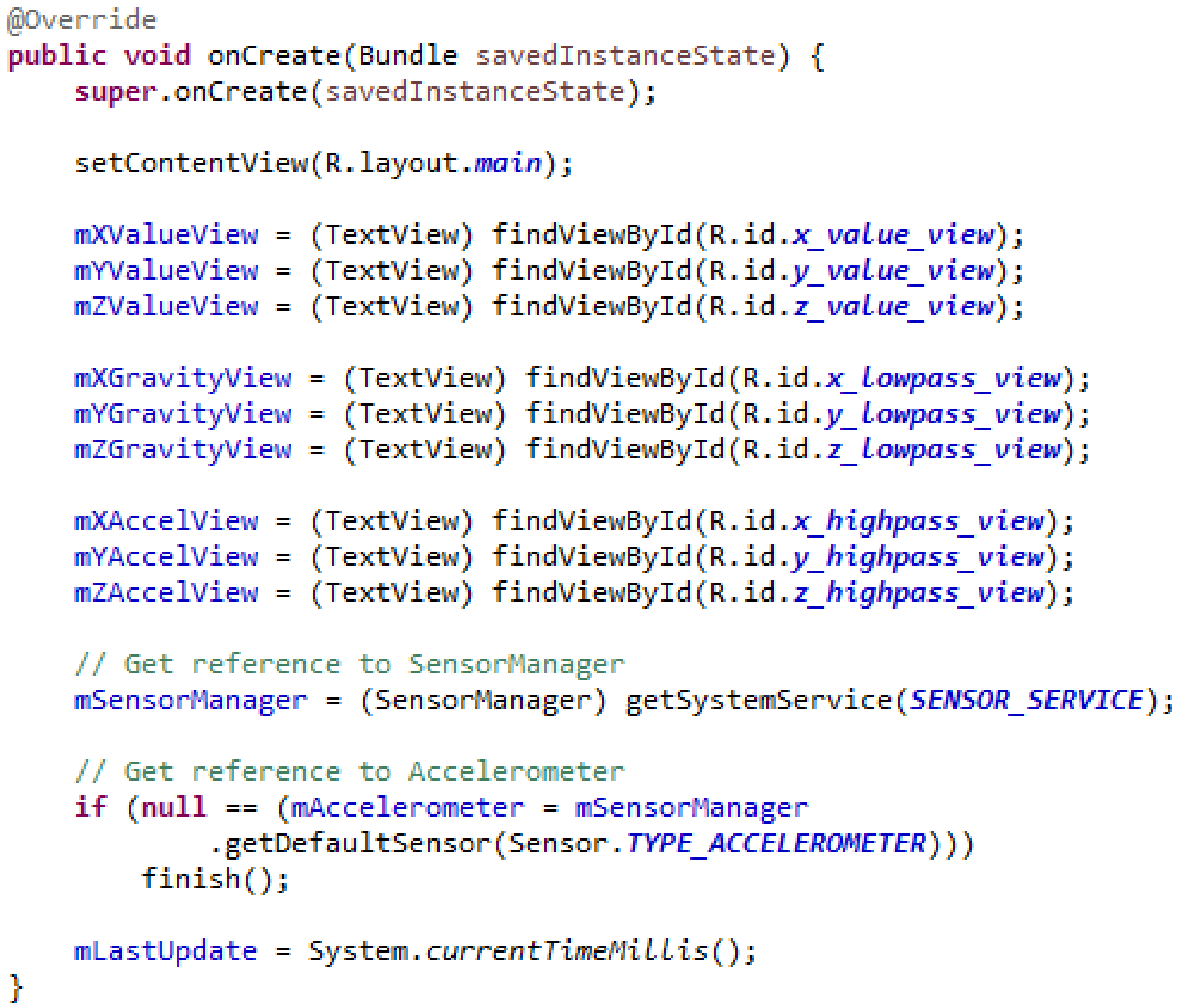
Displays the filtered values



SensorFiltererdValues

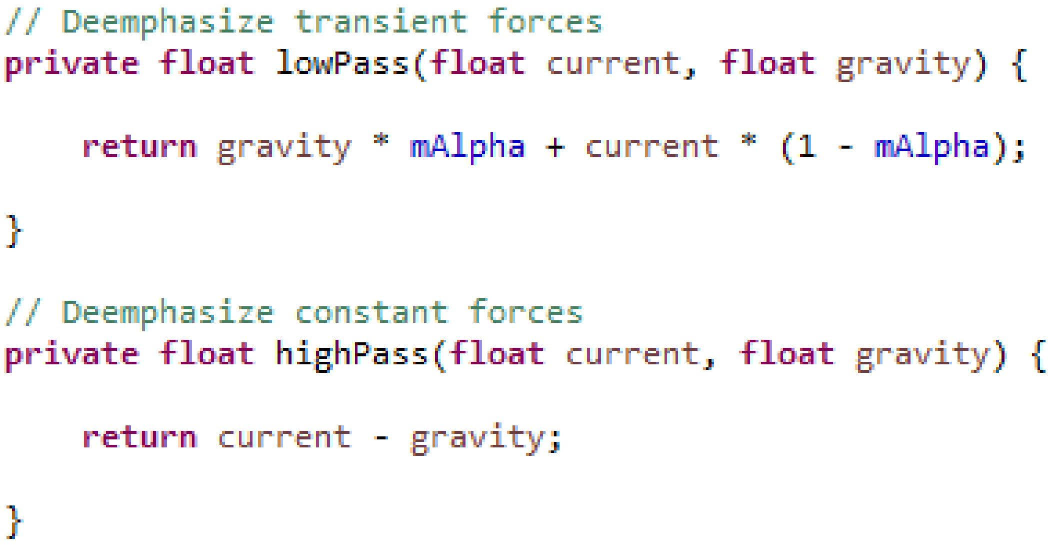
```
Raw X:         0.030166626
Raw Y:         9.675522
Raw Z:         0.20509338

LowPass X  0.025992874
LowPass Y: 9.705229
LowPass Z: 0.18410519

HighPass X 0.004173752
HighPass Y:-0.029706955
HighPass Z:0.020988196
```

SensorFilteredAccelerometer



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

    setContentView(R.layout.main);

    mXValueView = (TextView) findViewById(R.id.x_value_view);
    mYValueView = (TextView) findViewById(R.id.y_value_view);
    mZValueView = (TextView) findViewById(R.id.z_value_view);

    mXGravityView = (TextView) findViewById(R.id.x_lowpass_view);
    mYGravityView = (TextView) findViewById(R.id.y_lowpass_view);
    mZGravityView = (TextView) findViewById(R.id.z_lowpass_view);

    mXAccelView = (TextView) findViewById(R.id.x_highpass_view);
    mYAccelView = (TextView) findViewById(R.id.y_highpass_view);
    mZAccelView = (TextView) findViewById(R.id.z_highpass_view);

    // Get reference to SensorManager
    mSensorManager = (SensorManager) getSystemService(SENSOR_SERVICE);

    // Get reference to Accelerometer
    if (null == (mAccelerometer = mSensorManager
            .getDefaultSensor(Sensor.TYPE_ACCELEROMETER)))
        finish();

    mLastUpdate = System.currentTimeMillis();
}
```

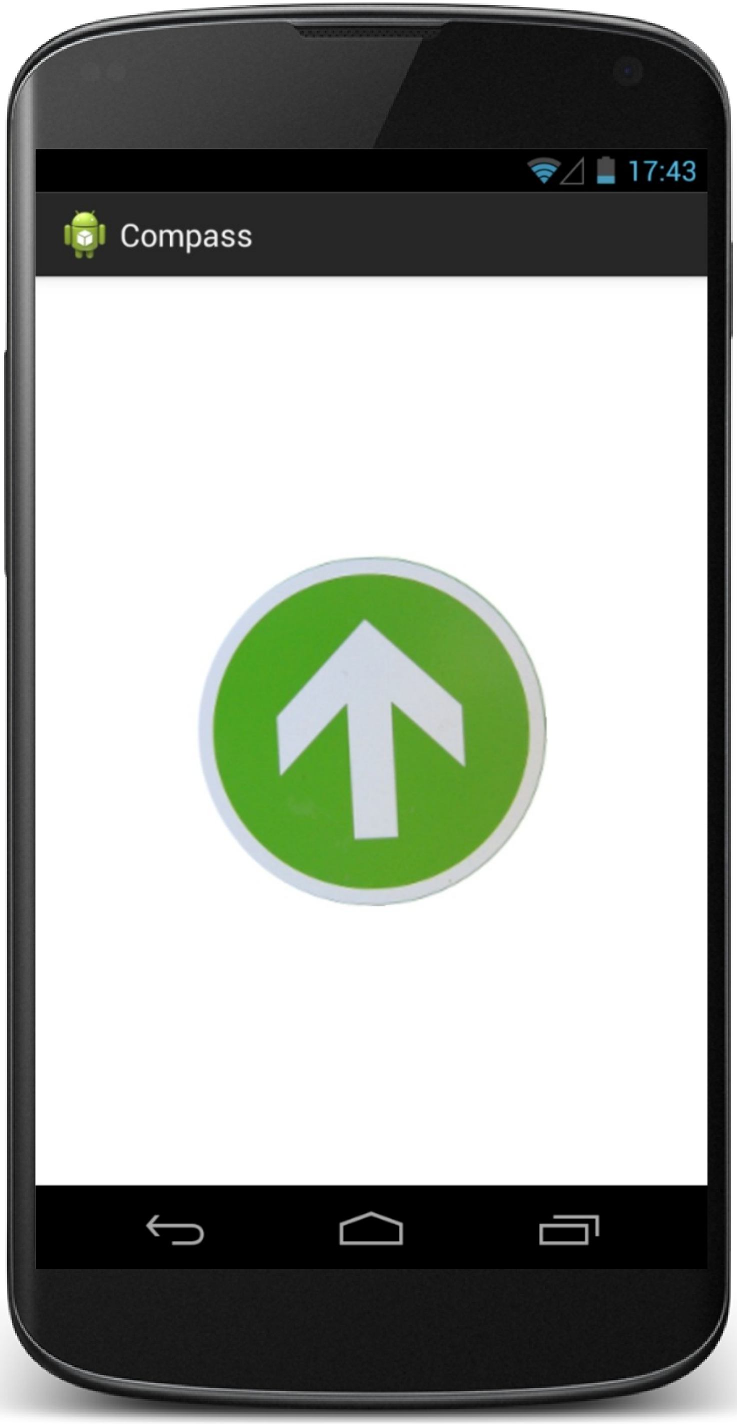
SensorFilteredAccelerometer



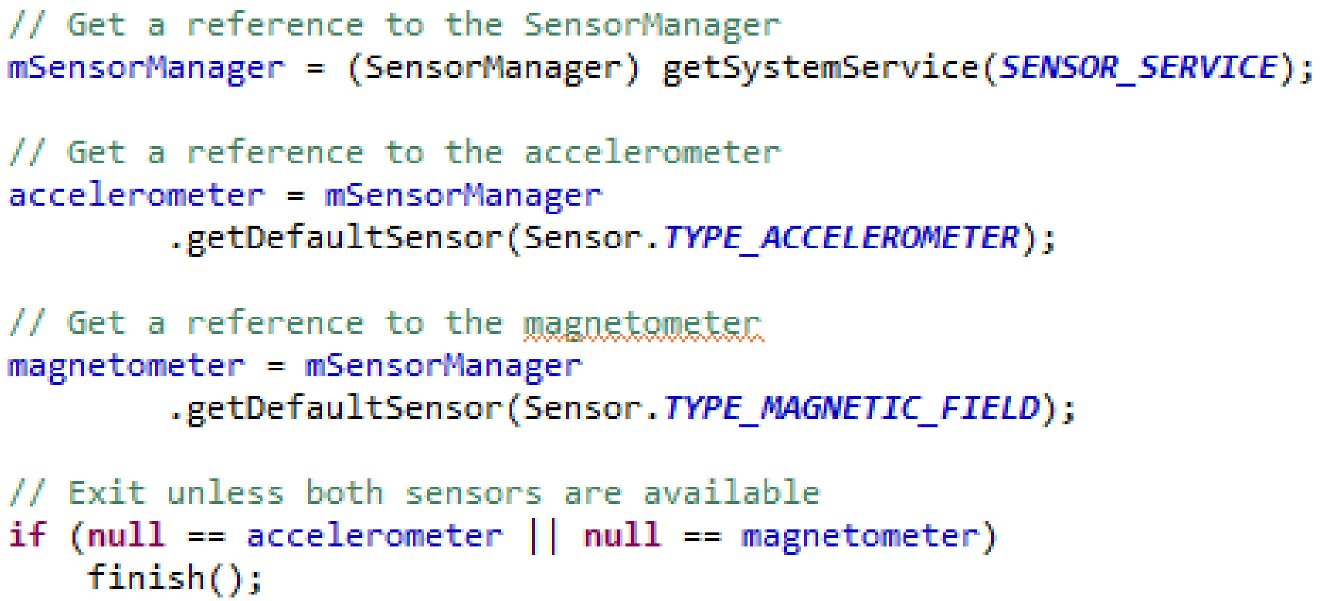
```
// Deemphasize transient forces
private float lowPass(float current, float gravity) {

    return gravity * mAlpha + current * (1 - mAlpha);

}

// Deemphasize constant forces
private float highPass(float current, float gravity) {

    return current - gravity;

}
```

SensorCompass

Uses the device’s accelerometer and magnetometer to orient a compass

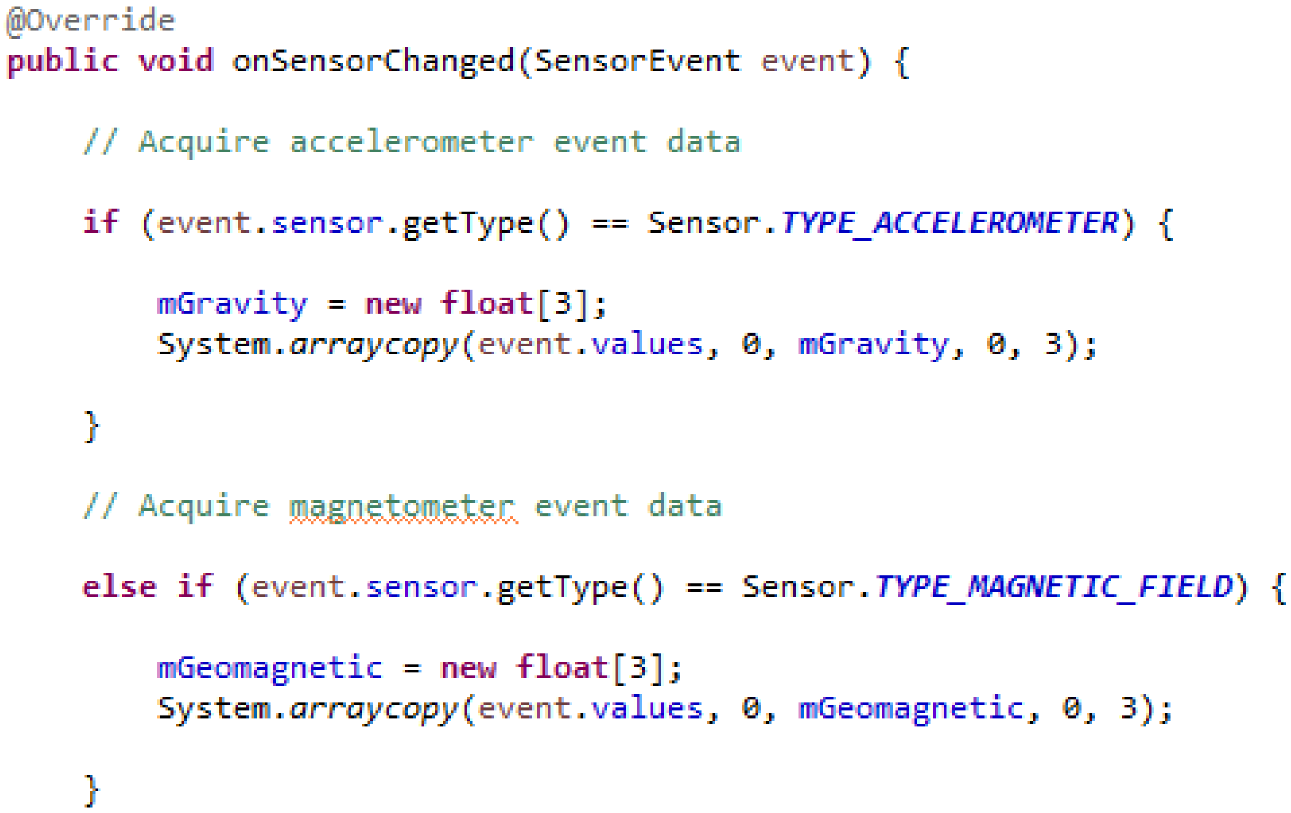

17:43
Compass

SensorCompass



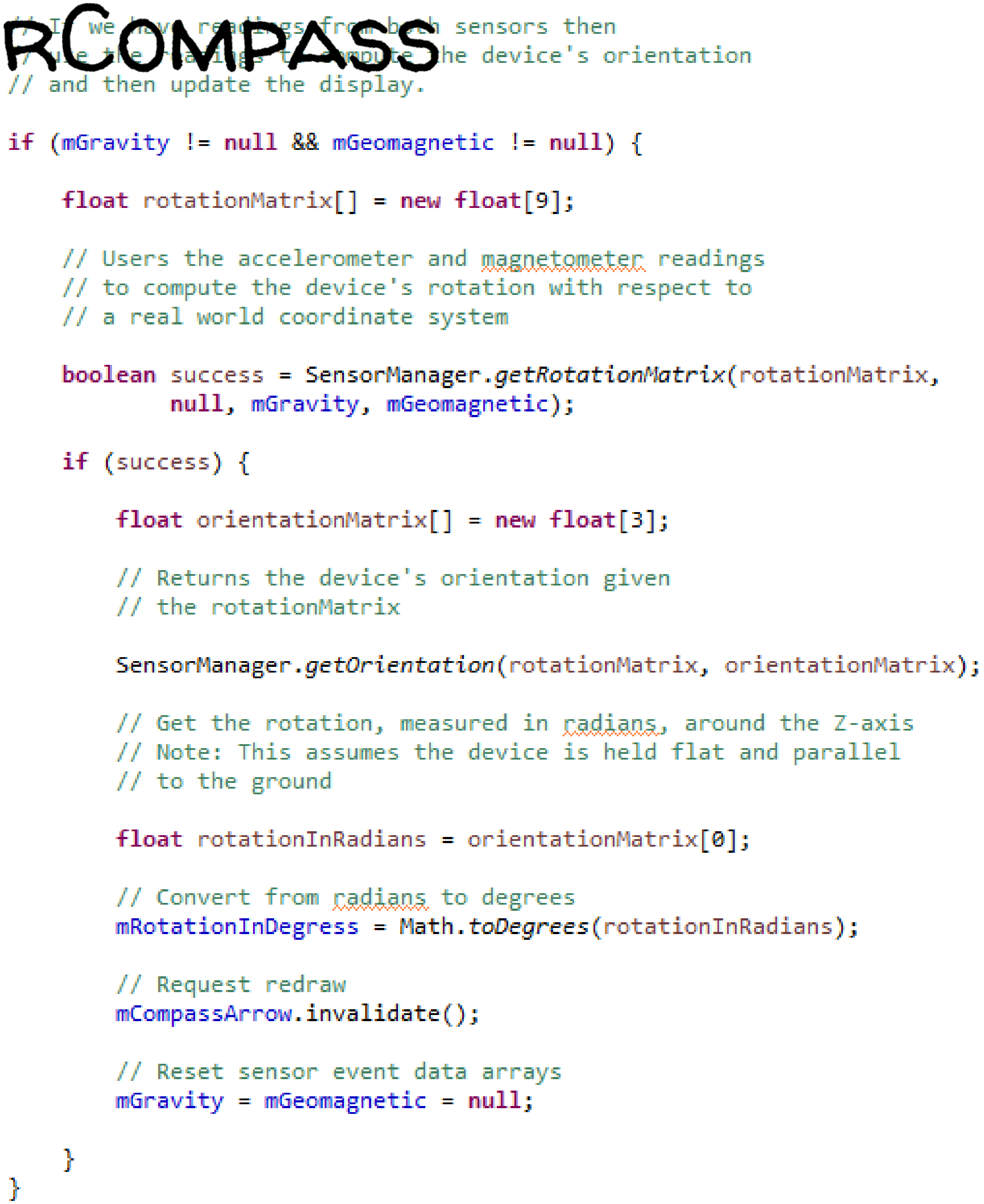
```
// Get a reference to the SensorManager
mSensorManager = (SensorManager) getSystemService(SENSOR_SERVICE);

// Get a reference to the accelerometer
accelerometer = mSensorManager
        .getDefaultSensor(Sensor.TYPE_ACCELEROMETER);

// Get a reference to the magnetometer
magnetometer = mSensorManager
        .getDefaultSensor(Sensor.TYPE_MAGNETIC_FIELD);

// Exit unless both sensors are available
if (null == accelerometer || null == magnetometer)
    finish();
```

SensorCompass



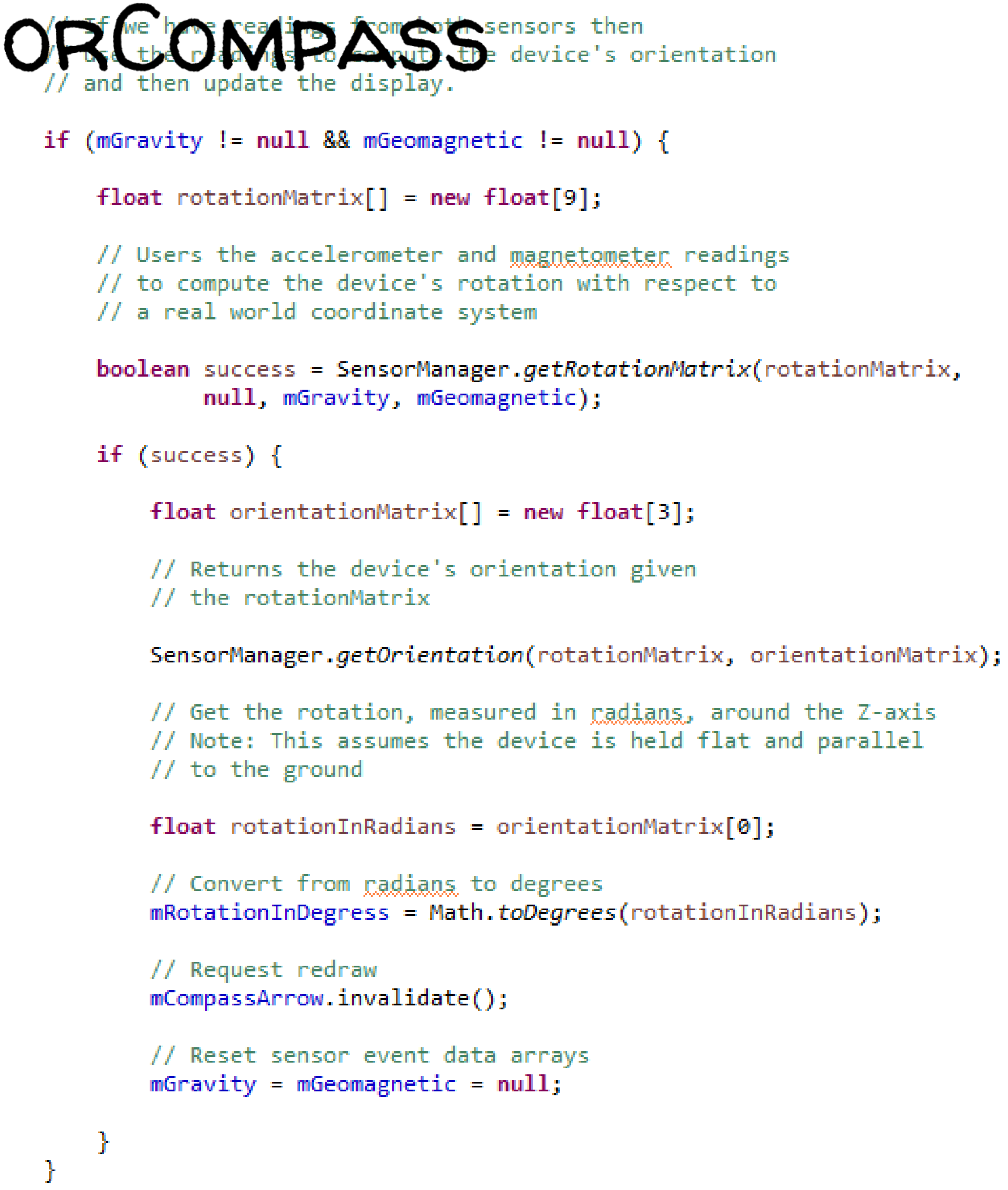
```
@Override
public void onSensorChanged(SensorEvent event) {

    // Acquire accelerometer event data

    if (event.sensor.getType() == Sensor.TYPE_ACCELEROMETER) {

        mGravity = new float[3];
        System.arraycopy(event.values, 0, mGravity, 0, 3);

    }

    // Acquire magnetometer event data

    else if (event.sensor.getType() == Sensor.TYPE_MAGNETIC_FIELD) {

        mGeomagnetic = new float[3];
        System.arraycopy(event.values, 0, mGeomagnetic, 0, 3);

    }
```

SensorCompass



```java
// If we have readings from both sensors then
// use the readings to compute the device's orientation
// and then update the display.

if (mGravity != null && mGeomagnetic != null) {

    float rotationMatrix[] = new float[9];

    // Users the accelerometer and magnetometer readings
    // to compute the device's rotation with respect to
    // a real world coordinate system

    boolean success = SensorManager.getRotationMatrix(rotationMatrix,
            null, mGravity, mGeomagnetic);

    if (success) {

        float orientationMatrix[] = new float[3];

        // Returns the device's orientation given
        // the rotationMatrix

        SensorManager.getOrientation(rotationMatrix, orientationMatrix);

        // Get the rotation, measured in radians, around the Z-axis
        // Note: This assumes the device is held flat and parallel
        // to the ground

        float rotationInRadians = orientationMatrix[0];

        // Convert from radians to degrees
        mRotationInDegress = Math.toDegrees(rotationInRadians);

        // Request redraw
        mCompassArrow.invalidate();

        // Reset sensor event data arrays
        mGravity = mGeomagnetic = null;

    }
}
```

SensorCompass



```
// If we have readings from both sensors then
// use the readings to compute the device's orientation
// and then update the display.

if (mGravity != null && mGeomagnetic != null) {

    float rotationMatrix[] = new float[9];

    // Users the accelerometer and magnetometer readings
    // to compute the device's rotation with respect to
    // a real world coordinate system

    boolean success = SensorManager.getRotationMatrix(rotationMatrix,
            null, mGravity, mGeomagnetic);

    if (success) {

        float orientationMatrix[] = new float[3];

        // Returns the device's orientation given
        // the rotationMatrix

        SensorManager.getOrientation(rotationMatrix, orientationMatrix);

        // Get the rotation, measured in radians, around the Z-axis
        // Note: This assumes the device is held flat and parallel
        // to the ground

        float rotationInRadians = orientationMatrix[0];

        // Convert from radians to degrees
        mRotationInDegress = Math.toDegrees(rotationInRadians);

        // Request redraw
        mCompassArrow.invalidate();

        // Reset sensor event data arrays
        mGravity = mGeomagnetic = null;

    }
}
```