Motion Sensors

The Android platform provides several sensors that let you monitor the motion of a device. Two of these sensors are always hardware-based (the accelerometer and gyroscope), and three of these sensors can be either hardware-based or software-based (the gravity, linear acceleration, and rotation vector sensors). For example, on some devices the software-based sensors derive their data from the accelerometer and magnetometer, but on other devices they may also use the gyroscope to derive their data. Most Android-powered devices have an accelerometer, and many now include a gyroscope. The availability of the softare-based sensors is more variable because they often rely on one or more hardware sensors to derive their data.

Motion sensors are useful for monitoring device movement, such as tilt, shake, rotation, or swing. The movement is usually a reflection of direct user input (for example, a user steering a car in a game or a user controlling a ball in a game), but it can also be a reflection of the physical environment in which the device is sitting (for example, moving with you while you drive your car). In the first case, you are monitoring motion relative to the device's frame of reference or your application's frame of reference; in the second case you are monitoring motion relative to the world's frame of reference. Motion sensors by themselves are not typically used to monitor device position, but they can be used with other sensors, such as the geomagnetic field sensor, to determine a device's position relative to the world's frame of reference (see Position Sensors (../../../guide/topics/sensors/sensors_position.html) for more information).

All of the motion sensors return multi-dimensional arrays of sensor values for each `SensorEvent` (../../../reference/android/hardware/SensorEvent.html). For example, during a single sensor event the accelerometer returns acceleration force data for the three coordinate axes, and the gyroscope returns rate of rotation data for the three coordinate axes. These data values are returned in a `float` array (`values` (../../../reference/android/hardware/SensorEvent.html#values)) along with other `SensorEvent` (../../../reference/android/hardware/SensorEvent.html) parameters. Table 1 summarizes the motion sensors that are available on the Android platform.

Table 1. Motion sensors that are supported on the Android platform.

Sensor	Sensor event data	Description	Units of measure
`TYPE_ACCELEROMETER`	`SensorEvent.values[0]`	Acceleration force along the x axis (including gravity).	m/s^2
	`SensorEvent.values[1]`	Acceleration force along the y axis (including gravity).	

IN THIS DOCUMENT

KEY CLASSES AND INTERFACES

- `Sensor`
- `SensorEvent`
- `SensorManager`
- `SensorEventListener`

RELATED SAMPLES

- Accelerometer Play
- API Demos (OS - RotationVectorDemo)
- API Demos (OS - Sensors)

SEE ALSO

- Sensors
- Sensors Overview
- Position Sensors
- Environment Sensors

	`SensorEvent.values[2]`	Acceleration force along the z axis (including gravity).	
	`SensorEvent.values[0]`	Force of gravity along the x axis.	
`TYPE_GRAVITY`	`SensorEvent.values[1]`	Force of gravity along the y axis.	m/s^2
	`SensorEvent.values[2]`	Force of gravity along the z axis.	
	`SensorEvent.values[0]`	Rate of rotation around the x axis.	
`TYPE_GYROSCOPE`	`SensorEvent.values[1]`	Rate of rotation around the y axis.	rad/s
	`SensorEvent.values[2]`	Rate of rotation around the z axis.	
	`SensorEvent.values[0]`	Acceleration force along the x axis (excluding gravity).	
`TYPE_LINEAR_ACCELERATION`	`SensorEvent.values[1]`	Acceleration force along the y axis (excluding gravity).	m/s^2
	`SensorEvent.values[2]`	Acceleration force along the z axis (excluding gravity).	
`TYPE_ROTATION_VECTOR`	`SensorEvent.values[0]`	Rotation vector component along the x axis (x * sin(θ/2)).	Unitless
	`SensorEvent.values[1]`	Rotation vector component along the y axis (y * sin(θ/2)).	
	`SensorEvent.values[2]`	Rotation vector component along the z axis (z * sin(θ/2)).	
	`SensorEvent.values[3]`	Scalar component of the rotation vector ((cos(θ/2)).[1]	

[1] The scalar component is an optional value.

The rotation vector sensor and the gravity sensor are the most frequently used sensors for motion detection and monitoring. The rotational vector sensor is particularly versatile and can be used for a wide range of motion-related tasks, such as detecting gestures, monitoring angular change, and monitoring relative orientation changes. For example, the rotational vector sensor is ideal if you are developing a game, an augmented reality application, a 2-dimensional or 3-dimensional compass, or a camera stabilization app. In most cases, using these sensors is a better choice than using the accelerometer and geomagnetic field sensor or the orientation sensor.

Android Open Source Project Sensors

The Android Open Source Project (AOSP) provides three software-based motion sensors: a gravity sensor, a linear acceleration sensor, and a rotation vector sensor. These sensors were updated in Android 4.0 and now use a device's gyroscope (in addition to other sensors) to improve stability and performance. If you want to try these sensors, you can identify them by using the `getVendor()`

(../../../reference/android/hardware/Sensor.html#getVendor()) method and the getVersion() (../../../reference/android/hardware/Sensor.html#getVersion()) method (the vendor is Google Inc.; the version number is 3). Identifying these sensors by vendor and version number is necessary because the Android system considers these three sensors to be secondary sensors. For example, if a device manufacturer provides their own gravity sensor, then the AOSP gravity sensor shows up as a secondary gravity sensor. All three of these sensors rely on a gyroscope: if a device does not have a gyroscope, these sensors do not show up and are not available for use.

Using the Accelerometer

An acceleration sensor measures the acceleration applied to the device, including the force of gravity. The following code shows you how to get an instance of the default acceleration sensor:

```
private SensorManager mSensorManager;
private Sensor mSensor;
  ...
mSensorManager = (SensorManager) getSystemService(Context.SENSOR_SERVICE);
mSensor = mSensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER);
```

Conceptually, an acceleration sensor determines the acceleration that is applied to a device (A_d) by measuring the forces that are applied to the sensor itself (F_s) using the following relationship:

$$A_d = - \sum F_s / mass$$

However, the force of gravity is always influencing the measured acceleration according to the following relationship:

$$A_d = -g - \sum F / mass$$

For this reason, when the device is sitting on a table (and not accelerating), the accelerometer reads a magnitude of g = 9.81 m/s^2. Similarly, when the device is in free fall and therefore rapidly accelerating toward the ground at 9.81 m/s^2, its accelerometer reads a magnitude of g = 0 m/s^2. Therefore, to measure the real acceleration of the device, the contribution of the force of gravity must be removed from the accelerometer data. This can be achieved by applying a high-pass filter. Conversely, a low-pass filter can be used to isolate the force of gravity. The following example shows how you can do this:

```
public void onSensorChanged(SensorEvent event){
  // In this example, alpha is calculated as t / (t + dT),
  // where t is the low-pass filter's time-constant and
  // dT is the event delivery rate.

  final float alpha = 0.8;

  // Isolate the force of gravity with the low-pass filter.
  gravity[0] = alpha * gravity[0] + (1 - alpha) * event.values[0];
```

```
    gravity[1] = alpha * gravity[1] + (1 - alpha) * event.values[1];
    gravity[2] = alpha * gravity[2] + (1 - alpha) * event.values[2];

    // Remove the gravity contribution with the high-pass filter.
    linear_acceleration[0] = event.values[0] - gravity[0];
    linear_acceleration[1] = event.values[1] - gravity[1];
    linear_acceleration[2] = event.values[2] - gravity[2];
  }
```

> Note: You can use many different techniques to filter sensor data. The code sample above uses a simple filter constant (alpha) to create a low-pass filter. This filter constant is derived from a time constant (t), which is a rough representation of the latency that the filter adds to the sensor events, and the sensor's event delivery rate (dt). The code sample uses an alpha value of 0.8 for demonstration purposes. If you use this filtering method you may need to choose a different alpha value.

Accelerometers use the standard sensor coordinate system (../../../guide/topics/sensors/sensors_overview.html#sensors-coords). In practice, this means that the following conditions apply when a device is laying flat on a table in its natural orientation:

- If you push the device on the left side (so it moves to the right), the x acceleration value is positive.
- If you push the device on the bottom (so it moves away from you), the y acceleration value is positive.
- If you push the device toward the sky with an acceleration of A m/s^2, the z acceleration value is equal to A + 9.81, which corresponds to the acceleration of the device (+A m/s^2) minus the force of gravity (-9.81 m/s^2).
- The stationary device will have an acceleration value of +9.81, which corresponds to the acceleration of the device (0 m/s^2 minus the force of gravity, which is -9.81 m/s^2).

In general, the accelerometer is a good sensor to use if you are monitoring device motion. Almost every Android-powered handset and tablet has an accelerometer, and it uses about 10 times less power than the other motion sensors. One drawback is that you might have to implement low-pass and high-pass filters to eliminate gravitational forces and reduce noise.

The Android SDK provides a sample application that shows how to use the acceleration sensor (Accelerometer Play (../../../resources/samples/AccelerometerPlay/index.html)).

Using the Gravity Sensor

The gravity sensor provides a three dimensional vector indicating the direction and magnitude of gravity. The following code shows you how to get an instance of the default gravity sensor:

```
private SensorManager mSensorManager;
private Sensor mSensor;
...
mSensorManager = (SensorManager) getSystemService(Context.SENSOR_SERVICE);
mSensor = mSensorManager.getDefaultSensor(Sensor.TYPE_GRAVITY);
```

The units are the same as those used by the acceleration sensor (m/s^2), and the coordinate system is the same as the one used by the acceleration sensor.

> Note: When a device is at rest, the output of the gravity sensor should be identical to that of the accelerometer.

Using the Gyroscope

The gyroscope measures the rate or rotation in rad/s around a device's x, y, and z axis. The following code shows you how to get an instance of the default gyroscope:

```
private SensorManager mSensorManager;
private Sensor mSensor;
...
mSensorManager = (SensorManager) getSystemService(Context.SENSOR_SERVICE);
mSensor = mSensorManager.getDefaultSensor(Sensor.TYPE_GYROSCOPE);
```

The sensor's coordinate system (../../../guide/topics/sensors/sensors_overview.html#sensors-coords) is the same as the one used for the acceleration sensor. Rotation is positive in the counter-clockwise direction; that is, an observer looking from some positive location on the x, y or z axis at a device positioned on the origin would report positive rotation if the device appeared to be rotating counter clockwise. This is the standard mathematical definition of positive rotation and is not the same as the definition for roll that is used by the orientation sensor.

Usually, the output of the gyroscope is integrated over time to calculate a rotation describing the change of angles over the timestep. For example:

```
// Create a constant to convert nanoseconds to seconds.
private static final float NS2S = 1.0f / 1000000000.0f;
private final float[] deltaRotationVector = new float[4]();
private float timestamp;

public void onSensorChanged(SensorEvent event) {
  // This timestep's delta rotation to be multiplied by the current rotation
  // after computing it from the gyro sample data.
  if (timestamp != 0) {
    final float dT = (event.timestamp - timestamp) * NS2S;
    // Axis of the rotation sample, not normalized yet.
    float axisX = event.values[0];
    float axisY = event.values[1];
    float axisZ = event.values[2];

    // Calculate the angular speed of the sample
    float omegaMagnitude = sqrt(axisX*axisX + axisY*axisY + axisZ*axisZ);

    // Normalize the rotation vector if it's big enough to get the axis
    // (that is, EPSILON should represent your maximum allowable margin of error)
```

```
    if (omegaMagnitude > EPSILON) {
      axisX /= omegaMagnitude;
      axisY /= omegaMagnitude;
      axisZ /= omegaMagnitude;
    }

    // Integrate around this axis with the angular speed by the timestep
    // in order to get a delta rotation from this sample over the timestep
    // We will convert this axis-angle representation of the delta rotation
    // into a quaternion before turning it into the rotation matrix.
    float thetaOverTwo = omegaMagnitude * dT / 2.0f;
    float sinThetaOverTwo = sin(thetaOverTwo);
    float cosThetaOverTwo = cos(thetaOverTwo);
    deltaRotationVector[0] = sinThetaOverTwo * axisX;
    deltaRotationVector[1] = sinThetaOverTwo * axisY;
    deltaRotationVector[2] = sinThetaOverTwo * axisZ;
    deltaRotationVector[3] = cosThetaOverTwo;
  }
  timestamp = event.timestamp;
  float[] deltaRotationMatrix = new float[9];
  SensorManager.getRotationMatrixFromVector(deltaRotationMatrix, deltaRotationVect
    // User code should concatenate the delta rotation we computed with the curre
    // in order to get the updated rotation.
    // rotationCurrent = rotationCurrent * deltaRotationMatrix;
    }
  }
```

Standard gyroscopes provide raw rotational data without any filtering or correction for noise and drift (bias). In practice, gyroscope noise and drift will introduce errors that need to be compensated for. You usually determine the drift (bias) and noise by monitoring other sensors, such as the gravity sensor or accelerometer.

Using the Linear Accelerometer

The linear acceleration sensor provides you with a three-dimensional vector representing acceleration along each device axis, excluding gravity. The following code shows you how to get an instance of the default linear acceleration sensor:

```
private SensorManager mSensorManager;
private Sensor mSensor;
...
mSensorManager = (SensorManager) getSystemService(Context.SENSOR_SERVICE);
mSensor = mSensorManager.getDefaultSensor(Sensor.TYPE_LINEAR_ACCELERATION);
```

Conceptually, this sensor provides you with acceleration data according to the following relationship:

```
linear acceleration = acceleration - acceleration due to gravity
```

You typically use this sensor when you want to obtain acceleration data without the influence of gravity. For example, you could use this sensor to see how fast your car is going. The linear acceleration sensor always has an offset, which you need to remove. The simplest way to do this is to build a calibration step into your application. During calibration you can ask the user to set the device on a table, and then read the offsets for all three axes. You can then subtract that offset from the acceleration sensor's direct readings to get the actual linear acceleration.

The sensor coordinate system (../../../guide/topics/sensors/sensors_overview.html#sensors-coords) is the same as the one used by the acceleration sensor, as are the units of measure (m/s^2).

Using the Rotation Vector Sensor

The rotation vector represents the orientation of the device as a combination of an angle and an axis, in which the device has rotated through an angle θ around an axis (x, y, or z). The following code shows you how to get an instance of the default rotation vector sensor:

```
private SensorManager mSensorManager;
private Sensor mSensor;
...
mSensorManager = (SensorManager) getSystemService(Context.SENSOR_SERVICE);
mSensor = mSensorManager.getDefaultSensor(Sensor.TYPE_ROTATION_VECTOR);
```

The three elements of the rotation vector are expressed as follows:

```
x*sin(θ/2)
y*sin(θ/2)
z*sin(θ/2)
```

Where the magnitude of the rotation vector is equal to sin(θ/2), and the direction of the rotation vector is equal to the direction of the axis of rotation.

The three elements of the rotation vector are equal to the last three components of a unit quaternion (cos(θ/2), x*sin(θ/2), y*sin(θ/2), z*sin(θ/2)). Elements of the rotation vector are unitless. The x, y, and z axes are defined in the same way as the acceleration sensor. The reference coordinate system is defined as a direct orthonormal basis (see figure 1). This coordinate system has the following characteristics:

- X is defined as the vector product Y x Z. It is tangential to the ground at the device's current location and points approximately East.
- Y is tangential to the ground at the device's current location and points toward the geomagnetic North Pole.
- Z points toward the sky and is perpendicular to the ground plane.

The Android SDK provides a sample application that shows how to use the rotation vector sensor. The sample application is located in the API Demos code (OS -

RotationVectorDemo

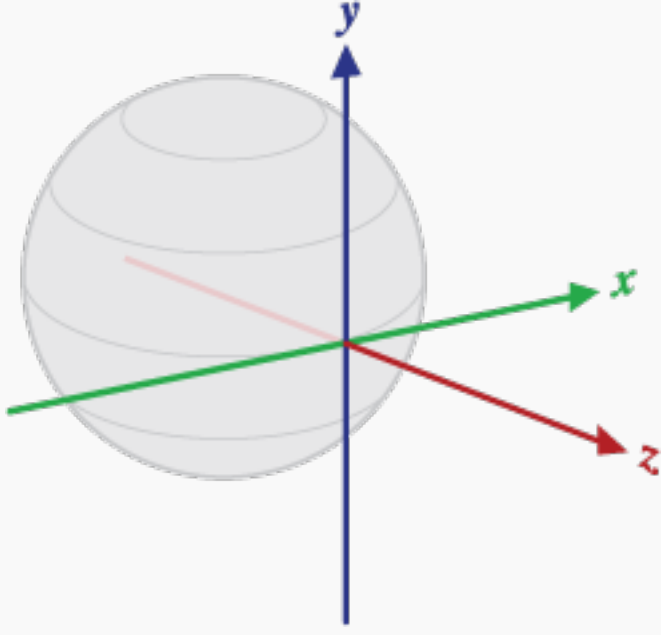

Figure 1. Coordinate system used by the rotation vector sensor.

(../../../resources/samples/ApiDemos/src/com/example/android/apis/os/RotationVectorDemo.html)).