CMSC436: Programming Handheld Systems

Sensors

Today's Topics

Sensor & SensorManager

SensorEvent & SensorEventListener

Filtering sensor values

Example applications

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

Sensor Types

int TYPE_MOTION_DETECT

int TYPE_GRAVITY

int TYPE_AMBIENT_TEMPERATURE

int TYPE_ACCELEROMETER

int TYPE_ALL

Some Sensor Methods

float getResolution()

float getPower()

int getReportingMode()

int getMinDelay()

float getMaximumRange()

SensorEvent

Represents a Sensor event

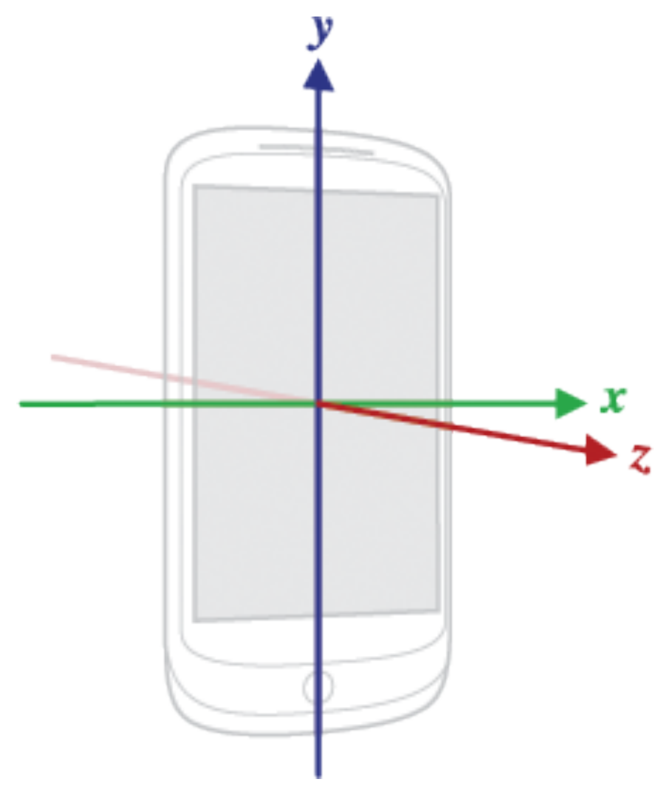
Data includes

- Sensor type
- Time-stamp
- Accuracy
- Sensor-specific measurement data

Sensor Coordinate System

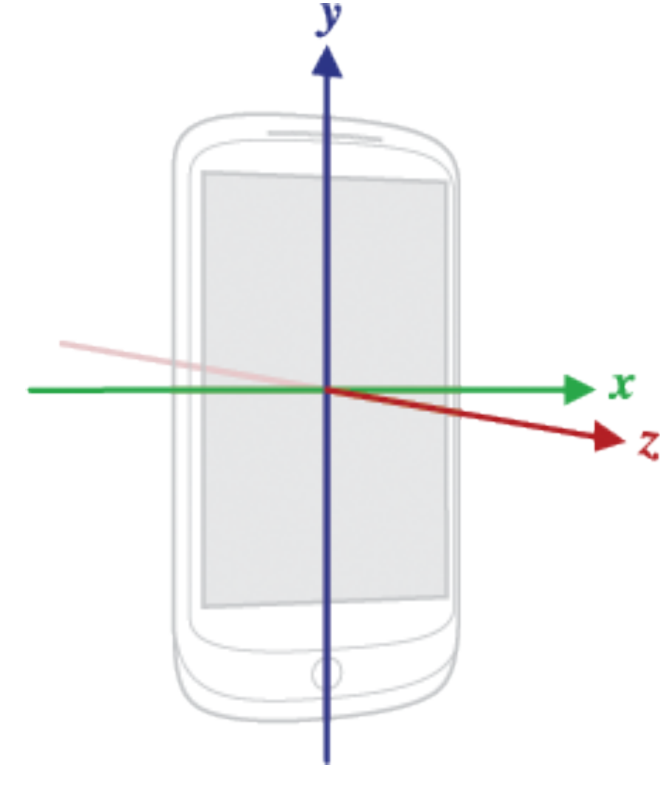
When default orientation is portrait & the device is lying flat, face-up on a table, axes run

- X – Left to right
- Y – Bottom to top
- Z – Down to up



Sensor Coordinate System

Coordinate system does not change when device orientation changes



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

Some SensorManager Methods

```
open fun getSensorList(type: Int): MutableList<Sensor!>!

open fun getDefaultSensor(type: Int): Sensor!
```

SensorEventListener

Interface for SensorEvent callbacks

SensorEventListener

Called when a sensor's accuracy has changed

```
abstract fun onAccuracyChanged(
                    sensor: Sensor!, accuracy: Int): Unit
```

Accuracy Constants

SENSOR_STATUS_ACCURACY_HIGH

SENSOR_STATUS_ACCURACY_MEDIUM

SENSOR_STATUS_ACCURACY_LOW

SENSOR_STATUS_NO_CONTACT

SENSOR_STATUS_UNRELIABLE

SensorEventListener

Called when sensor values have changed

```
abstract fun onSensorChanged(event: SensorEvent!): Unit
```

Note: This method should not keep a reference to the SensorEvent

Registering for SensorEvents

Use the SensorManager to register/unregister for SensorEvents

Registering for SensorEvents

Register SensorEventListener for a given sensor

registerListener(listener: SensorEventListener!,
sensor: Sensor!, samplingPeriodUs: Int): Boolean

Registering for SensorEvents

Unregisters a listener for the sensors with which it is registered

```
unregisterListener(listener: SensorEventListener!,
                   sensor: Sensor!): Unit
```

SensorRawAccelerometer

Displays the raw values read from the device's accelerometer

Extended controls

Location
Cellular
Battery
Phone
Directional pad
Microphone
Fingerprint
Virtual sensors
Bug report
Settings
Help

Accelerometer Additional sensors

Rotate Move

Yaw -180 180 0.0
Pitch -180 180 0.0
Roll -180 180 0.0

Device rotation

Resulting values

Accelerometer (m/s^2):	0.00	9.81	0.00
Gyroscope (rad/s):	0.00	0.00	0.00
Magnetometer (μT):	22.00	5.90	43.10
Rotation:	ROTATION_0		

SensorRawAccelerometer

Extended controls

Location
Cellular
Battery
Phone
Directional pad
Microphone
Fingerprint
Virtual sensors
Bug report
Settings
Help

Accelerometer Additional sensors

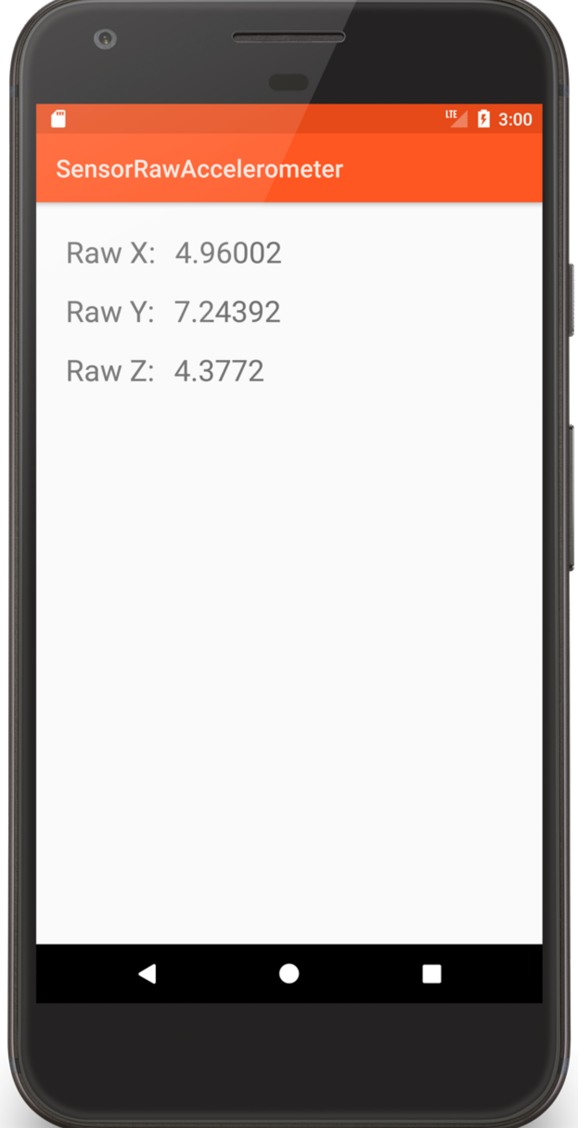
Rotate Move

Yaw -180 180 34.4
Pitch -180 180 -26.5
Roll -180 180 31.8

Device rotation

Resulting values

Accelerometer (m/s^2):	4.96	7.24	4.38
Gyroscope (rad/s):	0.00	0.00	-0.00
Magnetometer (μT):	-12.49	-11.13	45.79
Rotation:	ROTATION_0		



SensorRawAccelerometerActivity.kt

```
public override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(R.layout.main)

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

    // Get reference to SensorManager
    mSensorManager = getSystemService(Context.SENSOR_SERVICE)
                                                       as SensorManager
    // Get reference to Accelerometer
    mAccelerometer = mSensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER)
    if (null == mAccelerometer)
        Log.i(localClassName, "Accelerometer not available")
}
```

SensorRawAccelerometerActivity.kt

```
// Register listener
override fun onResume() {
    super.onResume()
    mSensorManager.registerListener(this, mAccelerometer,
                                    SensorManager.SENSOR_DELAY_UI)
    mLastUpdate = System.currentTimeMillis()
}

// Unregister listener
override fun onPause() {
    mSensorManager.unregisterListener(this)
    super.onPause()
}
```

SensorRawAccelerometerActivity.kt

```
// Process new reading
override fun onSensorChanged(event: SensorEvent) {
    if (event.sensor.type == Sensor.TYPE_ACCELEROMETER) {
        val actualTime = System.currentTimeMillis()
        if (actualTime - mLastUpdate > UPDATE_THRESHOLD) {
            mLastUpdate = actualTime
            val x = event.values[0]
            val y = event.values[1]
            val z = event.values[2]
            …
        }
    }
}
```

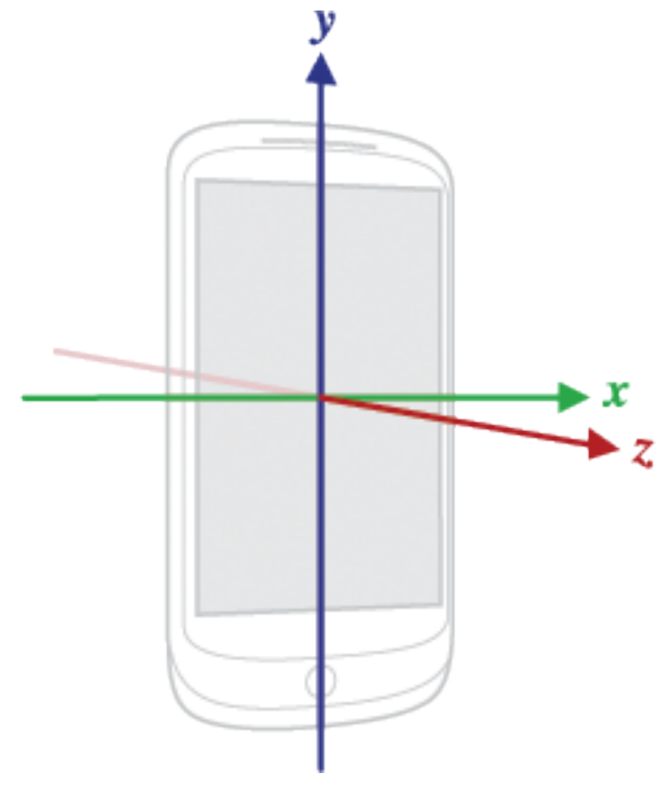
Accelerometer Values

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

X ≈ 0 m/s2

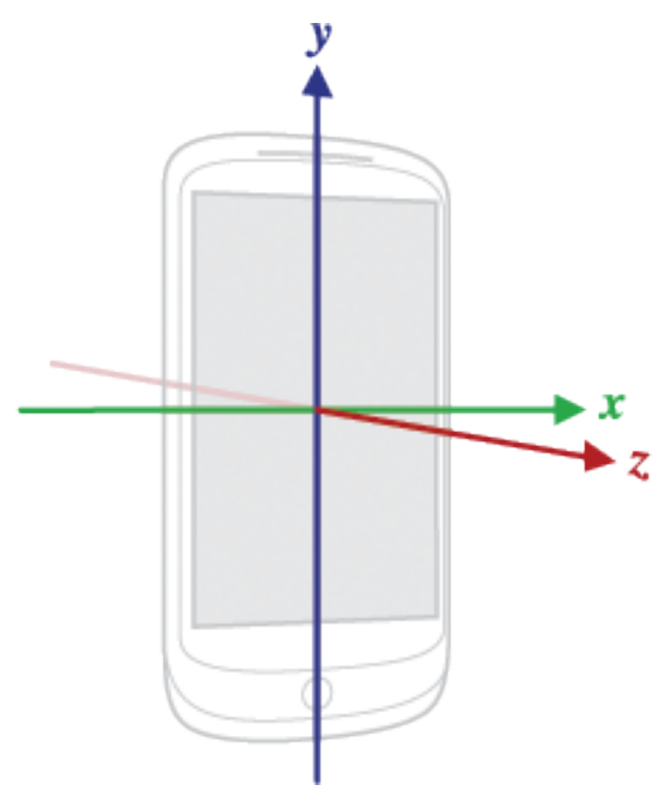
Y ≈ 9.81 m/s2

Z ≈ 0 m/s2



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 Filter

Deemphasize transient force changes

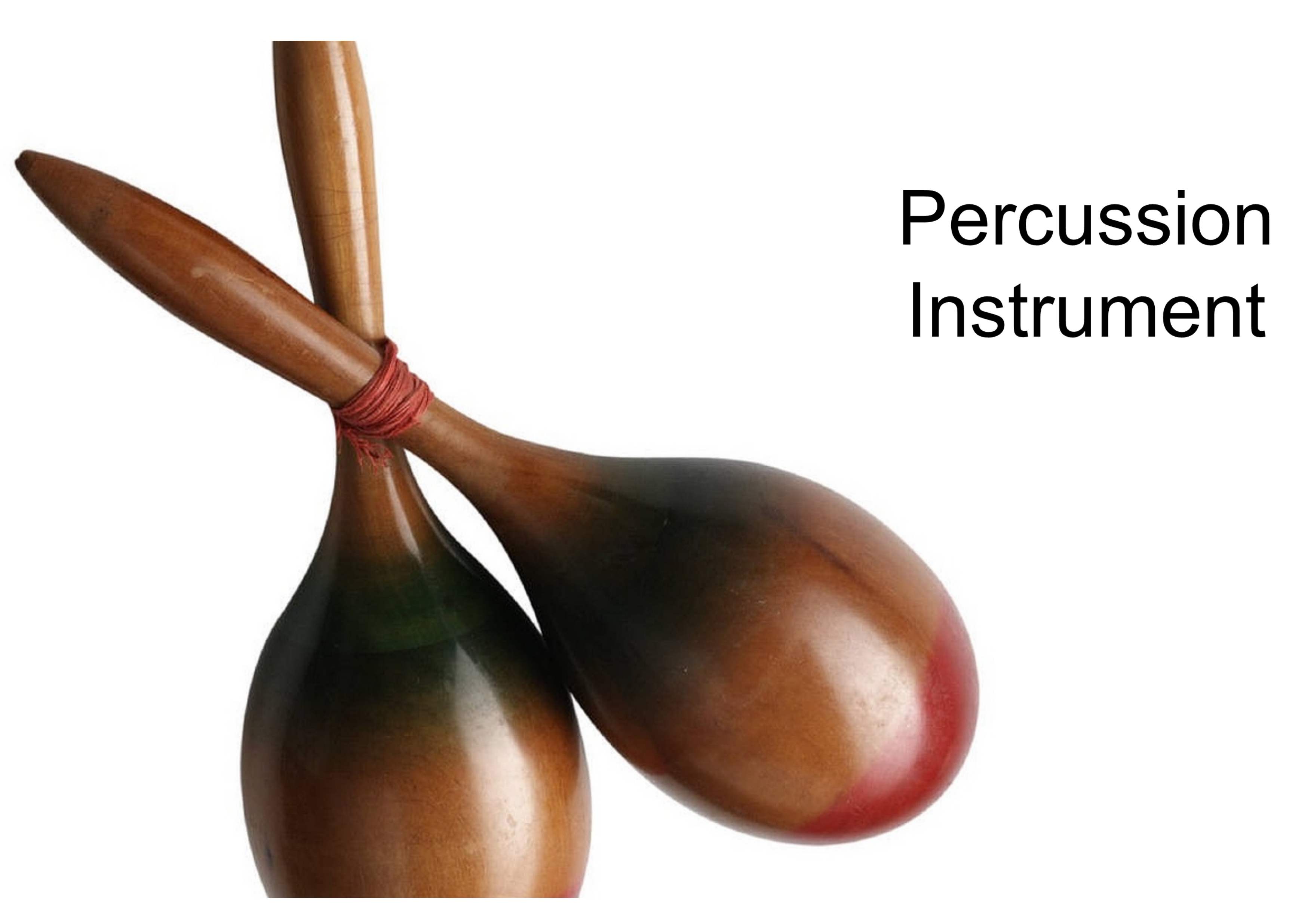
Emphasize constant force components

Carpenter's Level

High-Pass Filter

Emphasize transient force changes

Deemphasize constant force components


Percussion
Instrument

SensorFilteredAccelerometer

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

Displays the filtered values

Extended controls

Location
Cellular
Battery
Phone
Directional pad
Microphone
Fingerprint
Virtual sensors
Bug report
Settings
Help

Accelerometer Additional sensors

Rotate Move

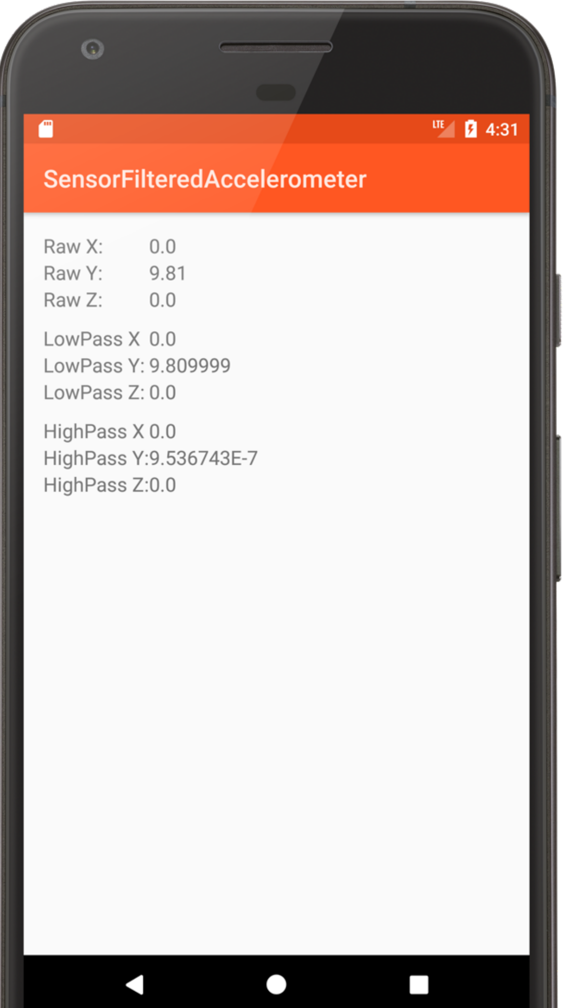
X -7 7 0.0

Y -4 4 0.0

Device rotation

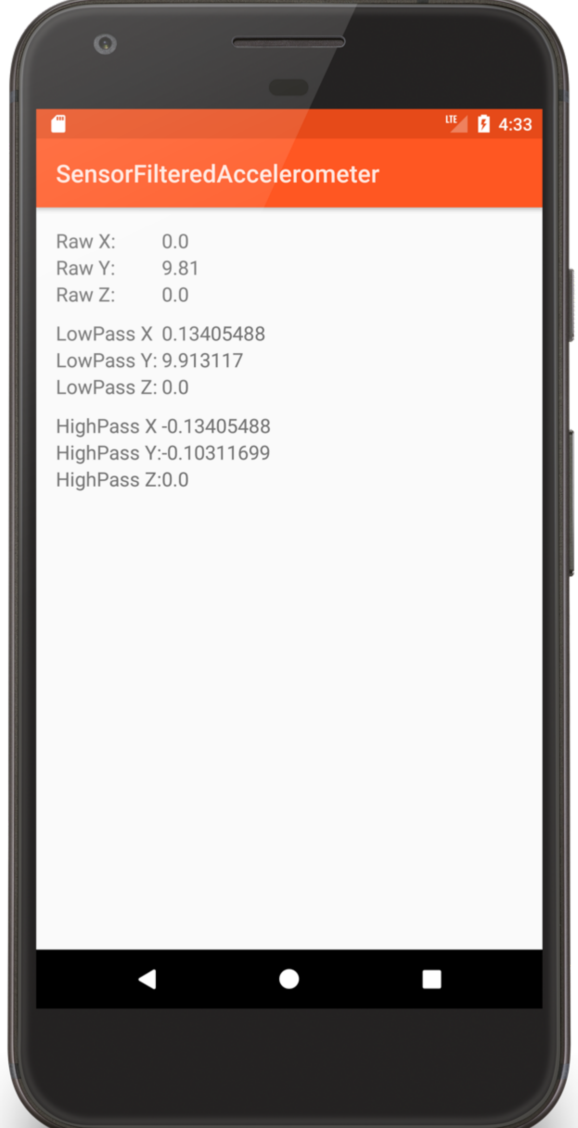
Resulting values

Accelerometer (m/s^2):	0.00	9.81	0.00
Gyroscope (rad/s):	0.00	0.00	0.00
Magnetometer (μT):	22.00	5.90	43.10
Rotation:	ROTATION_0		



SensorFilteredAccelerometer

Extended controls
Location
Cellular
Battery
Phone
Directional pad
Microphone
Fingerprint
Virtual sensors
Bug report
Settings
Help
Accelerometer
Additional sensors
Rotate
Move
X
-7
7
-1.9
Y
-4
4
-0.7
Device rotation
Resulting values
Accelerometer (m/s²): 0.00 9.81 0.00
Gyroscope (rad/s): 0.00 0.00 0.00
Magnetometer (μT): 22.00 5.90 43.10
Rotation: ROTATION_0


4:33
SensorFilteredAccelerometer
Raw X: 0.0
Raw Y: 9.81
Raw Z: 0.0
LowPass X 0.13405488
LowPass Y: 9.913117
LowPass Z: 0.0
HighPass X -0.13405488
HighPass Y:-0.10311699
HighPass Z:0.0

SensorFilteredValuesActivity.kt

```
public override fun onCreate(savedInstanceState: Bundle?) {
   …
   // Get reference to SensorManager
   mSensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager
    // Get reference to Accelerometer
    mAccelerometer =
                 mSensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER)
    if (null == mAccelerometer) {
        Log.i(localClassName, "Accelerometer not found")
    }
    mLastUpdate = System.currentTimeMillis()
}
```

SensorFilteredValuesActivity.kt

```
// Register listener
override fun onResume() {
    super.onResume()
    mSensorManager.registerListener(this, mAccelerometer,
                                            SensorManager.SENSOR_DELAY_UI)
}

// Unregister listener
override fun onPause() {
    super.onPause()
    mSensorManager.unregisterListener(this)
}
```

SensorFilteredValuesActivity.kt

```
// Process new reading
override fun onSensorChanged(event: SensorEvent) {
  if (event.sensor.type == Sensor.TYPE_ACCELEROMETER) {
    val actualTime = System.currentTimeMillis()
    if (actualTime - mLastUpdate > 500) {
      mLastUpdate = actualTime
      val rawX = event.values[0]
      val rawY = event.values[1]
      val rawZ = event.values[2]
      // Apply low-pass filter
      mGravity[0] = lowPass(rawX, mGravity[0])
      mGravity[1] = lowPass(rawY, mGravity[1])
      mGravity[2] = lowPass(rawZ, mGravity[2])
```

SensorFilteredValuesActivity.kt

```
// Apply high-pass filter
      mAccel[0] = highPass(rawX, mGravity[0])
      mAccel[1] = highPass(rawY, mGravity[1])
      mAccel[2] = highPass(rawZ, mGravity[2])
      mXValueView.text = rawX.toString()
      mYValueView.text = rawY.toString()
      mZValueView.text = rawZ.toString()
      mXGravityView.text = mGravity[0].toString()
      mYGravityView.text = mGravity[1].toString()
      mZGravityView.text = mGravity[2].toString()
      mXAccelView.text = mAccel[0].toString()
      mYAccelView.text = mAccel[1].toString()
      mZAccelView.text = mAccel[2].toString()
    …
```

SensorFilteredValuesActivity.kt

```
// Deemphasize transient forces
private fun lowPass(current: Float, gravity: Float): Float {
    val mAlpha = 0.8f
    return gravity * mAlpha + current * (1 - mAlpha)
}

// Deemphasize constant forces
private fun highPass(current: Float, gravity: Float): Float {
    return current - gravity
}
```

SensorCompass

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

Extended controls

Location
Cellular
Battery
Phone
Directional pad
Microphone
Fingerprint
Virtual sensors
Bug report
Settings
Help

Accelerometer Additional sensors

Rotate Move

Yaw -180 180 -180.0

Pitch -180 180 -45.0

Roll -180 180 0.0

Device rotation

Resulting values

Accelerometer (m/s²):	0.00	-6.94	6.94
Gyroscope (rad/s):	-0.00	0.00	-0.00
Magnetometer (μT):	-22.00	26.30	34.65
Rotation:	ROTATION_0		

Extended controls

Location
Cellular
Battery
Phone
Directional pad
Microphone
Fingerprint
Virtual sensors
Bug report
Settings
Help

Accelerometer | Additional sensors

Rotate
Move

Yaw -180 180 -52.9
Pitch -180 180 -45.0
Roll -180 180 0.0

Device rotation

Resulting values

Accelerometer (m/s²):	-5.53	4.18	6.94
Gyroscope (rad/s):	0.00	-0.00	-0.00
Magnetometer (μT):	34.25	1.68	34.65
Rotation:	ROTATION_0		

CompassActivity.kt

```
override fun onCreate(savedInstanceState: Bundle?) {
    …
    // Get a reference to the SensorManager
    mSensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager

    // 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()
}
```

CompassActivity.kt

```
override fun onResume() {
    super.onResume()
    // Register for sensor updates
    mSensorManager.registerListener(this, accelerometer,
                                          SensorManager.SENSOR_DELAY_NORMAL)
    mSensorManager.registerListener(this, magnetometer,
                                          SensorManager.SENSOR_DELAY_NORMAL)
}

override fun onPause() {
    super.onPause()
    // Unregister all sensors
    mSensorManager.unregisterListener(this)
}
```

CompassActivity.kt

```
override fun onSensorChanged(event: SensorEvent) {
    // Acquire accelerometer event data
    if (event.sensor.type == Sensor.TYPE_ACCELEROMETER) {
        mGravity = FloatArray(3)
        System.arraycopy(event.values, 0, mGravity, 0, 3)
    } else if (event.sensor.type == Sensor.TYPE_MAGNETIC_FIELD) {
        mGeomagnetic = FloatArray(3)
        System.arraycopy(event.values, 0, mGeomagnetic, 0, 3)
    }
    // 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) {
        val rotationMatrix = FloatArray(9)
```

CompassActivity.kt

```
// Users the accelerometer and magnetometer readings to compute the
//device's rotation with respect to a real-world coordinate system
   val success = SensorManager.getRotationMatrix(
                          rotationMatrix,null, mGravity, mGeomagnetic)
   if (success) {
       val orientationMatrix = FloatArray(3)
       // Returns the device's orientation given the rotationMatrix
       SensorManager.getOrientation(rotationMatrix, orientationMatrix)
       // Get the rotation, measured in radians, around the Z-axis
       val rotationInRadians = orientationMatrix[0]
       // Convert from radians to degrees
       mRotationInDegrees = Math.toDegrees(rotationInRadians.toDouble())
       // Request redraw
       mCompassArrow.invalidate()
       // Reset sensor event data arrays
       mGeomagnetic = null
       mGravity = null
   }
```

Next Time

Maps & Location

Example Applications

SensorRawAccelerometer

SensorFilteredAccelerometer

SensorCompass