Sensing and Sensors

• *a capability that can capture measurements about the device and its external environment*

• Can detects and responds to some type of input from the physical environment.

• The specific input could be light, heat, motion, moisture, pressure, etc.

• Convert the measurement into a signal that can be read.
Example Sensor Data

1 | MPU6500 Acceleration Sensor | [-1.6741456, 9.370906, 2.6886885] | 1441670212915

1 | MPU6500 Gyroscope Sensor | [-0.02263687, -0.016777916, -0.008788432] | 1441670213508

1 | AK09911C Magnetic field Sensor | [16.86, -64.26, -62.7] | 1441670213400

1 | GPS |{"mProvider":"fused","mResults":[0.0,0.0],"mAccuracy":29.0,"mAltitude":83.0,"mLatitude":40.748431","mLongitude":73.985741 ... } | 1441573552851

1 | WiFi |{"BSSID":"00:21:6c:87:02:d1","SSID":"eduroam","capabilities":[WPA2-EAP-CCMP","frequency": 2462,"level":-82} | 1392465248466
Android OS Fragmentation

http://opensignal.com/reports/2015/08/android-fragmentation/
http://opensignal.com/reports/2015/08/android-fragmentation/
Sensor Availability

- Varies from device to device
- May vary between Android versions
Newer Sensors Over Time
Sensors

- **Position sensors**
  GPS, orientation sensors and magnetometers.

- **Motion sensors**
  accelerometers, gravity sensors, gyroscopes, etc.

- **Environmental sensors**
  barometers, photometers, and thermometers.
Sensor Framework

Access sensors and acquire raw sensor data.

• Determine which sensors are available on a device.
• Determine an individual sensor's capabilities, such as its maximum range, manufacturer, power requirements, and resolution.
• Acquire raw sensor data and define the minimum rate at which you acquire sensor data.
• Register and unregister sensor event listeners that monitor sensor changes.
Sensors

- **Hardware-based** or **Software-based**
- **Hardware-based sensors** - physical components built into a handset or tablet device
  - directly measuring specific environmental properties.
    - acceleration, geomagnetic field strength, or angular change.
- **Software-based sensors** - mimic hardware-based sensors
  - derive their data from one or more of the hardware-based sensors and are sometimes called virtual sensors or synthetic sensors.
    - The linear acceleration sensor and the gravity sensor.
Sensor Framework

Access sensors and acquire raw sensor data. Android Sensor Framework includes three classes and one interface.

- SensorManager
- Sensor
- SensorEvent
- SensorEventListener

Identifying sensors and sensor capabilities
Monitor sensor events

public class SensorActivity extends Activity, implements SensorEventListener {
    private final SensorManager mSensorManager;
    private final Sensor mAccelerometer;

    public SensorActivity() {
        mSensorManager = (SensorManager) getSystemService(SENSOR_SERVICE);
        mAccelerometer = mSensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER);
    }

    protected void onResume() {
        super.onResume();
        mSensorManager.registerListener(this, mAccelerometer, SensorManager.SENSOR_DELAY_NORMAL);
    }

    protected void onPause() {
        super.onPause();
        mSensorManager.unregisterListener(this);
    }

    public void onAccuracyChanged(Sensor sensor, int accuracy) {
    }

    public void onSensorChanged(SensorEvent event) {
    }
}
SensorManager

• System Service that manages sensors

• First the application needs to get a reference to the SensorManager
  • `getSystemService(SENSOR_SERVICE);`

• Access a specific sensor with
  • `SensorManager.getDefaultSensor(int type)`
Sensor

- Accelerometer
  - Sensor.TYPE_ACCELEROMETER
- Magnetic Field
  - Sensor.TYPE_MAGNETIC_FIELD
- Pressure
  - Sensor.TYPE_PRESSURE
Get a List of All Sensors

```
SensorManager sensorManager = 
    (SensorManager) getSystemService(Activity.SENSOR_SERVICE);
List<Sensor> sensors = sensorManager.getSensorList(Sensor.TYPE_ALL);
```
SensorEventListener

• For an application to receive information from a Sensor
  • It needs to implement a SensorEventListener
  • Before starting to receive sensorEvents

```java
protected void onResume() {
    super.onResume();
    mSensorManager.registerListener(this, mAccelerometer, SensorManager.SENSOR_DELAY_NORMAL);
}
```

• Type of Sensor
• Delay
SensorEventListener

registerListener

```java
boolean registerListener (SensorEventListener listener,
                        Sensor sensor,
                        int samplingPeriodUs)
```

<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>sensor</td>
<td>Sensor: The Sensor to register to.</td>
</tr>
<tr>
<td>samplingPeriodUs</td>
<td>int: The rate sensor events are delivered at. This is only a hint to the system. Events may be received faster or slower than the specified rate. Usually events are received faster. The value must be one of SENSOR_DELAY_NORMAL, SENSOR_DELAY_UI, SENSOR_DELAY_GAME, or SENSOR_DELAY_FASTEST or, the desired delay between events in microseconds. Specifying the delay in microseconds only works from Android 2.3 (API level 9) onwards. For earlier releases, you must use one of the SENSOR_DELAY_* constants.</td>
</tr>
</tbody>
</table>
SensorDelay

SENSOR_DELAY_FASTEST

int SENSOR_DELAY_FASTEST

get sensor data as fast as possible

Constant Value: 0 (0x00000000)

SENSOR_DELAY_UI

int SENSOR_DELAY_UI

rate suitable for the user interface

Constant Value: 2 (0x00000002)

SENSOR_DELAY_GAME

int SENSOR_DELAY_GAME

rate suitable for games

Constant Value: 1 (0x00000001)

SENSOR_DELAY_NORMAL

int SENSOR_DELAY_NORMAL

rate (default) suitable for screen orientation changes

Constant Value: 3 (0x00000003)
SensorEventListener

registerListener

```java
boolean registerListener (SensorEventListener listener,
    Sensor sensor,
    int samplingPeriodUs,
    Handler handler)
```

| handler | **Handler**: The **Handler** the **sensor events** will be delivered to. |
SensorEventListener

registerListener

boolean registerListener (SensorEventListener listener,
    Sensor sensor,
    int samplingPeriodUs,
    int maxReportLatencyUs,
    Handler handler)

maxReportLatencyUs: int: Maximum time in microseconds that events can be delayed before being reported to the application. A large value allows reducing the power consumption associated with the sensor. If maxReportLatencyUs is set to zero, events are delivered as soon as they are available, which is equivalent to calling registerListener(SensorEventListener, Sensor, int).
SensorEventListener

• For an application to receive information from a Sensor
  • It needs to implement a SensorEventListener
    • Invoked when accuracy of a sensor changes
    • When the sensor acquires a new reading

<table>
<thead>
<tr>
<th>abstract void</th>
<th>onAccuracyChanged (Sensor sensor, int accuracy)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Called when the accuracy of the registered sensor has changed.</td>
</tr>
<tr>
<td>abstract void</td>
<td>onSensorChanged (SensorEvent event)</td>
</tr>
<tr>
<td></td>
<td>Called when sensor values have changed.</td>
</tr>
</tbody>
</table>
onAccuracyChanged

```java
public void onAccuracyChanged(Sensor sensor, int accuracy) {
}
```

- Accuracy is represented by one of four status constants:
  - `SENSOR_STATUS_UNRELIABLE`  
    - Constant Value: 0 (0x00000000)
  - `SENSOR_STATUS_ACCURACY_LOW`  
    - Constant Value: 1 (0x00000001)
  - `SENSOR_STATUS_ACCURACY_MEDIUM`  
    - Constant Value: 2 (0x00000002)
  - `SENSOR_STATUS_ACCURACY_HIGH`  
    - Constant Value: 3 (0x00000003)

onAccuracyChanged

```java
public void onAccuracyChanged(Sensor sensor, int accuracy) {
    // Do something here if sensor accuracy changes.
    // You must implement this callback in your code.
    if (sensor == mValuen) {
        switch (accuracy) {
            case 0:
                System.out.println("Unreliable");
                con=0;
                break;
            case 1:
                System.out.println("Low Accuracy");
                con=0;
                break;
            case 2:
                System.out.println("Medium Accuracy");
                con=0;
                break;
            case 3:
                System.out.println("High Accuracy");
                con=1;
                break;
        }
    }
}
```
onSensorChanged

```java
public void onSensorChanged(SensorEvent se) {
    float x = se.values[0];
    float y = se.values[1];
    float z = se.values[2];
    mAccelLast = mAccelCurrent;
    mAccelCurrent = (float) Math.sqrt((double) (x*x + y*y + z*z));
    float delta = mAccelCurrent - mAccelLast;
    mAccel = mAccel * 0.9f + delta; // perform low-cut filter
}
```
public void onSensorChanged(SensorEvent event) {
    if (event.sensor.getType() == Sensor.TYPE_MAGNETIC_FIELD)
        magnetic = event.values;
    if (event.sensor.getType() == Sensor.TYPE_ACCELEROMETER)
        gravity = event.values;
    if ((gravity == null) || (magnetic == null))
        return;
}
SensorEventListener

• Once you are done Using the Sensor

```java
protected void onPause() {
    super.onPause();
    mSensorManager.unregisterListener(this);
}
```
public class SensorActivity extends Activity, implements SensorEventListener {
    private final SensorManager mSensorManager;
    private final Sensor mAccelerometer;

    public SensorActivity() {
        mSensorManager = (SensorManager) getSystemService(SENSOR_SERVICE);
        mAccelerometer = mSensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER);
    }

    protected void onResume() {
        super.onResume();
        mSensorManager.registerListener(this, mAccelerometer, SensorManager.SENSOR_DELAY_NORMAL);
    }

    protected void onPause() {
        super.onPause();
        mSensorManager.unregisterListener(this);
    }

    public void onAccuracyChanged(Sensor sensor, int accuracy) {
    }

    public void onSensorChanged(SensorEvent event) {
    }
}
Sensor Coordinate System

• The sensor framework uses a standard 3-axis coordinate system to express data values.
  • **X axis** is horizontal and points to the right
  • **Y axis** is vertical and points up
  • **Z axis** points toward the outside of the screen face
    • coordinates behind the screen have negative values
Sensor Coordinate System

Such a coordinate system is used by:

- Acceleration sensor
- Gravity sensor
- Gyroscope
- Linear acceleration sensor
- Geomagnetic field sensor
Points to Remember

• Your application must not assume that a device's natural (default) orientation is portrait.
  • The sensor coordinate system is always based on the natural orientation of a device.
  • The natural orientation for many tablet devices is landscape.

• Verify sensors before you use them
  • Verify that a sensor exists on a device before you attempt to acquire data from it
  The sensor's coordinate system never changes as the device moves

• You must test your sensor code on a physical device.
  • You currently can't test sensor code on the emulator because the emulator cannot emulate sensors.
  • There are, however, sensor simulators that you can use to simulate sensor output.
Verify sensors before you use them

If you are publishing your application on Google Play you can **use the `<uses-feature>` element in your manifest file** to filter your application from devices that do not have the appropriate sensor configuration for your application.

```java
private SensorManager mSensorManager;
...
if (mSensorManager.getDefaultSensor(Sensor.TYPE_PRESSURE) != null){
// Success! There's a pressure sensor.
}
else {
// Failure! No pressure sensor.
}
```

```xml
<uses-feature android:name="android.hardware.sensor.accelerometer"
    android:required="true"/>
```
Points to Remember

• Unregister sensor listeners
  • when you are done using the sensor or when the sensor activity pauses.
  • If a sensor listener is registered and its activity is paused, the sensor will continue to acquire data and use battery resources unless you unregister the sensor.

• Don't block the onSensorChanged() method
  • Sensor data can change at a high rate - system may call the onSensorChanged(SensorEvent) method quite often
  • Do as little as possible within the onSensorChanged(SensorEvent) method so you don't block it

• Choose sensor delays wisely
  • Sensors can provide data at very high rates.
  • Sending extra data that you don't need wastes system resources and uses battery power.
Orientation Sensors

- **TYPE_ACCELEROMETER** uses the accelerometer and only the accelerometer. It returns raw accelerometer events, with minimal or no processing at all.

- **TYPE_LINEAR_ACCELERATION** (if present) uses the gyroscope and only the gyroscope. Like above, it returns raw events (angular speed un rad/s) with no processing at all (no offset / scale compensation).

- **TYPE_ORIENTATION** is deprecated. It returns the orientation as yaw/pitch/roll in degrees.
  - This sensor uses a combination of the accelerometer and the magnetometer.
  - Marginally better results can be obtained using SensorManager's helpers.
  - This sensor is heavily "processed".
Orientations

Accelerometer

Gyroscope
# Orientation Sensors

<table>
<thead>
<tr>
<th>TYPE_ACCELEROMETER</th>
<th>SensorEvent.values[0]</th>
<th>Acceleration force along the x axis (including gravity).</th>
<th>m/s²</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>SensorEvent.values[1]</td>
<td>Acceleration force along the y axis (including gravity).</td>
<td></td>
</tr>
<tr>
<td></td>
<td>SensorEvent.values[2]</td>
<td>Acceleration force along the z axis (including gravity).</td>
<td></td>
</tr>
<tr>
<td>TYPE_GRAVITY</td>
<td>SensorEvent.values[0]</td>
<td>Force of gravity along the x axis.</td>
<td>m/s²</td>
</tr>
<tr>
<td>TYPE_GYROSCOPE</td>
<td>SensorEvent.values[0]</td>
<td>Rate of rotation around the x axis.</td>
<td>rad/s</td>
</tr>
<tr>
<td></td>
<td>SensorEvent.values[1]</td>
<td>Rate of rotation around the y axis.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>SensorEvent.values[2]</td>
<td>Rate of rotation around the z axis.</td>
<td></td>
</tr>
</tbody>
</table>
Orientation Sensors.

- **TYPE_LINEAR_ACCELERATION**, **TYPE_GRAVITY**, **TYPE_ROTATION VECTOR** are "fused" sensors which return respectively
  - the linear acceleration,
  - gravity and
  - rotation vector (a quaternion).
- On some devices they are implemented in h/w,
- On some devices they use the accelerometer + the magnetometer
- On some other devices they use the gyro.

<table>
<thead>
<tr>
<th>TYPE_LINEAR_ACCELERATION</th>
<th>SensorEvent.values[0]</th>
<th>Acceleration force along the x axis (excluding gravity).</th>
<th>m/s²</th>
</tr>
</thead>
<tbody>
<tr>
<td>SensorEvent.values[1]</td>
<td></td>
<td>Acceleration force along the y axis (excluding gravity).</td>
<td></td>
</tr>
</tbody>
</table>
MEMS accelerometers are tiny masses on tiny springs.

They can sense

- Speeding up or slowing down in a straight line
- Shaking the device
- Earth’s gravity, which is 1 g downward
TYPE_ACCELEROMETER

At rest

The graph shows the acceleration over time. At rest, the graph indicates that there is no change in acceleration over time. The acceleration values are constant, indicating a lack of movement or change in the system being measured.
TYPE_LINEAR_ACCELERATION

At rest
TYPE_ACCELEROMETER

Rotation around y axis

![Acceleration Graph](image)

- Red: z
- Blue: y
- Green: x
Measures rate of rotation.
You cannot directly measure angle using a gyroscope.
You can integrate the rate of rotation over time to get angle.
Activity Recognition

Physical Activity Level Inference

\[ m = \sqrt{a_x^2 + a_y^2 + a_z^2} \]
Walking
Bus
Driving
Train
Activity Recognition Study

Running
## Android "DetectedActivity" API

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IN_VEHICLE</td>
<td>The device is in a vehicle, such as a car.</td>
</tr>
<tr>
<td>ON_BICYCLE</td>
<td>The device is on a bicycle.</td>
</tr>
<tr>
<td>ON_FOOT</td>
<td>The device is on a user who is walking or running.</td>
</tr>
<tr>
<td>RUNNING</td>
<td>The device is on a user who is running.</td>
</tr>
<tr>
<td>STILL</td>
<td>The device is still (not moving).</td>
</tr>
<tr>
<td>TILTING</td>
<td>The device angle relative to gravity changed significantly.</td>
</tr>
<tr>
<td>UNKNOWN</td>
<td>Unable to detect the current activity.</td>
</tr>
<tr>
<td>WALKING</td>
<td>The device is on a user who is walking.</td>
</tr>
</tbody>
</table>

TYPE_MAGNETIC_FIELD

• Hardware Sensor
• Mostly Hall effect Sensors
• Android reports magnetic fields in microtesla.
• Earth’s magnetic field can vary from 30 microtesla to 60 microtesla
• Uses
  • Compass
  • The magnetic field sensor can be influenced by nearby metal, some people have used the sensor to make an Android device into a crude metal detector
    • Due to an effect called hysteresis
TYPE_PROXIMITY

• Hardware Sensor
• Lets you determine how far away an object is from a device
  • Some proximity sensors provide a boolean value (near/far).
    • Typically, the far value is a value > 5 cm, but this can vary from sensor to sensor.
• Usually used to determine how far away a person's head is from the face of a handset device
  • To lock the screen when a user is on a call
public class SensorActivity extends Activity implements SensorEventListener {
    private SensorManager mSensorManager;
    private Sensor mProximity;

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

        // Get an instance of the sensor service, and use that to get an instance of
        // a particular sensor.
        mSensorManager = (SensorManager) getSystemService(Context.SENSOR_SERVICE);
        mProximity = mSensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY);
    }
```java
@Override
public final void onSensorChanged(SensorEvent event) {
    float distance = event.values[0];
    // Do something with this sensor data.
}

@Override
protected void onResume() {
    // Register a listener for the sensor.
    super.onResume();
    mSensorManager.registerListener(this, mProximity, SensorManager.SENSOR_DELAY_NORMAL);
}

@Override
protected void onPause() {
    // Be sure to unregister the sensor when the activity pauses.
    super.onPause();
    mSensorManager.unregisterListener(this);
}
```
TYPE_STEP_COUNTER

• Returns the number of steps taken by the user since the last reboot while activated.
• Reset to zero only on a system reboot.
• The timestamp of the event is set to the time when the last step for that event was taken.
• This sensor is implemented in hardware and is expected to be low power.
• Application needs to stay registered for this sensor because step counter does not count steps if it is not activated.
TYPE_STEP_DETECTOR

• Triggers an event each time a step is taken by the user.
• The only allowed value to return is 1.0 and an event is generated for each step.
• The timestamp indicates when the event (here the step) occurred
  • When the foot hits the ground - generating a high variation in acceleration.
ENVIRONMENT SENSORS

• The Android platform provides four sensors that let you monitor various environmental properties.
  • Ambient Pressure
    • Measures the ambient air pressure in hPa or mbar.
  • Ambient Humidity
    • Ambient humidity near the phone (expressed as % atmospheric humidity)
  • Illuminance
    • Used to control screen brightness (measured in lux)
  • Ambient temperature
    • Ambient humidity near the phone (measured in degree centigrade)

• They are all hardware sensors.
TYPE_HEART_RATE

• Found in Android Wearables.
• The reported value is the heart rate in beats per minute.
• This sensor requires permission `android.permission.BODY_SENSORS`
• It will not be returned by `SensorManager.getSensorsList` or `SensorManager.getDefaultSensor` if the permission is missing.
HEART RATE SENSOR

• Measured using a Photoplethysmography (PPG sensor).
• Measures the differential reflection of light by oxygenated and deoxygenated blood
HEART RATE SENSOR

- Similar to the principle of Android apps to measure heart rate using the camera.
ARTIFACTS IN HEART RATE

- Sensor Movement Artifact
- Nervous Fidgeting Artifact

Removal of Local Fidgeting

Changes in Wristband Acc + Physiological signal = YES
Changes in Smartphone Acc/Gyro = NO

Remove next 30 seconds
Artifact Removal – Activity Recognition Use Case.
Removal Artifacts using Filtering

• Two common filtering techniques
  • Low-pass filters
    • Pass frequencies lower than cut off frequency
    • Deemphasize transient force change (vibrations)
    • Emphasize constant force components
    • e.g., for a bubble level
  • High-pass filters
    • Pass frequencies higher than cut off frequency
    • Emphasize transient force changes
    • Deemphasize constant force components (gravity)
    • e.g., for a game controller
Signal Preprocessing

Input Streams

Preprocessing

\[ A_{\text{norm}} = \sqrt{A_x^2 + A_y^2 + A_z^2} \]

\[ G_{\text{norm}} = \sqrt{G_x^2 + G_y^2 + G_z^2} \]
Signal Preprocessing

\[ A_{norm} = \sqrt{A_x^2 + A_y^2 + A_z^2} \]

\[ G_{norm} = \sqrt{G_x^2 + G_y^2 + G_z^2} \]

Boundary Removal

\[ N = \begin{cases} 
T/10 \text{ if } N < 300 \\
30 \text{ secs Otherwise} 
\end{cases} \]
Signal Preprocessing

40 Hz
Accelometer
40 Hz
Gyroscope
8 KHz
Audio

Input Streams

Preprocessing

Screen-on checking and removal

For controlled scenario: If screen-unlocked > 10 seconds

\[ A_{norm} = \sqrt{A_x^2 + A_y^2 + A_z^2} \]

\[ G_{norm} = \sqrt{G_x^2 + G_y^2 + G_z^2} \]
Signal Preprocessing

Input Streams
- Accelerometer
- Gyroscope
- Audio

Preprocessing

\[ A_{\text{norm}} = \sqrt{A_x^2 + A_y^2 + A_z^2} \]

\[ G_{\text{norm}} = \sqrt{G_x^2 + G_y^2 + G_z^2} \]

Boundary Removal And Filtering

Segment into 3 second window

Segmentation

Human Average Stride rate is between 80-120 steps a minute
Signal Preprocessing

Input Streams

Preprocessing

Segmentation

Feature Extraction

Accelerometer

Gyroscope

Audio

Boundary Removal
And Filtering

Segment into 3 second window

Preprocessing

Segmentation

Feature Extraction

Input Streams

Preprocessing

Segmentation

Feature Extraction

Mean

Standard deviation

Number of peaks

Inter peak distances

minimum

maximum

Zero Crossing rate

RMS Energy

MFCCs

Signal Preprocessing

\[ A_{\text{norm}} = \sqrt{A_x^2 + A_y^2 + A_z^2} \]

\[ G_{\text{norm}} = \sqrt{G_x^2 + G_y^2 + G_z^2} \]
Signal Preprocessing

Input Streams

- Accelerometer
- Gyroscope
- Audio

Preprocessing

\[ A_{\text{norm}} = \sqrt{A_x^2 + A_y^2 + A_z^2} \]

\[ G_{\text{norm}} = \sqrt{G_x^2 + G_y^2 + G_z^2} \]

Segmentation into 3-second window

Boundary Removal
And Filtering

Feature Extraction

- Mean
- Standard deviation
- Number of peaks
- Inter peak distances
- Minimum
- Maximum
- Zero Crossing rate
- RMS Energy
- MFCCs

Classification

Random Forest
LOSO

ACC
GYRO
ACC+GYRO
ACC+GYRO+AUDI0
Challenge

High sampling rate drains battery
Effects of lowering sampling rates
Publication Related to this section

Recognizing Human Activities from Smartphone Sensor Signals. ACM Multimedia 2014,
**Ghosh, Arindam**, and Riccardi, Giuseppe