Mobile and Ubiquitous Computing on Smartphones

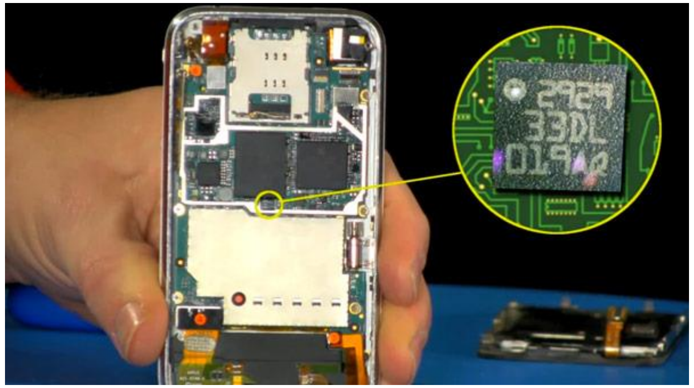
Lecture 7b: Android Sensors and Step Counting

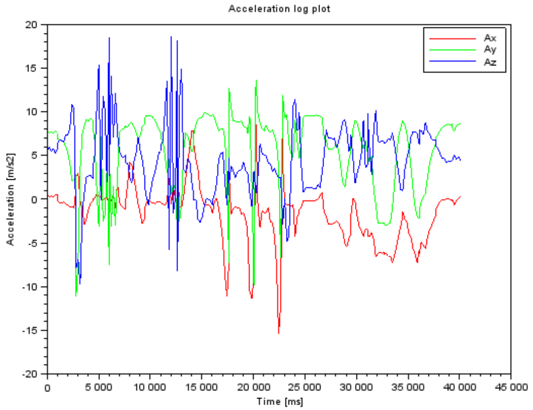
Emmanuel Agu

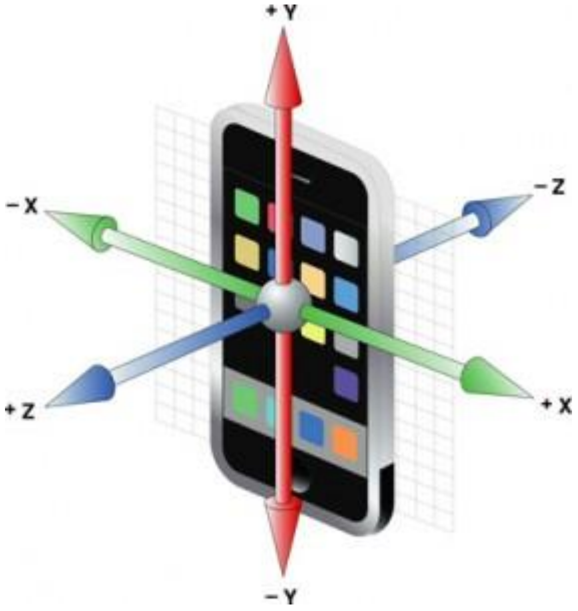
Android Sensors

What is a Sensor?

- Converts physical quantity (e.g. light, acceleration, magnetic field) into a signal
- **Example:** accelerometer converts acceleration along X,Y,Z axes into signal

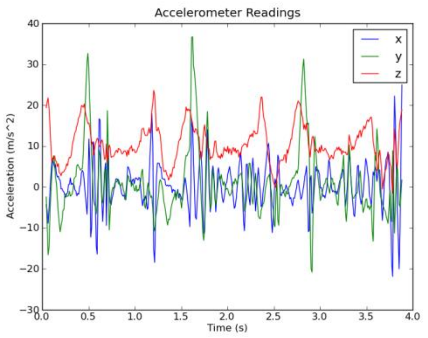




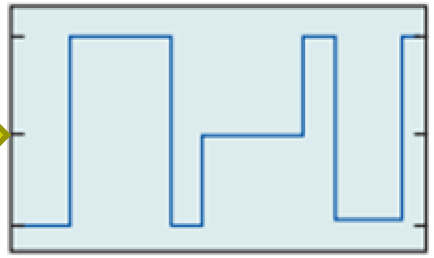


So What?

- Raw sensor data can be processed into useful info
- **Example:** Raw accelerometer data can be processed/classified to infer/guess user's activity (e.g. walking running, etc)
- Voice samples can be processed/classified to infer/guess whether speaker is nervous or not



Raw accelerometer readings



Machine learning
Feature extraction and classification

Walking
Running
Jumping
Step count
Calories burned
Falling

Android Sensors: Categories

Motion sensors

- Accelerometer
- Gyroscope (orientation)

Environmental sensors

- Temperature
- Pressure
- Light levels

Position sensors

- Orientation
- Magnetometer
- Proximity

- **Important:** Different phones have different sets of sensor types!!

Apps to play around with Android Sensors

AndroSensor

Sensor Box

Android Sensor Framework

https://developer.android.com/guide/topics/sensors/sensors_overview

- Can be used to programmatically:
  - Determine **which sensors** are available on phone
  - Determine **capabilities of sensors** (e.g. max. values, range, resolution, manufacturer, power requirements)
  - **Register and unregister** sensor event listeners
  - **Acquire raw sensor data** and define data rate

Android Sensor Framework

https://developer.android.com/guide/topics/sensors/sensors_overview

- Android sensors can be either hardware or software

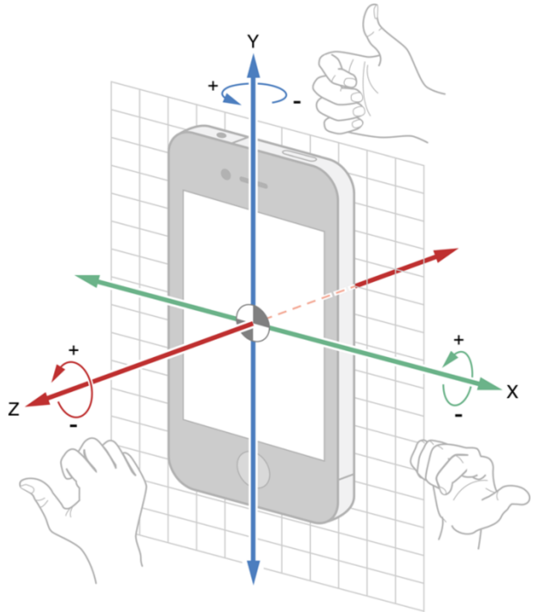
- **Hardware sensor:**
  - physical components (integrated circuit) built into phone,
  - **Example:** temperature

- **Software sensor (or virtual sensor):**
  - Not physical device
  - Their values calculated from one or more hardware sensors (a formula)
  - **Example:** gravity sensor

Sensor Types Supported by Android

- TYPE_PROXIMITY
  - Measures an **object's proximity to device's screen**
  - **Common uses:** determine if handset is held to ear

- TYPE_GYROSCOPE
  - Measures device's **rate of rotation** around X,Y,Z axes in rad/s
  - **Common uses:** rotational speed detection (spin, turn, etc)



Types of Sensors

Sensor	HW/SW	Description	Use
TYPE_ACCELEROMETER	HW	Rate of change of velocity	Shake, Tilt
TYPE_AMBIENT_TEMPERATURE	HW	Room temperature	Monitor Room temp
TYPE_GRAVITY	SW/HW	Gravity along X,Y,Z axes	Shake, Tilt
TYPE_GYROSCOPE	HW	Rate of rotation	Spin, Turn
TYPE_LIGHT	HW	Illumination level	Control Brightness
TYPE_LINEAR_ACCELERATION	SW/HW	Acceleration along X,Y,Z – g	Accel. Along an axis
TYPE_MAGNETIC_FIELD	HW	Magnetic field	Create Compass
TYPE_ORIENTATION	SW	Rotation about (x,y,z) axes	Device position
TYPE_PRESSURE	HW	Air pressure	Air pressure
TYPE_PROXIMITY	HW	Any object close to device?	Phone close to face?
TYPE_RELATIVE_HUMIDITY	HW	% of max possible humidity	Dew point
TYPE_ROTATION_VECTOR	SW/HW	Device's rotation vector	Device's orientation
TYPE_TEMPERATURE	HW	Temp. of device	Device temp.

2 New Hardware Sensor introduced in Android 4.4

- TYPE_STEP_DETECTOR
  - Sensor event triggered each user step (**single step**)
  - Delivered event has value of 1.0 + timestamp of step

- TYPE_STEP_COUNTER
  - Also triggers a sensor event each time user takes a step
  - Delivers total ***accumulated number of steps since this sensor was first registered by an app,***
  - Tries to analyze multiple steps, eliminate false positives

- **Common uses:** step counting, pedometer apps
- Requires hardware support, available on some phones (E.g. Google Nexus 5)
- Alternatively step counting available through Google Play Services (Google Fit)
  - https://developers.google.com/fit/scenarios/record-steps
  - https://developers.google.com/fit/scenarios/read-daily-step-total

Sensor Programming

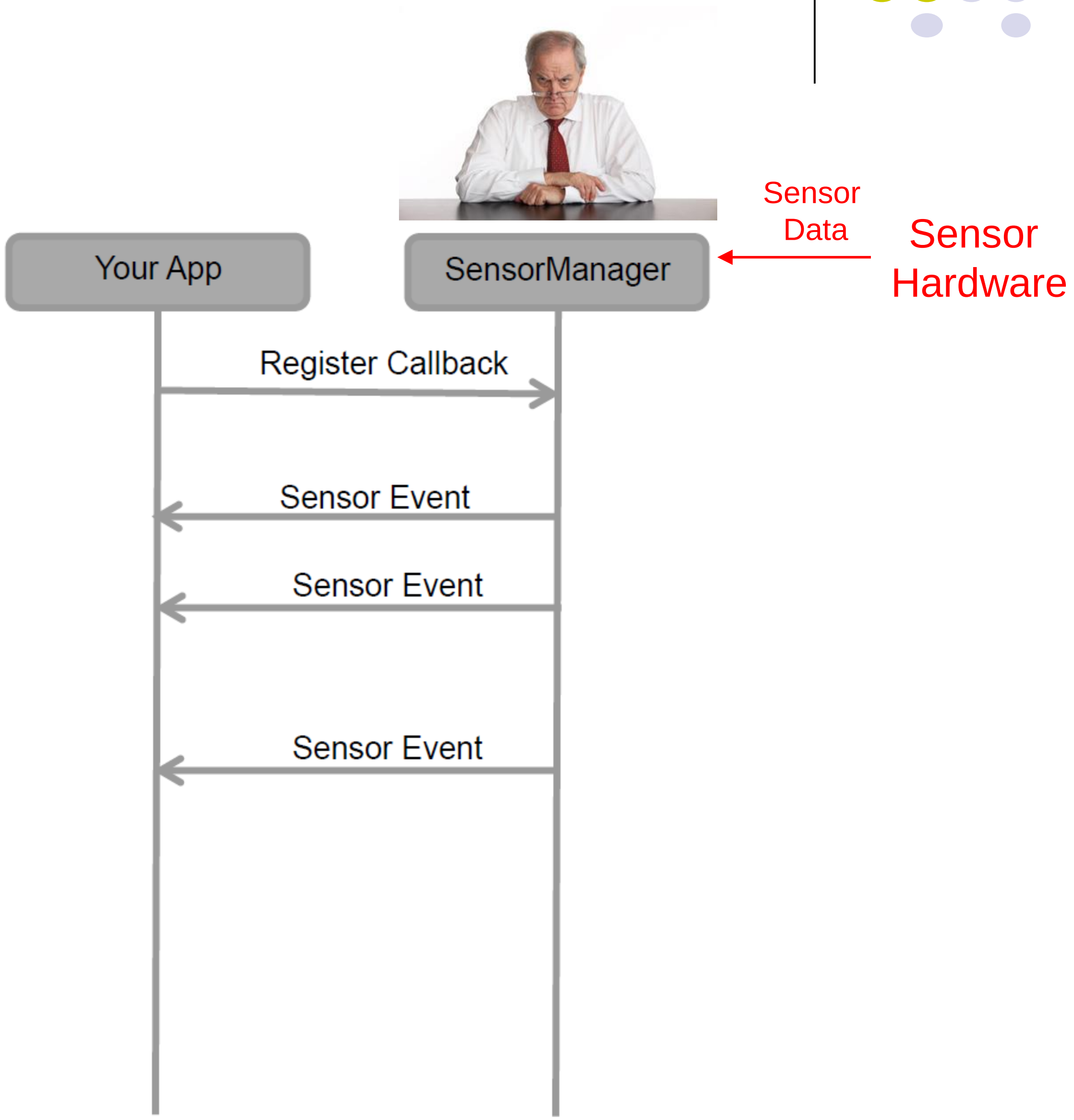
- Sensor framework is part of **android.hardware**

- Classes and interfaces include:
  - **SensorManager**
  - **Sensor**
  - **SensorEvent**
  - **SensorEventListener**

- These sensor-APIs used for:
  1. Identifying sensors and sensor capabilities
  2. Monitoring sensor events

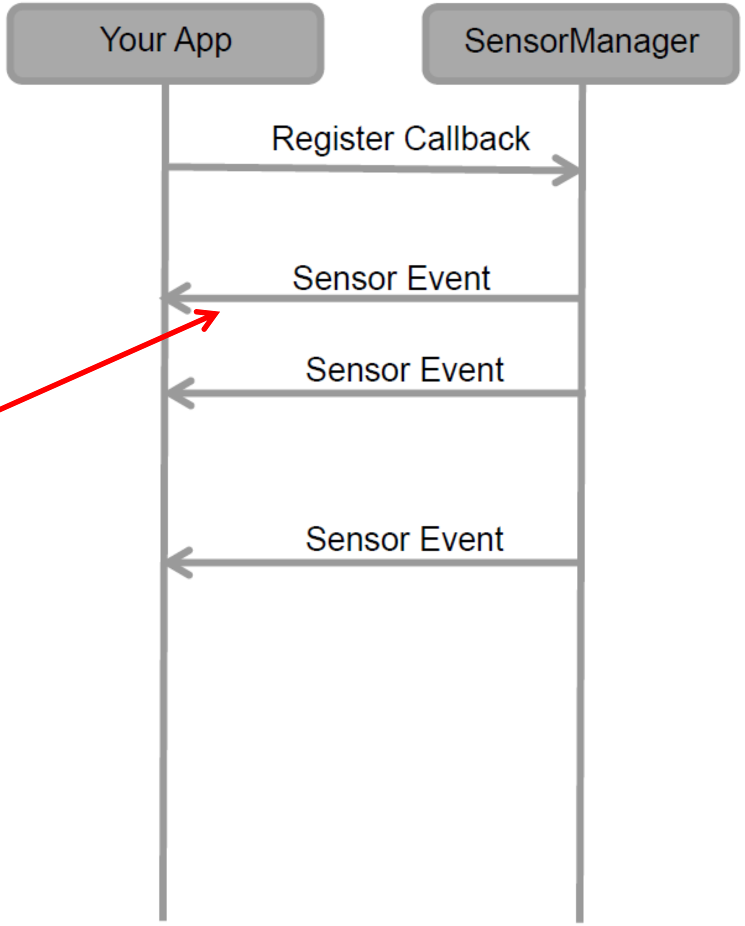
Sensor Events and Callbacks

- Sensors send data to sensor manager asynchronously, when new data arrives
- General approach:
  - App registers callbacks
  - **SensorManager** notifies app of sensor event whenever:
    - New data arrives, or
    - Accuracy changes
- Sensor app needs to create instance of **SensorManager**



Sensor

- A class used to create instance of a specific sensor
  - E.g instance of accelerometer
- Has methods used to determine a sensor’s capabilities
- Included in sensor event object

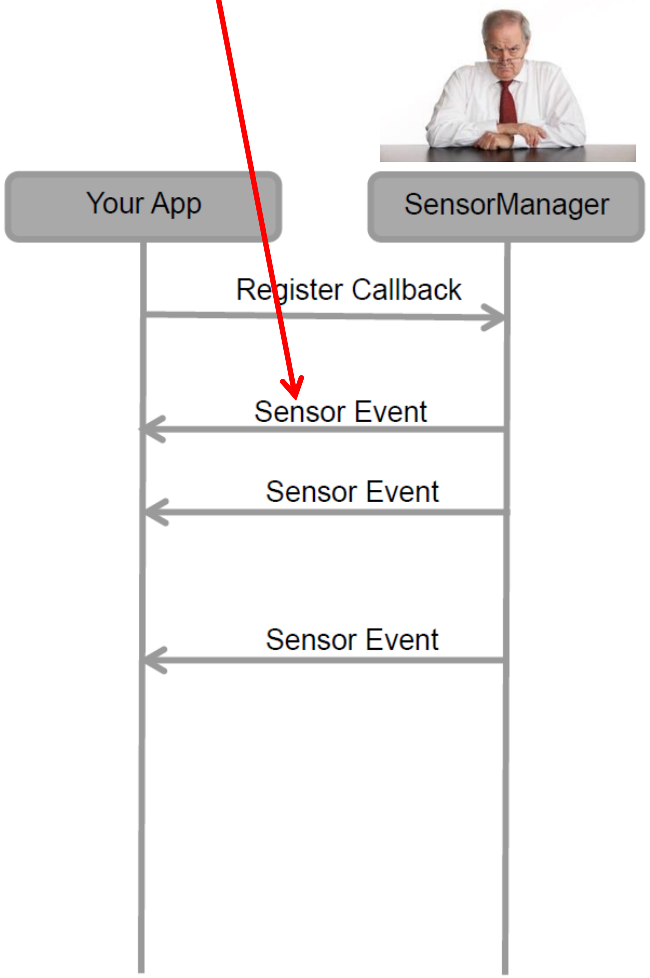


SensorEvent

- SensorManager sends sensor event information as a **sensor event object**

- **Sensor event object** includes:
  - ***Sensor:*** Type of sensor that generated the event
  - ***Values:*** Raw sensor data
  - ***Accuracy:*** Accuracy of the data
  - ***Timestamp:*** Event timestamp

Sensor value depends on sensor type



Sensor Values Depend on Sensor Type

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).	
	SensorEvent.values[2]	Acceleration force along the z axis (including gravity).	
TYPE_GRAVITY	SensorEvent.values[0]	Force of gravity along the x axis.	m/s^2
	SensorEvent.values[1]	Force of gravity along the y axis.	
	SensorEvent.values[2]	Force of gravity along the z axis.	
TYPE_GYROSCOPE	SensorEvent.values[0]	Rate of rotation around the x axis.	rad/s
	SensorEvent.values[1]	Rate of rotation around the y axis.	
	SensorEvent.values[2]	Rate of rotation around the z axis.	

https://developer.android.com/guide/topics/sensors/sensors_motion

SensorEventListener

- Interface used to create 2 callbacks that receive sensor events when:
  - Sensor values change **(onSensorChange( ))** or
  - When sensor accuracy changes **(onAccuracyChanged( ))**

Called when sensor values change

Called when sensor accuracy changes

```
//Sensor program

onSensorChange( ) {

}

onAccuracyChange( ) {

}
```

Sensor API Tasks: Examples

- **Sensor API Task 1: Identify sensors on phone and their capabilities at runtime**
- Why?
  - Disable app features that use sensors not present, or
  - Choose best option if multiple sensors of same type (e.g. multiple accelerometers)

- **Sensor API Task 2: Monitor sensor events**
- Why?
  - Acquire raw sensor data
  - Sensor event occurs every time sensor detects change in parameters it is measuring
    - E.g. change in phone's rotational velocity triggers gyroscope sensor event

Sensor Availability

https://developer.android.com/guide/topics/sensors/sensors_motion

- Different sensors are available on different **Android versions**

Sensor	Android 4.0 (API Level 14)	Android 2.3 (API Level 9)	Android 2.2 (API Level 8)	Android 1.5 (API Level 3)
TYPE_ACCELEROMETER	Yes	Yes	Yes	Yes
TYPE_AMBIENT_TEMPERATURE	Yes	n/a	n/a	n/a
TYPE_GRAVITY	Yes	Yes	n/a	n/a
TYPE_GYROSCOPE	Yes	Yes	n/a[1]	n/a[1]
TYPE_LIGHT	Yes	Yes	Yes	Yes
TYPE_LINEAR_ACCELERATION	Yes	Yes	n/a	n/a
TYPE_MAGNETIC_FIELD	Yes	Yes	Yes	Yes
TYPE_ORIENTATION	Yes[2]	Yes[2]	Yes[2]	Yes
TYPE_PRESSURE	Yes	Yes	n/a[1]	n/a[1]
TYPE_PROXIMITY	Yes	Yes	Yes	Yes
TYPE_RELATIVE_HUMIDITY	Yes	n/a	n/a	n/a
TYPE_ROTATION_VECTOR	Yes	Yes	n/a	n/a
TYPE_TEMPERATURE	Yes[2]	Yes	Yes	Yes

Identifying Sensors and Sensor Capabilities

https://developer.android.com/guide/topics/sensors/sensors_overview

- First create instance of **SensorManager** by calling **getSystemService( )**

```
private lateinit var sensorManager: SensorManager
...
sensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager
```

- Then list sensors available on device by calling **getSensorList( )**

```
val deviceSensors: List<Sensor> = sensorManager.getSensorList(Sensor.TYPE_ALL)
```

- Can list only available sensors of particular type. E.g. **TYPE_GYROSCOPE, TYPE_GRAVITY**, etc.

Checking if Phone has at least one of particular Sensor Type

- Device may have multiple sensors of a particular type.
  - E.g. multiple magnetometers
  - If so, one of them must be designated “default sensor” of that type (E.g. Default magnetometer)
- To determine if specific sensor type exists use **getDefaultSensor( )**
- **Example:** To check whether device has at least one magnetometer

```
private lateinit var sensorManager: SensorManager
...
sensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager
if (sensorManager.getDefaultSensor(Sensor.TYPE_MAGNETIC_FIELD) != null) {
    // Success! There's a magnetometer.
} else {
    // Failure! No magnetometer.
}
```

Example: Monitoring Light Sensor Data

- **Goal:** Monitor light sensor data using **onSensorChanged( )**

```
class SensorActivity : Activity(), SensorEventListener {
    private lateinit var sensorManager: SensorManager
    private var mLight: Sensor? = null

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

        sensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager
        mLight = sensorManager.getDefaultSensor(Sensor.TYPE_LIGHT)
    }

    override fun onAccuracyChanged(sensor: Sensor, accuracy: Int) {
        // Do something here if sensor accuracy changes.
    }
```

Create instance of Sensor manager

Get default Light sensor

Called by Android system when accuracy of sensor being monitored changes

Example: Monitoring Light Sensor Data (Contd)

Called by Android system to report new sensor value
Provides SensorEvent object containing new sensor data

```
    override fun onSensorChanged(event: SensorEvent) {
        // The light sensor returns a single value.
        // Many sensors return 3 values, one for each axis.
        val lux = event.values[0]
        // Do something with this sensor value.
    }

    override fun onResume() {
        super.onResume()
        mLight?.also { light ->
            sensorManager.registerListener(this, light, SensorManager.SENSOR_DELAY_NORMAL)
        }
    }

    override fun onPause() {
        super.onPause()
        sensorManager.unregisterListener(this)
    }
}
```

Get new light sensor value

Register sensor when app is in foreground

Unregister sensor if app is no longer in foreground to reduce battery drain

Handling Different Sensor Configurations

- Different phones have different sensors built in
  - **E.g.** Motorola Xoom has pressure sensor, Samsung Nexus S doesn't
- If app uses a specific sensor, how to ensure this sensor exists on target phone?
- Two options
  - **Option 1:** Detect device sensors at runtime, enable/disable app features as appropriate
  - Following code checks if device has at least one pressure sensor

```
private lateinit var sensorManager: SensorManager
...
sensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager

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

Option 2: Use Google Play Filters + AndroidManifest.xml

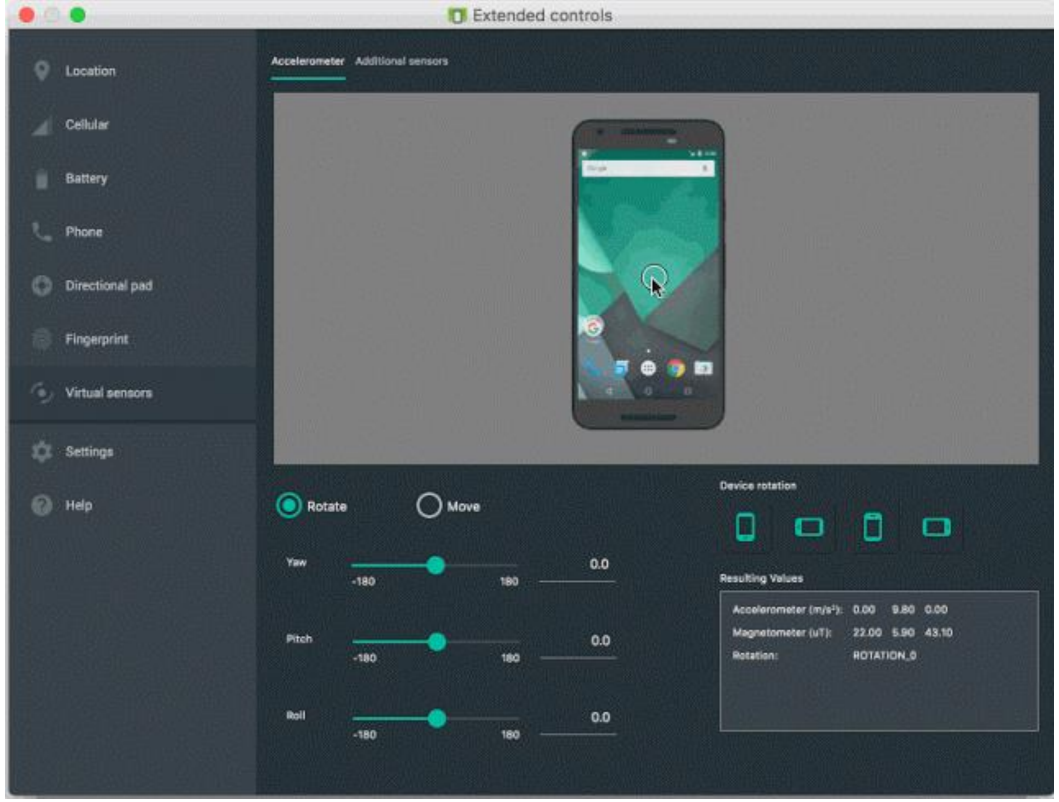
- **Option 2:** Use Google Play filters + AndroidManifest.xml “uses-feature” entries to ensure that only devices that have required sensor can **see** app on Google Play store
  - **E.g.** following manifest entry in AndroidManifest ensures that only devices with accelerometers will **see** this app on Google Play

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

Sensor Programming: Best Practices

https://developer.android.com/guide/topics/sensors/sensors_overview

- Only gather sensor data when app is in foreground, unless crucial (e.g. pedometer)
- Unregister sensor listeners when done using sensor
- Test sensor code on emulator (if sensors used are available there)



Sensor Programming: Best Practices (Contd.)

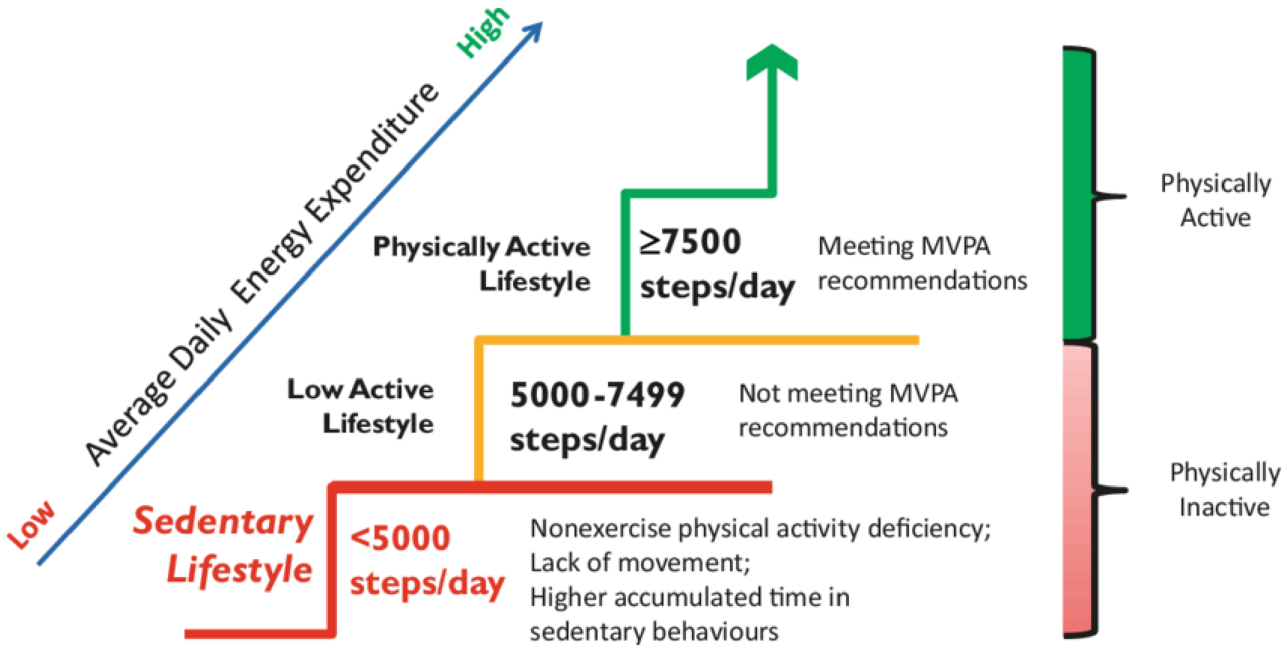
https://developer.android.com/guide/topics/sensors/sensors_overview

- Do as little work in, and don’t block **onSensorChanged( )** method
  - Sensor data can change at high rate, causing onSensorChanged to be called a lot
- Avoid using deprecated methods and sensor types. E.g.
  - TYPE_TEMPERATURE deprecated. Use TYPE_AMBIENT_TEMPERATURE instead
- Verify sensors before using them
- Choose sensor delays carefully
  - Do not choose delivery rate that is too high for your needs

Step Counting
(How Step Counting Works)

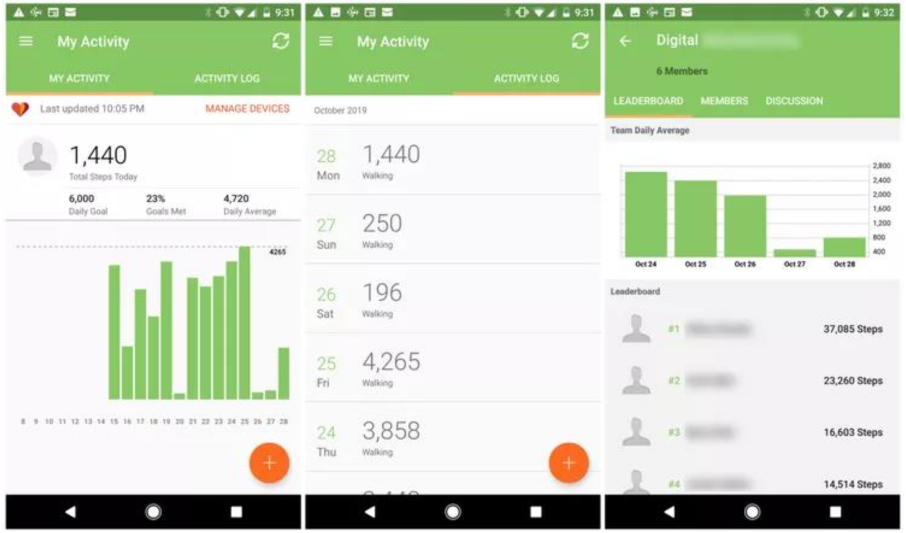
Sedentary Lifestyle

- Sedentary lifestyle
  - increases risk of diabetes, heart disease, dying younger, etc
  - Kills more than smoking!!
- Categorization of sedentary lifestyle based on daily step count by paper:
  - "Catrine Tudor-Locke, Cora L. Craig, John P. Thyfault, and John C. Spence, A step-defined sedentary lifestyle index: < 5000 steps/day", Appl. Physiol. Nutr. Metab. 38: 100–114 (2013)



Step Count Mania

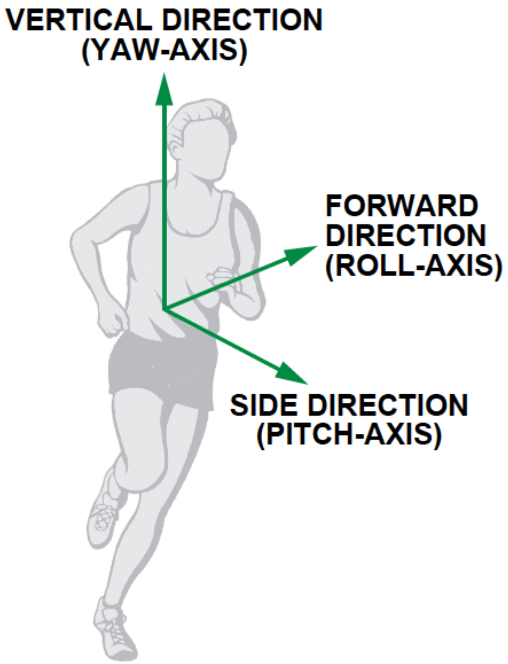
- Everyone is crazy about step count these days
- Pedometer apps, pedometers, fitness trackers, Fitbits, etc
- Tracking makes user aware of activity levels, motivates them to exercise more

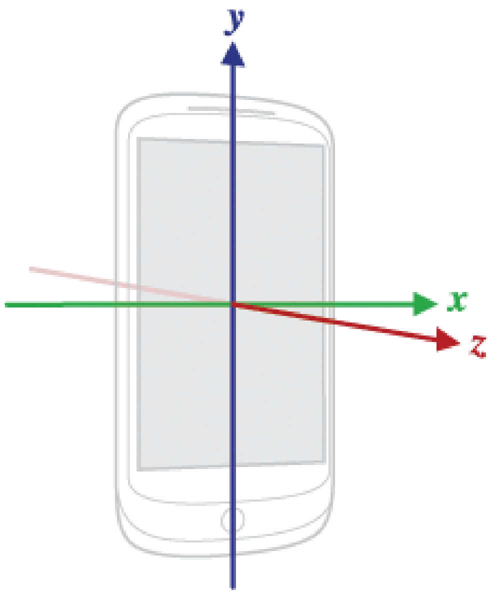


How does a Pedometer Detect/Count Steps

Ref: Deepak Ganesan, Ch 2 Designing a Pedometer and Calorie Counter

- As example of processing Accelerometer data
- Walking or running results in motion along the 3 body axes (forward, vertical, side)
- Smartphone has similar axes
  - Phone orientation: difference between alignment between body and phone axes

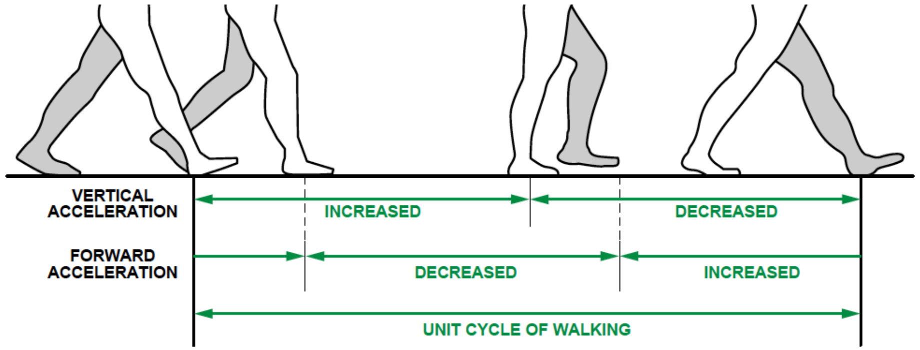


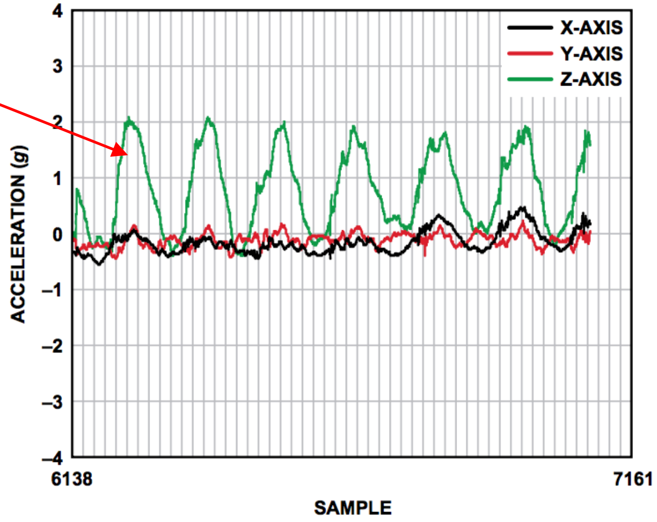


The Nature of Walking

Ref: Deepak Ganesan, Ch 2 Designing a Pedometer and Calorie Counter

- Vertical and forward acceleration increases/decreases during different phases of walking
- Walking causes a large periodic spike on one of the accelerometer axes
- Which axes (x, y or z) and magnitude depends on phone orientation
  - Use axis with largest spike (amplitude)

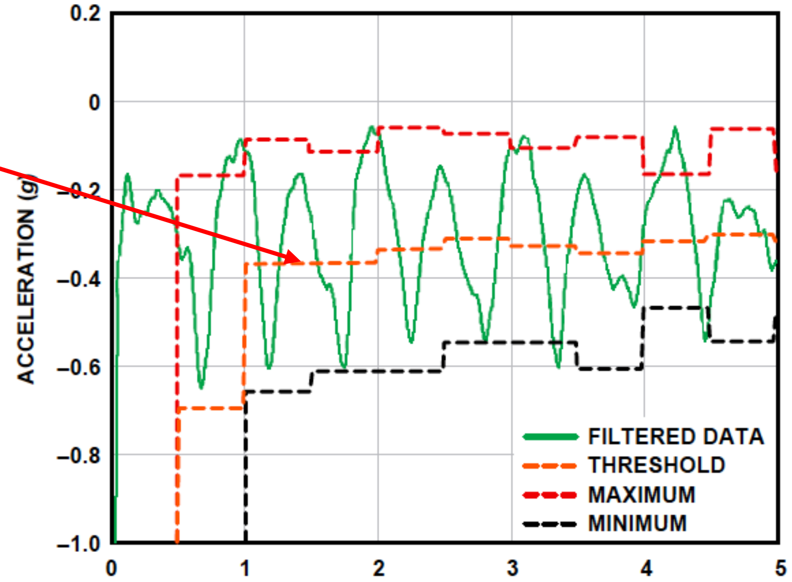




Step Detection Algorithm

Ref: Deepak Ganesan, Ch 2 Designing a Pedometer and Calorie Counter

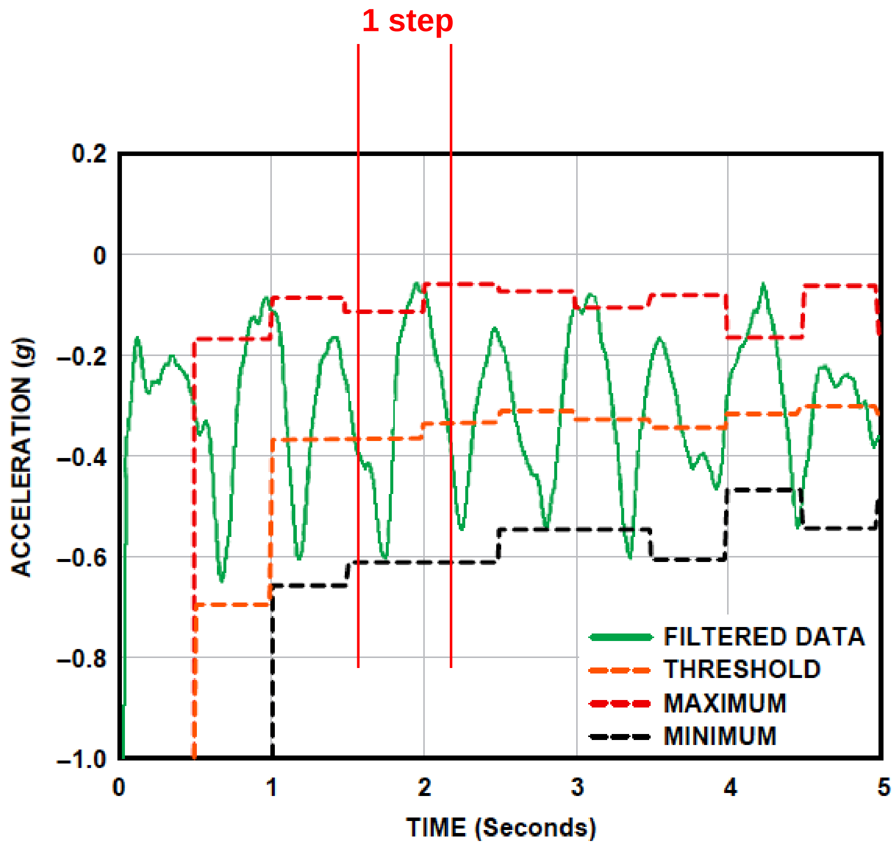
- **Step 1: smoothing**
  - Signal looks choppy
  - Smooth by replacing each sample with average of N
  - E.g. = 3 averages current, prior and next accelerometer sample (Window of 3)
- **Step 2: Dynamic Threshold Detection**
  - Focus on accelerometer axis with largest peak
  - Would like to find threshold such that whenever signal crosses it, is a step
  - But cannot assume fixed threshold (magnitude depends on phone orientation)
  - Track min, max values observed every 50 samples
  - Compute ***dynamic threshold: (Max + Min)/2***



Step Detection Algorithm

Ref: Deepak Ganesan, Ch 2 Designing a Pedometer and Calorie Counter

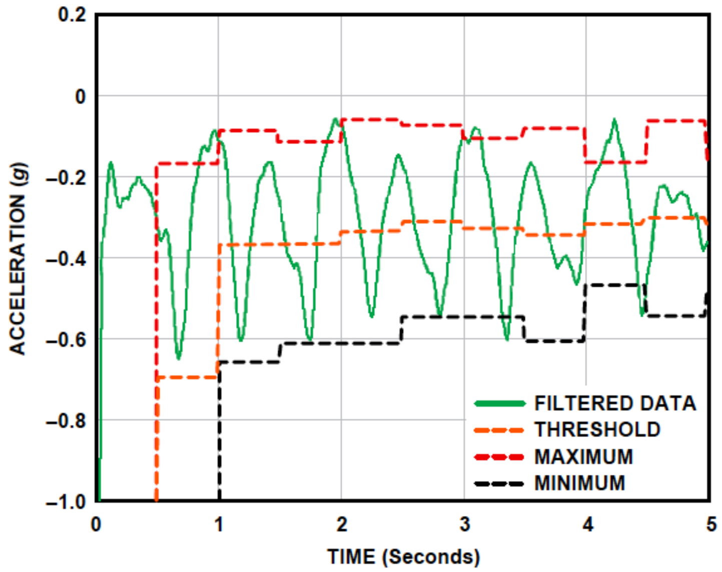
- A step is indicated by crossings of dynamic threshold
- Defined step: interval between consecutive times when negative slope (sample_new < sample_old) of smoothed waveform crosses dynamic threshold.
- Negative slope? Values reducing at point when waveform crosses threshold



Step Detection Algorithms

Ref: Deepak Ganesan, Ch 2 Designing a Pedometer and Calorie Counter

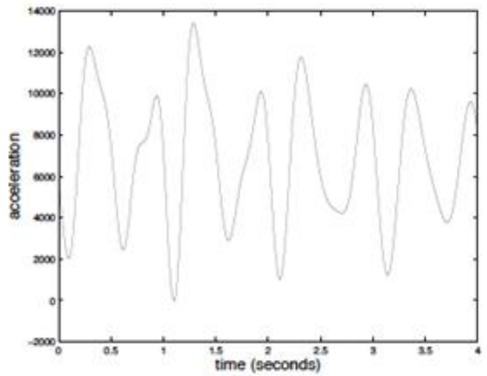
- **Problem:** vibrations (e.g. mowing lawn, plane taking off) could be counted as a step
- **Optimization:** Fix by exploiting periodicity of walking/running
- Assume people can:
  - **Run:** 5 steps per second => 0.2 seconds per step
  - **Walk:** 1 step every 2 seconds => 2 seconds per step
  - So, eliminate "negative crossings" that occur outside period [0.2 – 2 seconds] (e.g. vibrations)



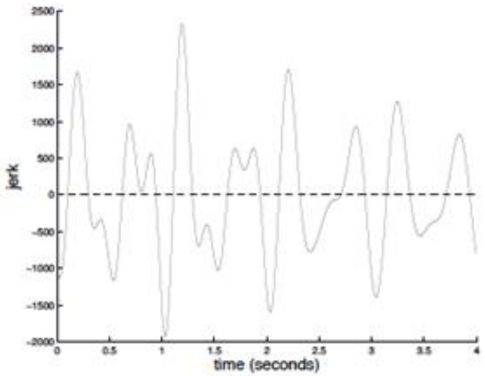
Step Detection Algorithms

Ref: Deepak Ganesan, Ch 2 Designing a Pedometer and Calorie Counter

- Previous step detection algorithm is simple.
- Can use more sophisticated signal processing algorithms for smoothing
- Frequency domain processing (E.g. Fourier transform + low-pass filter)



(c) Output of the low-pass filter.



(d) Derivative of the low-pass filter.

Estimate Distance Traveled

Ref: Deepak Ganesan, Ch 2 Designing a Pedometer and Calorie Counter

- Calculate distance covered based on number of steps taken

Distance = number of steps × distance per step (1)

- Distance per step (stride) depends on user's height (taller people, longer strides)
- Can use number of steps taken per 2 seconds, can estimate stride length based on their height

Steps per 2 s	Stride (m/s)
0~2	Height/5
2~3	Height/4
3~4	Height/3
4~5	Height/2
5~6	Height/1.2
6~8	Height
>=8	1.2 × Height

Estimating Calories Burned

Ref: Deepak Ganesan, Ch 2 Designing a Pedometer and Calorie Counter

- To estimate speed, remember that speed = distance/time. Thus,

Speed (in m/s) = (no. steps per 2 s × stride (in meters))/2s (2)

- Can also convert to calorie expenditure, which depends on many factors E.g
  - Body weight, workout intensity, fitness level, etc
- Rough relationship given in table

Running Speed (km/h)	Calories Expended (C/kg/h)
8	10
12	15
16	20
20	25

x / y = 1.25

- Expressed as an equation

Calories (C/kg/h) = 1.25 × running speed (km/h) (3)

- First convert from speed in m/s to km/h

Calories (C/kg/h) = 1.25 × speed (m/s) × 3600/1000 = 4.5 × speed (m/s) (4)

The Rest of the Class

The Rest of this class

- **Part 1: Course and Android Introduction**
  - Introduce mobile computing, ubiquitous Computing, Android,
  - Basics of Android programming, UI, Android Lifecycle
- **Part 2: Mobile and ubicomp Android programming**
  - mobile Android components (location, Google Places, maps, geofencing)
  - Ubicomp Android components (camera, face detection, activity recognition, sensor programming, etc)
- **Part 3: Mobile Computing/Ubicomp Research**
  - Ubicomp research (smartphone sensing examples, human mood detection, etc) using machine learning
  - Mobile computing research papers (new directions app usage studies, energy consumption, etc)
  - Machine learning (classification) in ubicomp.

Next!!

References

- Google Android Tutorials
- Deepak Ganesan, Step Counting Notes