Administrivia

- Groups should submit 1-slide on their final project (due next class)
- Quiz
  - Covers lectures 5-6
  - All code in those lectures handed out
  - Papers and handouts
- Project 3 posted
  - I’ve covered everything you need to do it EXCEPT Activity Recognition (Next week)
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 user’s activity (e.g. walking, running, etc)
- Voice samples can be processed/classified to infer whether speaker is nervous or not
Android Sensors

- Microphone (sound)
- Camera
- Temperature
- Location (GPS, A-GPS)
- Accelerometer
- Gyroscope (orientation)
- Proximity
- Pressure
- Light

Different phones do not have all sensor types!!
Android Sensor Framework

- Enables apps to:
  - Access sensors available on device and
  - Acquire raw sensor data

Specifically, using the Android Sensor Framework, you can:
- Determine **which sensors** are available on phone
- Determine **capabilities of sensors** (e.g. max. range, manufacturer, power requirements, resolution)
- **Register and unregister** sensor event listeners
- **Acquire raw sensor data** and define data rate
Android sensors can be either hardware or software

- **Hardware sensor:**
  - physical components built into phone,
  - **Example:** temperature

- **Software sensor (or virtual sensor):**
  - Not physical device
  - Derives their data 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: rotation detection (spin, turn, etc)
## Types of Sensors

<table>
<thead>
<tr>
<th>Sensor</th>
<th>HW/SW</th>
<th>Description</th>
<th>Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>TYPE_ACCELEROMETER</td>
<td>HW</td>
<td>Rate of change of velocity</td>
<td>Shake, Tilt</td>
</tr>
<tr>
<td>TYPE_AMBIENT_TEMPERATURE</td>
<td>HW</td>
<td>Room temperature</td>
<td>Monitor Room temp</td>
</tr>
<tr>
<td>TYPE_GRAVITY</td>
<td>SW/HW</td>
<td>Gravity along X,Y,Z axes</td>
<td>Shake, Tilt</td>
</tr>
<tr>
<td>TYPE_GYROSCOPE</td>
<td>HW</td>
<td>Rate of rotation</td>
<td>Spin, Turn</td>
</tr>
<tr>
<td>TYPE_LIGHT</td>
<td>HW</td>
<td>Illumination level</td>
<td>Control Brightness</td>
</tr>
<tr>
<td>TYPE_LINEAR_ACCELERATION</td>
<td>SW/HW</td>
<td>Acceleration along X,Y,Z – g</td>
<td>Accel. Along an axis</td>
</tr>
<tr>
<td>TYPE_MAGNETIC_FIELD</td>
<td>HW</td>
<td>Magnetic field</td>
<td>Create Compass</td>
</tr>
<tr>
<td>TYPE_ORIENTATION</td>
<td>SW</td>
<td>Rotation about X,Y,Z axes</td>
<td>Device position</td>
</tr>
<tr>
<td>TYPE_PRESSURE</td>
<td>HW</td>
<td>Air pressure</td>
<td>Air pressure</td>
</tr>
<tr>
<td>TYPE_PROXIMITY</td>
<td>HW</td>
<td>Any object close to device?</td>
<td>Phone close to face?</td>
</tr>
<tr>
<td>TYPE_RELATIVE_HUMIDITY</td>
<td>HW</td>
<td>% of max possible humidity</td>
<td>Dew point</td>
</tr>
<tr>
<td>TYPE_ROTATION_VECTOR</td>
<td>SW/HW</td>
<td>Device’s rotation vector</td>
<td>Device’s orientation</td>
</tr>
<tr>
<td>TYPE_TEMPERATURE</td>
<td>HW</td>
<td>Phone’s temperature</td>
<td>Monitor temp</td>
</tr>
</tbody>
</table>
2 New Hardware Sensor introduced in Android 4.4

- **TYPE_STEP_DETECTOR**
  - Triggers sensor event each time user takes a step (*single step*)
  - Delivered event has value of $1.0 + \text{timestamp of step}$

- **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 eliminate false positives

- **Common uses**: step counting, pedometer apps
- Requires hardware support, available in Nexus 5
- Alternatively step counting available through Google Play Services (more later)
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 events 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

- A class that can be 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**

- Android system 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
<table>
<thead>
<tr>
<th>Sensor</th>
<th>Sensor event data</th>
<th>Description</th>
<th>Units of measure</th>
</tr>
</thead>
<tbody>
<tr>
<td>TYPE_ACCELEROMETER</td>
<td>SensorEvent.values[0]</td>
<td>Acceleration force along the x axis (including gravity).</td>
<td>m/s²</td>
</tr>
<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>
<tr>
<td>TYPE_GYROSCOPE_UNCALIBRATED</td>
<td>SensorEvent.values[0]</td>
<td>Rate of rotation (without drift compensation) around the x axis.</td>
<td>rad/s</td>
</tr>
<tr>
<td></td>
<td>SensorEvent.values[1]</td>
<td>Rate of rotation (without drift compensation) around the y axis.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>SensorEvent.values[2]</td>
<td>Rate of rotation (without drift compensation) around the z axis.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>SensorEvent.values[3]</td>
<td>Estimated drift around the x axis.</td>
<td></td>
</tr>
</tbody>
</table>
## Sensor Values Depend on Sensor Type

<table>
<thead>
<tr>
<th>Sensor</th>
<th>Sensor event data</th>
<th>Description</th>
<th>Units of measure</th>
</tr>
</thead>
<tbody>
<tr>
<td>TYPE_LINEAR_ACCELERATION</td>
<td>SensorEvent.values[0]</td>
<td>Acceleration force along the x axis (excluding gravity).</td>
<td>m/s²</td>
</tr>
<tr>
<td></td>
<td>SensorEvent.values[1]</td>
<td>Acceleration force along the y axis (excluding gravity).</td>
<td></td>
</tr>
<tr>
<td>TYPE_ROTATION_VECTOR</td>
<td>SensorEvent.values[0]</td>
<td>Rotation vector component along the x axis ((x \times \sin(\theta/2))).</td>
<td>Unitless</td>
</tr>
<tr>
<td></td>
<td>SensorEvent.values[1]</td>
<td>Rotation vector component along the y axis ((y \times \sin(\theta/2))).</td>
<td></td>
</tr>
<tr>
<td></td>
<td>SensorEvent.values[2]</td>
<td>Rotation vector component along the z axis ((z \times \sin(\theta/2))).</td>
<td></td>
</tr>
<tr>
<td></td>
<td>SensorEvent.values[3]</td>
<td>Scalar component of the rotation vector ((\cos(\theta/2))).¹</td>
<td></td>
</tr>
<tr>
<td>TYPE_SIGNIFICANT_MOTION</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>TYPE_STEP_COUNTER</td>
<td>SensorEvent.values[0]</td>
<td>Number of steps taken by the user since the last reboot while the sensor was activated.</td>
<td>Steps</td>
</tr>
<tr>
<td>TYPE_STEP_DETECTOR</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
</tr>
</tbody>
</table>
SensorEventListener

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

- **Sensor API Task 1: Identifying sensors and their capabilities**
  - Why identify sensor and their capabilities at runtime?
    - Disable app features using sensors not present, or
    - If multiple sensors of 1 type, choose implementation with best performance

- **Sensor API Task 2: Monitor sensor events**
  - Why monitor sensor events?
    - To 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

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

<table>
<thead>
<tr>
<th>Sensor</th>
<th>Android 4.0 (API Level 14)</th>
<th>Android 2.3 (API Level 9)</th>
<th>Android 2.2 (API Level 8)</th>
<th>Android 1.5 (API Level 3)</th>
</tr>
</thead>
<tbody>
<tr>
<td>TYPE_ACCELEROMETER</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>TYPE_AMBIENT_TEMPERATURE</td>
<td>Yes</td>
<td>n/a</td>
<td>n/a</td>
<td>n/a</td>
</tr>
<tr>
<td>TYPE_GRAVITY</td>
<td>Yes</td>
<td>Yes</td>
<td>n/a</td>
<td>n/a</td>
</tr>
<tr>
<td>TYPE_GYROSCOPE</td>
<td>Yes</td>
<td>Yes</td>
<td>n/a(^1)</td>
<td>n/a(^1)</td>
</tr>
<tr>
<td>TYPE_LIGHT</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>TYPE_LINEAR_ACCELERATION</td>
<td>Yes</td>
<td>Yes</td>
<td>n/a</td>
<td>n/a</td>
</tr>
<tr>
<td>TYPE_MAGNETIC_FIELD</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>TYPE_ORIENTATION</td>
<td>Yes(^2)</td>
<td>Yes(^2)</td>
<td>Yes(^2)</td>
<td>Yes</td>
</tr>
<tr>
<td>TYPE_PRESSURE</td>
<td>Yes</td>
<td>Yes</td>
<td>n/a(^1)</td>
<td>n/a(^1)</td>
</tr>
<tr>
<td>TYPE_PROXIMITY</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>TYPE_RELATIVE_HUMIDITY</td>
<td>Yes</td>
<td>n/a</td>
<td>n/a</td>
<td>n/a</td>
</tr>
<tr>
<td>TYPE_ROTATION_VECTOR</td>
<td>Yes</td>
<td>Yes</td>
<td>n/a</td>
<td>n/a</td>
</tr>
<tr>
<td>TYPE_TEMPERATURE</td>
<td>Yes(^2)</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>
Identifying Sensors and Sensor Capabilities

- First create instance of `SensorManager` by calling `getSystemService( )` and passing in `SENSOR_SERVICE` argument

```java
private SensorManager mSensorManager;

mSensorManager = (SensorManager) getSystemService(Context.SENSOR_SERVICE);
```

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

```java
List<Sensor> deviceSensors = mSensorManager.getSensorList(Sensor.TYPE_ALL);
```

- To list particular type, use `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 multiple sensors of a given type exist, one of them must be designated “the default sensor” of that type
- To determine if specific sensor type exists use `getDefaultSensor()`
- **Example:** To check whether device has at least one magnetometer

```java
private SensorManager mSensorManager;
...

mSensorManager = (SensorManager) getSystemService(Context.SENSOR_SERVICE);
if (mSensorManager.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()`, display it in a `TextView` defined in `main.xml`
Example: Monitoring Light Sensor Data (Contd)

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

@Override
public final void onSensorChanged(SensorEvent event) {
    // The light sensor returns a single value.
    // Many sensors return 3 values, one for each axis.
    float lux = event.values[0];
    // Do something with this sensor value.
}

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

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

Get new light sensor value
Register sensor when app becomes visible
Unregister sensor if app is no longer visible 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 device?
- Two options
  - **Option 1**: Detect device sensors at runtime, enable/disable app features as appropriate
  - **Option 2**: Use AndroidManifest.xml entries to ensure that only devices possessing required sensor can see app on Google Play
    - E.g. following manifest entry in AndroidManifest ensures that only devices with accelerometers will see this app on Google Play

```xml
<uses-feature android:name="android.hardware.sensor.accelerometer"
              android:required="true" />
```
Option 1: Detecting Sensors at Runtime

- Following code checks if device has at least one pressure sensor

```java
private SensorManager mSensorManager;
...

mSensorManager = (SensorManager) getSystemService(Context.SENSOR_SERVICE);
if (mSensorManager.getDefaultSensor(Sensor.TYPE_PRESSURE) != null){
    // Success! There's a pressure sensor.
}
else {
    // Failure! No pressure sensor.
}
```
Example Step Counter App

- **Goal:** Track user’s steps, display it in TextView
- **Note:** Phone hardware must support step counting

```java
package com.starboardland.pedometer;

import android.app.Activity;
import android.content.Context;
import android.hardware.*;
import android.os.Bundle;
import android.widget.TextView;
import android.widget.Toast;

public class CounterActivity extends Activity implements SensorEventListener {

    private SensorManager sensorManager;
    private TextView count;
    boolean activityRunning;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        count = (TextView) findViewById(R.id.count);

        sensorManager = (SensorManager) getSystemService(Context.SENSOR_SERVICE);
    }
}
```

https://theelfismike.wordpress.com/2013/11/10/android-4-4-kitkat-step-detector-code/
Example Step Counter App (Contd)

```java
@Override
protected void onResume() {
    super.onResume();
    activityRunning = true;
    Sensor countSensor = sensorManager.getDefaultSensor(Sensor.TYPE_STEP_COUNTER);
    if (countSensor != null) {
        sensorManager.registerListener(this, countSensor, SensorManager.SENSOR_DELAY_UI);
    } else {
        Toast.makeText(this, "Count sensor not available!", Toast.LENGTH_LONG).show();
    }
}

@Override
protected void onPause() {
    super.onPause();
    activityRunning = false;
    // if you unregister the last listener, the hardware will stop detecting step events
    // sensorManager.unregisterListener(this);
}
```

https://theelfismike.wordpress.com/2013/11/10/android-4-4-kitkat-step-detector-code/
Example Step Counter App (Contd)

```java
@Override
public void onSensorChanged(SensorEvent event) {
    if (activityRunning) {
        count.setText(String.valueOf(event.values[0]));
    }
}

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

https://theelfismike.wordpress.com/2013/11/10/android-4-4-kitkat-step-detector-code/
Step Counting
(How Step Counting Works)
Sedentary Lifestyle

- Sedentary lifestyle
  - increases risk of diabetes, heart disease, dying earlier, etc
  - Kills more than smoking!!

- Categorization of sedentary lifestyle based on step count by paper:
Step Count Mania

- Everyone is crazy about step count these days
- Pedometer apps, pedometers, fitness trackers, 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
  - Alignment depends on phone orientation
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 in one of the accelerometer axes
- Which axes (x, y or z) and magnitude depends on phone orientation
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 current, prior and next sample (Window of 3)

- **Step 2: Dynamic Threshold Detection**
  - Focus on accelerometer axis with largest peak
  - Would like a threshold such that each crossing 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 as negative slope (sample_new < sample_old) when smoothed waveform crosses dynamic 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)
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 \times distance per step (1)

- Distance per step (stride) depends on user’s height (taller people, longer strides)
- Using person’s height, can estimate their stride, then number of steps taken per 2 seconds

<table>
<thead>
<tr>
<th>Steps per 2 s</th>
<th>Stride (m/s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0~2</td>
<td>Height/5</td>
</tr>
<tr>
<td>2~3</td>
<td>Height/4</td>
</tr>
<tr>
<td>3~4</td>
<td>Height/3</td>
</tr>
<tr>
<td>4~5</td>
<td>Height/2</td>
</tr>
<tr>
<td>5~6</td>
<td>Height/1.2</td>
</tr>
<tr>
<td>6~8</td>
<td>Height</td>
</tr>
<tr>
<td>\geq 8</td>
<td>1.2 \times Height</td>
</tr>
</tbody>
</table>
Estimating Calories Burned
Ref: Deepak Ganesan, Ch 2 Designing a Pedometer and Calorie Counter

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

\[ \text{Speed (in m/s)} = \frac{\text{(no. steps per 2 s} \times \text{stride (in meters))}}{2 \text{s}} \] (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

- Expressed as an equation

\[ \text{Calories (C/kg/h)} = 1.25 \times \text{running speed (km/h)} \] (3)

<table>
<thead>
<tr>
<th>Running Speed (km/h)</th>
<th>Calories Expended (C/kg/h)</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>10</td>
</tr>
<tr>
<td>12</td>
<td>15</td>
</tr>
<tr>
<td>16</td>
<td>20</td>
</tr>
<tr>
<td>20</td>
<td>25</td>
</tr>
</tbody>
</table>

\[ x / y = 1.25 \]

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

\[ \text{Calories (C/kg/h)} = 1.25 \times \text{speed (m/s)} \times 3600/1000 = 4.5 \times \text{speed (m/s)} \] (4)
References

- Busy Coder’s guide to Android version 6.3
- CS 65/165 slides, Dartmouth College, Spring 2014
- CS 371M slides, U of Texas Austin, Spring 2014
References

- John Corpuz, 10 Best Location Aware Apps
- Liane Cassavoy, 21 Awesome GPS and Location-Aware Apps for Android,
- Head First Android
- Android Nerd Ranch, 2nd edition
- Busy Coder’s guide to Android version 6.3
- CS 65/165 slides, Dartmouth College, Spring 2014
- CS 371M slides, U of Texas Austin, Spring 2014