Sensors

Programming the Android Platform
Sensors

- Hardware devices that take measurements of the physical environment
- Some examples
  - 3-axis Accelerometer
  - 3-axis Magnetic field sensor
  - Temperature sensor
  - Proximity sensor
  - Orientation sensor
  - Light sensor
SensorManager

- System service that manages device’s sensors
- Get instance with
  - `getSystemService(Context.SENSOR_SERVICE)`
- Access specific sensor with
  - `SensorManager.getDefaultSensor(int type)`
Some Sensor Type Constants

- TYPE_ACCELEROMETER
- TYPE_GRAVITY
- TYPE_GYROSCOPE
- TYPE_LIGHT
- TYPE_MAGNETIC_FIELD
- TYPE_PRESSURE
- TYPE_PROXIMITY
- TYPE_TEMPERATURE
SensorEvent

- Represents a Sensor event
- Holds Sensor-specific data
  - e.g., sensor's type, the time-stamp, accuracy & measurement data
When device is lying flat, face-up on a table, axes run
- X – right to left
- Y – top to bottom
- Z – up to down

Coordinate system does not depend on device orientation (portrait vs. landscape)
SensorEventListener

- Interface for SensorEvent callbacks
  - void onAccuracyChanged(Sensor sensor, int accuracy)
    - Called when the accuracy of a sensor has changed.
  - void onSensorChanged(SensorEvent event)
    - Called when sensor values have changed.
Registering for SensorEvents

- Use SensorManager to register/unregister for SensorEvents
  - public boolean registerListener (SensorEventListener listener, Sensor sensor, int rate)
    - Registers a SensorEventListener for the given sensor.
  - public void unregisterListener (SensorEventListener listener, Sensor sensor)
    - Unregisters a listener for the sensors with which it is registered.
public class SensorShowValuesActivity extends Activity
  implements SensorEventListener {

  ... 

  public void onCreate(Bundle savedInstanceState) {
    ...
    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() {
    mSensorManager.unregisterListener(this);
    super.onPause();
}
public void onSensorChanged(SensorEvent event) {
    if (event.sensor.getType() == Sensor.TYPE_ACCELEROMETER) {
        long actualTime = System.currentTimeMillis();
        if (actualTime - mLastUpdate > 500) {
            mLastUpdate = actualTime;
            float x = event.values[0], y = event.values[1], z = event.values[2];
            xView.setText(String.valueOf(x));
            yView.setText(String.valueOf(y));
            zView.setText(String.valueOf(z));
        }
    }
}
When device is lying flat, face-up on a table, accelerometer ideally reports the following forces:

- $x \approx 0 \text{ m/s}^2$
- $y \approx 0 \text{ m/s}^2$
- $z \approx 9.81 \text{ m/s}^2$

But these values will vary from natural movements, non-flat surfaces, noise, etc.
Two common data transforms

- Low-pass filters
  - Deemphasize transient force changes
  - Emphasize constant force components
  - e.g., for a bubble level

- High-pass filters
  - Emphasize transient force changes
  - Deemphasize constant force components
  - e.g., for a game controller
mAlpha = 0.9f;
// simple low-pass filter
float lowPass(float current, float filtered) {
  return mAlpha * current + (1.0f - mAlpha) * filtered;
}
// simple high-pass filter
float highPass(float current, float last, float filtered) {
  return mAlpha * (filtered + current - last);
}
x = event.values[0];
y = event.values[1];
z = event.values[2];

mLowPassX = lowPass(x, mLowPassX);
mLowPassY = lowPass(y, mLowPassY);
mLowPassZ = lowPass(z, mLowPassZ);

mHighPassX = highPass(x, mLastX, mHighPassX);
mHighPassY = highPass(y, mLastY, mHighPassY);
mHighPassZ = highPass(z, mLastZ, mHighPassZ);

mLastX = x;
mLastY = y;
mLastZ = z;
Lab Assignment
Source Code Examples

- SensorShowValues