Internet of Things
Android-Part 2
Data Acquisition with Sensors

Aveek Dutta
Assistant Professor
Electrical Engineering and Computer Science
University of Kansas
e-mail: aveekd@ku.edu
http://www.ittc.ku.edu/~aveekd
Lesson Plan

- Sensors in Android Phones, taxonomy and operation
- Errors in sensor signal processing
- Algorithms for filtering data and sensor fusion schemes
- Android Open Accessory
- Android APIs for accessing the sensors (self-study)
MEMS Sensors

- MEMS sensors incorporate some part that physically moves or vibrates
  - Examples - Pressure, Accelerometer, Gyroscope
  - All Android sensors are made using these techniques

- The sensors referenced through the “Sensor” class may be of two types:
  - Raw or Synthetic (or composite or virtual) sensor
    - Raw sensors correspond to one physical component inside the Android device, e.g., Pressure
    - Synthetic sensors provide an abstraction layer between application code and device and combine raw data of multiple raw sensors, e.g., Orientation (bearing and tilt)
Sensor Taxonomy

- Raw Sensors
  - Light, Proximity, Pressure, Accelerometer, Gyroscope, Magnetic Field, Relative Humidity, etc.
- Synthetic sensors
  - Rotation Vector, Linear Acceleration, Gravity, Orientation
- Binary sensors report only one of two values, e.g., Proximity
- Continuous sensors measure any of a range of values within a max and min limit
Key Properties

- **Dynamic range** is the range of values the sensor can measure.
  - Example: Dynamic range of a light sensor may be 1 to 10,000 lux

- **Saturation** occurs when a sensor attempts to sense an input greater than its maximum measurable value.
  - Returns to zero (with error) once stimulus removed

- **Sampling Frequency** is the time between two measurements.
  - Depends on hardware and software control (Sensor.getMinimumDelay())

- **Resolution** is the smallest detectable difference between physical values.
  - It is often limited by noise
  - Example - 8 bit accelerometer with range of $39.24 \text{ m/s}^2 = 0.15328126 \text{ m/s}^2$ (How?)
Sensing the Environment

This section introduces the sensors that can be used to sense properties of the physical environment that a device is currently in.

We will discuss the below sensors in the coming sections:

- Sensor.TYPE_LIGHT
- Sensor.TYPE_PROXIMITY
- Sensor.TYPE_PRESSURE
- Sensor.TYPE_RELATIVE_HUMIDITY
- Sensor.TYPE_AMBIENT_TEMPERATURE
Sensor.TYPE _LIGHT

- The light sensor is often visible on the face of the device
- It is simply a photodiode, generating voltage when light is incident on it
- The light sensor reports its values in lux, and has a typical dynamic range between 1 and 30,000 lux
- The light sensor also has a resolution of 1 lux. A value of 0.25 lux is like the indirect brightness from a full moon
- Screen brightness is controlled by OS instead of the light sensors
- Can also be used as proximity sensors
Sensor.TYPE_PROXIMITY

- Weak IR LED next to a photodetector
- The LED pulses on and off and the photodetector is locked to that frequency
  - Eliminates spurious events and false positives
- Some proximity sensors report the distance to an object in centimeters
  - Typical threshold distance is usually around 2–4 cm
- High-level firmware algorithms using different threshold
  - Swipe rejection
  - Touch rejection (in top area, near the sensors)
  - Holding the phone in hand
Sensor.TYPE_PRESSURE

- This refers to a MEMS barometer, which measures air pressure
  - Its primary use is to determine altitude in absence of GPS, such as indoor locations
- MEMS pressure sensors look like a drum skin over a chamber with known pressure
  - As the outside pressure changes, the drum skin bulges in or out with the differential pressure
  - Altitude is calculated from air pressure using the SensorManager.getAltitude() method

\[
h(p_0, p) = \frac{T_0}{L} \left(1 - \left(\frac{p}{p_0}\right)^{\frac{R_L}{gM}}\right) = 44330 \times \left(1 - \left(\frac{p}{p_0}\right)^{\frac{1}{5.255}}\right)
\]

- Possible application: Pressure difference between two floors of a shopping mall
  - A difference of -1 mbar, correspond to a descent of 8.34 m (inversely, 10m change = 1.2 bar)
Relative humidity is the amount of water vapor in the air compared to the maximum amount of water vapor that the air can hold at a given temperature.

- The relative humidity along with the ambient temperature can be used to calculate the dew point and absolute humidity.
- Dew point is the temperature at which water vapor condenses. Absolute humidity is the mass of water in a given volume of air.
Sensing Orientation and Movement

- **Inertial sensors** describe what the device is doing in its environment as opposed to describing the environment.

- For orientation and movement sensors, Android uses two coordinate systems:
  - Global coordinate system $x_E, y_E, z_E$, and a Device coordinate system $x, y, z$.

- Figure shows the device positioned at the equator of Earth, with some tilt with respect to Earth.

Coordinate Systems

- **Global**
  - $y_E$ points toward magnetic north, which is approximately true north
  - $x_E$ points approximately east, parallel to Earth’s surface but 90 degrees from $y_E$
  - $z_E$ points away from the center of the earth

- **Device**
  - Raw three-axis inertial sensors (accelerometer, magnetometer, and gyroscope) report values corresponding to the device coordinate system
  - The device coordinate system is partially defined by the default orientation
    - The x-axis is horizontal with positive values to the right
    - The y-axis is vertical with positive values upward
    - The z-axis is positive values in front of the screen

- **How to convert the rotation from the device to the global coordinate system?**
Rotation Angles

- Angular quantities around axes are given by either a 3-vector, rotation matrix, or quaternion
  
  - For example, a 3-vector gyroscope reading of (0.1, –0.2, 0.0) indicates that the rotation rate is +0.1 radians per second around the x-axis, –0.2 radians per second around the y-axis, and not rotating around the z-axis.

- The direction of angular three-vectors is determined by the “right-hand rule”

- The components of angular three-vectors are also called azimuth (or heading or yaw), pitch, and roll

Azimuth (Yaw) - Z axis
Pitch - X axis
Roll - Y Axis
Other Sensor Type

- **Accelerometer** - acceleration in three axes
  - measure the acceleration of the device minus the force of gravity along the 3 sensor axes.
  - When the device lies flat on a table, the acceleration value along z is +9.81 m/s\(^2\), which corresponds to the acceleration of the device (0 m/s\(^2\)) minus the force of gravity (-9.81 m/s\(^2\)).

- **Gyroscope** - reports the rate of rotation of the device around the 3 sensor axes

- **Magnetometer** - Components of magnetic field in three dimensions

- **Orientation** is a fusion of these three sensors
  - The orientation is represented by the rotation necessary to align the Global coordinates with the phone's coordinates. That is, applying the rotation to the world frame (X,Y,Z) would align them with the phone coordinates (x,y,z)

- How can an application take advantage of the rotation mapping function?
Sensor.TYPE_Rotation_Vector() and getOrientation()

- **values[0]** = Azimuth (or heading or yaw) = Rotation about z-axis:
  - Assume the device is flat on its back in portrait mode, with the top pointing toward north. The device reports 0 radians in this orientation, $\pi/2$ radians when pointing east, $-\pi/2$ radians when pointing west, and $\pi$ radians when pointing south.

- **values[1]** = Pitch = Rotation about x-axis:
  - Assume the device is flat on its back in portrait mode. The device reports 0 radians in this orientation, $-\pi/2$ radians when you lift the top upward so it is standing upright with the screen facing toward you, $+\pi/2$ radians when you lower the top so it is standing upright with the screen facing away from you, and $\pi$ radians when the device is face down.

- **values[2]** = Roll = Rotation about y-axis:
  - Assume the device is flat on its back in portrait mode. The device reports 0 radians in this orientation, $-\pi/2$ radians when you lift the right side so it is standing upright on its side with the screen facing west, $\pi/2$ radians when you lift the left side so it is standing upright with the screen facing east, and $\pi$ radians when the device is face down.
Errors and Sensor Signal Processing

- Sensors do not measure values perfectly
  - Noise or degradation over time
- Algorithms and techniques exist to address these errors
  - Filtering the raw output
  - Fusion of results from multiple sensors
  - Android’s synthetic sensors execute filtering algorithms
- Let’s look at “Types of error” and “Technique to address Error”
Accuracy and Precision

- Actual value vs Measured value
- **Accuracy** - The measured value is close to the actual value
- Precision - Measurements are more tightly clustered, regardless of accuracy
- Notice the cluster of values in either high accuracy or low accuracy situations that have high precision, while low precision measurements scatter the data points.
Errors and Sensor Signal Processing - Types of Errors

- **Human Error, Systematic Error, and Random Error:**
  - Systematic errors are a constant offset from the true value
  - Noise (white or colored)

- **Drift**
  - Long-term wandering of data away from the real-world value

- **Zero Offset (or "Offset," or "Bias")**
  - Accelerometer and Gyro have offsets.

- **Time Delays and Dropped Data**
  - Some measured data values can be delayed, resulting in incorrect timestamps

- **Integration Error**
  - The gyroscope reports in radians per second, integrate to get the rotation angle in radians (zero offset gets added as well)
Techniques to Address Error

- Re-zeroing by calibration
- Filters
  - Low Pass
  - High Pass
  - Band Pass
  - Others, band stop, notch, Kalman etc.
- Sensor Fusion
  - Take advantage of the strengths of each sensor and mitigate the effects of the weaknesses
  - For example, the accelerometer can give a relatively accurate measurement of the “downward” direction, but not yaw. However, the compass can supplement that measurement to give yaw.
  - Add integrated gyroscope data to give an app access to faster and lower-noise changes but use the accelerometer and compass to reduce the effects of normal gyroscope drift.
Sampling and Reconstruction

Digital processing of analog signals proceeds in three stages:

- The analog is sampled and quantized to finite number of bits - A/D conversion
- The digitized samples are processed by a digital signal processor
- The resulting output samples may be converted back to analog - (D/A conversion)

Sampling Theorem

- The analog signal $x(t)$ is periodically measured every $T$ seconds. Thus, time is discretized in units of the **sampling interval** $T$.

- Every frequency component of the original signal is periodically replicated over the entire frequency axis, with period given by the **sampling rate** $f_s$.

$$f_s = \frac{1}{T}$$
Sampling Theorem

- For accurate reconstruction of $x(t)$ from its time samples $x(nT)$
  - The signal $x(t)$ must be bandlimited, that is, its frequency spectrum must be limited to contain frequencies up to some maximum frequency, say $f_{\text{max}}$, and no frequencies beyond that.
  - The sampling rate $f_s$ must be chosen to be at least twice the maximum frequency $f_{\text{max}}$

\[
 f_s \geq 2f_{\text{max}}
\]

\[
 [-\frac{f_s}{2}, \frac{f_s}{2}] = \text{Nyquist Interval}
\]

\[
 f_s = 8f \quad f_s = 4f \quad f_s = 2f
\]
Fourier Transform Examples
Noise Reduction Filters

One of the most common problems in signal processing is to extract a desired signal, say $s(n)$, from a noisy measured signal:

$$x(n) = s(n) + v(n)$$
**Low Pass Filter**

- The filter is better if
  - Decreasing $A_{\text{pass}}$ so that the passband becomes flatter
  - Increasing $A_{\text{stop}}$ so that the stopband becomes deeper
  - Moving $f_{\text{stop}}$ closer to $f_{\text{pass}}$ so that the transition region becomes narrower
High Pass Filter
Bandpass Filter

The diagram illustrates a desired digital bandpass filter, showing the frequency response $|H(f)|^2$ with the passband and stopbands indicated. The frequency $f$ is plotted on the x-axis with the passband regions $f_p$ and stopband regions $f_s$. The filter characteristics include the passband gain $A_{pass}$ and the stopband attenuation $A_{stop}$, with the transition between passband and stopband regions $f_{pa}$ and $f_{pb}$. The diagram also includes the stopband regions $f_{sa}$ and $f_{sb}$.
Moving Average Filter

- Averaging smoothes the signal losing high frequency components
- MA filter is an example of LPF
Weighted Moving Average (EWMA)

- Define a weight $a$ to include certain fraction of the historical data
- Note, if $a = 1$, no weight on history, only use the current value

\[
\text{value}[n] = \text{value}[n-1] \times (1-a) + x[n] \times a
\]
HPF Example

- HPF are essentially inverse LPF
- \( \text{HPF}(\text{value}[n]) = \text{value}[n] - \text{LPF}(\text{value}[n]) \)
- Filters low frequency components like Drift over time

```java
public void onSensorChanged(SensorEvent event) {
    final float alpha = 0.8;

    gravity[0] = a * gravity[0] + (1 - a) * event.values[0];
    gravity[1] = a * gravity[1] + (1 - a) * event.values[1];

    linear_acceleration[0] = event.values[0] - gravity[0];
    linear_acceleration[1] = event.values[1] - gravity[1];
}
```
Drift Removal - Example

**FIGURE 6-3:** The effect of high-pass filtering on simulated accelerometer data for different values of $\alpha$. Notice that high-frequency shaking in (A) passes through but low-frequency drift and offset in (A) and (B) do not pass. (C) and (D) give two extreme values of $\alpha$ for comparison, plotted on a separate graph from (A) and (B) for clarity.
Noise Removal Example with Notch Filter
Sensor Fusion

- Map the sensor outputs to get desired quantities
- The sensor can indicate which direction is “north” and “down” (and therefore provide pitch, roll, and yaw) and angular velocity
  - However, the accelerometer and compass are inherently noisy and give poor results
- GPS can be used to get the heading, instead of the compass, if the device is moving

![Diagram of sensor fusion](image)
Sensor Fusion

**FIGURE 6-5:** Determining orientation by directly mapping sensor inputs to the desired outputs.
Sensor Fusion

FIGURE 6-6: Quick and dirty use of a low-pass filter to determine orientation
Sensor Fusion

- Fusion algorithms are not available to most developers, developers continue to consider other methods to determine orientation.
- Integrating gyro measurement will accumulate drift in the computed angle
Sensor Fusion - The Balance Filter

- This integrates the gyroscope to get angle, then high-pass filters the result to remove drift, and adds it to the smoothed accelerometer and compass results.
- Newest gyro data point stored in \texttt{gyro}, the newest angle measurement from the accelerometer is stored in \texttt{angle\_acc}, and \texttt{dt} is the time from the last gyro data until now. Then your new angle is:

  \[
  \text{angle} = b \ast (\text{angle} + \text{gyro}\ast\text{dt}) + (1 - b) \ast \text{angle\_acc}
  \]
Advanced Filtering

- The balance filter is useful and simple to implement, but is not the ideal
  - Kalman filter and will provide superior orientation results

**FIGURE 6-9:** Use of Kalman filters to determine device orientation
Android Open Accessory

- **Android Open Accessory (AOA)** is a protocol that allows an Android device to interact with external sensors and actuators via USB
  - Tools: AOA APIs in the Android SDK and Android Development Kit (ADK)
  - Apple’s MFi (Made for iPhone/iPad/iPod Touch) create 30-pin dock accessories

- Without AOA, a USB host (phone) is incapable of sending commands
  - The Android device enters a special accessory to send commands
  - Meanwhile, the external hardware can send sensor information to the Android device
AOA Limitations

- Arduino has a finite sampling frequency
  - Arduino Mega analog inputs may sample up to 10 kHz (10,000 times per second), whereas its digital inputs can read once per instruction cycle, or 16 MHz (6 million times per second).
- Sampling below Nyquist rate cause aliasing errors
- Connect a phone to an external radio (RTL-SDR, SDR Touch)
- Video [link](#)
- Something more exotic (ZynQ PicoSDR or USRP-Mini) - summer REU (contact me)
Overview of Android API
(Self Study)
The Android Sensor API consists of classes for requesting and processing sensor information from a device’s hardware.

The entry point to the API is the SensorManager class, which allows an app to request sensor information and register to receive sensor data.

When registered, sensor data values are sent to a SensorEventListener in the form of a SensorEvent that contains information produced from a given Sensor.

Android Sensor API outlines the classes within the Android Sensor API and illustrates how to use the classes.
SensorManager is the Android system service that gives an app access to hardware sensors.

Like other system services, it allows apps to register and unregister for sensor-related events. Once registered, an app will receive sensor data from the hardware.

In addition to allowing an app to register for sensor data, the SensorManager also provides methods that process sensor data.

SensorManager.getOrientation() is an example of such a method that uses sensor data to generate device orientation information.
ANDROID SENSOR API - Sensor Class

The Sensor class is the Android representation of a hardware sensor on a device. This class exposes information about the sensor, such as:

- Maximum range
- Minimum delay
- Name
- Power
- Resolution
- Type
- Vendor
- Version

SensorManager provides two methods to access Sensor objects: getSensorList() and getDefaultSensor().
The `getSensorList()` method retrieves all the sensors of a given type while `getDefaultSensor()` returns the default sensor for the specified type.

The sensor returned from `getDefaultSensor()` may be either a raw sensor or a synthetic sensor that manipulates raw sensor data.

It is important for an app to examine the output from these methods because devices may or may not support a particular sensor that an app needs. The following code sample is a generally foolproof method for checking for an accelerometer with `getSensorList()`. Checks for other sensors follow a similar pattern.

```java
public static boolean isAccelerometerSupported(Context context) {
    SensorManager sm =
        (SensorManager) context.getSystemService(Context.SENSOR_SERVICE);
    List<Sensor> sensors = sm.getSensorList(Sensor.TYPE_ACCELEROMETER);
    return sensors.size() > 0;
}
```
When you register a listener, you specify the delay or measurement rate for the listener. The predefined rates are:

- `SENSOR_DELAY_FASTEST`
- `SENSOR_DELAY_GAME`
- `SENSOR_DELAY_UI` (Suitable for usual user interface functions, like rotating the screen orientation.)
- `SENSOR_DELAY_NORMAL` (The default value.)

In Android 4.0.3, these are hard-coded to be 0, 20, 67, and 200 milliseconds, respectively. You can also specify your own delay in microseconds by passing a sensor rate value to the registration that is not one of the aforementioned constants.

However, these rates are only intended to be hints to the system, as events may be received faster or slower than the specified delay. Events are usually received faster if the hardware and garbage collection can keep up.
Perhaps the most useful methods of the Sensor class are Sensor.getMaximumRange() and Sensor.getResolution(), both of which take no arguments and return a floating-point number.

getMaximumRange() returns the maximum range the sensor can measure in the regular units reported by the sensor. A measured value of 19.6133 m/s² (equivalent to 2 g, where g is a unit of acceleration) — as in STMicroelectronics’ KR3DM 3-axis accelerometer, for instance — means the sensor can measure accelerations from +2g to –2g.

getResolution() reports the resolution of the sensor, in the regular units reported by the sensor. However, the resolution here is a digital resolution figure that is independent of the sensor noise. Android sensors output digital signals, for example, 8-bit (256 possible values), 10-bit (1024 possible values), and 12-bit (4096 possible values) accelerometers are common. The maximum range divided by the number of possible values gives the resolution reported here.
The SensorEventListener is an interface that provides the callbacks to alert an app to sensor related events.

To be made aware of these events, an app registers a concrete class that implements SensorEventListener with the SensorManager.
Resources

