Android Sensors Part 1

Today

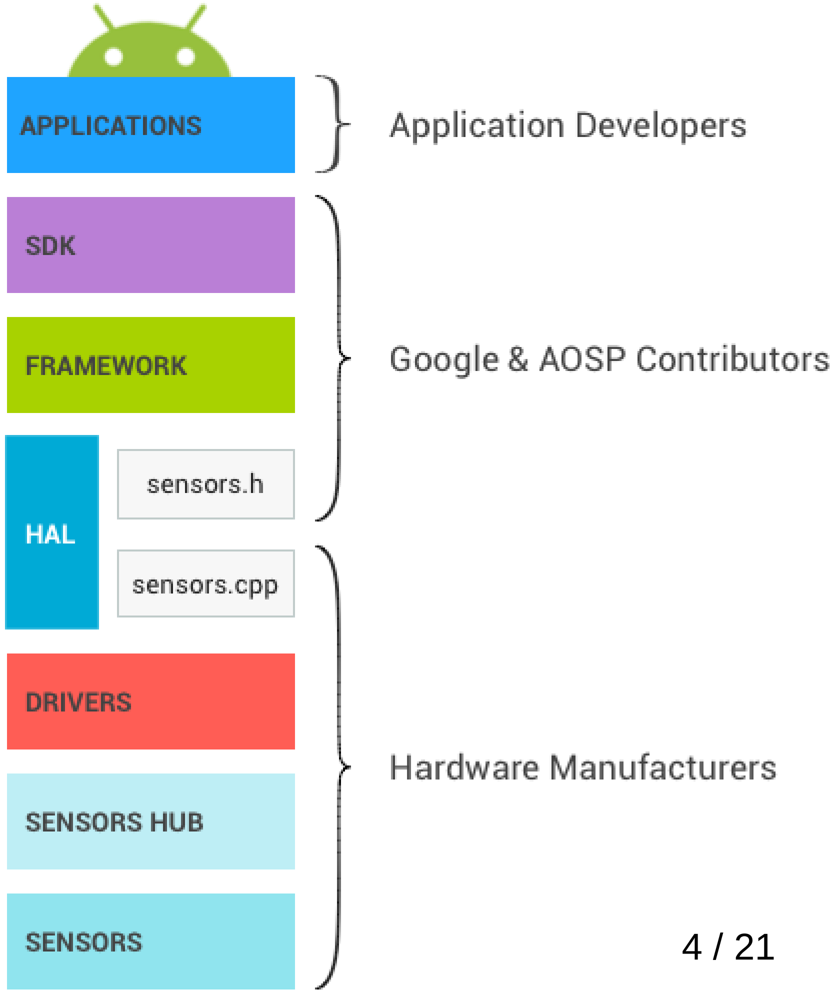
- We’ll begin our discussion on sensor programming for Android.
- We’ll see the different categories of sensors supported by the Android platform.
- We’ll discuss the sensor framework in detail.
- We’ll see two simple examples.

Sensors: Overview

- Sensors measure a physical quantity, such as force, light, temperature, humidity, etc.
- Most sensors are Microelectromechanics systems (MEMS).
  - MEMs range from 20-1000 micrometers in size.
- Sensors can be viewed as *transducers*: you convert one kind of energy to another.
- In general, all sensors convert some physical quantity to analog electrical signals, which are in turn converted to digital values.
- Sensors barely use any battery or CPU time.

Sensors: Android Sensor Stack

- Android contains a full sensor stack.
- Note that sensors are little devices of their own.
- They have their own embedded languages.
- HAL: Hardware Abstraction Layer.
- HAL maps between sensor-specific software and the Android framework.



Sensors: Categories

- There are two main types of sensors.
- Physical sensors (aka hardware sensors), mapped to actual MEMs on the phone. Eg. Accelerometer, gyroscope, magnetometer.
- Synthetic sensors (aka virtual sensors) are derived by combining one or more physical sensors. Eg. Gravity sensor, step detector.
- Android doesn’t distinguish between physical and synthetic sensors!

Sensors: Categories

- Android supports three categories of sensors (regardless of whether they are physical or synthetic).
- Motion sensors: measure forces that could create motion on the phone’s axes.
  - This motion can be linear (translation) or angular (rotation).
- Position sensors: measure physical position of the phone in the world frame. Eg. Combine the geomagnetic sensor with the accelerometer to obtain a compass.
- Both of these sensors give vector-valued data.
- Environmental sensors: measure environmental properties. This is scalar-valued data.

Digression: Scalars and vectors

- A scalar is a number indicating magnitude of some quantity. Eg. Concentration, density, energy.
- A vector is a collection of scalars.
- Typically, the scalar parts (components) of a vector indicate magnitudes along different axes.
- Imagine you’re moving in 2D or 3D space. A vector can tell you how far you’ve moved in the horizontal, vertical, and the third direction.
- You can also figure out the total amount moved in all directions by combining the scalars.
- Force, acceleration, velocity, momentum, position, are all vectors.

Sensors: Sensor values

- Sensor values come in three types:
  - Raw: this is sensor data that is directly collected from the sensor. Eg. Accelerometer, light sensors, proximity sensors.
  - Calibrated: the Android OS postprocesses raw sensor data to remove noise, drift, bias etc, and reports the postprocessed values. Eg. Step detector.
  - Fused: combined data from one or more sensors. Eg. Gravity sensor data is a combination of accelerometer and gyroscope data.

Sensors: Coordinate System

- Sensors use a 3-axis coordinate system.
  - The origin of this system is the center of the screen.
  - The default orientation is typically portrait mode.
  - The x-axis is the horizontal along the screen.
  - The y-axis is the vertical along the screen.
  - The z-axis is the direction into and out of the screen.
  - If the default orientation is landscape, then x and y are swapped.
- In general, better to pin your apps to default orientation.

Sensors: Coordinate System

- Certain sensors actually don’t center their coordinate system on the phone.
- Rather, they use a coordinate system relative to the earth.
- You’ll need to learn to convert between the two types freely.
- These two coordinate systems (object-frame and world-frame) are extremely common in engineering, mathematics, and computer graphics.

Sensors: The Sensor Framework

- Android gives you a full sensor framework to access and work with sensors and sensor data.
- This framework is a part of the android.hardware package.
- There are four components:
  - SensorManager.
  - Sensor.
  - SensorEvent.
  - SensorEventListener.

Sensors: SensorManager

- This class allows you to create an instance of the sensor service (a system service).
- Lets you access and list sensors.
- Lets you register and unregister sensor event listeners.
- Provides methods to obtain orientation information.
- Provides sensor constants for various purposes.

Sensors: Sensor (class)

- This class is used to create instances of specific sensors.
- Can query the sensor for its capabilities.
  - Maximum range.
  - Minimum delay.
  - Name.
  - Power.
  - Reporting mode.
  - Resolution.

Sensors: SensorEvent and SensorEventListener

- SensorEventListener is an interface.
- Allows you to create callback methods for notifications
  - When sensor values change: onSensorChanged().
  - When sensor accuracy changes: onAccuracyChanged().
- Sensor data is sent to registered SensorEventListeners in the form of the SensorEvent class.
- SensorEvent packages the data from a given sensor:
  - values[]: a multidimensional array holding sensor data.
  - timestamp: time in nanoseconds at which event happened.
  - accuracy: an integer constant representing accuracy.
  - sensor: the sensor type that generated this data.

Sensors: Sensor Modes and Callbacks

- Callbacks are onSensorChanged() and onAccuracyChanged().
- Sensors can generate events in different reporting modes:
  - Continuous: events are generated at a constant rate.
  - On-change: events are generated only if measured values have changed.
  - One-shot: the sensor deactivates itself upon detection of an event, then generates a single event.
- Callbacks depend on reporting modes.

Sensors: Sensor Modes and Callbacks

- onSensorChanged() is called at regular frequency if the sensor is in continuous mode. Eg. Accelerometer.
- onSensorChanged() is called whenever there’s a change in the sensor data if the sensor is in on-change mode.
- Beware: the OS may choose to report sensor values at a different frequency from what you specify due to job scheduling.
- onAccuracyChanged() is called much more rarely (whenever sensor accuracy changes).
- Typically tied to battery-saver mode or heavy CPU processing.

Sensors: Workflow

- Here’s the typical sensor workflow:
  - Instantiate SensorManager.
  - Get the required Sensor object from SensorManager.
  - Implement the SensorEventListener, and override the callbacks.
  - Register the SensorEventListener with SensorManager.
  - Handle sensor events in onSensorChanged().
  - Unregister the SensorEventListener when you don’t need sensor data.
- Always remember to register and unregister in pairs.

Sensors: Example 36 (List of all sensors)

- We’ll get a list of all available sensors.
- We’ll display them in a RecyclerView.
- We list the sensor type, power, resolution. There are many other sensor properties too.
- In your app, you may need to get a list of valid sensors before proceeding with some operation.

Sensors: Example 37 (Display sensor values)

- We’ll display the actual sensor values for a sensor.
- We’ll register the listener in onResume(), and unregister in onPause().
- This way, the app isn’t trying to listen to stuff all the time.
- Consider how this would work with the MVVM architecture:
  - Is the sensor data part of the Model? Then it should be acquired and listened to in the repository.
  - Is it part of the ViewModel? Maybe if it’s lightweight?

Sensors: Foreground vs Background

- In general, one acquires sensor data on the UI thread.
- This protects against draining battery by polling the sensor on a background thread.
- If you absolutely need to acquire sensor data in the background thread, use WorkManager, ensure that battery won’t be drained.
- We’ll stop here.

Summary

- We saw what Android sensors are.
- We saw how to use the Sensor framework to read sensor data.
- We discussed the different categories of sensors.