Designing An Android Sensor Subsystem

Pitfalls and Considerations

Jen Costillo
jen@rebelbot.com
Simple Choices

User experience:
- Latency
- Gesture Recognition
- Sensor Sampling

Battery performance:
- Wake Up event processing
- Gesture Processing
- Calibration Strategy
- Co-processor architecture
Established or Innovative Product?

Established
• Will I be making another new product in 6 months?
• Is the reference design considered good enough for the application?

Innovation-Driven
• Do I have new sensors types?
• Are features more important than release date?
• Are money and resources no problem?
Forsaking Reference Designs
Going On Your Own

• If you make your own,
  – You’re on your own
  – Integration pains
  – Test time ↑
  – Gesture testing becomes a challenge
  – Calibration blues
  – Larger mechanical footprint

• But...
  – power ↓
  – Control code size
  – Control mechanical footprint
  – In-house expertise
Android Universe

Android Application

SensorManager

Sensor JNI

Sensor Service

Sensor Manager

Sensor HAL

Interface Kernel Driver

Sensor Driver

Sensor Manager

Sensors

Sensor Driver

Sensors

Sensor Hub/ Coprocessor

Sensors

Sensor HAL

Hardware

Linux Kernel

Frameworks

Libraries

Application
Hardware Architecture
Sampling Rates: ★ The 3 Rates

**Under-sampling**
- Inaccurate, sluggish response
- Slight power savings

**Over-sampling**
- Accurate, smooth response
- Power-hungry
## Wake up events and power considerations

<table>
<thead>
<tr>
<th>Application Processor only</th>
<th>Internal Coprocessor</th>
<th>External Processor</th>
</tr>
</thead>
<tbody>
<tr>
<td>![Thumb Down]</td>
<td>![Thumb Up]</td>
<td>![Unicorns]</td>
</tr>
<tr>
<td>Reference supported</td>
<td>Reference supported</td>
<td>More processor selection</td>
</tr>
<tr>
<td>Most power hungry</td>
<td>Most work done for you</td>
<td>More outcome control</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Most customized</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Footprint impact</td>
</tr>
</tbody>
</table>
Hardware Summary

Power Consumption = $\sum \text{sensors}_n + \text{any dedicated processor}$

Latency = $\text{Max}(\text{sensors}_n) + \text{dedicated processing time}$

Sensor Solution

• Use tie-breaker criteria
Kernel Driver

Application Processor

Microcontroller

Peripheral Interface

Sensor

Shared Memory

Coprocessor
LIBRARIES AND SERVICES

- Application
- Frameworks
- Libraries
- Linux Kernel
- Hardware
Sensor HAL and Services

• **HAL**
  device/<vendor>/<board name>/libsensors

• **Service**
  frameworks/base/services/sensorservice

• **Manager**
  frameworks/base/libs/gui
Sensor fusion is the combining of sensory data or data derived from sensory data from disparate sources such that the resulting information is in some sense better than would be possible when these sources were used individually. The term

Gesture Detection Algorithm

Application
Processor

Sensor Hub

Sensors

Android SensorService

Co-Processor

MPU with Gyro/Accel

Barometer

Proximity

Compass
Gesture Detection Comparison

Make

Off-load AppPro
In-house expertise
Compact code

Buy

Minimal Schedule Impact
Already Tested &tuned
Complete solution

Application
• Powerful processor

Sensor hub
• Off-load to cheaper power
• Wake up Event Handling
Calibration

Use of Calibration Gesture in the Compass App by Catch.com
FRAMEWORKS

Application

Frameworks

Libraries

Linux Kernel

Hardware
Virtual Sensors

• Leverages 1+ physical or other virtual sensors

• Multiple Options
  – Google (version 3)
  – Reference Vendor
  – Sensor Vendors
Android Virtual Sensors

- Gyro + Accel = Orientation
- Compass + Accel = Rotation
Virtual Sensors Challenges

• Garbage In- Garbage Out
• Latency
• Non-Synchronized samples
• Implementation Dependencies
• Multi-vendor problems, verify Vendor ID

http://www.invensense.com/midc/presentations/James%20Lim.pdf
APPLICATIONS

Application

Frameworks

Libraries

Linux Kernel

Hardware
Using Sensors

mSensorManager = 
    getSystemService(Context.SENSOR_SERVICE);

mSensorManager.registerListener(
    mSensorListener,
    mSensorManager.getDefaultSensor(
        Sensor.TYPE_ACCELEROMETER),
    SensorManager.SENSOR_DELAY_NORMAL);

void onSensorChanged(SensorEvent event) {
    // get sensor data
    float x =
        event.values[SensorManager.DATA_X];
}

Using Sensor (Continued)

```java
SensorManager.getRotationMatrix(
    m_rotationMatrix,
    null,
    m_Mag,
    m_Accels);
SensorManager.getOrientation(
    m_rotationMatrix,
    m_orientation);

float yaw_deg = m_orientation[0] * 57.2957795f;
float pitch_deg = m_orientation[1] * 57.2957795f;
float roll_deg = m_orientation[2] * 57.2957795f;
```
Types of Sensor Problems

• Bias
• Drift
• Settling Time
• Jitter/Noise
• Environmental Interference
Bias

• Problem: Data is off by a constant value.

• Sources:
  – static calibration failure

• Solutions:
  – Calculate linear offset at start of application
  – Recalibrate locally
Drift

• Problem: Shift of data without cause

• Sources:
  – Magnetic interference
  – Poor HW calibration

• Solutions:
  – Increase smoothing techniques
Settling Time

• Problem: Extended time before finalized steady data.

• Sources:
  – Latency
  – Sensitivity

• Solutions:
  – Limit additional processing
Noise

• Problem: Data jumps around constantly

• Sources:
  – Sensor
  – Calibration
  – Poor filtering

• Solutions:
  – High pass filter
  – Linear averaging
  – FFT
Environmental Interference

• Problem: Inconsistent results

• Sources:
  – Magnetometer
  – EMI

• Solutions:
  – Reference Device
  – Calibration gesture
Best Practices in Application Development

• Select the right sensor for the job.
• Use the Correct Data Rate.
  – UI or GAMING are the most common.
• Use Sensor In Context
• Customize for your hardware and system capabilities
• Magnetometer-based sensors are the most touchy.
• Keep the Gesture UI simple.
QUESTIONS?
JEN@REBELBOT.COM

Additional resources
http://www.kandroid.org/online-pdk/guide/sensors.html
http://invensense.com/midc/