E0-245: ASP

Lecture 16+17: Physical Sensors

Dipanjan Gope

Module 2: Android Sensor Applications

- Location Sensors
  - Theory of location sensing
  - Package android.location
- Physical Sensors
  - Sensor Manager
  - Accelerometer
  - Gyroscope
  - Magnetometer
  - Sensor fusion
- Multimedia
  - Camera
  - Microphone
- NFC

Coverage

- ☑ Activity
- ☑ Views
- ☐ Intent
- ☐ ContentProvider
- ☐ BroadcastReceiver
- ☐ Service

References

- Greg Milette, Adam Stroud: Professional Android Sensor Programing, 2012, Wiley India

Popular Apps with Physical Sensors

MAGNETOMETER APPS

GYROSCOPE GAMING


ACCELEROMETER FITNESS

Classification based on source of data

Raw Sensor Data

Sensor	Type	Description	Common Uses
TYPE_ACCELEROMETER	Hardware	Measures the acceleration force in m/s^2 that is applied to a device on all three physical axes (x, y, and z), including the force of gravity.	Motion detection (shake, tilt, etc.).
TYPE_GYROSCOPE	Hardware	Measures a device's rate of rotation in rad/s around each of the three physical axes (x, y, and z).	Rotation detection (spin, turn, etc.).
TYPE_MAGNETIC_FIELD	Hardware	Measures the ambient geomagnetic field for all three physical axes (x, y, z) in μT.	Creating a compass.
TYPE_PRESSURE	Hardware	Measures the ambient air pressure in hPa or mbar.	Monitoring air pressure changes.
TYPE_PROXIMITY	Hardware	Measures the proximity of an object in cm relative to the view screen of a device. This sensor is typically used to determine whether a handset is being held up to a person's ear.	Phone position during a call.
TYPE_RELATIVE_HUMIDITY	Hardware	Measures the relative ambient humidity in percent (%).	Monitoring dewpoint, absolute, and relative humidity.

Raw Sensor Data

Sensor	Type	Description	Common Uses
TYPE_LIGHT	Hardware	Measures the ambient light level (illumination) in lx.	Controlling screen brightness.
TYPE_AMBIENT_TEMPERATURE	Hardware	Measures the ambient room temperature in degrees Celsius (°C). See note below.	Monitoring air temperatures.
TYPE_TEMPERATURE	Hardware	Measures the temperature of the device in degrees Celsius (°C). This sensor implementation varies across devices and this sensor was replaced with the TYPE_AMBIENT_TEMPERATURE sensor in API Level 14	Monitoring temperatures.

http://developer.android.com/guide/topics/sensors/sensors_overview.html

Synthetic Sensor Data

Sensor	Type	Description	Common Uses
`TYPE_GRAVITY`	Software or Hardware	Measures the force of gravity in m/s^2 that is applied to a device on all three physical axes (x, y, z).	Motion detection (shake, tilt, etc.).
`TYPE_LINEAR_ACCELERATION`	Software or Hardware	Measures the acceleration force in m/s^2 that is applied to a device on all three physical axes (x, y, and z), excluding the force of gravity.	Monitoring acceleration along a single axis.
`TYPE_ROTATION_VECTOR`	Software or Hardware	Measures the orientation of a device by providing the three elements of the device's rotation vector.	Motion detection and rotation detection.
`TYPE_ORIENTATION`	Software	Measures degrees of rotation that a device makes around all three physical axes (x, y, z). As of API level 3 you can obtain the inclination matrix and rotation matrix for a device by using the gravity sensor and the geomagnetic field sensor in conjunction with the `getRotationMatrix()` method.	Determining device position.

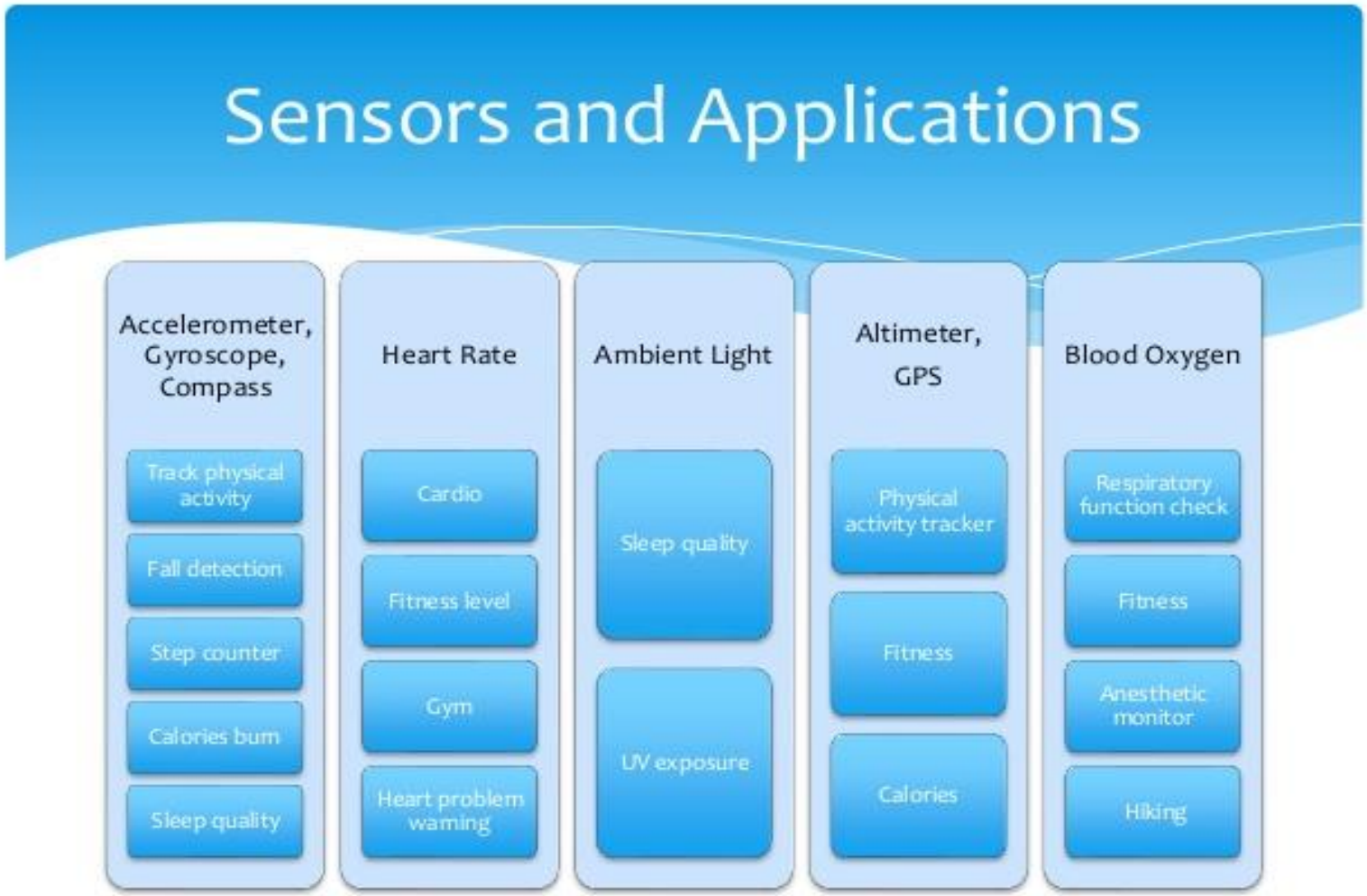
http://developer.android.com/guide/topics/sensors/sensors_overview.html

Classification based on application

Physical Sensors

- Motion
  - Accelerometer
  - Gyroscope
  - Linear acceleration
  - Gravity
- Position
  - Magnetic field
  - Proximity
  - Rotation vector
- Environment
  - Light
  - Barometer
  - Ambient temperature
  - Relative humidity

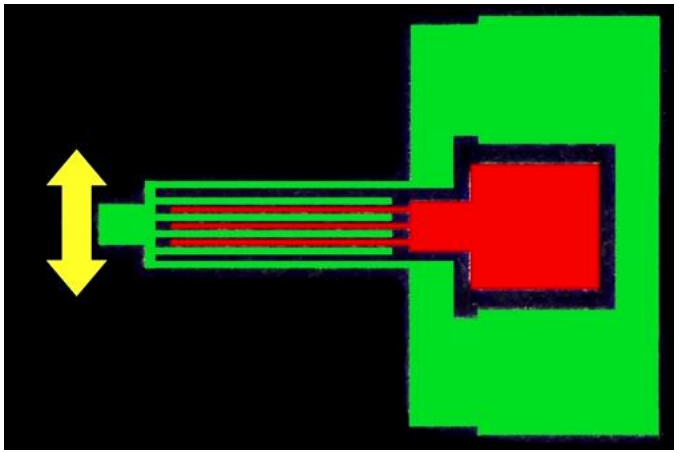
Smart Watches Today



https://www.google.co.in/url?sa=i&rct=j&q=&esrc=s&source=images&cd=&cad=rja&uact=8&ved=0CAcQjRw&url=http%3A%2F%2Fwww.slideshare.net%2FLeybzon%2Fapple-iwatch-android-wear-and-other-wristbased-sensor-platforms&ei=J-EQVe6sDNWdugS84oGwBg&psig=AFQjCNFwN3VY9DQjg7lYqPiWdZqxh0qR8A&ust=1427255772724923

Sensor Hardware

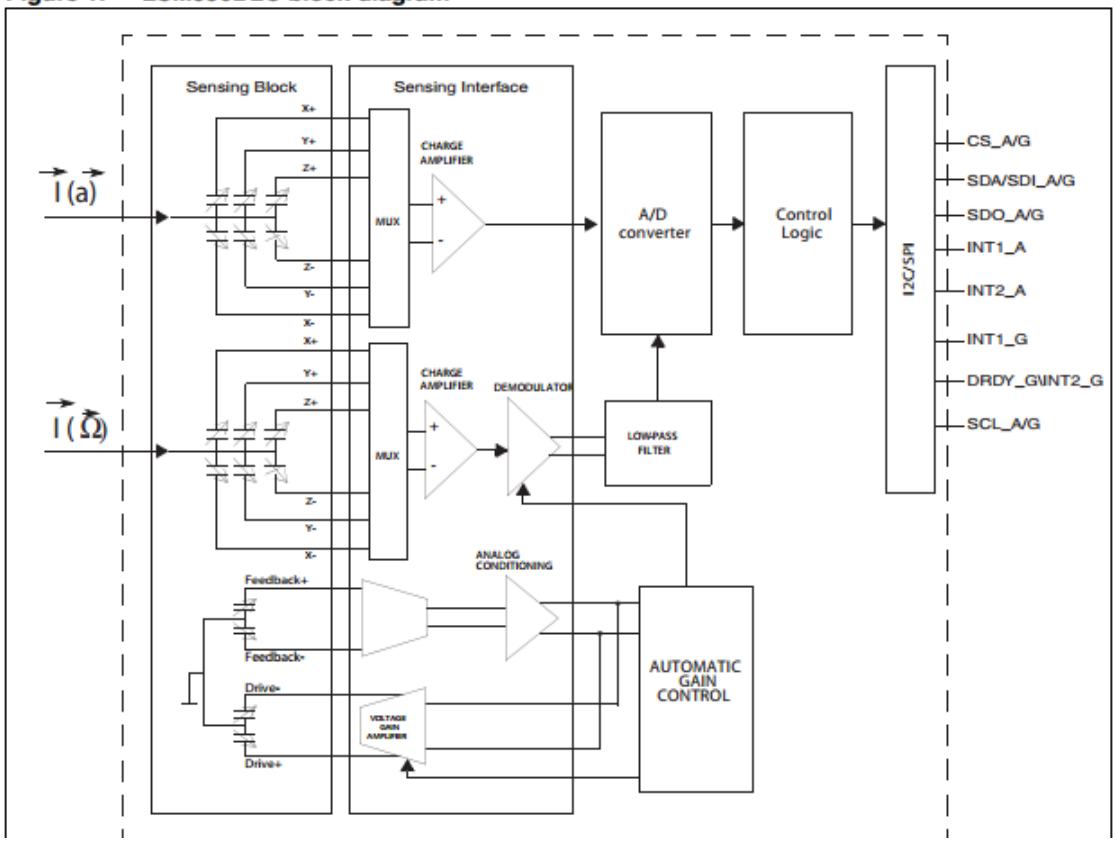
Accelerometer



F=mA

MEMS Capacitive-based SIP

Figure 1. LSM330DLC block diagram



https://www.youtube.com/watch?v=i2U49usFo10

http://www.st.com/st-web-ui/static/active/en/resource/technical/document/datasheet/DM00037200.pdf

Gyroscope

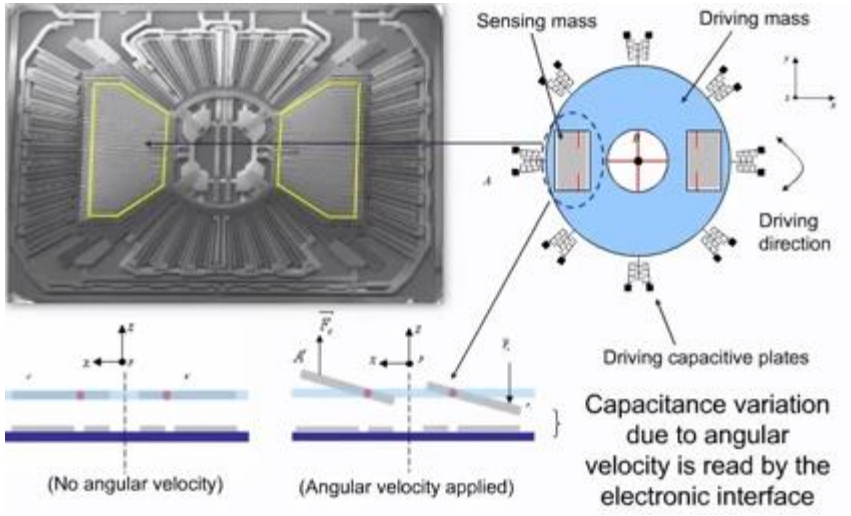
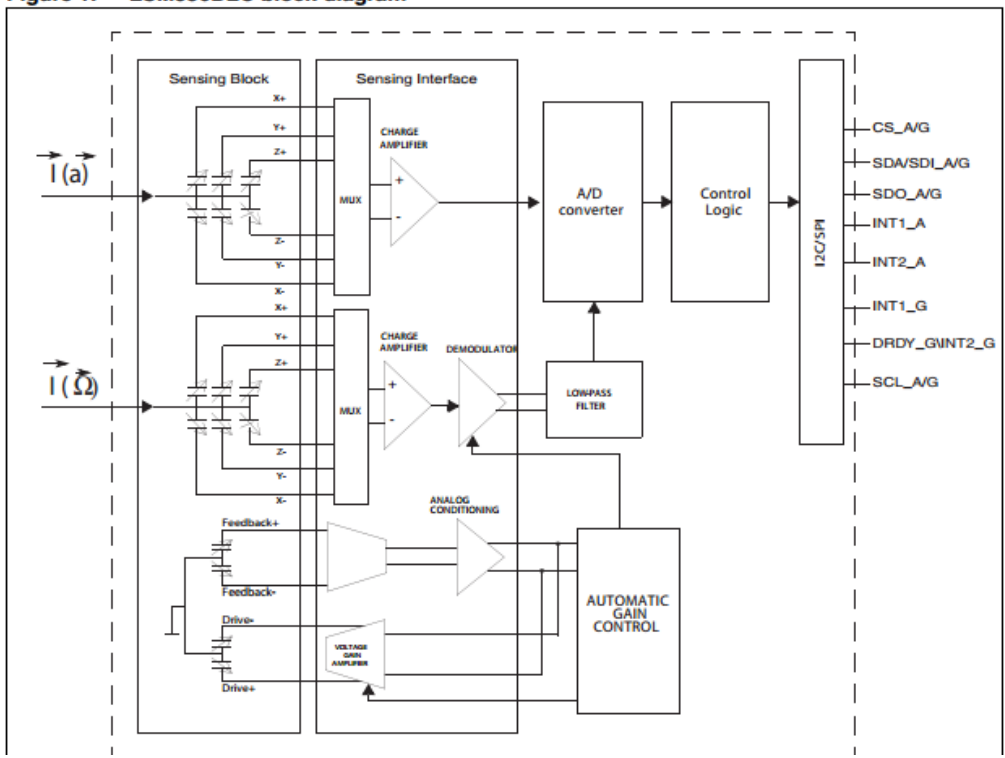
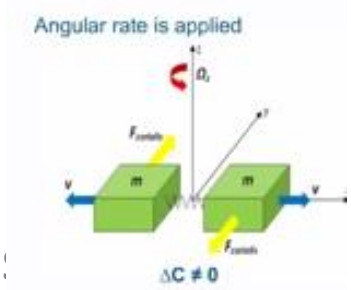


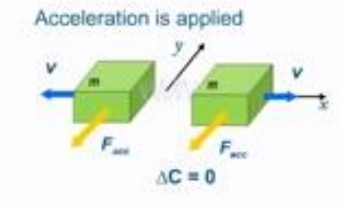
Figure 1. LSM330DLC block diagram



F=-2m Ω x v

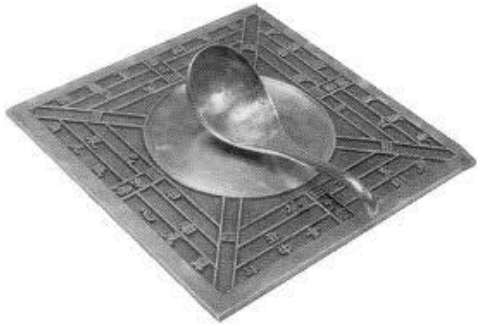
MEMS Capacitive based



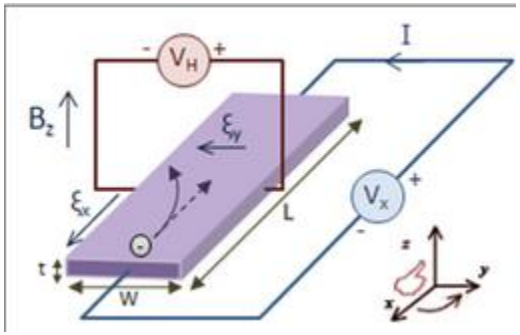


Ref: MEMS gyroscopes - A revolutionary way to interface with the real world, STMicroelectronics

Magnetic Field



Anonymous, 400BC



Lorentz Force

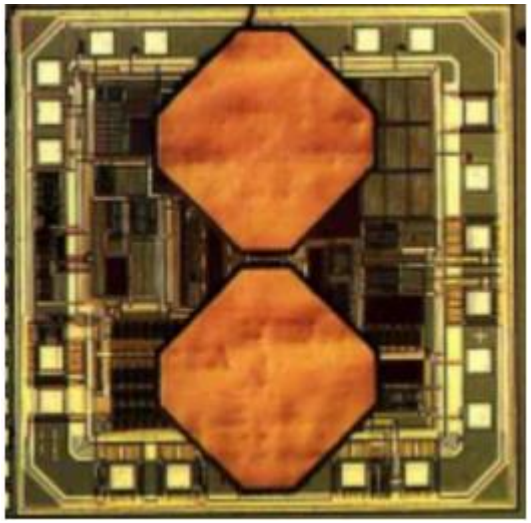
$$\mathbf{F} = q\left(\mathbf{E} + \mathbf{v} \times \mathbf{B}\right) = 0 \text{ where}$$

$E = V_H/w$, $v = L/T$, $I = Q/T$, $Q = n\ Lwt\ e$

Therefore,

$$V_H = -\frac{IB}{nte}$$

http://en.wikipedia.org/wiki/Hall_effect



Hall ASICs with Integrated Magnetic Concentrators

Asahi Kasei: AK8975C

Others: Anisotropic Magneto Resistance

Android Sensor Data Structures

Android.hardware

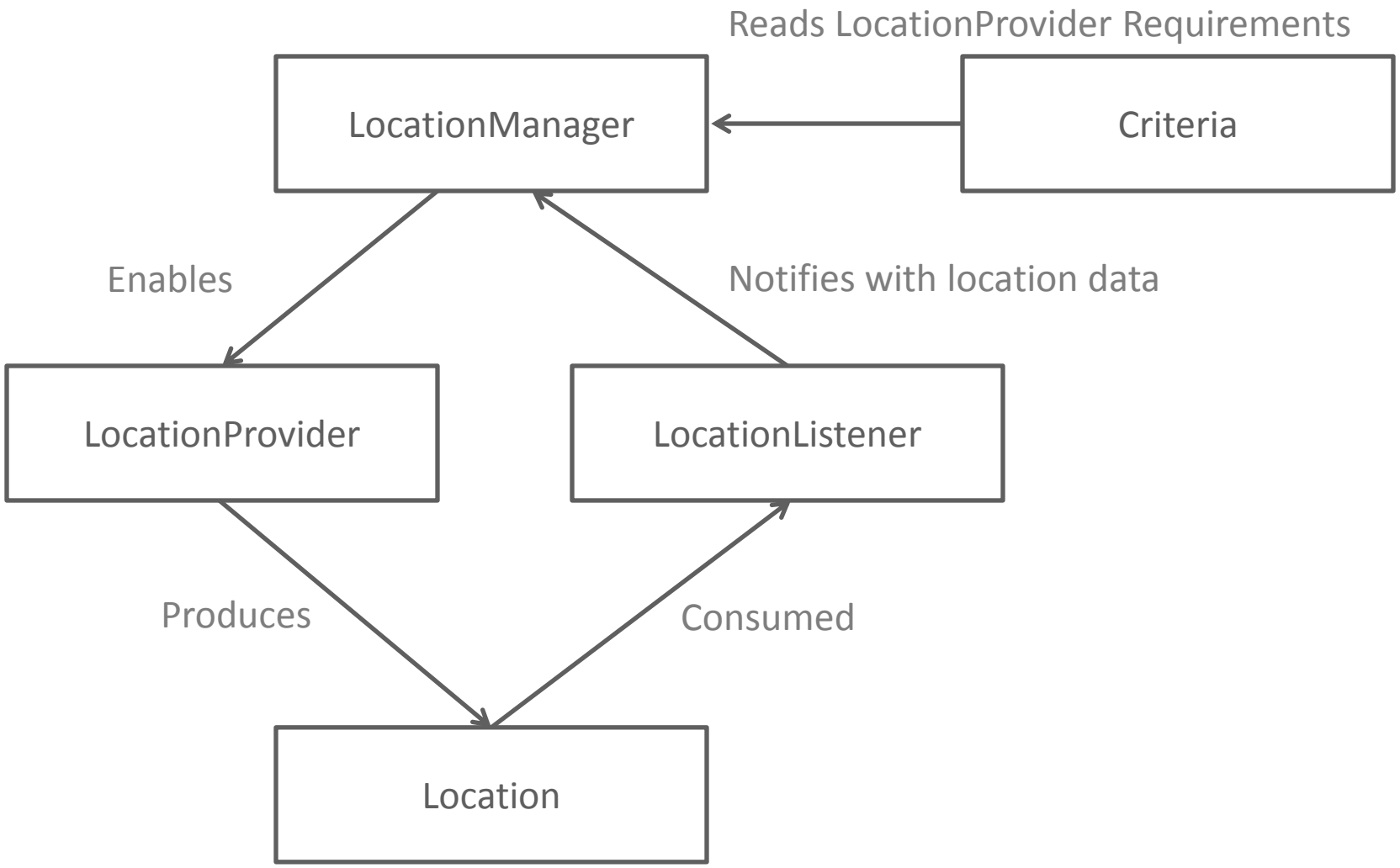
Class:

- SensorManager
- Sensor
- SensorEvent

Interface:

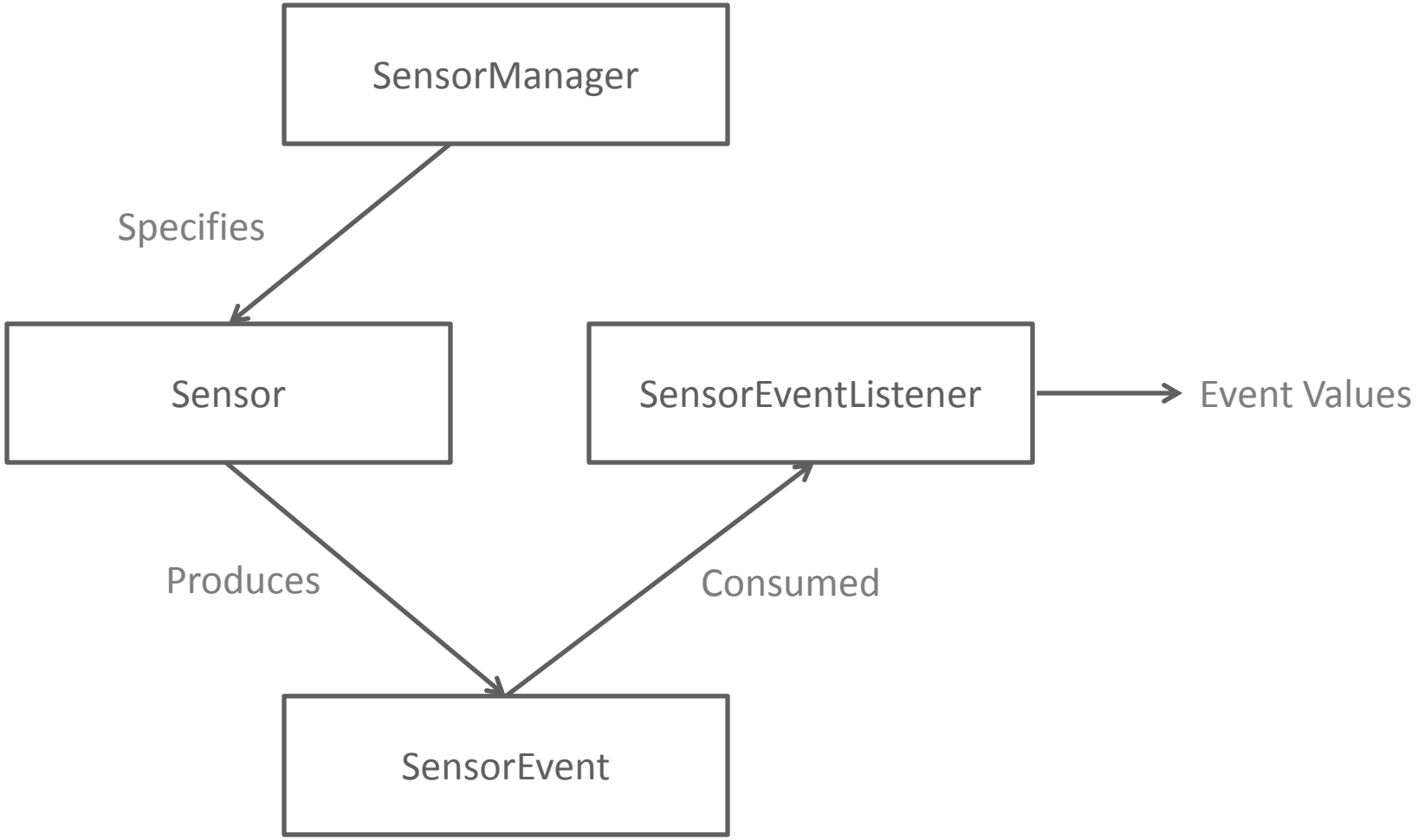
- SensorEventListener

Package: Android.location



Greg Milette, Adam Stroud: Professional Android Sensor Programing, 2012, Wiley India

Package: Android.hardware



Manifest File

```
<uses-feature android:name="android.hardware.sensor.accelerometer"
android:required="true" />
```

```
<uses-feature android:name="android.hardware.sensor.compass"
android:required="false" />
```

Manifest File

- android.hardware.sensor.accelerometer
- android.hardware.sensor.gyroscope
- android.hardware.sensor.compass
- android.hardware.sensor.barometer
- android.hardware.sensor.light
- android.hardware.sensor.proximity

SensorManager

- private SensorManager sensorManager;
- sensorManager = (SensorManager) getSystemService(*SENSOR_SERVICE);*

- List<Sensor> sensors = sensorManager.getSensorList(Sensor.TYPE_****);
- Sensor = sensorManager.getDefaultSensor(Sensor.TYPE_***);

Sensor Class

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

SensorEvent

- Accuracy
  - SENSOR_STATUS_ACCURACY_HIGH
  - SENSOR_STATUS_ACCURACY_MEDIUM
  - SENSOR_STATUS_ACCURACY_LOW
  - SENSOR_STATUS_ACCURACY_UNRELIABLE
- Sensor
- Timestamp
- Values

SensorEventListener

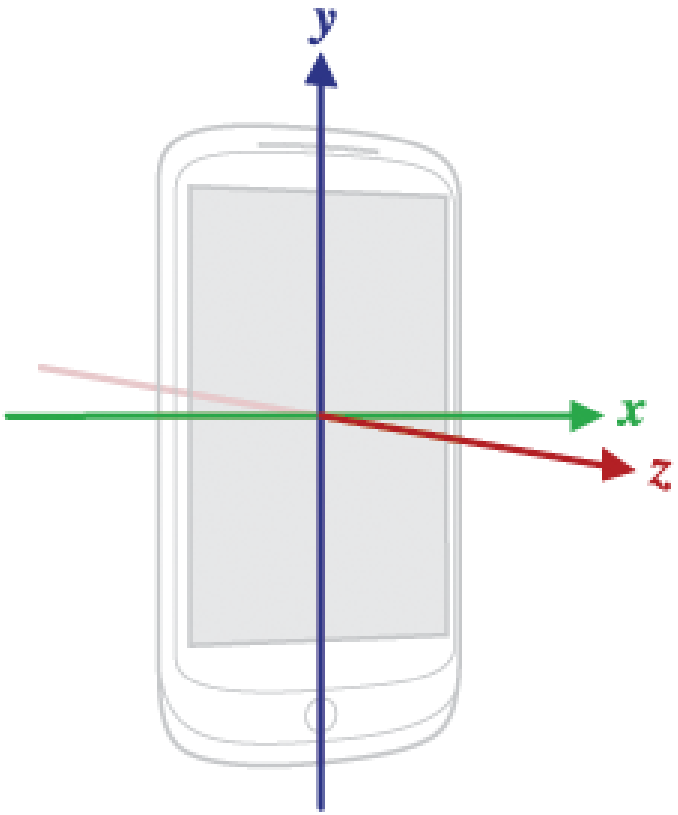
- onAccuracyChanged

  - when the accuracy from the sensor changes

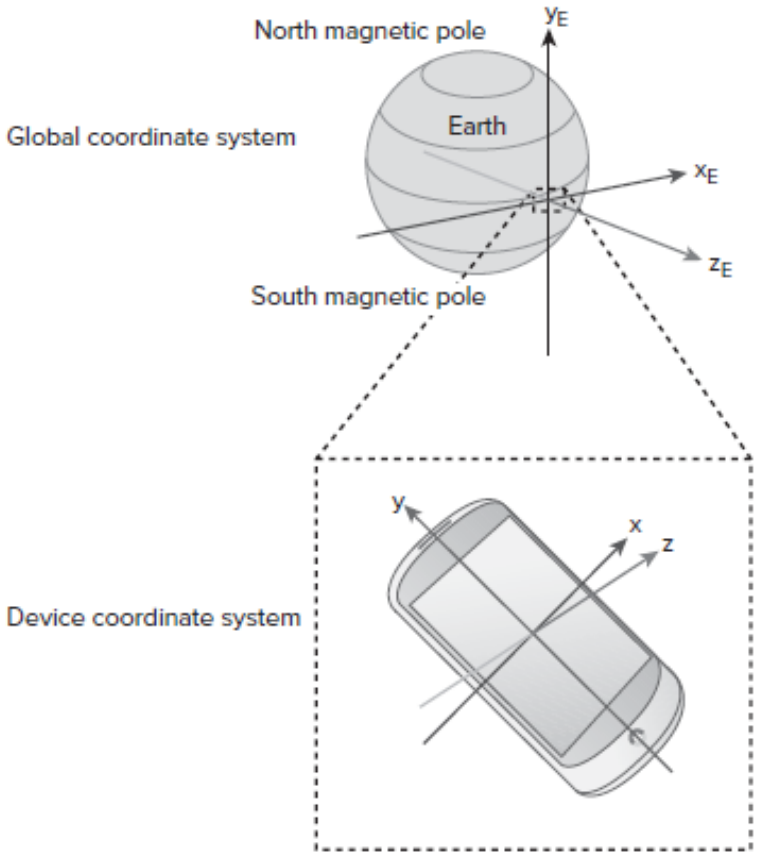
- onSensorChanged

  - when the values from the sensor changes

Sensor Data Reference



http://developer.android.com/guide/topics/sensors/sensors_overview.html

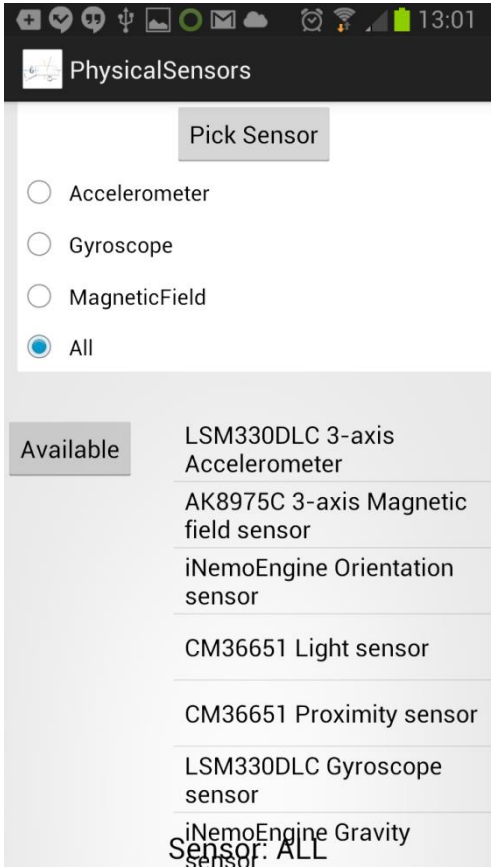


SOURCE: HTTP://DEVELOPER.ANDROID.COM/REFERENCE/ANDROID/HARDWARE/SENSOREVENT.HTML

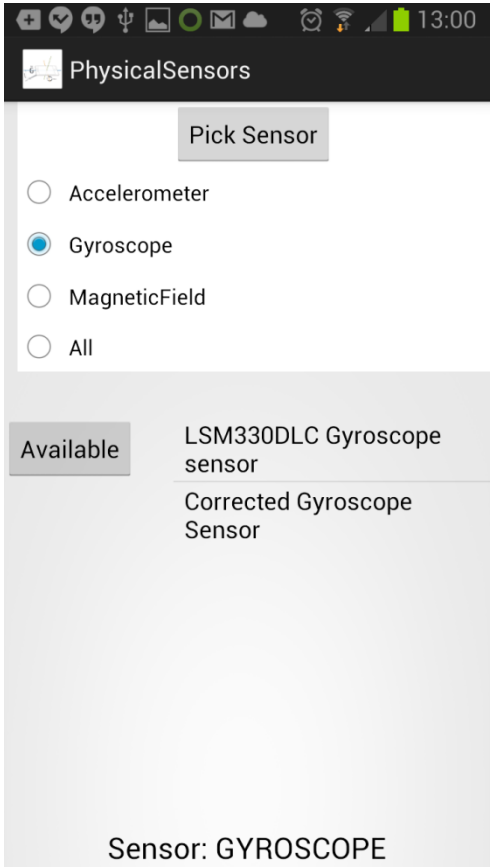
Lets Code …

1. Determine Available Sensors

```
List<Sensor> sensors = sensorManager.getSensorList(mySensor);
List<String> listSensorType = new ArrayList<String>();
for(int i=0; i<sensors.size(); i++)
{
    listSensorType.add(sensors.get(i).getName());
}
ListView availableList = (ListView) findViewById(R.id.availableList);
final ArrayAdapter adapter = new ArrayAdapter(this, android.R.layout.simple_list_item_1, listSensorType);
availableList.setAdapter(adapter);
break;
```



Sensor: ALL

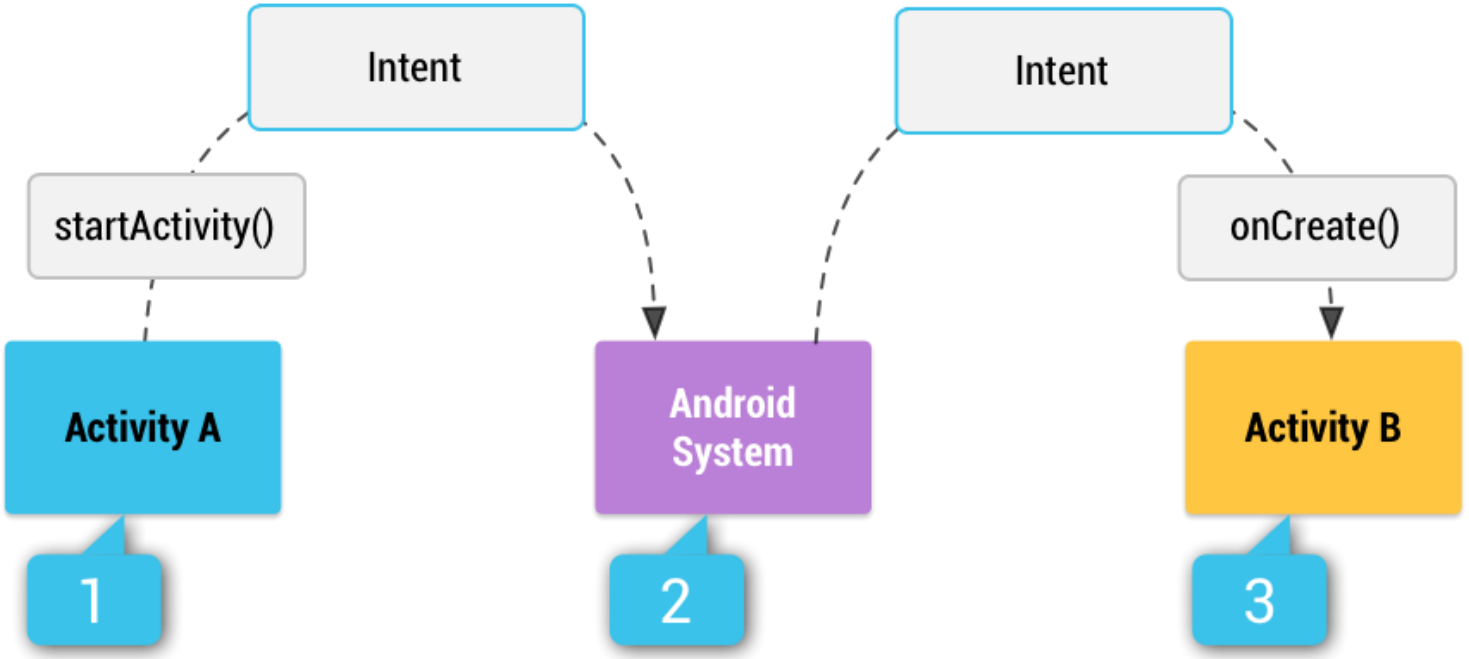


Sensor: GYROSCOPE

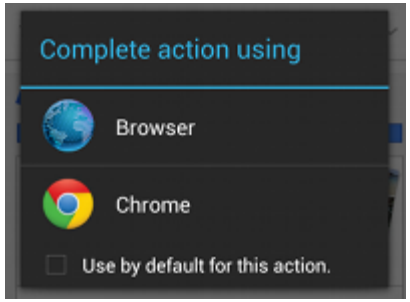
Building Block: Intent

- Move from one screen (activity) to another
- Can also open other applications
- 3 main: start an activity/service/broadcast
- Asynchronous
- Bound at run-time
- Explicit or implicit

Implicit Intent



http://developer.android.com/guide/components/intents-filters.html



Passing Data: Extras

- Primitive

```
Intent intent = new Intent(getApplicationContext(), SecondActivity.class);
intent.putExtra("Variable Name","Value you want to pass");
startActivity(intent);
```

Now on the OnCreate method of your SecondActivity you can fetch the extras like this

If the value you sent was in "long"

```
long value = getIntent().getLongExtra("Variable Name which you sent as an extra", defaultVal
```

If the value you sent was a "String"

```
String value = getIntent().getStringExtra("Variable Name which you sent as an extra");
```

If the value you sent was a "Boolean"

```
Boolean value = getIntent().getBooleanExtra("Variable Name which you sent as an extra",defau
```

Passing Data: Application

- Persistent Objects

```
public class SensorApplication extends Application
{
    private int mySensor = Sensor.TYPE_ALL;
    private SensorManager sensorManager;

    public void onCreate()
    {
        super.onCreate();
        sensorManager = (SensorManager) getSystemService(SENSOR_SERVICE);
    }

    public void setMySensor(int value)
    {
        mySensor = value;
    }

    public int getMySensor()
    {
        return mySensor;
    }

    public SensorManager getSensorManager()
    {
        return sensorManager;
    }
```

```
<application
    android:name="SensorApplication"
    android:allowBackup="true"

    android:icon="@drawable/ic_launcher"
    android:label="@string/app_name"
    android:theme="@style/AppTheme" >
```

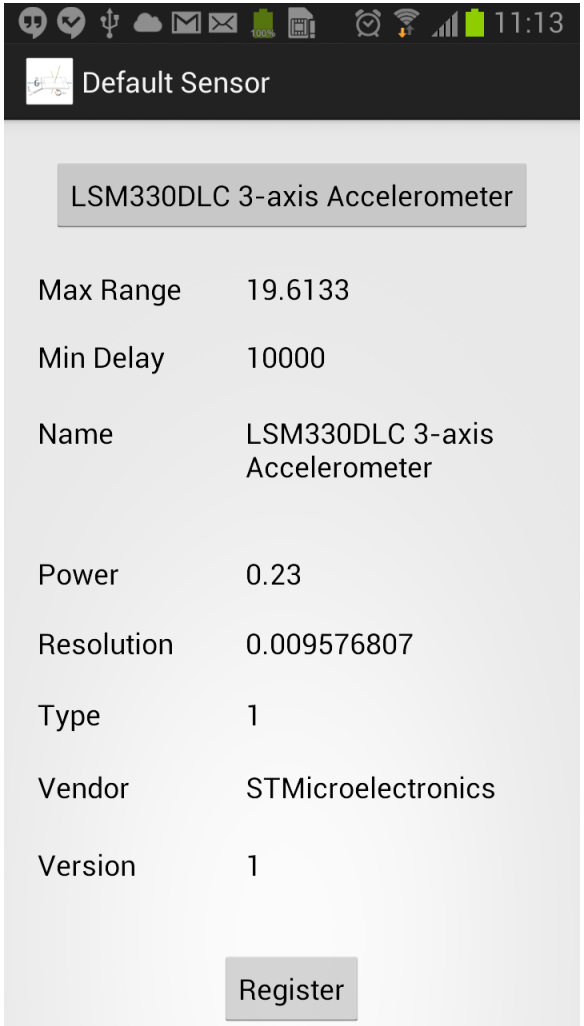
```
public class DefaultSensor extends Activity implements OnClickListener
{
    private SensorApplication app;

    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_default_sensor);
        app = (SensorApplication) getApplication();

        Button defaultButton;
        defaultButton = (Button) findViewById(R.id.sensorNameButton);
        defaultButton.setOnClickListener(this);
        defaultButton.setText(app.getSensorManager().getDefaultSensor(app.getMySensor()).getName());
    }

    public void onClick(View v) {
        // TODO Auto-generated method stub
        switch (v.getId())
        {
        case R.id.sensorNameButton:
            break;
        }
    }
}
```

2. Determine Sensor Range and Resolution




```
public void onCreate(Bundle savedInstanceState)
{
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_default_sensor);
    app = (SensorApplication) getApplication();

    Button defaultButton;
    defaultButton = (Button) findViewById(R.id.sensorNameButton);
    defaultButton.setOnClickListener(this);
    defaultSensor = app.getSensorManager().getDefaultSensor(app.getMySensor());
    defaultButton.setText(defaultSensor.getName());
}

@TargetApi(9)
public void onClick(View v) {
    // TODO Auto-generated method stub
    switch (v.getId())
    {
    case R.id.sensorNameButton:
        TextView maxRange = (TextView) findViewById(R.id.maxRange);
        maxRange.setText(""+defaultSensor.getMaximumRange());
        TextView minDelay = (TextView) findViewById(R.id.minDelay);
        minDelay.setText(""+defaultSensor.getMinDelay());
        TextView name = (TextView) findViewById(R.id.name);
        name.setText(""+defaultSensor.getName());
        TextView power = (TextView) findViewById(R.id.power);
        power.setText(""+defaultSensor.getPower());
        TextView resolution = (TextView) findViewById(R.id.resolution);
```

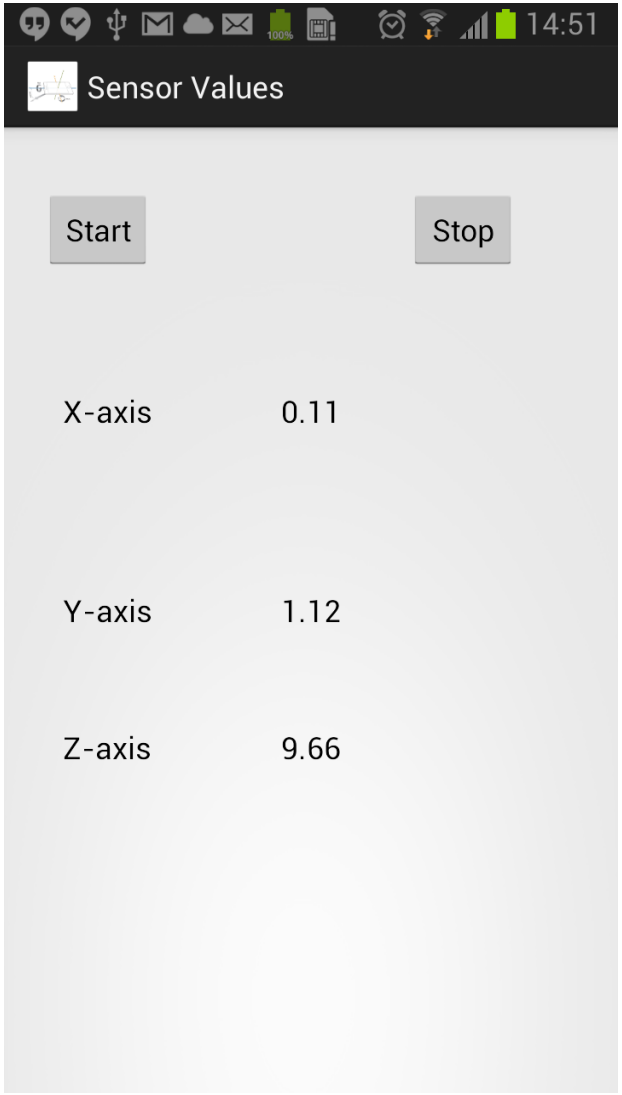
Units

- Acceleration: m/s^2
- Gyroscope: Deg/s or Rad/s
- Magnetic Field: uT
- Min Delay: us
- Power: mA

3. Define Sensor Rate

- SENSOR_DELAY_FASTEST 0ms
- SENSOR_DELAY_GAME 20ms
- SENSOR_DELAY_UI 67ms
- SENSOR_DELAY_NORMAL 200ms

Sensor Results Display




```
updateTimer = new Timer("aUpdate");
updateTimer.scheduleAtFixedRate(new TimerTask() {
    public void run() {
        updateGUI();
    }
}, 0, 100);
```

```
private void updateGUI() {
    runOnUiThread(new Runnable() {
        public void run() {
            String sX = String.format("%.2f", x);
            TextView p1 = (TextView)findViewById(R.id.xValueText);
            p1.setText(sX);

            String sY = String.format("%.2f", y);;
            TextView p2 = (TextView)findViewById(R.id.yValueText);
            p2.setText(sY);

            String sZ = String.format("%.2f", z);;
            TextView p3 = (TextView)findViewById(R.id.zValueText);
            p3.setText(sZ);
        }
    });
};
```

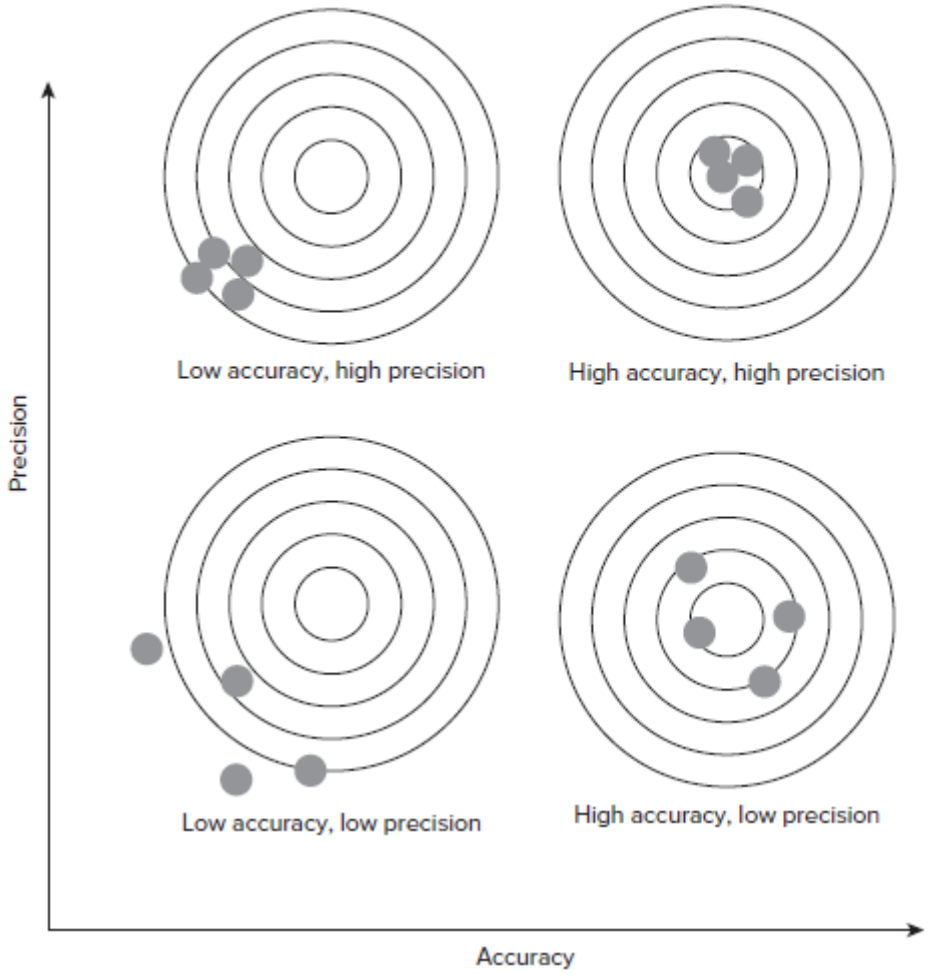
4. Register/Unregister SensorListeners

```
public void onClick(View v) {
    // TODO Auto-generated method stub
    switch (v.getId())
    {
        case R.id.startButton:
            app.getSensorManager().registerListener(mySensorListener, app.getSensorManager().getDefaul
            break;

        case R.id.stopButton:
            app.getSensorManager().unregisterListener(mySensorListener, app.getSensorManager().getDefa
            break;
    }
}
```

Signal Processing and Sensor Fusion

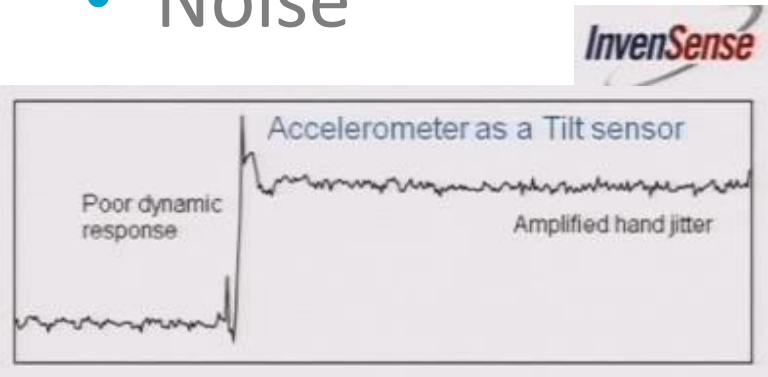
Accuracy and Precision

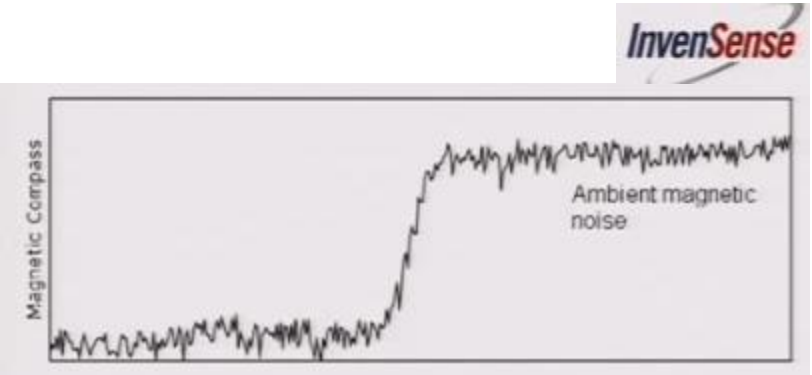


Greg Milette, Adam Stroud: Professional Android Sensor Programing, 2012, Wiley India

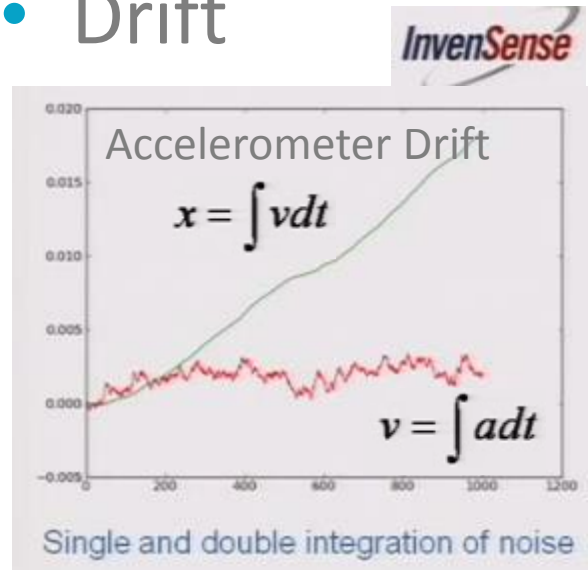
Type of Error

- Noise

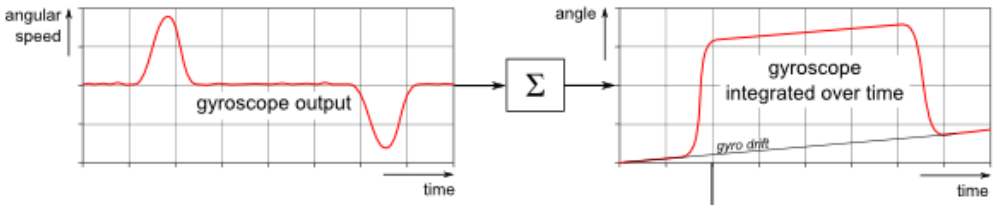




- Drift



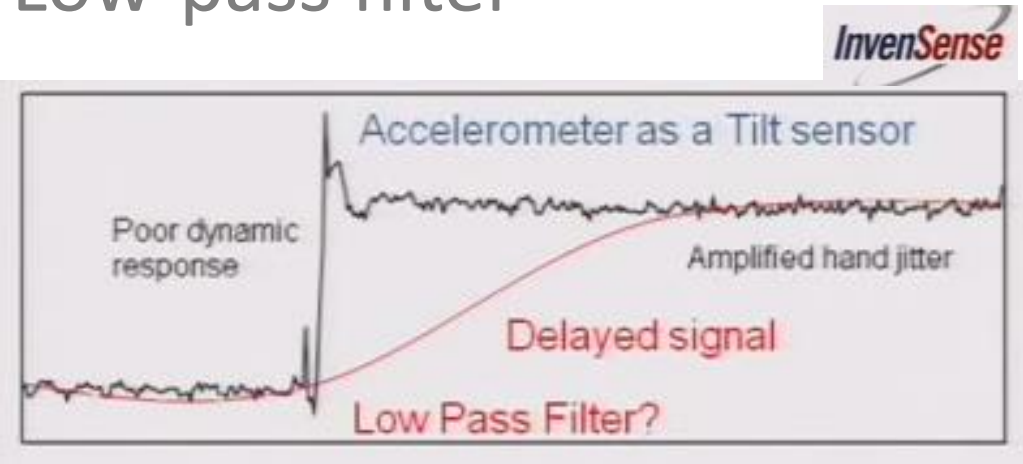
Gyroscope Drift



Paul Lawitski, Sensor fusion

Filtering

- Low-pass filter

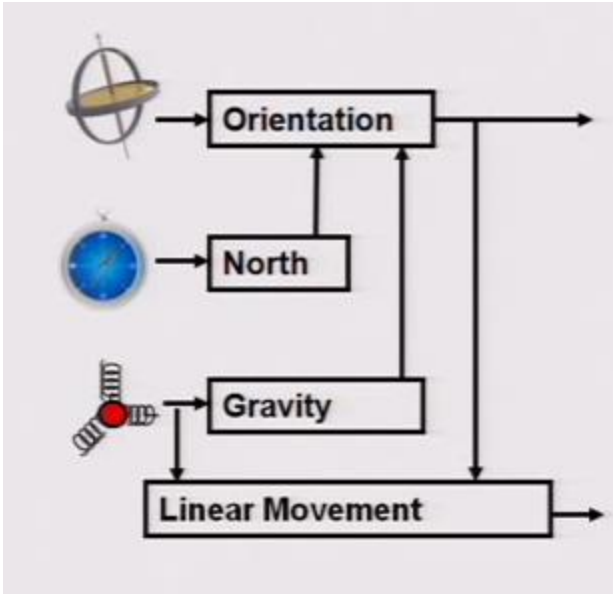


- High-pass filter to filter noise in gravity

- Kalman filter

  - systems dynamic model and constraints

Sensor Fusion



TYPE_GRAVITY

TYPE_LINEAR_ACCELERATION

TYPE_ROTATION_VECTOR

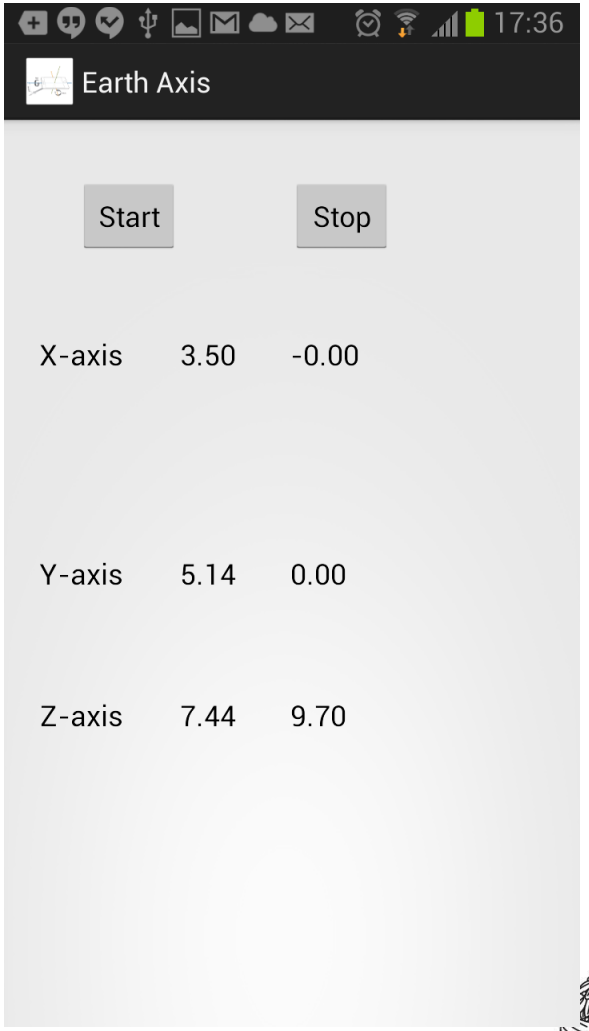
Acceleration in Earth's axis

```
case Sensor.TYPE_MAGNETIC_FIELD:
    Hfield = event.values.clone();
    break;
}

 final int matrix_size = 16;
 float[] R = new float[matrix_size];
 float[] outR = new float[matrix_size];
 float[] I = new float[matrix_size];

 if (Hfield != null && acceleration != null)
 {
     SensorManager.getRotationMatrix(R, I, acceleration, Hfield);
     //SensorManager.remapCoordinateSystem(R, SensorManager.AXIS_Y, Ser
     //SensorManager.getOrientation(R, values);
 }

 xE = R[0]*acceleration[0]+R[1]*acceleration[1]+R[2]*acceleration[2];
 yE = R[4]*acceleration[0]+R[5]*acceleration[1]+R[6]*acceleration[2];
 zE = R[8]*acceleration[0]+R[9]*acceleration[1]+R[10]*acceleration[2];
```



Coverage

- ☑ Activity
- ☑ Views
- ☑ Intent
- ☐ ContentProvider
- ☐ BroadcastReceiver
- ☐ Service