Project Description

- **Form Activity:** main activity which is presented to the user when launching the application for the first time. It displays a form and allows the user to fill and submit the form. When submitted, it retrieves the values filled by the user and stores as Bundle in Intent and starts another activity – DisplayActivity. Uses **form.xml as layout file** to display the form.
- **Display Activity:** it retrieves the Bundle from Intent which has values filled by the user and display it by setting the value of view elements. Uses **display.xml as layout file** to display the result.

Code...

Display code in Android Studio

Android Introduction

Third Part: Sensors

Wireless Systems Lab - 2014

Android Sensors Overview

- Android Sensors:

- MIC
- Camera
- Temperature
- Location (GPS or Network)
- Orientation
- Accelerometer
- Proximity
- Pressure
- Light

Sensors on Android

A sensor (also called detector) is a device that measures a physical quantity and converts it into a signal which can be read by an observer or by an instrument.

REAL (HARDWARE)

1. ACCELEROMETER
2. GYROSCOPE
3. MAGNETIC_FIELD
4. LIGHT
5. PRESSURE
6. AMBIENT_TEMP
7. RELATIVE_HUMIDITY

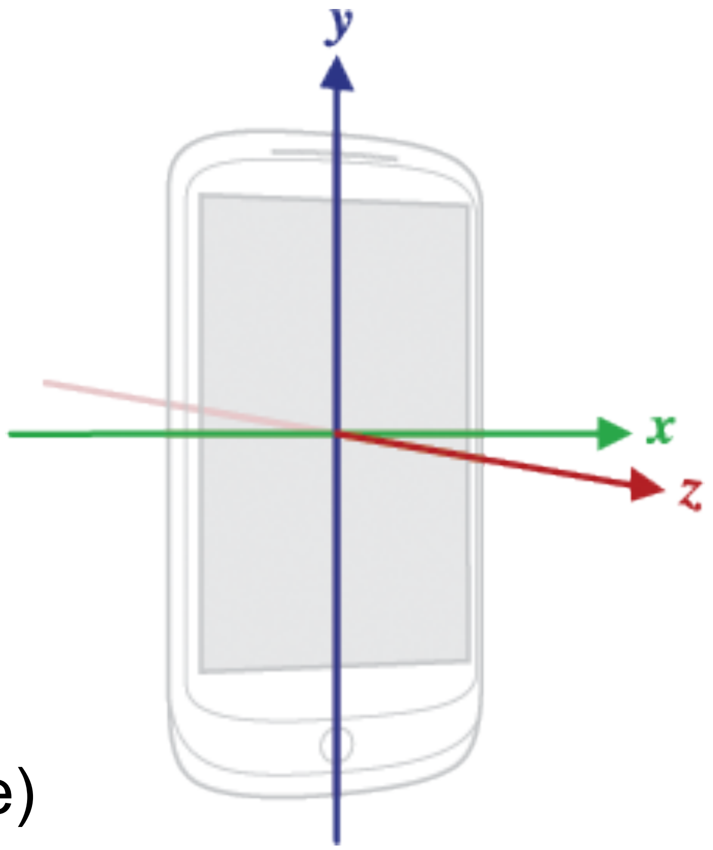
VIRTUAL (SOFTWARE)

1. PROXIMITY
2. LINEAR_ACCELERATION
3. GRAVITY
4. ROTATION_VECTOR

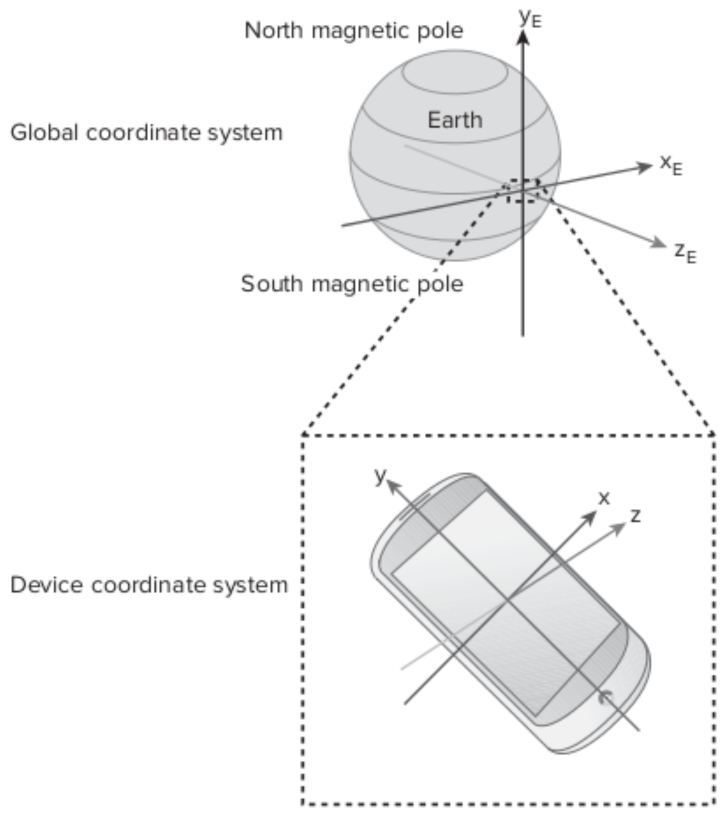
Useful reference: http://developer.android.com/guide/topics/sensors/sensors_motion.html

Sensor Coordinate System

- Standard 3-axis coordinate system to express data values.
- Used by
  - Acceleration sensor
  - Gravity sensor
  - Gyroscope
  - Linear acceleration sensor
  - Geomagnetic field sensor
- Defined relative to the device's natural screen (portrait or landscape)



Global Coordinate System



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

Map the device coordinate system on to the global coordinate system

In the global coordinate system:

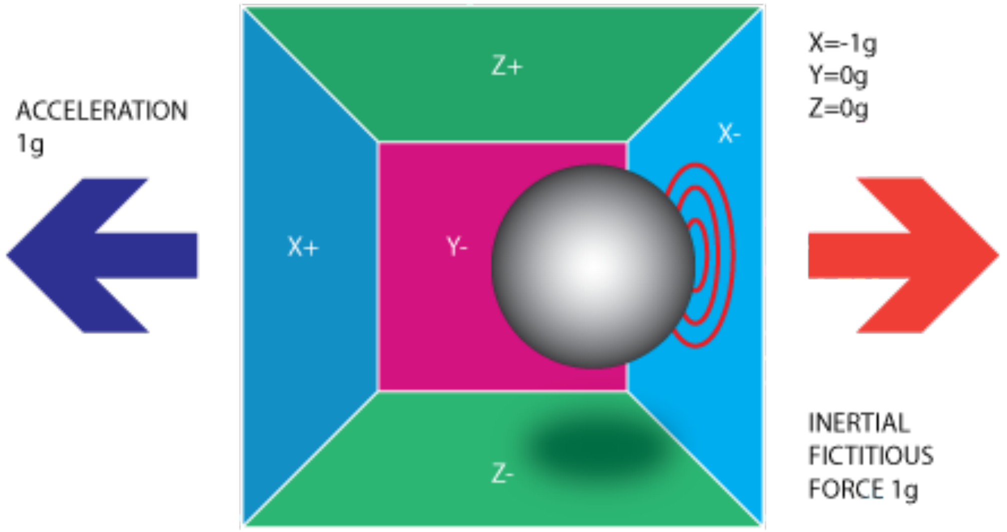
- Y_E points toward magnetic north, which is approximately true north.
- X_E points approximately east parallel to Earth's surface but 90 degrees from Y_E
- Z_E points away from the center of the earth

Accelerometer

Conceptually, an acceleration sensor determines the acceleration that is applied to a device (A_d) by measuring the forces that are applied to the sensor itself (F_s) using the following relationship:

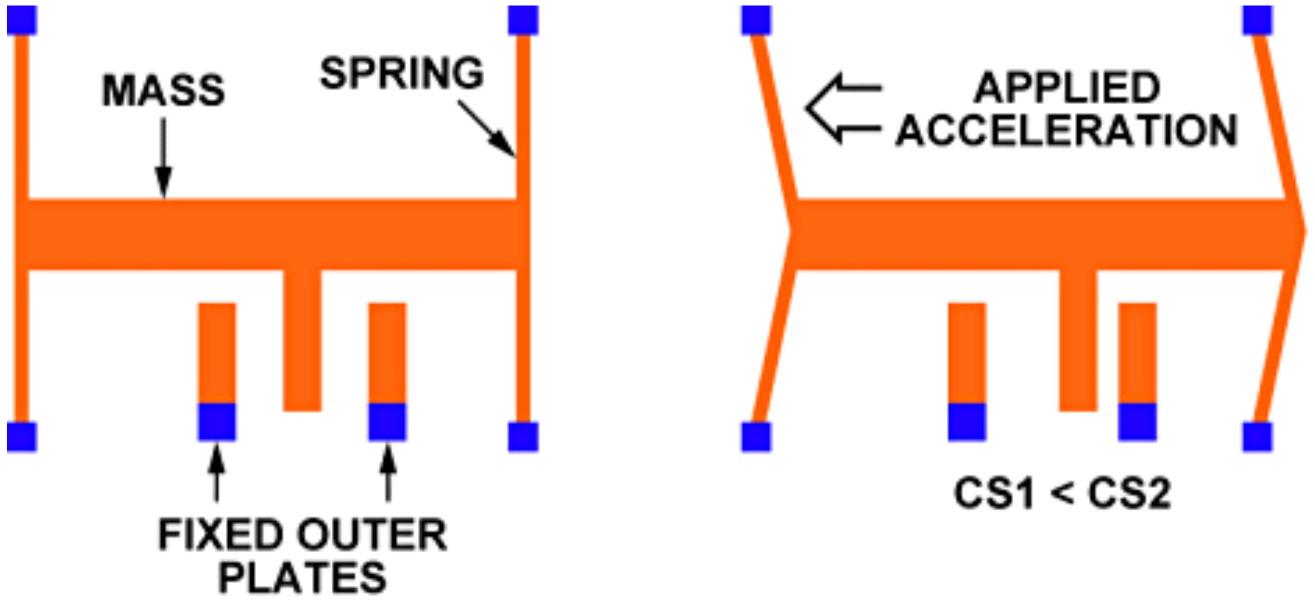
$$A_d = -g - \sum F / mass$$

where $g = 9.81\ m/s^2$



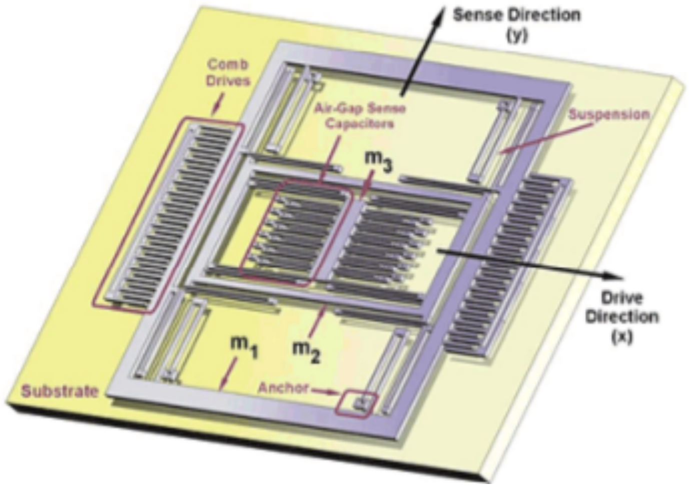
Sensors Hardware

- Accelerometer
  - Microelectromechanical sensors (MEMS)



Sensors Hardware

- Gyroscope

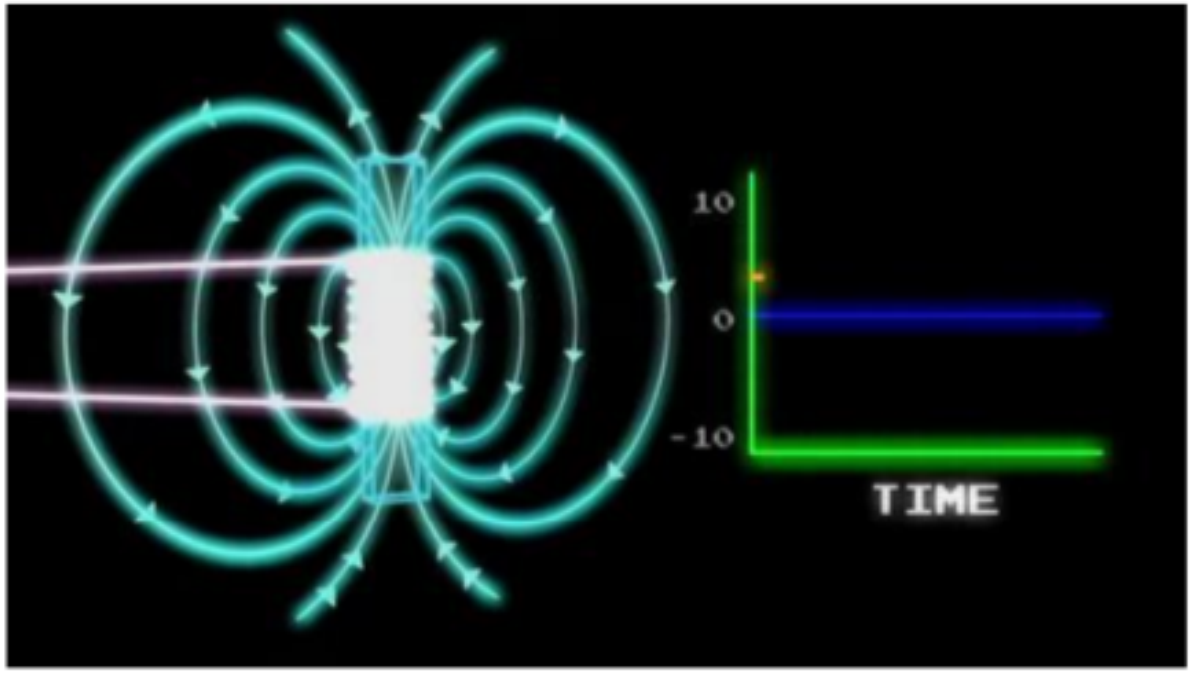


Gyroscope

- MEMS gyroscopes are tiny masses on tiny springs, designed to measure the Coriolis force due to rotation.
- The Coriolis force is the tendency for a free object to veer off course when viewed from a rotating reference frame.
- Gyroscopes measure only angular velocity, or, the speed at which the device is rotating. When the device is stationary, regardless of which direction the device is pointing, all three axes of the gyroscope will measure zero.
- Android reports values in radians per second around the standard x, y, and z axes

Sensors Hardware

- Compass (Magnometer)

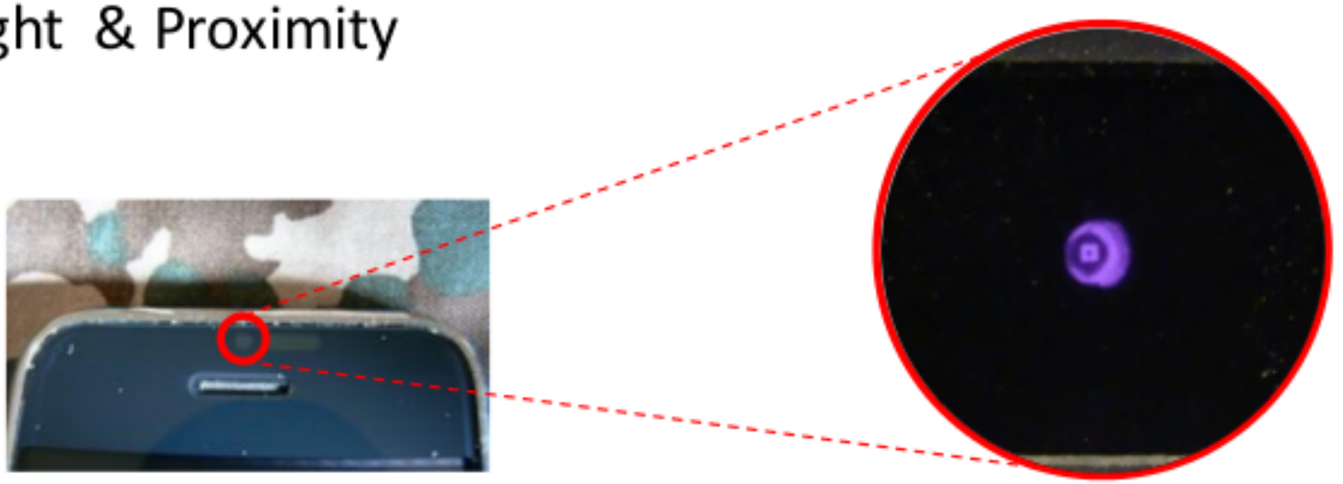


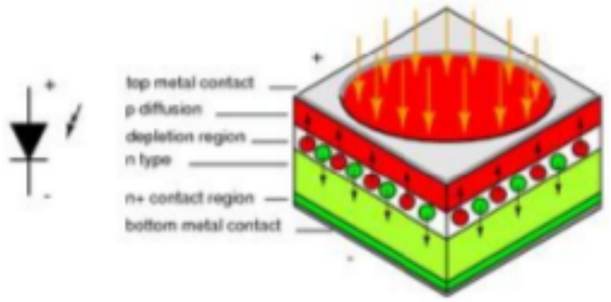
Magnetic field sensors

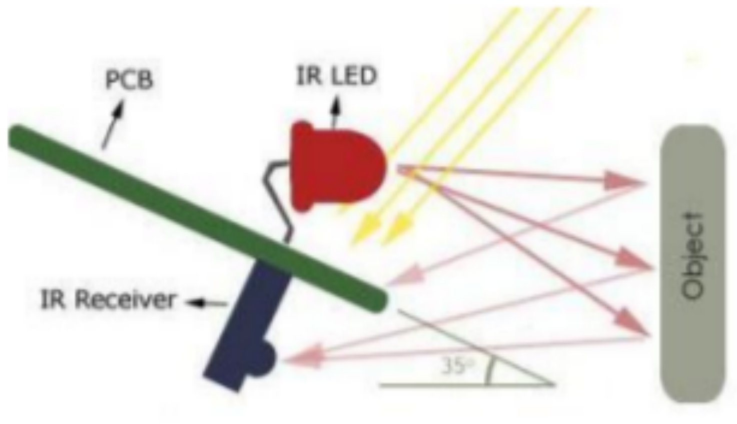
- Hall effect:
  - It works by simply passing a current through a wire.
- A magnetic field component perpendicular to that wire causes the electrons to have higher density on one side of the wire compared to the other, which results in a voltage across the width of the wire that is proportional to the magnetic field.
- Magnetic field sensors will report the magnetic field in x, y, and z in microTesla.

Sensors Hardware

- Light & Proximity







Light Sensor

- It is simply a photodiode, which operates on the same physical principle as an LED (light-emitting diode) but in reverse. Instead of generating light when a voltage is applied, it generates a voltage when light is incident on it.
- The light sensor reports its values in lux:
  - LIGHT_NO_MOON : 0.001
  - LIGHT_FULLMOON : 0.25
  - LIGHT_CLOUDY : 100
  - LIGHT_SUNRISE : 400
  - LIGHT_OVERCAST : 10000
  - LIGHT_SHADE : 20000
  - LIGHT_SUNLIGHT : 110000
  - LIGHT_SUNLIGHT_MAX : 120000

(Constants in SensorManager class)

Proximity Sensor

- It consists of a weak infrared LED (light-emitting diode) next to a photodetector. When something (such as the ear of a person making a phone call) comes close enough to the sensor, the photodetector detects the reflected infrared light.
- Some proximity sensors report the distance to an object in centimeters. Others are not designed to measure the distance to an object, but only the presence or absence of an object at a distance closer than some threshold (around 5 cm)

Sensor.TYPE_ROTATION_VECTOR

- It’s a synthetic sensor that calculates rotation angle of the global coordinate system with respect to the device coordinate system using the accelerometer, the magnetometer, and possibly the gyroscope if available.
- **getRotationMatrix()** : map coordinates between local reference frame and global coordinate system
- **SensorManager.getOrientation**(rotationMatrix) get the orientation (azimuth, pitch and roll) with respect to earth
  - Azimuth (or heading or yaw) = Rotation about z-axis
  - Pitch = Rotation about x-axis
  - Roll = Rotation about y-axis
- Sensor.TYPE_ORIENTATION is deprecated

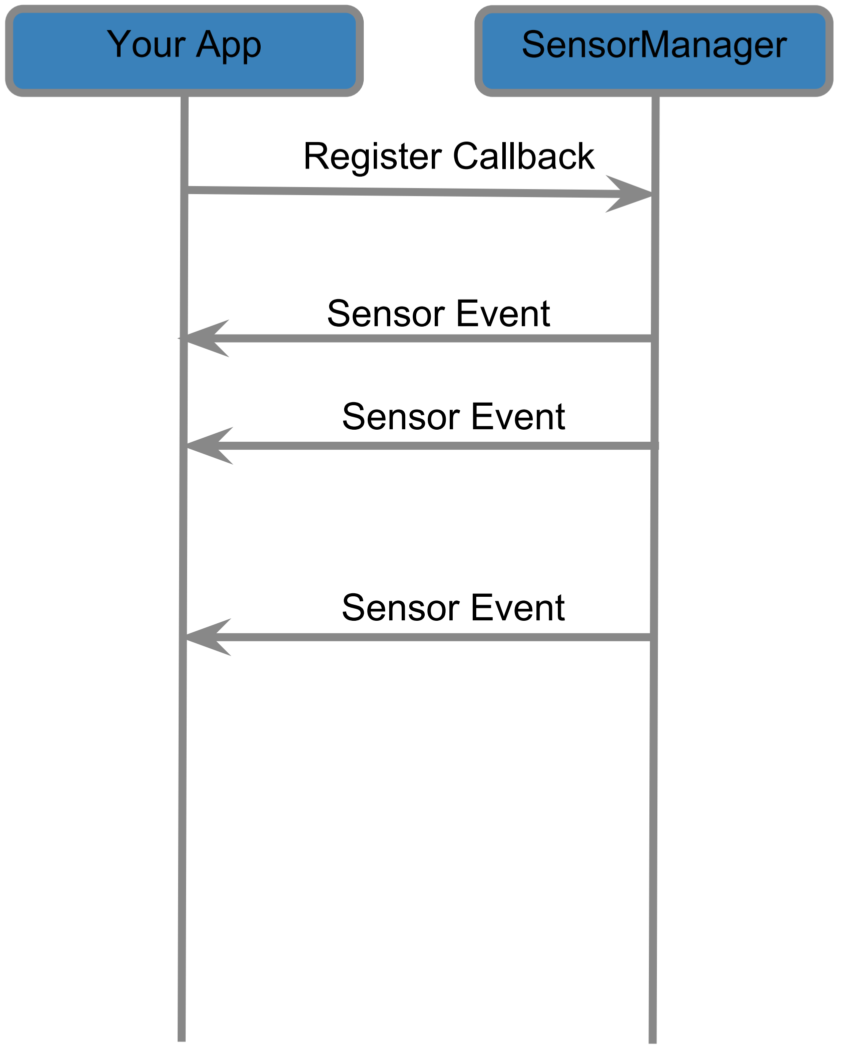
Async Callbacks

•Android's sensors are controlled by external services and only send events when they choose to

•An app must register a callback to be notified of a sensor event

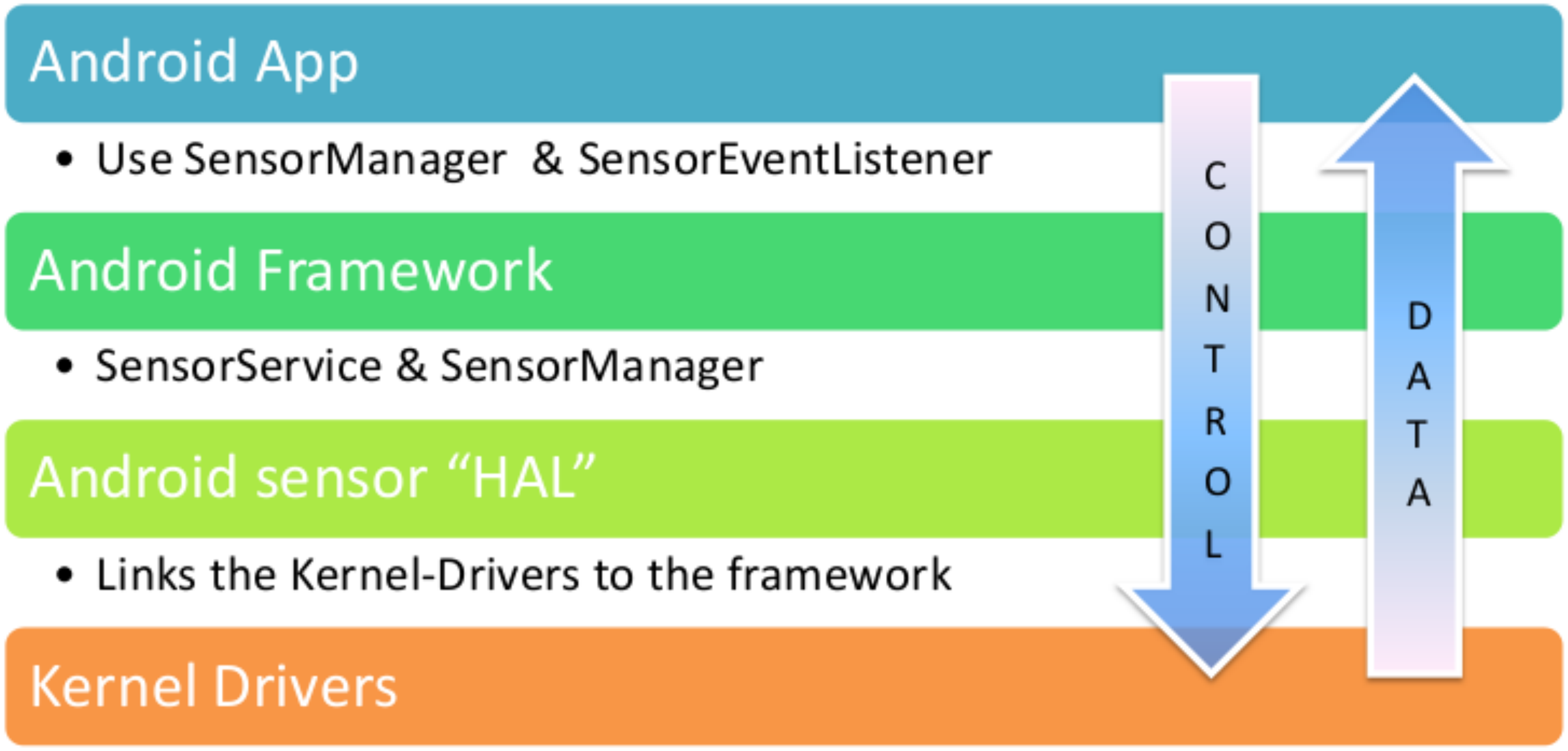
•Each sensor has a related XXXListener interface that your callback must implement

- E.g. LocationListener



SAPIENZA
UNIVERSITÀ DI ROMA

Sensors Framework Overview



Android App

- Use SensorManager & SensorEventListener

Android Framework

- SensorService & SensorManager

Android sensor "HAL"

- Links the Kernel-Drivers to the framework

Kernel Drivers

- Device drivers to control and gather data from the actual hardware.

Android Sensors API

- 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
- 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

Getting the Relevant System Service

- The non-media (e.g. not camera) sensors are managed by a variety of XXXXManager classes:
  - LocationManager (GPS)
  - SensorManager (accelerometer, gyro, proximity, light, temp)
- The first step in registering is to obtain a reference to the relevant manager
- Every Activity has a getSystemService() method that can be used to obtain a reference to the needed manager

```
public class MyActivity … {

  private SensorManager sensorManager_;

  public void onCreate(){
     …

     sensorManager_ = (SensorManager) getSystemService(SENSOR_SERVICE);
  }

}
```

Registering for Sensor Updates

- The SensorManager handles registrations for
  - Accelerometer, Temp, Light, Gyro
- In order for an object to receive updates from GPS, it must implement the SensorEventListener interface
- Once the SensorManager is obtained, you must obtain a reference to the specific sensor you are interested in updates from
- The arguments passed into the registerListener method determine the sensor that you are connected to and the rate at which it will send you updates

```
public class MyActivity … implements SensorListener{
  private Sensor accelerometer_;
  private SensorManager sensorManager_;

  public void connectToAccelerometer() {
     sensorManager_ = (SensorManager)getSystemService(SENSOR_MANAGER);
     accelerometer_ = sensorManager_
                    .getDefaultSensor(Sensor.TYPE_ACCELEROMETER);
     sensorManager_.registerListener(this, accelerometer_,
                    SensorManager.SENSOR_DELAY_NORMAL);


}
```

Best practices for accessing and using sensors

1. Unregister sensor listeners.
2. Don’t test your code on the emulator.
3. Don’t block the onSensorChanged() method.
4. Avoid using deprecated methods or sensor types.
5. Verify sensors before you use them.
6. Choose sensor delays carefully.
7. Filter the values received in onSensorChanged(). Allow only those that are needed.

Being a Good Citizen...

- It is very important that you unregister your App when you no longer need updates
- For example, you should always unregister your listener when your Activity is paused!
- If you unregister when you pause, you must also re-register when you resume
  - This is true for all sensors!

```
public class MyActivity … {

     private LocationManager locationManager_;

     public void onCreate(Bundle savedInstanceState) {
          …
          locationManager_ = (LocationManager)getSystemService(LOCATION_SERVICE);
     }
     protected void onPause() {
          super.onPause();
          locationManager_.removeUpdates(this);
     }
     protected void onResume() {
          super.onResume();
          locationManager_.requestLocationUpdates(LocationManager.GPS_PROVIDER, 10,
                                                  Criteria.ACCURACY_FINE, this);
     }
…
}
```

Sensor Event Listener

- After we register the Sensors, the sensor readings get notified in SensorEventListener‘s onSensorChanged() method.
- To avoid that small changes (noise) jump within a large range of values we can specify the SensorManager‘s delay properties from one of these:
  1. SENSOR_DELAY_FASTEST (0 ms)
  2. SENSOR_DELAY_GAME (20 ms)
  3. SENSOR_DELAY_UI (67 ms)
  4. SENSOR_DELAY_NORMAL (200 ms)

Remember:

Allow only those values which are useful and discard the unnecessary noise.

Sensor Manager

- Because there is one interface for multiple types of sensors, listening to multiple sensors requires switching on the type of event (or creating separate listener objects)

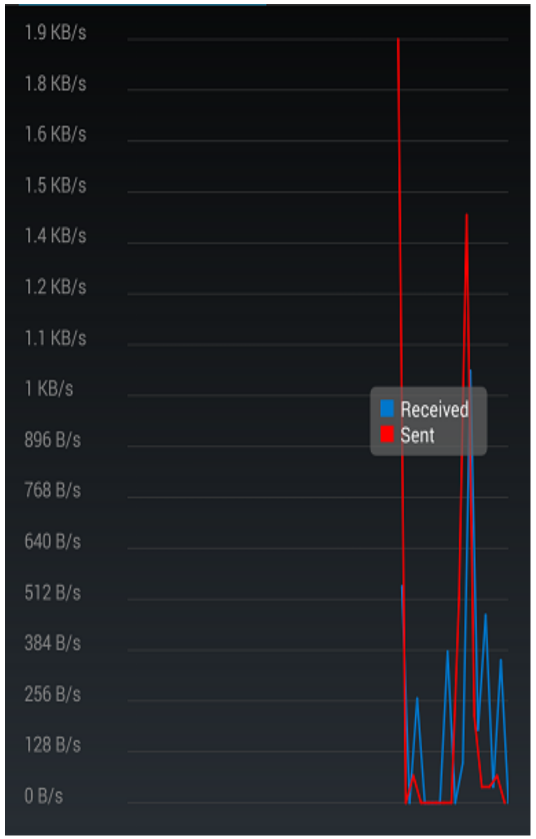
```
private void initAccel(){
    mSensorManager = (SensorManager) getSystemService(SENSOR_SERVICE);
    mSens = mSensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER);
    mSensorManager.registerListener(this, mSens,
                                    SensorManager.SENSOR_DELAY_GAME);
}

@Override
public void onSensorChanged(SensorEvent event) {
  if (event.sensor.getType() == SensorManager.SENSOR_ACCELEROMETER) {
       float x = event.values[SensorManager.DATA_X];
       float y = event.values[SensorManager.DATA_Y];
       float z = event.values[SensorManager.DATA_Z];
    }
}
```

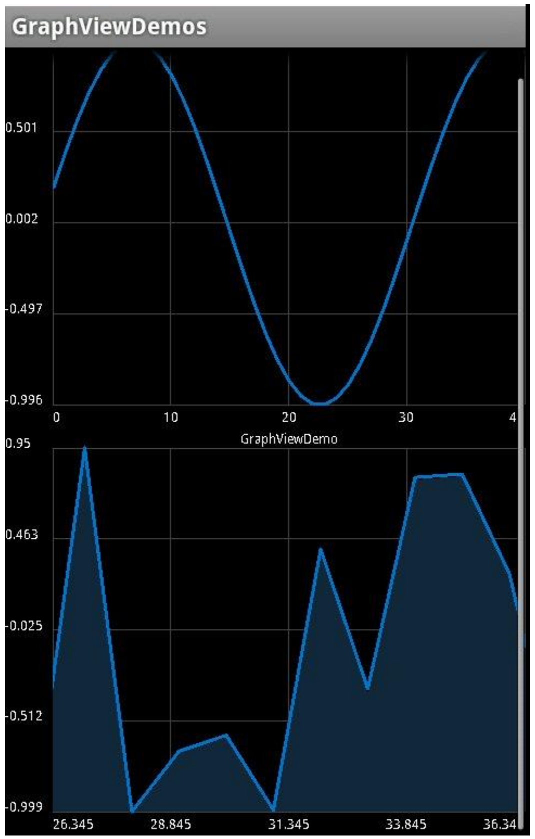
- Another approach for multiple sensors:

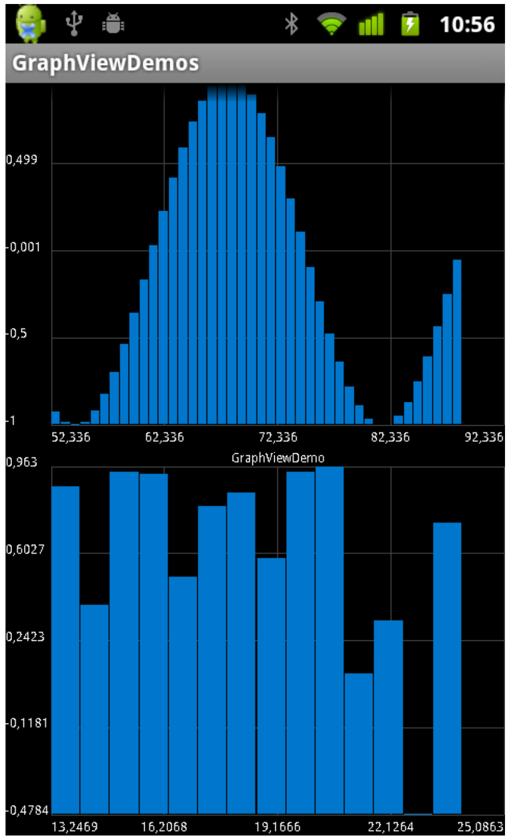
```
public class MyActivity … {

         private class AccelListener implements SensorListener {
               public void onSensorChanged(SensorEvent sensorEvent) {
                    …
               }
               public void onAccuracyChanged(Sensor arg0, int arg1) {}
          }

     private class LightListener implements SensorListener {
               public void onSensorChanged(SensorEvent sensorEvent) {
                    …
               }
               public void onAccuracyChanged(Sensor arg0, int arg1) {}
          }

          private SensorListener accelListener_ = new AccelListener();
          private SensorListener lightListener_ = new LightListener();

          …
          public void onResume(){
             …
           sensorManager_.registerListener(accelListener, accelerometer,
                                SensorManager.SENSOR_DELAY_GAME);
           sensorManager_.registerListener(lightListener, lightsensor,
                                SensorManager.SENSOR_DELAY_NORMAL);

          }
          public void onPause(){
            sensorManager_.unregisterListener(accelListener_);
            sensorManager_.unregisterListener(lightListener_);
          }
```

Android GraphView Library

Reference: http://android-graphview.org/#



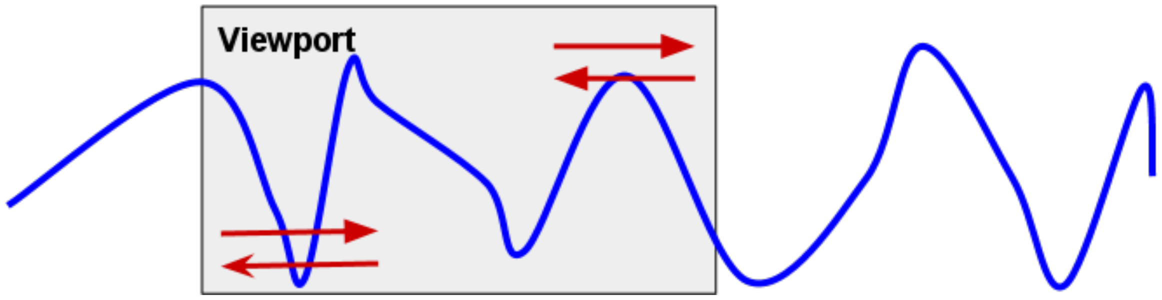




A simple graph

```
// init example series data
GraphViewSeries exampleSeries = new GraphViewSeries(new GraphViewData[] {
    new GraphViewData(1, 2.0d)
    , new GraphViewData(2, 1.5d)
    , new GraphViewData(3, 2.5d)
    , new GraphViewData(4, 1.0d)
});

GraphView graphView = new LineGraphView(
    this // context
    , "GraphViewDemo" // heading
);
graphView.addSeries(exampleSeries); // data

LinearLayout layout = (LinearLayout) findViewById(R.id.layout);
layout.addView(graphView);
```

Viewport



```
...
GraphView graphView = new LineGraphView(
    this
    , "GraphViewDemo"
);
// add data
graphView.addSeries(new GraphViewSeries(data));
// set view port, start=2, size=40
graphView.setViewPort(2, 40);
graphView.setScrollable(true);
// optional - activate scaling / zooming
graphView.setScalable(true);

LinearLayout layout = (LinearLayout) findViewById(R.id.layout);
layout.addView(graphView);
```

More styles...

```
graphView.getGraphViewStyle().setGridColor(Color.GREEN);
graphView.getGraphViewStyle().setHorizontalLabelsColor(Color.YELLOW);
graphView.getGraphViewStyle().setVerticalLabelsColor(Color.RED);
graphView.getGraphViewStyle().setTextSize(getResources().getDimension(R.
dimen.big));
graphView.getGraphViewStyle().setNumHorizontalLabels(5);
graphView.getGraphViewStyle().setNumVerticalLabels(4);
graphView.getGraphViewStyle().setVerticalLabelsWidth(300);
```

Errors and sensors signal processing

- Sensors do not measure values perfectly. Instead, they can often produce data that is incorrect due to noise or because of degradation that occurs over time. Both of these problems may introduce errors in the resulting data.
- To reduce errors, an app can filter output from individual sensor readings or fuse results from multiple sensors.
  - **Noise**: random fluctuation of a measured value
  - **Drift**: slow, long-term wandering of data away from the real-world value
  - **Zero Offset ("Bias")**: if the output signal is not zero when the measured property is zero, the sensor has an offset or bias.

Techniques to Address Error

- **Re-zeroing**: if there is an offset present that is affecting your application, it may be useful to re-zero the sensor measurements. This is as simple as storing a calibrated value and subtracting it from each measured value.
- **Filters**: Low-pass filters filter out any high-frequency signal or noise and have a “smoothing” effect on data. High-pass filters filter out slow drift and offset and just give the higher frequency changes.
- **Sensor fusion**: it refers to using more than one sensor to take advantage of the strengths of each sensor and mitigate the effects of the weaknesses.

Low-pass filter

- A low-pass filter passes low-frequency signals/values and attenuates (reduces the amplitude of) signals/values with frequencies higher than the cutoff frequency.

- Take an example of simple signal with values ranging from 0 to 1.

- Due to an external source (environmental factors such as jerks or vibrations), a considerable amount of noise is added to these signals. These high frequency signals (noise) cause the readings to hop between considerable high and low values.

Apply Low-pass filter

New mean= Last value * (1– α) + x[i] * α

Here is the algorithm implementation:

```
for i from 1 to n
   y[i] := y[i-1] + α * (x[i] - y[i-1])
```

Here, α is the cut-off/threshold.

Lets implement it in Android:

lowPass(**float**[] input, **float**[] output)

The above method filters the input values and applies LPF and outputs the filtered signals.

```
static final float ALPHA = 0.25f; // if ALPHA = 1 OR 0, no filter applies.

protected float[] lowPass( float[] input, float[] output ) {
        if ( output == null ) return input;
        for ( int i=0; i<input.length; i++ ) {
            output[i] = output[i-1] + ALPHA * (input[i] - output[i-1]);
        }
        return output;
    }
```

High Pass Filter

- Emphasizes the higher frequency or transient components
- Inverse of Low Pass Filter

```
final float a = 0.8;

/* Note: this is a low pass filter with input the new event.value[0] */

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

/* Linear acceleration is filtered by removing “static” values */

higher_frequency_value=event.values[0] - change_slow
```

Lets try them all ...

- Build an App that :
  - List all available sensor from the phone
  - Choose a sensor to read data
  - Display sensor data
  - Plot the related data over time
  - Implement the two filters above
-
- Follow the hint code :)

Activities

Two main Activity:

- MainActivity
  - List all type of sensors (string:name, int:type) available on cellphone
  - Create a button for each of them
  - Pass all the data through an Intent to Sensor Activity
- SensorActivity
  - Get the type of sensor clicked from the first activity
  - Register to get the values
  - Display and plot the values

Activity Main Layout

Used ScrollView with linear layout e.g.,

```
<?xml version="1.0" encoding="utf-8"?>
<ScrollView  xmlns:android="http://schemas.android.com/apk/res/android"
        android:id="@+id/scroll"
        android:layout_width="match_parent"
        android:layout_height="wrap_content">

<LinearLayout
        android:orientation="vertical"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:id="@+id/linearlayout1">
 </LinearLayout>

</ScrollView>
```

Sensor Activity Layout

```
<?xml version="1.0" encoding="utf-8"?>

<ScrollView  xmlns:android="http://schemas.android.com/apk/res/android"
     android:id="@+id/scroll"
        android:layout_width="match_parent"
       android:layout_height="wrap_content
/*Linear layout to print sensor data*/
<LinearLayout
         android:layout_width="match_parent"
         android:layout_height="wrap_content"
         android:orientation="vertical"  >

<TextView
       android:id="@+id/name"
       android:textAppearance="?android:attr/textAppearanceSmall" />
/*Linear Layout to print the graph*/
    <LinearLayout
       android:id="@+id/graph" >
   </LinearLayout>

</LinearLayout>
</ScrollView>
```

Activity Main: Sensor List

- Get all the information from the type of sensors

```
public LinkedList<SensorData> getInfo(){
        /* Sensor Manager */
        sMgr = (SensorManager) this.getSystemService(SENSOR_SERVICE);
        List<Sensor> list = sMgr.getSensorList(Sensor.TYPE_ALL);
        /* Define a class Sensor Data */
        LinkedList<SensorData> list = new LinkedList<SensorData>();

        /* For each sensor get the data about the sensor and add to a data structure */
                for (Sensor sensor : list) { }
}
```

List all the Buttons

```
protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        LinkedList<SensorData> listSensorData ; Type and properties of sensors
        MAX_BUTTONS ; //Max number of the buttons
        /*Create an array of buttons*/
        btnWord = new Button[MAX_BUTTONS];
        /* Get the linear layout to assign the button array*/
        linear = (LinearLayout) findViewById(R.id.linearlayout1);
        for (int i = 0; i < btnWord.length; i++) {
            name =.. // name of sensor
            type  = .. // type of sensor
            btnWord[i] = new Button(this);
            btnWord[i].setText(name);
            btnWord[i].setId(i + 1); // ID start from 1
            btnWord[i].setOnClickListener(new MyOnClickListener(type));
            linear.addView(btnWord[i]); // add the button to the linear layout
```

Define a custom clickListener

```
class MyOnClickListener implements  View.OnClickListener {

            private final int sensorType;
            /* Define myOnClickListener with a custom ID*/
              public MyOnClickListener(int id) {
                sensorType = id;
              }

              public void onClick(View v) {
                        /*Create an intent to start the SensorActivity with the choosen sensor type */
                        Intent myIntent = new Intent(v.getContext(), SensorActivity.class);
                        myIntent.putExtra("sensor_type",sensorType );
                        v.getContext().startActivity(myIntent);
               }


            }
```

SensorActivity

```
protected void onCreate(Bundle savedInstanceState) {
                super.onCreate(savedInstanceState);
                setContentView(R.layout.sensor_activity);
                /* Get the intent with the selected type of sensor */
                Intent intent = getIntent();
                int type = intent.getIntExtra("sensor_type", -1);
                SensorManager senSensorManager = (SensorManager) getSystemService(Context.
    SENSOR_SERVICE);
                 /* Register for the type of the sensor */
     Sensor senGeneral = senSensorManager.getDefaultSensor(type);
     senSensorManager.registerListener(this, senGeneral , SensorManager.SENSOR_DELAY_NORMAL);
     /*senGeneral is the type of sensor choosen, displaySensor prints information about the type of the sensor e.g.,
    name power consumpiton etc*/
     displaySensor(senGeneral);
    /*Create the graph */
      createGraph(); }
```

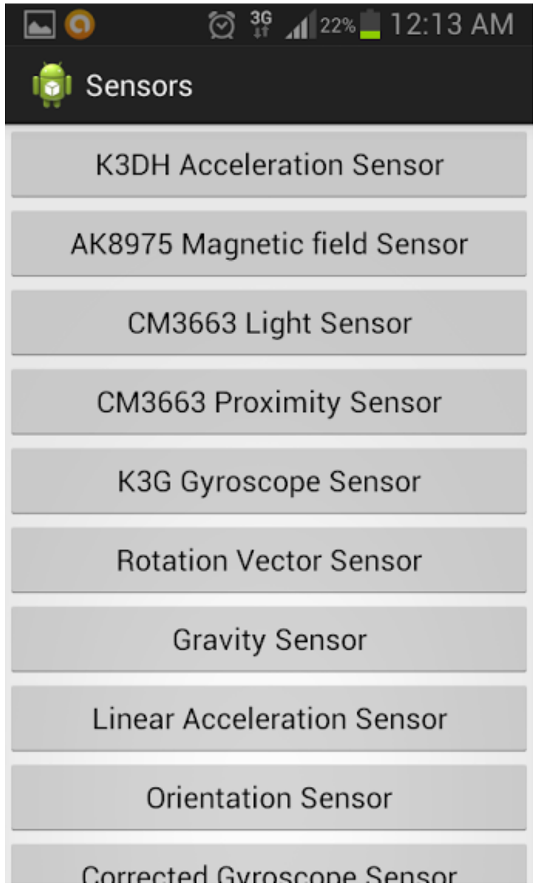
Switch between sensor events

```
public void onSensorChanged(SensorEvent event) {
    /*E.g., get the TextView defined in layout file for the Sensor Activity and print the time*/
        TextView timestamp = (TextView) findViewById(R.id.timestamp);
        timestamp.setText(String.valueOf("Timestamp: "+event.timestamp+" ns"));

        switch (event.sensor.getType()) {
        case Sensor.TYPE_ACCELEROMETER:
            /* Elabore and show the data received from the sensor */
            showEventData("Acceleration - gravity on axis", “m/s*s”,
                    event.values[0], event.values[1], event.values[2]);
            break;
        case Sensor.TYPE_MAGNETIC_FIELD:
            showEventData("Abient Magnetic Field", "uT", event.values[0],
                    event.values[1], event.values[2]);
            break;
        case Sensor.TYPE_GYROSCOPE:
            showEventData("Angular speed around axis", "radians/sec",
                    event.values[0], event.values[1], event.values[2]);
            break;
    ................
```

createGraph() function

```
public void createGraph(){
        graphView = new LineGraphView(   this , "SensorGraph"  );    }
      /*The value for the graph*/
      exampleSeries1 = new GraphViewSeries("X",null,new GraphViewData[] {   new GraphViewData(0, 0.0d)   });
      graphView.addSeries(exampleSeries1);
      graphView.setViewPort(1, 100);
      graphView.setScalable(true);
      graphView.setScrollable(true);
      graphView.setManualYAxisBounds((int)max,(int)min);
      graphView.getGraphViewStyle().setGridColor(Color.BLACK);
      graphView.getGraphViewStyle().setHorizontalLabelsColor(Color.BLACK);
      graphView.getGraphViewStyle().setVerticalLabelsColor(Color.BLACK);
      graphView.setShowLegend(true);
      graphView.setLegendAlign(LegendAlign.TOP);
      //graphView.setLegendWidth(200);
      graphView.getGraphViewStyle().setNumHorizontalLabels(5);
      graphView.getGraphViewStyle().setNumVerticalLabels(5);
       LinearLayout layout = (LinearLayout) findViewById(R.id.graph);
      layout.addView(graphView); }
```

Output


Sensors

K3DH Acceleration Sensor

AK8975 Magnetic field Sensor

CM3663 Light Sensor

CM3663 Proximity Sensor

K3G Gyroscope Sensor

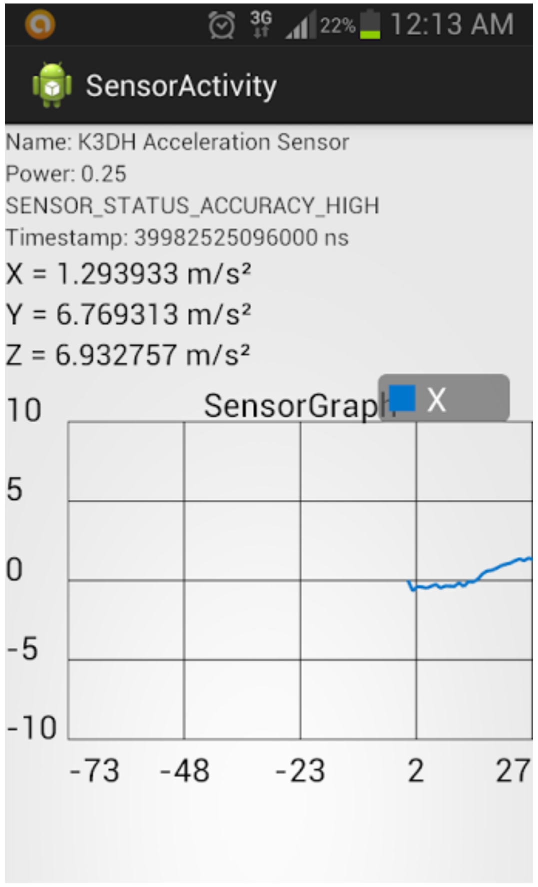
Rotation Vector Sensor

Gravity Sensor

Linear Acceleration Sensor

Orientation Sensor

Corrected Gyroscope Sensor


SensorActivity

```
Name: K3DH Acceleration Sensor
Power: 0.25
SENSOR_STATUS_ACCURACY_HIGH
Timestamp: 39982525096000 ns
X = 1.293933 m/s²
Y = 6.769313 m/s²
Z = 6.932757 m/s²
```