

Sensori

PROVA DI
ESONERO DEL
CORSO DI
MOBILE
PROGRAMMING

UNIVERSITÀ
DEGLI STUDI DI
ROMA TOR
VERGATA

Cosa vedremo

- Introduzione
 - Categorie
 - Tipi Supportati
 - Sistema di coordinate
- Sensor Framework
 - Identificare
 - Monitorare
 - Gestire
 - Best Practices
- Qualche applicazione

Introduzione

Categorie

- **Sensori di movimento**

Misurano le forze di accelerazione e rotazione sui 3 assi.

Include accelerometri, giroscopi e sensori di gravità.

- **Sensori ambientali**

Misurano parametri ambientali come luminosità, temperatura e umidità.

Include barometri, fotometri e termometri.

- **Sensori di posizione**

Misurano la posizione fisica di un dispositivo.

Include sensori di orientamento e magnetometri.

Introduzione

Tipi supportati

- **Sensori hardware**

Componenti fisici interni a un dispositivo. Misurano direttamente le proprietà dell'ambiente circostante, come accelerazione e campo geomagnetico.

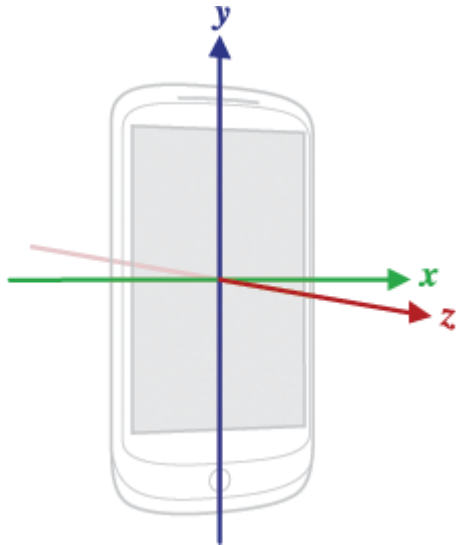
- **Sensori software (a.k.a. virtuali o sintetici)**

Si avvalgono delle misure sui dati acquisite da uno o più sensori hardware. Esempi di sensori virtuali sono il sensore di gravità (utilizza accelerometro e giroscopio) e il sensore di vettore di rotazione (utilizza accelerometro, magnetometro e, se presente, il giroscopio).

Introduzione

Sistema di coordinate

Dispositivi mobili



Autoveicoli



Le rotazioni positive sono in senso antiorario per ogni asse

Sensor framework

Consente l'accesso ai sensori e l'acquisizione dei dati misurati.
Fa parte del package `android.hardware` e include le seguenti classi e interfacce:

- **SensorManager**

Permette di creare un'istanza del sensor service e fornisce metodi per registrare un sensor event listener e accedere ai sensori. Fornisce inoltre diverse costanti per impostare il tempo di campionatura e per calibrare i sensori.

- **Sensor**

Permette di creare un'istanza di uno specifico sensore. Fornisce vari metodi per determinare le capacità di un sensore.

Sensor framework (2)

- **SensorEvent**

Permette di creare un oggetto **SensorEvent** per catturare gli eventi e contiene info su dati misurati, il tipo di sensore che ha generato l'evento, la precisione dei dati e il tempo di cattura.

- **SensorEventListener**

Interfaccia per metodi che ricevono la notifica dell'evento quando i valori misurati dal sensore o la precisione cambiano.

Sensor Framework

Identificare

```
private SensorManager sensorManager;
sensorManager = (SensorManager) getSystemService(Context.SENSOR_SERVICE);

// Ottenere tutti i sensori del dispositivo.
List<Sensor> deviceSensors = sensorManager.getSensorList(Sensor.TYPE_ALL);

if (sensorManager.getDefaultSensor(Sensor.TYPE_MAGNETIC_FIELD) != null){
    // Molto bene! C'è un magnetometro.
} else {
    // Accidenti! Il dispositivo non ha un magnetometro.
}
```


Sensor Framework

Identificare (2)

```
sensorManager.getSensorList( int TYPE )
```

```
sensorManager.getDefaultSensor(int TYPE)
```

| Sensor. | TIPO |
|---------------------------|---------|
| TYPE_ACCELEROMETER | HW |
| TYPE_AMBIENT_TEMPERATURE | HW |
| TYPE_GRAVITY | SW o HW |
| TYPE_GYROSCOPE | HW |
| TYPE_LIGHT | HW |
| TYPE_LINEAR_ACCELEROMETER | SF o HW |
| TYPE_MAGNETIC_FIELD | HW |
| TYPE_ORIENTATION | SF |
| TYPE_PRESSURE | HW |
| TYPE_PROXIMITY | HW |
| TYPE_RELATIVE_HUMIDITY | HW |
| TYPE_ROTATION_VECTOR | SF o HW |
| TYPE_TEMPERATURE | HW |

Sensor Framework

Monitorare

```
public class SensorActivity extends Activity implements SensorEventListener {
    private SensorManager sM;
    private Sensor mySensor;

    @Override
    public void onCreate(Bundle bundle) {
        ...
        sM = (SensorManager) getSystemService(Context.SENSOR_SERVICE);
        mySensor = sM.getDefaultSensor(Sensor.TYPE_LIGHT); ... }

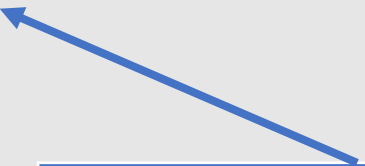
    @Override
    public void onAccuracyChanged(Sensor sensor, int accuracy) {
        // Fai qualcosa se cambia accuracy del sensore
    }

    @Override
    public void onSensorChanged(SensorEvent event) {
        // I sensori possono ritornare 1 o 3 valori
        float dataFromSensor = event.values[0];
    } ...
}
```

Sensor Framework

Monitorare (2)

```
...  
  
@Override  
protected void onResume() {  
    super.onResume();  
    // Inizia ad ascoltare il sensore  
    sm.registerListener(this, mySensor,  
                        SensorManager.SENSOR_DELAY_NORMAL);  
}  
  
@Override  
protected void onPause() {  
    super.onPause();  
    // Smetti di ascoltare il sensore  
    sm.unregisterListener(this);  
}  
}
```



| SensorManager. | microsec |
|----------------------|----------|
| SENSOR_DELAY_NORMAL | 200,000 |
| SENSOR_DELAY_UI | 60,000 |
| SENSOR_DELAY_GAME | 20,000 |
| SENSOR_DELAY_FASTEST | 0 |

Sensor Framework

Gestire

- Verificare che esista il sensore da usare

✓ vedi slide [8]

- Nel caso servissero più sensori di tipi diversi

✓ nel metodo `onSensorChanged()` filtrare le diverse letture con un `if` :

```
public void onSensorChanged(SensorEvent event) {  
    // se i dati letti provengono dall'accelerometro...  
    if (event.sensor.getType() == Sensor.TYPE_ACCELEROMETER)  
        // fai qualcosa.  
    ... }  
}
```

- Filtrare i dispositivi che non hanno i sensori utilizzati dall'app

✓ nel Manifest:

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

Sensor Framework

Best Practices

- Raccogliere informazioni nel foreground

Con le versioni più recenti di Android, alcuni sensori non ricevono eventi in background.

- Annullare la registrazione di un sensor listener

Nel metodo `onPause()` .

- Verificare la disponibilità sensori

Non è detto che un qualsiasi dispositivo abbia il sensore che vogliamo utilizzare ⇒ inserire `uses-feature` nel manifest.

- Testare i sensori con l'Android Emulator

Sensor Framework

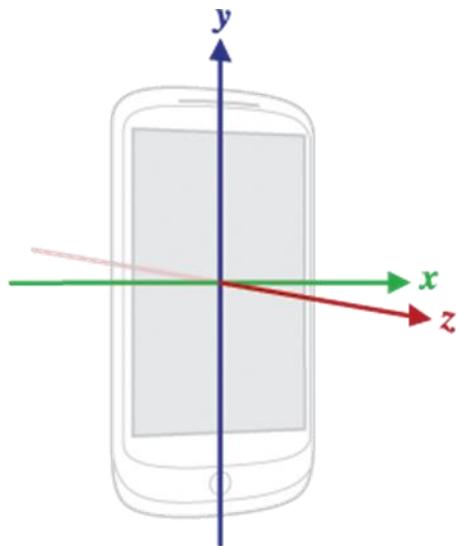
Best Practices (2)

- **Evitare di usare metodi e tipi di sensore deprecated**
Cambiamenti importanti (es.: Android 4.0 [API level 14])
- **Non bloccare il metodo `sensorOnChanged()`**
Il sistema chiama questo metodo ogni volta che i dati del sensore cambiano (molto frequentemente), per cui è bene evitare di appesantire questo metodo per non bloccarlo.
- **Scegliere il tempo di campionatura attentamente**
Per non sprecare risorse ed energia della batteria.

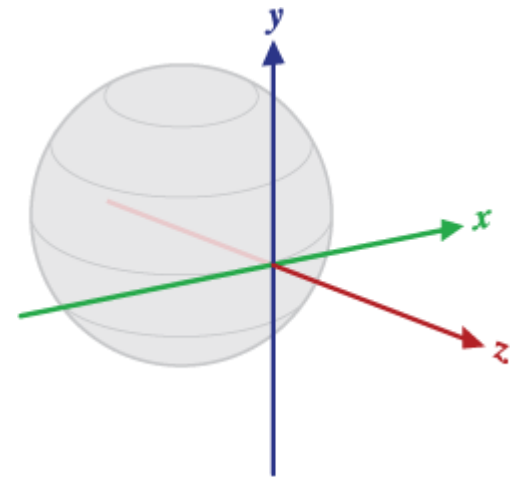
Qualche applicazione

Bussola

1. Registrare e leggere Accelerometro e Magnetometro.
2. Calcolare la Matrice di Rotazione.
3. Calcolare il Vettore Orientamento del dispositivo.
4. Leggere l'Azimuth dal Vettore Orientamento.



Azimuth rappresenta l'angolo tra l'asse y del dispositivo e il Polo Nord magnetico



Qualche applicazione

Bussola (2)

2. Calcolare la Matrice di Rotazione.

```
public static boolean getRotationMatrix (float[] R,  
                                         float[] I,  
                                         float[] accelerometerReading,  
                                         float[] magnetometerReading)
```

Calcola i valori per le matrici **R** e **I** in input, utilizzando le letture di Accelerometro e Magnetometro.

R è la Matrice di Rotazione. Questa è uguale alla matrice identità quando il sistema di coordinate del dispositivo coincide con quello del mondo.

Qualche applicazione

Bussola (3)

3. Calcolare la Matrice di Orientamento del dispositivo.

```
public static float[] getOrientation (float[] R,  
                                     float[] values)
```

Calcola l'orientamento del dispositivo (i.e. `values[]`) basandosi sulla matrice di rotazione `R`.

4. Leggere l'Azimuth dal Vettore Orientamento.

```
float azimuth = values[0];
```

Il primo valore del vettore `values[]` è l'Azimuth. Il secondo e il terzo sono rispettivamente *Pitch* e *Roll*.

Qualche applicazione Chiamata

Uno dei principali utilizzi di un *sensore di prossimità* in un dispositivo mobile è quello di evitare azioni indesiderate a causa della pressione sullo schermo con un tocco accidentale, ad esempio con l'orecchio mentre si è in una chiamata.

Una possibile applicazione che implementa questo comportamento può fare uso della classe `PowerManager` per controllare l'energia del dispositivo creando un oggetto `PowerManager.wakeLock`.

```
...  
pm = (PowerManager) getSystemService(Context.POWER_SERVICE);  
mWakeLock =  
    pm.newWakeLock(PowerManager.PROXIMITY_SCREEN_OFF_WAKE_LOCK,  
        "Call:incall");
```

Qualche applicazione

Chiamata (2)

`PROXIMITY_SCREEN_OFF_WAKE_LOCK` = costante che permette di spegnere lo schermo quando il sensore di prossimità si attiva.

Nel metodo `sensorOnChanged()` si chiama la `wakeLock.acquire()` per acquisire la wakeLock con il comportamento definito dalla costante. Si è fatto uso di 2 toggle switch per simulare chiamata e vivavoce.

```
@Override
public final void onSensorChanged(SensorEvent event) {
    if (tgl_callSwitch.isChecked() &&
        !tgl_vivavoce.isChecked()) {
        float distance = event.values[0];
        if (!mWakeLock.isHeld() &&
            distance < proximitySensor.getMaximumRange())
            mWakeLock.acquire();
    }
}
```

Qualche applicazione

Chiamata (3)

Come consuetudine si annulla la registrazione del sensor listener nella `onPause()` e si chiama la `wakeLock.release()` per rilasciare la `wakeLock`.

Per evitare crash è bene verificare prima se una `wakeLock` è già stata acquisita ma non rilasciata con il metodo `wakeLock.isHeld()`.

```
@Override
protected void onPause() {
    super.onPause();
    ...
    sensorManager.unregisterListener(this);
    if (mWakeLock.isHeld())
        mWakeLock.release();
}
```

Qualche applicazione Chiamata (4)

N.B.: Per poter utilizzare un oggetto `WakeLock` un'applicazione deve richiedere il permesso in una `<uses-permission>` del manifest.

```
<uses-permission android:name="android.permission.WAKE_LOCK"/>
```

Di seguito è mostrata la `onResume()` dove si registra un listener per il sensore di prossimità:

```
@Override
protected void onResume() {
    super.onResume();
    sensorManager.registerListener(this, proximitySensor,
        SensorManager.SENSOR_DELAY_NORMAL);
    ...
}
```

○ Introduzione

- Categorie
- Tipi Supportati
- Sistema di coordinate

○ Sensor Framework

- Identificare
- Monitorare
- Gestire
- Best Practices

○ Qualche applicazione

Cosa
abbiamo
visto