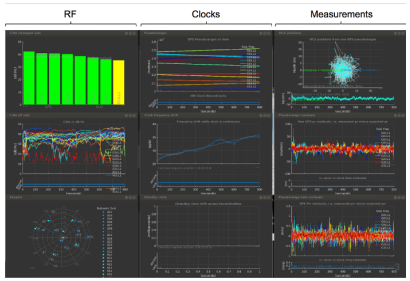
PMDM10.- Sensores.

Caso práctico



Google Developers (Uso educativo nc)

De nuevo, el equipo se enfrenta a un nuevo reto a partir de las necesidades concretas de un cliente de BK Programación que se dedica a la venta de bicicletas.

- Tenemos una petición de un cliente que desea desarrollar una aplicación Android para los compradores de sus bicicletas -informa Ada a su equipo, que espera expectante su nuevo reto-. El objetivo principal de la aplicación es enviar mensajes publicitarios, cupones de descuento y otras ofertas para fidelizar a sus clientes.

- Pero -señala María, siempre muy atenta a las explicaciones de Ada-, para que la aplicación sea instalada por los usuarios, esta debe ofrecer algún servicio.

- Buena puntualización -continúa explicando Ada-. En este caso se desea ofrecer un conjunto de valores relacionados con una actividad deportiva, mostrando la velocidad a la que se desplazan los ciclistas, la distancia recorrida, la temperatura actual, etc. y otros parámetros calculados como las calorías quemadas durante el ejercicio, la velocidad media durante la actividad, etc.

- Vamos a investigar qué tipo de sensores tenemos a nuestro alcance y cómo manejarlos -anima Pedro al equipo.

El equipo descubre que los dispositivos Android incluyen un conjunto importante de sensores bastante precisos capaces de capturar información muy interesante para desarrollar este tipo de aplicaciones. Aprender a manejarlos es el reto que tienen por delante para ofrecer a los usuarios datos calculados a partir de la información real capturada desde el dispositivo.

Materiales formativos de FP Online propiedad del Ministerio de Educación y Formación Profesional.

Aviso Legal

1.- Tipos de sensores.

Existen sensores basados en hardware y sensores basados en software:

- Los **sensores basados en hardware** son componentes físicos integrados en el dispositivo. Derivan sus datos midiendo directamente las propiedades ambientales específicas, como la aceleración, la intensidad del campo geomagnético o el cambio angular.
- Los **sensores basados en software** no son dispositivos físicos, aunque imitan a los sensores basados en hardware. Los sensores basados en software derivan sus datos de uno o más de los sensores basados en hardware y, a veces, se los denomina sensores virtuales o sensores sintéticos. El sensor de aceleración lineal y el sensor de gravedad son ejemplos de sensores basados en software.

Stanley Ng (Pexels)

Pocos dispositivos con Android tienen todos los sensores especificados en la tabla que son los compatibles con Android. Por ejemplo, la mayoría de los dispositivos y tabletas tienen un acelerómetro y un magnetómetro, pero pocos dispositivos tienen barómetros o termómetros. Además, un dispositivo puede tener más de un sensor de un tipo determinado. Por ejemplo, un dispositivo puede tener dos sensores de gravedad, cada uno con un rango diferente.

Tipos de sensores compatibles con la plataforma Android

Sensor	Tipo	Descripción	Usos comunes
`TYPE_ACCELEROMETER`	Hardware	Mide, en m/s^2, la fuerza de aceleración que se aplica a un dispositivo en los tres ejes físicos (x, y, z), incluida la fuerza de gravedad.	Detección de movimiento (agitación, inclinación, etc.).
`TYPE_AMBIENT_TEMPERATURE`	Hardware	Mide la temperatura ambiente de la habitación en grados Celsius (°C). Consulta la siguiente nota.	Supervisión de la temperatura del aire.
`TYPE_GRAVITY`	Software o hardware	Mide, en m/s^2, la fuerza de gravedad que se aplica a un dispositivo en los tres ejes físicos (x, y, z).	Detección de movimiento (agitación, inclinación, etc.).

TYPE_GYROSCOPE	Hardware	Mide, en rad/s, la velocidad de rotación de un dispositivo alrededor de cada uno de los tres ejes físicos (x, y, z).	Detección de rotación (agitación, giro, etc.).
TYPE_LIGHT	Hardware	Mide el nivel de luz ambiental (iluminación) en lx.	Control del brillo de la pantalla.
TYPE_LINEAR_ACCELERATION	Software o hardware	Mide, en m/s^2, la fuerza de aceleración que se aplica a un dispositivo en los tres ejes físicos (x, y, z), excluyendo la fuerza de gravedad.	Supervisión de la aceleración a lo largo de un solo eje.
TYPE_MAGNETIC_FIELD	Hardware	Mide el campo geomagnético ambiental de los tres ejes físicos (x, y, z) en μT.	Creación de una brújula.
TYPE_ORIENTATION	Software	Mide los grados de rotación de un dispositivo alrededor de los tres ejes físicos (x, y, z). A partir de la API nivel 3, puedes obtener las matrices de inclinación y de rotación mediante el uso de los sensores de gravedad y de campo geomagnético del dispositivo, usando el sensor de gravedad junto con el método getRotationMatrix() .	Determinación de la posición del dispositivo.
TYPE_PRESSURE	Hardware	Mide la presión del aire del ambiente en hPa o mbar.	Supervisa los cambios de la presión del aire.
TYPE_PROXIMITY	Hardware	Mide, en cm, la proximidad de un objeto con respecto a la pantalla de visualización de un dispositivo. Este sensor en general se usa para determinar si un dispositivo manual se está sosteniendo cerca del oído de una persona.	Posición del teléfono durante una llamada.
TYPE_RELATIVE_HUMIDITY	Hardware	Mide en valor de porcentaje (%) la humedad relativa del ambiente.	Supervisa el punto de condensación, la humedad absoluta y la humedad relativa.
TYPE_ROTATION_VECTOR	Software o hardware	Mide la orientación de un dispositivo mediante los tres elementos del vector de rotación del dispositivo.	Detección de movimiento y detección de rotación.

`TYPE_TEMPERATURE`	Hardware	Mide la temperatura del dispositivo en grados Celsius (°C). La implementación de este sensor varía según el dispositivo; en la API nivel 14 se reemplazó por el sensor `TYPE_AMBIENT_TEMPERATURE` .	Supervisión de temperaturas.

2.- Framework o entorno de trabajo del sensor.

Podemos acceder a estos sensores y adquirir datos sin procesar del sensor utilizando el *framework* de sensores de Android. El *framework* del sensor es parte del paquete **android.hardware** e incluye las siguientes clases e interfaces:

- **SensorManager**: Podemos usar esta clase para crear una instancia del servicio del sensor. Esta clase proporciona varios métodos para acceder y enumerar sensores, registrar y anular el registro de receptores de eventos de sensores y adquirir información de orientación. Esta clase también proporciona varias constantes de sensor que se utilizan para informar la precisión del sensor, establecer velocidades de adquisición de datos y calibrar sensores.
- **Sensor**: Podemos usar esta clase para crear una instancia de un sensor específico. Esta clase proporciona varios métodos que nos permiten determinar las capacidades de un sensor.
- **SensorEvent**: El sistema usa esta clase para crear un objeto de evento sensor, que proporciona información sobre un evento sensor. Un objeto de evento de sensor incluye la siguiente información:
  - los datos del sensor en bruto,
  - el tipo de sensor que generó el evento,
  - la precisión de los datos
  - y la marca de tiempo del evento.
- **SensorEventListener**: Podemos usar esta interfaz para crear dos métodos de devolución de llamada que reciben notificaciones (eventos del sensor) cuando cambian los valores del sensor o cuando cambia la precisión del sensor.

En una aplicación típica, utilizaremos estas API relacionadas con el sensor para realizar dos tareas básicas:

- **Identificación de sensores y capacidades del sensor**

  La identificación de sensores y capacidades del sensor en el tiempo de ejecución es útil si su aplicación tiene características que dependen de capacidades o tipos de sensores específicos. Por ejemplo, es posible que desee identificar todos los sensores que están presentes en un dispositivo y desactivar las funciones de la aplicación que dependen de sensores que no están presentes. Del mismo modo, es posible que desee identificar todos los sensores de un tipo determinado para que pueda elegir la implementación del sensor que tenga el rendimiento óptimo para su aplicación.

- **Control de los eventos del sensor**

  La monitorización de los eventos del sensor es la forma de adquirir los datos del sensor sin procesar. Se produce un evento de [illegible]ensor detecta un cambio en los parámetros que está midiendo. Un evento de sensor le proporciona

cuatro elementos de información: el nombre del sensor que activó el evento, la marca de tiempo del evento, la precisión del evento y los datos brutos del sensor que desencadenaron el evento.

3.- Disponibilidad del sensor.

Si bien la disponibilidad del sensor varía de un dispositivo a otro, también puede variar entre las versiones de Android. Esto se debe a que los sensores de Android se han introducido a lo largo de varias versiones de la plataforma. Por ejemplo, se introdujeron muchos sensores en Android 1.5 (nivel 3 de la API), pero algunos no se implementaron y no estuvieron disponibles para su uso hasta Android 2.3 (API de nivel 9).

Asimismo, se introdujeron varios sensores en Android 2.3 (Nivel API 9) y Android 4.0 (Nivel API 14). Actualmente, prácticamente todos los dispositivos incluyen estos sensores, aunque puedes buscar aquellos permisos peligrosos en esta página de referencia de Disponibilidad de los sensores

Dos sensores (**TYPE_ORIENTATION** y **TYPE_TEMPERATURE**) han quedado en desuso y reemplazados por sensores nuevos y mejores.

4.- Identificación de sensores.

El marco del sensor de Android proporciona varios métodos que le permiten determinar en tiempo de ejecución qué sensores están en un dispositivo. La API también proporciona métodos que le permiten determinar las capacidades de cada sensor, como su rango máximo, su resolución y sus requisitos de potencia.

Para identificar los sensores que están en un dispositivo, primero necesitamos obtener una referencia al servicio del sensor. Para hacerlo, crearemos una instancia de la clase **SensorManager** llamando al método **getSystemService()** y pasando el argumento **SENSOR_SERVICE**.

Ejemplo

```
1  private SensorManager mSensorManager;
2  ...
3  mSensorManager = (SensorManager) getSystemService (Context.SENSOR_SERVICE);
```

A continuación, podemos obtener una lista de cada sensor en un dispositivo llamando al método getSensorList() y utilizando la constante TYPE_ALL.

Ejemplo

```
sors = mSensorManager.getSensorList (Sensor.TYPE_ALL);
```

Si deseamos enumerar todos los sensores de un tipo determinado, podemos usar otra constante en lugar de **TYPE_ALL** como **TYPE_GYROSCOPE**, **TYPE_LINEAR_ACCELERATION** o **TYPE_GRAVITY**.

También podemos determinar si existe un tipo específico de sensor en un dispositivo utilizando el método **getDefaultSensor()** y pasando la constante de tipo para un sensor específico. Si un dispositivo tiene más de un sensor de un tipo determinado, uno de los sensores debe designarse como el sensor predeterminado. Si no existe un sensor predeterminado para un tipo determinado de sensor, la llamada al método devuelve nulo, lo que significa que el dispositivo no tiene ese tipo de sensor.

Ejemplo

El siguiente código verifica si hay un magnetómetro en un dispositivo:

```
private SensorManager mSensorManager;
...
mSensorManager = (SensorManager) getSystemService (Context.SENSOR_SERVICE);
if (mSensorManager.getDefaultSensor (Sensor.TYPE_MAGNETIC_FIELD)! = null) {
  // ¡Éxito!  Hay un magnetómetro.
}
else {
  // ¡Fallo!  Sin magnetómetro
}
```

Android no requiere que los fabricantes de dispositivos creen ningún tipo particular de sensores en sus dispositivos, por lo que los dispositivos pueden tener una amplia gama de configuraciones de sensores.

Otro método útil es el método **getMinDelay()**, que devuelve el intervalo de tiempo mínimo (en microsegundos) que un sensor puede usar para detectar los datos. Cualquier sensor que devuelve un valor distinto de cero para el método **getMinDelay()** es un sensor de transmisión. Los sensores de transmisión por secuencias detectan los datos a intervalos regulares y se introdujeron en Android 2.3 (nivel

de API 9). Si un sensor devuelve cero cuando llama al método **getMinDelay()**, significa que el sensor no es un sensor de transmisión porque informa los datos solo cuando hay un cambio en los parámetros que está detectando.

El método **getMinDelay()** es útil porque nos permite determinar la velocidad máxima a la que un sensor puede adquirir datos. Si ciertas funciones en su aplicación requieren altas tasas de adquisición de datos o un sensor de transmisión, puede utilizar este método para determinar si un sensor cumple con esos requisitos y luego habilitar o deshabilitar, en consecuencia, las características relevantes en su aplicación.

Precaución: la velocidad máxima de adquisición de datos de un sensor no es necesariamente la velocidad a la que el marco del sensor entrega los datos del sensor a su aplicación. El marco del sensor informa los datos a través de los eventos del sensor, y varios factores influyen en la velocidad a la que su aplicación recibe los eventos del sensor.

Autoevaluación

Para obtener una lista de todos los sensores de un dispositivo utilizamos la constante:

- ○ TYPE_GRAVITY
- ○ TYPE_ALL
- ○ TYPE_GYROSCOPE
- ○ TYPE_LINEAR_ACCELERATION

No es correcto, vuelve a leer el apartado.

Correcto.

No es correcto, repasa el texto de nuevo

No es correcto, esta constante proporciona otra información.

Solución

1. Incorrecto
2. Opción correcta
3. Incorrecto
4. Incorrecto

5.- Monitorización de eventos del sensor.

Para monitorizar datos de sensores necesitamos implementar dos métodos contemplados en la interfaz **SensorEventListener**, que serán **onAccuracyChanged():onAccuracyChanged()** y **onSensorChanged()**. El sistema Android llama a estos métodos siempre que ocurran ciertas circunstancias.

La precisión de un sensor cambia.

En este caso, el sistema invoca el método **onAccuracyChanged()**, que le proporciona una referencia al objeto Sensor que cambió y la nueva precisión del sensor. La precisión está representada por una de las cuatro constantes de estado: SENSOR_STATUS_ACCURACY_LOW, SENSOR_STATUS_ACCURACY_MEDIUM, SENSOR_STATUS_ACCURACY_HIGH o SENSOR_STATUS_UNRELIABLE.

Un sensor informa un nuevo valor.

En este caso, el sistema invoca el método **onSensorChanged()**, que le proporciona un objeto **SensorEvent**. Un objeto **SensorEvent** contiene información sobre los nuevos datos del sensor, que incluyen: la precisión de los datos, el sensor que generó los datos, la marca de tiempo en la que se generaron los datos y los nuevos datos que el sensor registró.

Ejemplo

El siguiente código muestra cómo usar el método **onSensorChanged()** para controlar los datos del sensor de luz. Este ejemplo muestra los datos del sensor en un *TextView* que está definido en el archivo **main.xml** como **sensor_data**.

```
1  public class SensorActivity extends Activity implements SensorEventListener {
2      private SensorManager mSensorManager;
3
4
```

```
    @Override
    public final void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.main);
      mSensorManager = (SensorManager) getSystemService(Context.SENSOR_SERVICE);
      mLight = mSensorManager.getDefaultSensor(Sensor.TYPE_LIGHT);
    }

    @Override
    public final void onAccuracyChanged(Sensor sensor, int accuracy) {
      // Do something here if sensor accuracy changes.
    }

    @Override
    public final void onSensorChanged(SensorEvent event) {
      // The light sensor returns a single value.
      // Many sensors return 3 values, one for each axis.
      float lux = event.values[0];
      // Do something with this sensor value.
    }

    @Override
    protected void onResume() {
      super.onResume();
      mSensorManager.registerListener(this, mLight, SensorManager.SENSOR_DELAY_NORMAL);
    }

    @Override
    protected void onPause() {
      super.onPause();
      mSensorManager.unregisterListener(this);
    }
}
```


atos predeterminado (**SENSOR_DELAY_NORMAL**) se especifica cuando se invoca el método **registerListener()**. El retardo de datos (o tasa de muestreo) controla el intervalo en el cual los eventos del sensor se envían a su aplicación

a través del método de devolución de llamada **onSensorChanged().** La demora de datos predeterminada es adecuada para monitorear cambios típicos de orientación de pantalla y utiliza un retraso de 200.000 microsegundos. Podemos especificar otros retrasos de datos, como **SENSOR_DELAY_GAME** (retardo de 20.000 microsegundos), **SENSOR_DELAY_UI** (retardo de 60.000 microsegundos) o **SENSOR_DELAY_FASTEST** (retardo de 0 microsegundos). A partir de Android 3.0 (API nivel 11) también podemos especificar el retraso como un valor absoluto (en microsegundos).

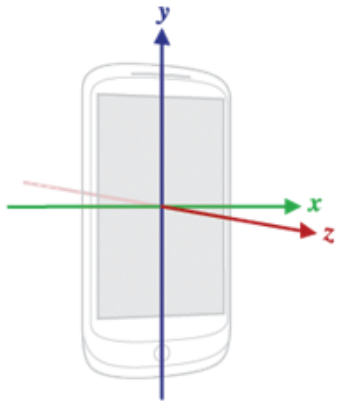
La demora que especifiquemos solo es un retraso sugerido. El sistema Android y otras aplicaciones pueden alterar este retraso. Como práctica recomendada, debemos especificar la demora más grande que podamos porque el sistema generalmente usa un retraso menor que el especificado por nosotros (es decir, debemos elegir la frecuencia de muestreo más lenta que aún satisfaga las necesidades de nuestra aplicación). Usar un retraso mayor impone una carga menor en el procesador y, por lo tanto, utilizará menos energía.

No hay un método público para determinar la velocidad a la que el framework del sensor está enviando eventos del sensor a nuestra aplicación; sin embargo, podemos usar las marcas de tiempo que están asociadas con cada evento del sensor para calcular la tasa de muestreo en varios eventos.

También es importante tener en cuenta que este ejemplo utiliza los métodos de devolución de llamada **onResume()** y **onPause()** para registrar y anular el registro del detector de eventos del sensor. Como práctica recomendada, siempre debemos desactivar los sensores que no necesitamos, especialmente cuando su actividad está en pausa. Si no lo hacemos, agotaremos la batería en unas pocas horas debido a que algunos sensores tienen requisitos de energía considerables y pueden agotar rápidamente la energía de la batería. El sistema no desactivará los sensores automáticamente cuando la pantalla se apaga.

6.- Sistema de coordenadas del sensor.

En general, el marco del sensor utiliza un sistema de coordenadas de tres ejes estándar para expresar valores de datos. Para la mayoría de los sensores, el sistema de coordenadas se define en relación con la pantalla del dispositivo cuando el dispositivo se mantiene en su orientación predeterminada (consulte la figura). Cuando un dispositivo se mantiene en su orientación predeterminada, el eje X es horizontal y apunta hacia la derecha, el eje Y es vertical y apunta hacia arriba, y el eje Z apunta hacia el exterior de la pantalla. En este sistema, las coordenadas detrás de la pantalla tienen valores Z negativos. Este sistema de coordenadas es utilizado por los siguientes sensores:



Google Developers (Uso educativo nc)

- Sensor de aceleración
- Sensor de gravedad
- Giroscopio
- Sensor de aceleración lineal
- Sensor de campo geomagnético

El punto más importante para entender sobre este sistema de coordenadas es que los ejes no se intercambian cuando cambia la orientación de la pantalla del dispositivo; es decir, el sistema de coordenadas del sensor nunca cambia a medida que el dispositivo se mueve. Este comportamiento es el mismo que el del sistema de coordenadas OpenGL.

Otro punto que debemos entender es que nuestra aplicación no debe asumir que la orientación natural (por defecto) de un dispositivo es vertical. La orientación natural para muchos dispositivos de tableta es el paisaje. Y el sistema de coordenadas del sensor siempre se basa en la orientación natural de un dispositivo.

Finalmente, si nuestra aplicación relaciona los datos del sensor con la visualización en pantalla, debemos usar el método **getRotation()** para determinar la rotación de la pantalla y luego usar el método **remapCoordinateSystem()** para mapear las coordenadas del sensor a las coordenadas de la pantalla. Debemos hacer esto incluso si nuestro fichero **AndroidManifest.xml** especifica la visualización solo retrato.

7.- Sensor de gravedad.

El sensor de gravedad proporciona un vector tridimensional que indica la dirección y la magnitud de la gravedad. Normalmente, este sensor se usa para determinar la orientación relativa del dispositivo en el espacio.

El siguiente código muestra cómo obtener una instancia del sensor de gravedad predeterminado:

Código

```
1  private SensorManager mSensorManager;
2  private Sensor mSensor;
3   ...
4  mSensorManager = (SensorManager) getSystemService(Context.SENSOR_SERVICE);
5  mSensor = mSensorManager.getDefaultSensor(Sensor.TYPE_GRAVITY);
```

Las unidades son las mismas que las utilizadas por el sensor de aceleración (m/s^2), y el sistema de coordenadas es el mismo que el usado por el sensor de aceleración.

Cuando un dispositivo está en reposo, la salida del sensor de gravedad debe ser idéntica a la del acelerómetro.

8.- Sensor de acelaración o acelerómetro.

Un sensor de aceleración mide la aceleración aplicada al dispositivo, incluida la fuerza de la gravedad. El siguiente código muestra cómo obtener una instancia del sensor de aceleración predeterminado:

Código

```
private SensorManager mSensorManager;
 sensor privado mSensor;
  ...
 mSensorManager = (SensorManager) getSystemService (Context.SENSOR_SERVICE);
 mSensor = mSensorManager.getDefaultSensor (Sensor.TYPE_ACCELEROMETER);
```

Conceptualmente, un sensor de aceleración determina la aceleración que se aplica a un dispositivo (A_D) al medir las fuerzas que se aplican al sensor en sí (F_S) usando la siguiente relación:

$$A_D=-\left (\displaystyle \frac{1}{mass} \right)\sum F_S$$

Sin embargo, la fuerza de la gravedad siempre influye en la aceleración medida de acuerdo con la siguiente relación:

$$A_D=-g-\left (\displaystyle \frac{1}{mass} \right)\sum F_S$$

Por esta razón, cuando el dispositivo está apoyado sobre una mesa (y no está acelerando), el acelerómetro muestra una magnitud de g = 9.81~m/s^2. De manera similar, cuando el dispositivo está en caída libre y por lo tanto acelera rápidamente hacia el suelo a 9.81~m/s^2, su acelerómetro lee una magnitud de g = 0 m/s^2. Por lo tanto, para medir la aceleración real del dispositivo, la contribución de la fuerza de gravedad debe eliminarse de los datos del acelerómetro. Esto se puede lograr aplicando un filtro de paso alto. Por el contrario, se puede usar un filtro de paso bajo para aislar la fuerza de la gravedad.

Ejemplo

Uso de filtro de paso bajo para aislar la fuerza de la gravedad:

```
public void onSensorChanged(SensorEvent event){

   // In this example, alpha is calculated as t / (t + dT),
   // where t is the low-pass filter's time-constant and
   // dT is the event delivery rate.

   final float alpha = 0.8;

   // Isolate the force of gravity with the low-pass filter.
   gravity[0] = alpha * gravity[0] + (1 - alpha) * event.values[0];
   gravity[1] = alpha * gravity[1] + (1 - alpha) * event.values[1];
   gravity[2] = alpha * gravity[2] + (1 - alpha) * event.values[2];

   // Remove the gravity contribution with the high-pass filter.
   linear_acceleration[0] = event.values[0] - gravity[0];
   linear_acceleration[1] = event.values[1] - gravity[1];
   linear_acceleration[2] = event.values[2] - gravity[2];

}
```

Podemos usar muchas técnicas diferentes para filtrar los datos del sensor. El ejemplo de código anterior usa una constante de filtro simple (alfa) para crear un filtro de paso bajo. Esta constante de filtro se deriva de una constante de tiempo (t), que es una representación aproximada de la latencia que el filtro agrega a los eventos del sensor, y la tasa de entrega de eventos del sensor (dt). El ejemplo de código usa un valor alfa de 0.8 para fines de demostración. Si usa este método de filtrado, puede que necesite elegir un valor alfa diferente.

Los acelerómetros usan el sistema de coordenadas del sensor estándar. En la práctica, esto significa que las siguientes condiciones se aplican cuando un dispositivo está tumbado sobre una mesa en su orientación natural:

- Si empujamos el dispositivo por la izquierda (para que se mueva hacia la derecha), el valor de la aceleración "x" es positivo.

- Si empujamos el dispositivo por abajo (para que se aleje de nosotros), el valor de aceleración "y" es positivo.
- Si empujamos el dispositivo hacia el cielo con una aceleración de A~m/s^2, el valor de aceleración "z" es A+9.81, que corresponde a la aceleración del dispositivo (+A~m/s^2) menos la fuerza de la gravedad (-9.81~m/s^2).
- El dispositivo estacionario o parado tendrá un valor de aceleración en "z" de +9.81, que corresponde a la aceleración del dispositivo (0~m/s^2 menos la fuerza de la gravedad, que es -9.81~m/s^2).

En general, el acelerómetro es un buen sensor para usar si monitorizamos el movimiento del dispositivo. Casi todos los teléfonos y tabletas con Android tienen un acelerómetro, y usan aproximadamente 10 veces menos energía que los otros sensores de movimiento. Una desventaja es que podríamos tener que implementar filtros de paso bajo y paso alto para eliminar las fuerzas gravitatorias y reducir el ruido.

9.- Giroscopio.

El giroscopio mide la velocidad de rotación en rad/s (radianes por segundo) alrededor del eje x, y y z de un dispositivo. El siguiente código muestra cómo obtener una instancia del giroscopio predeterminado:

Código

```
1 private SensorManager mSensorManager;
2 private Sensor mSensor;
3  ...
4  mSensorManager = (SensorManager) getSystemService(Context.SENSOR_SERVICE);
5  mSensor = mSensorManager.getDefaultSensor(Sensor.TYPE_GYROSCOPE);
```

El sistema de coordenadas del sensor es el mismo que el utilizado para el sensor de aceleración. La rotación es positiva en el sentido contrario a las agujas del reloj; es decir, un observador que mira desde una ubicación positiva en los ejes x, y o z en un dispositivo colocado en el origen informaría la rotación positiva si el dispositivo parecía estar girando en sentido contrario a las agujas del reloj. Esta es la definición matemática estándar de rotación positiva y no es lo mismo que la definición de balanceo que utiliza el sensor de orientación.

Normalmente, la salida del giroscopio se integra a lo largo del tiempo para calcular una rotación que describe el cambio de ángulos sobre el paso de tiempo.

Ejemplo

```
// Create a constant to convert nanoseconds to seconds.
private static final float NS2S = 1.0f / 1000000000.0f;
private final float[] deltaRotationVector = new float[4]();
private float timestamp;

public void onSensorChanged(SensorEvent event) {
  // This timestep's delta rotation to be multiplied by the current rotation
  // after computing it from the gyro sample data.
  if (timestamp != 0) {
    final float dT = (event.timestamp - timestamp) * NS2S;
    // Axis of the rotation sample, not normalized yet.
    float axisX = event.values[0];
    float axisY = event.values[1];
    float axisZ = event.values[2];

    // Calculate the angular speed of the sample
    float omegaMagnitude = sqrt(axisX*axisX + axisY*axisY + axisZ*axisZ);

    // Normalize the rotation vector if it's big enough to get the axis
    // (that is, EPSILON should represent your maximum allowable margin of error)
    if (omegaMagnitude > EPSILON) {
      axisX /= omegaMagnitude;
      axisY /= omegaMagnitude;
      axisZ /= omegaMagnitude;
      }

    // Integrate around this axis with the angular speed by the timestep
    // in order to get a delta rotation from this sample over the timestep
    // We will convert this axis-angle representation of the delta rotation
    // into a quaternion before turning it into the rotation matrix.
    float thetaOverTwo = omegaMagnitude * dT / 2.0f;
    float sinThetaOverTwo = sin(thetaOverTwo);
    float cosThetaOverTwo = cos(thetaOverTwo);
    deltaRotationVector[0] = sinThetaOverTwo * axisX;
    deltaRotationVector[1] = sinThetaOverTwo * axisY;
    deltaRotationVector[2] = sinThetaOverTwo * axisZ;
```

```
    deltaRotationVector[3] = cosThetaOverTwo;
  }

  timestamp = event.timestamp;
  float[] deltaRotationMatrix = new float[9];
  SensorManager.getRotationMatrixFromVector(deltaRotationMatrix, deltaRotationVector);
  // User code should concatenate the delta rotation we computed with the current rotation
  // in order to get the updated rotation.
  // rotationCurrent = rotationCurrent * deltaRotationMatrix;
  }
```

Los giroscopios estándar brindan datos rotativos brutos sin ningún filtrado o corrección por ruido y deriva (sesgo). En la práctica, el ruido y la deriva del giroscopio introducirán errores que deben ser compensados.

Autoevaluación

El sensor de aceleración...

- ○ es un buen sensor para usar si monitorizamos el movimiento del dispositivo.
- ○ se usa para determinar la orientación relativa del dispositivo en el espacio.
- ○ mide la velocidad de rotación.

Opción correcta

Incorrecto

Solución

1. Opción correcta
2. Incorrecto
3. Incorrecto

Autoevaluación

El sensor de gravedad...

- ○ es un buen sensor para usar si monitorizamos el movimiento del dispositivo.
- ○ se usa para determinar la orientación relativa del dispositivo en el espacio.
- ○ mide la velocidad de rotación.

Incorrecto

Opción correcta

Incorrecto

Solución

1. Incorrecto
2. Opción correcta
3. Incorrecto

10.- Sensor de aceleración lineal.

El sensor de aceleración lineal le proporciona un vector tridimensional que representa la aceleración a lo largo de cada eje del dispositivo, excluyendo la gravedad. Puede usar este valor para realizar la detección de gestos. El valor también puede servir como entrada para un sistema de navegación inercial, que utiliza la navegación a estima. El siguiente código muestra cómo obtener una instancia del sensor de aceleración lineal predeterminado:

Código

```
1 private SensorManager mSensorManager;
2 private Sensor mSensor;
3 ...
4 mSensorManager = (SensorManager) getSystemService(Context.SENSOR_SERVICE);
5 mSensor = mSensorManager.getDefaultSensor(Sensor.TYPE_LINEAR_ACCELERATION);
```

Conceptualmente, este sensor le proporciona datos de aceleración según la siguiente relación:

aceleración lineal = aceleración - aceleración debido a la gravedad

Normalmente utilizaremos este sensor cuando deseemos obtener datos de aceleración sin la influencia de la gravedad. Por ejemplo, podría usar este sensor para saber lo rápido que va mi automóvil. El sensor de aceleración lineal siempre tiene un desplazamiento, que debe eliminar. La forma más sencilla de hacerlo es crear un paso de calibración en nuestra aplicación. Durante la calibración, podemos pedirle al usuario que configure el dispositivo en una mesa, y luego lea los desplazamientos para los tres ejes. A continuación, deberíamos restar ese desplazamiento de las lecturas directas del sensor de aceleración para obtener la aceleración lineal real.

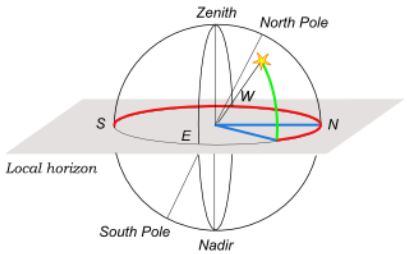
11.- Sensor de campo geomagnético.

El sensor de campo geomagnético le permite controlar los cambios en el campo magnético de la tierra. El siguiente código muestra cómo obtener una instancia del sensor de campo geomagnético predeterminado:

Código

```
1 private SensorManager mSensorManager;
2 private Sensor mSensor;
3 ...
4 mSensorManager = (SensorManager) getSystemService(Context.SENSOR_SERVICE);
5 mSensor = mSensorManager.getDefaultSensor(Sensor.TYPE_MAGNETIC_FIELD);
```

Este sensor proporciona datos de intensidad de campo sin procesar (en μT) para cada uno de los tres ejes de coordenadas. Por lo general, no es necesario que usemos este sensor directamente. En su lugar, podemos usar el sensor de vector de rotación para determinar el movimiento de rotación sin procesar o podemos usar el acelerómetro y el sensor de campo geomagnético junto con el método **getRotationMatrix()** para obtener la matriz de rotación y la matriz de inclinación. A continuación, podemos utilizar estas matrices con los **getOrientation()** y **getInclination()** para obtener datos de inclinación azimutal y geomagnética.



Al probar nuestra aplicación, podemos mejorar la precisión del sensor agitando el dispositivo describiendo un patrón con forma de 8.

12.- Sensor de proximidad.

El sensor de proximidad nos permite determinar lo lejos que está un objeto del dispositivo. El siguiente código muestra cómo obtener una instancia del sensor de proximidad predeterminado:

Código

```
1  private SensorManager mSensorManager;
2  private Sensor mSensor;
3  ...
4  mSensorManager = (SensorManager) getSystemService(Context.SENSOR_SERVICE);
5  mSensor = mSensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY);
```

El sensor de proximidad se usa generalmente para determinar lo lejos que está la cabeza de una persona del sensor del de proximidad, el cual suele estar junto al auricular del teléfono (por ejemplo, cuando un usuario realiza o recibe una llamada telefónica). La mayoría de los sensores de proximidad devuelven la distancia absoluta, en cm, pero algunos devuelven solo valores cercanos y lejanos.

Ejemplo

El siguiente código muestra cómo usar el sensor de proximidad:

```
1  public class SensorActivity extends Activity implements SensorEventListener {
2
3  private SensorManager mSensorManager;
```

```
private Sensor mProximity;

@Override
public final void onCreate(Bundle savedInstanceState) {
  super.onCreate(savedInstanceState);
  setContentView(R.layout.main);

  // Get an instance of the sensor service, and use that to get an instance of
  // a particular sensor.
  mSensorManager = (SensorManager) getSystemService(Context.SENSOR_SERVICE);
  mProximity = mSensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY);
}

@Override
public final void onAccuracyChanged(Sensor sensor, int accuracy) {
  // Do something here if sensor accuracy changes.
  }

@Override
public final void onSensorChanged(SensorEvent event) {
  float distance = event.values[0];
  // Do something with this sensor data.
  }

@Override
protected void onResume() {
  // Register a listener for the sensor.
  super.onResume();
  mSensorManager.registerListener(this, mProximity, SensorManager.SENSOR_DELAY_NORMAL);
  }

@Override
protected void onPause() {
  // Be sure to unregister the sensor when the activity pauses.
  super.onPause();
  mSensorManager.unregisterListener(this);
  }
}
```

Algunos sensores de proximidad devuelven valores binarios que representan "cerca" o "lejos". En este caso, el sensor generalmente informa su valor máximo de rango en el estado lejano y un valor menor en el estado cercano. Normalmente, el valor lejano tiene un valor > 5 cm, pero esto puede variar de un sensor a otro. Podemos determinar el rango máximo de un sensor utilizando el método **getMaximumRange()**.

Autoevaluación

El sensor de aceleración lineal se usa para la detección de gestos o para calcular por ejemplo la velocidad a la que viaja un vehículo.

○ Verdadero ○ Falso

Verdadero

13.- Ejercicio resuelto 2.

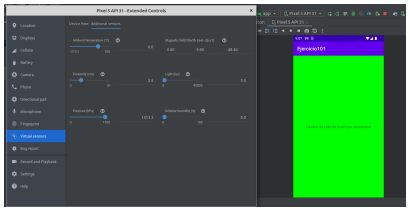
Ejercicio Resuelto

Crear una aplicación que utilice el sistema sensor de proximidad del dispositivo, de forma que teniendo un color de fondo, éste cambia dependiendo de si se sobrepasa una distancia concreta a dicho sensor.

Si entramos en la funcionalidad de sensores que tiene la emulación, y vamos dentro de los sensores adicionales (Additional sensors) al sensor de proximidad (Proximity) podemos marcar la distancia a la que virtualmente nos acercamos o alejamos, desplazando el scroll a valores menores o mayores de 1 cm.

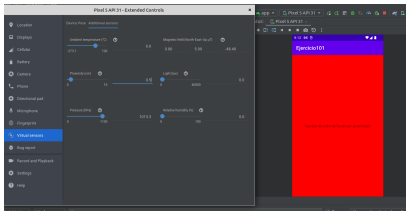
En el caso de estar por encima de este umbral se mostrará la pantalla de color verde (figura 1) y en caso contrario (distancia menor de 1 cm) se activará el color rojo en la pantalla (figura 2).

Figura 1. Distancia > 1 cm.



Google Developers (Uso educativo nc)

Figura 2. Distancia < 1 cm.



Google Developers (Uso educativo nc)

Mostrar retroalimentación

AndroidManifest.xml

Añadimos la siguiente línea en el archivo de manifiesto de nuestra aplicación:

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

activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res/android"
  xmlns:app="http://schemas.android.com/apk/res-auto"
  xmlns:tools="http://schemas.android.com/tools"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  tools:context=".MainActivity">

  <TextView
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:text="Cambio de color de fondo por proximidad"
  app:layout_constraintBottom_toBottomOf="parent"
  app:layout_constraintLeft_toLeftOf="parent"
  app:layout_constraintRight_toRightOf="parent"
  app:layout_constraintTop_toTopOf="parent" />
```

```
18
    </androidx.constraintlayout.widget.ConstraintLayout>
```

MainActivity.java

```
1  package com.cidead.pmdm.ejercicio101;
2
3  import androidx.appcompat.app.AppCompatActivity;
4
5  import android.graphics.Color;
6  import android.hardware.Sensor;
7  import android.hardware.SensorEvent;
8  import android.hardware.SensorEventListener;
9  import android.hardware.SensorManager;
10 import android.os.Bundle;
11 import android.widget.Toast;
12
13 public class MainActivity extends AppCompatActivity {
14 private SensorManager sensorManager;
15  private Sensor sensorProximidad;
16  private SensorEventListener escuchadorSensorProximidad;
17
18  @Override
19  protected void onCreate(Bundle savedInstanceState) {
20 super.onCreate(savedInstanceState);
21  setContentView(R.layout.activity_main);
22  sensorManager = (SensorManager)getSystemService(SENSOR_SERVICE);
23  sensorProximidad = sensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY);
24  if(sensorProximidad==null){
            this, "Sensor de proximidad no disponible", Toast.LENGTH_SHORT).show();
```

```
 finish();
 }

escuchadorSensorProximidad=new SensorEventListener() {

@Override
 public void onSensorChanged(SensorEvent sensorEvent) {
if(sensorEvent.values[0] < sensorProximidad.getMaximumRange()){
getWindow().getDecorView().setBackgroundColor(Color.RED);
 }
else {
getWindow().getDecorView().setBackgroundColor(Color.GREEN);
 }
}

@Override
 public void onAccuracyChanged(Sensor sensor, int i) {
}
};
 sensorManager.registerListener(escuchadorSensorProximidad, sensorProximidad, 2*1000*1000);
 }

@Override
 protected void onPause(){
super.onPause();
 sensorManager.unregisterListener(escuchadorSensorProximidad);
 }
}
```