## **Project Title**

**Sensor Dashboard in Android using Jetpack Compose**

## **Aim**

To build a **real-time sensor dashboard app** in Android using Jetpack Compose that integrates multiple hardware sensors to measure motion, environment, and orientation, and displays their values in a clean and interactive UI.

## **Features**

* Real-time monitoring of 9 key sensors.
* Motion sensors: Accelerometer, Gyroscope, Gravity, Step Counter, Step Detector.
* Environmental sensors: Ambient Light, Proximity.
* Position sensors: Magnetometer, Linear Acceleration.
* Intuitive dashboard UI built with Jetpack Compose.
* Smart logic such as metal detection (magnetometer), day/night detection (light), and torch control (proximity).

## **Tools & Technologies**

* **Language:** Kotlin
* **Framework:** Jetpack Compose
* **API:** Android SensorManager (android.hardware package)
* **IDE:** Android Studio
* **Dependencies:**
  + androidx.compose.ui:ui
  + androidx.compose.material3:material3
  + androidx.activity:activity-compose

## **Implementation**

### **General Sensor Flow**

1. Get SensorManager from the system.
2. Get the required sensor using getDefaultSensor().
3. Register a SensorEventListener to receive updates.
4. Extract values (event.values).
5. Update the UI with Compose.

### **Motion Sensors**

#### **1. Accelerometer**

* **Purpose:** Measures acceleration along X, Y, Z axes including gravity.
* **Uses:** Screen rotation, shake detection, step counting.
* **Code snippet:**

val accelerometer = sensorManager.getDefaultSensor(Sensor.TYPE\_ACCELEROMETER)

override fun onSensorChanged(event: SensorEvent) {

val x = event.values[0]

val y = event.values[1]

val z = event.values[2]

}

#### **2. Gyroscope**

* **Purpose:** Measures angular velocity (rotation speed) around X, Y, Z axes.

**Uses:** AR/VR, gaming, camera stabilization.

* **Code snippet:**

val gyroscope = sensorManager.getDefaultSensor(Sensor.TYPE\_GYROSCOPE)

override fun onSensorChanged(event: SensorEvent) {

val x = event.values[0]

val y = event.values[1]

val z = event.values[2]

}

#### **3. Gravity Sensor**

* **Purpose:** Detects gravity vector along X, Y, Z (no motion noise).
* **Uses:** Orientation, tilt detection, AR alignment.
* **Code snippet:**

val gravitySensor = sensorManager.getDefaultSensor(Sensor.TYPE\_GRAVITY)

override fun onSensorChanged(event: SensorEvent) {

val gx = event.values[0]

val gy = event.values[1]

val gz = event.values[2]

}

#### **4. Step Counter**

* **Purpose:** Cumulative step count since device reboot.
* **Uses:** Fitness apps, activity tracking.
* **Code snippet:**

val stepCounter = sensorManager.getDefaultSensor(Sensor.TYPE\_STEP\_COUNTER)

override fun onSensorChanged(event: SensorEvent) {

val steps = event.values[0].toInt()

}

#### **5. Step Detector**

* **Purpose:** Detects each individual step in real time.
* **Uses:** Fitness games, instant feedback apps.
* **Code snippet:**

val stepDetector = sensorManager.getDefaultSensor(Sensor.TYPE\_STEP\_DETECTOR)

override fun onSensorChanged(event: SensorEvent) {

stepsDetected++

}

### **Position Sensors**

#### **6. Magnetometer**

* **Purpose:** Measures magnetic field (X, Y, Z) in µT. Works like a compass.
* **Uses:** Compass apps, navigation, metal detection.
* **Logic:**
* Azimuth = direction relative to North.
* Field strength = magnitude of magnetic field.
* **Code snippet:**

val magnetometer = sensorManager.getDefaultSensor(Sensor.TYPE\_MAGNETIC\_FIELD)

override fun onSensorChanged(event: SensorEvent) {

val x = event.values[0]

val y = event.values[1]

val z = event.values[2]

val azimuth = (Math.toDegrees(atan2(y.toDouble(), x.toDouble())) + 360).toFloat() % 360

val fieldStrength = sqrt(x\*x + y\*y

+ z\*z)

}

#### **7. Linear Acceleration**

* **Purpose:** Measures acceleration excluding gravity.
* **Uses:** Detect shakes, falls, vehicle acceleration.
* **Code snippet:**

val linearAccel = sensorManager.getDefaultSensor(Sensor.TYPE\_LINEAR\_ACCELERATION)

override fun onSensorChanged(event: SensorEvent) {

val x = event.values[0]

val y = event.values[1]

val z = event.values[2]

val magnitude = sqrt(x.pow(2) + y.pow(2) + z.pow(2))

}

### **Environmental Sensors**

#### **8. Ambient Light Sensor**

* **Purpose:** Measures light intensity (lux).
* **Uses:** Auto-brightness, day/night mode.
* **Code snippet:**

val lightSensor = sensorManager.getDefaultSensor(Sensor.TYPE\_LIGHT)

override fun onSensorChanged(event: SensorEvent) {

val lux = event.values[0]

val isDark = lux < 40

}

#### **9. Proximity Sensor**

* **Purpose:** Detects if an object is near (cm).
* **Uses:** Screen off during calls, torch control, touchless systems.

**Code snippet:**

val proximity = sensorManager.getDefaultSensor(Sensor.TYPE\_PROXIMITY)

override fun onSensorChanged(event: SensorEvent) {

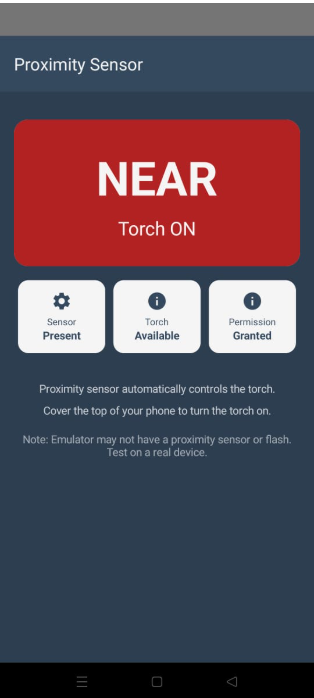
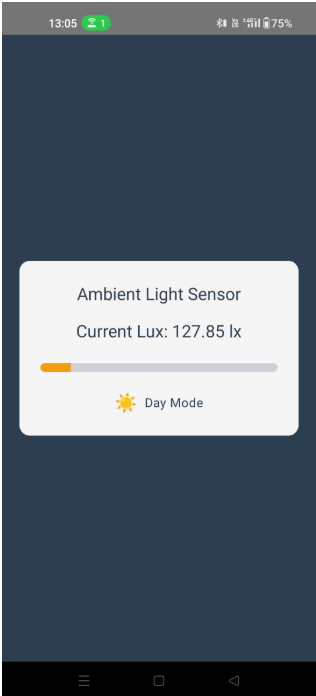
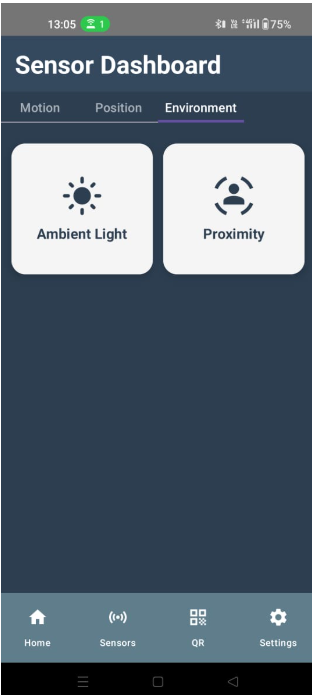
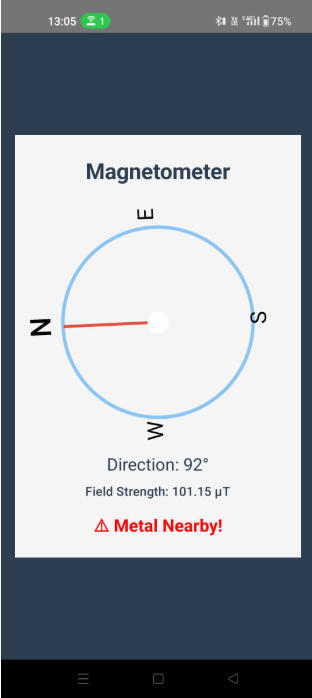
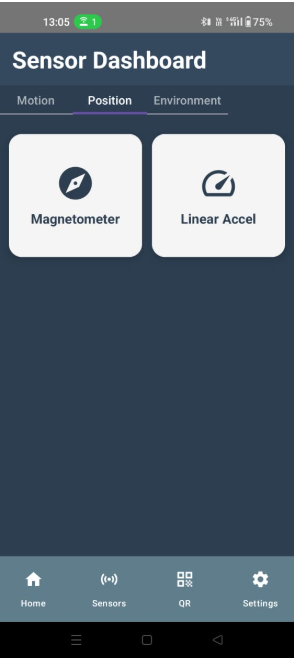
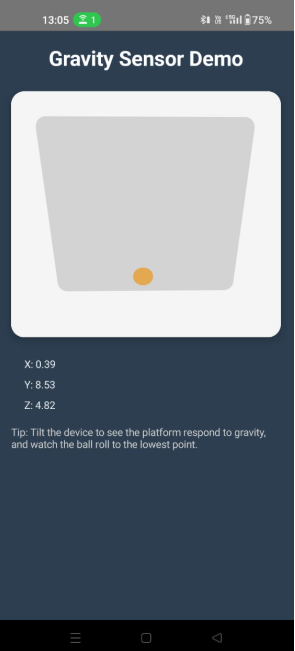
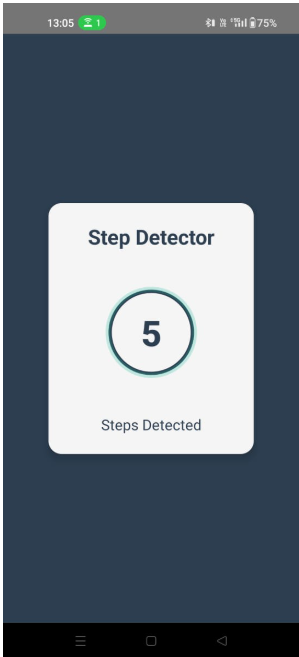
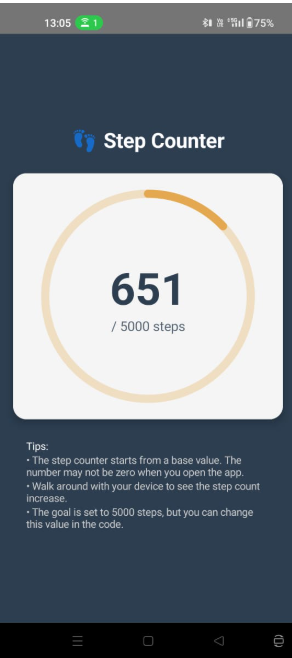
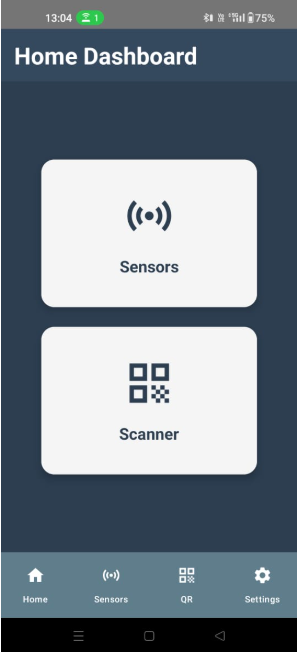
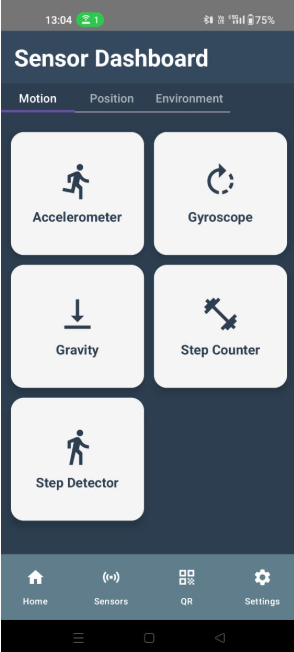
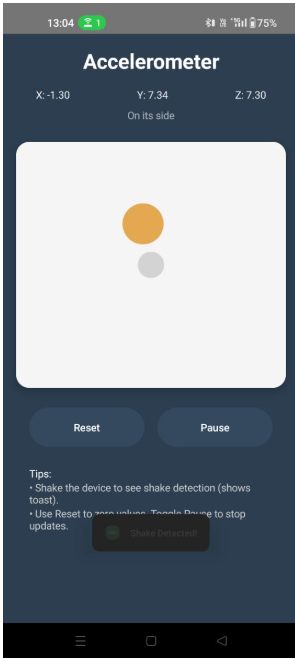
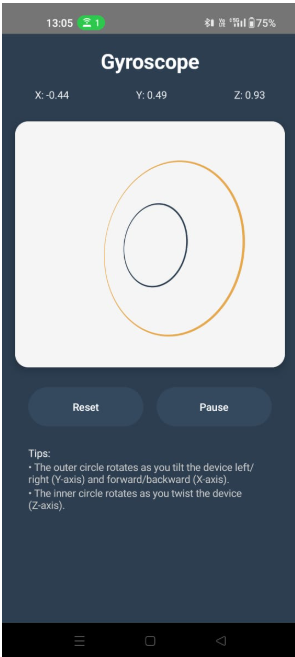
val isNear = event.values[0] < proximity.maximumRange

}

## **Results**

* The app successfully displays live readings for **9 different sensors**.
* Detects device orientation, tilt, steps, light intensity, proximity, and magnetic direction.
* Provides a **unified dashboard** for all sensors.

**ScreenShorts**



## **Applications**

* **Smartphones**: Auto-brightness, pocket detection, orientation handling.
* **Fitness**: Step tracking, activity monitoring.
* **Gaming/AR/VR**: Tilt and motion controls, rotation tracking.
* **IoT & Robotics**: Movement detection, environment monitoring.
* **Navigation**: Compass, location tracking.

## **Conclusion**

The project demonstrates how Android’s **SensorManager** can be used with Jetpack Compose to build a live **Sensor Dashboard**. It shows practical applications of motion, environmental, and position sensors, making devices more interactive and context-aware.

## **Future Scope**

* Add graphs and history logs of sensor data.
* Store and analyze readings in cloud.
* Use ML models for predictive analytics.
* Extend to wearables and IoT devices.