Міністерство освіти і науки України
Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського»
Факультет інформатики та обчислювальної техніки Кафедра інформатики та програмної інженерії

Звіт
до лабораторної роботи № 5 з дисципліни
«Розробка мобільних застосувань під Android»

Виконав ІК-24 Ніконов С.

Перевірив Орленко С.П.

Київ 2025

Лабораторна робота № 5

Тема: ДОСЛІДЖЕННЯ РОБОТИ З ВБУДОВАНИМИ ДАТЧИКАМИ

Мета роботи: ознайомитись з можливостями вбудованих датчиків мобільних пристроїв та дослідити способи їх використання для збору та обробки даних.

Завдання:

БАЗОВЕ (10/20 балів). Написати програму під платформу Андроїд, яка має інтерфейс для виведення даних з обраного вбудованого датчика (тип обирається самостійно, можна відслідковувати зміни значень і з декількох датчиків).

ПОВНЕ (20/20). Функціональність базового додатку додатково розширюється обробкою отриманих даних та виведенням їх у відповідній формі.

Примітка: конкретного варіанту не передбачено, студент сам обирає завдання та вигляд програми. Приклади очікуваних робіт:

- «будівельний рівень» з виведенням лінії горизонту та кутом нахилу;
- компас з ілюстрацією стрілки (циферблату з позначеними сторонами світу);
- крокомір (підрахунок кількості кроків);
- додаток для вимірювання перевантажень в авто (G-force meter);
- автоматичне регулювання яскравості та екрану в залежності від рівня освітлення, але ще б додати автозаглушення екрану при піднесенні до перешкоди (до вуха під час розмови або «в кишені»), щоб уникнути ненавмисних дотиків;
- барометр з прогнозом погоди (мова про опади – зміна атмосферного тиску, а, можливо, і вологості з температурою).

HARD TASK (не обов’язково). Оскільки конкретного варіанту немає, то і конкретного завдання модифікацій додатку теж немає, але слід додати або додаткову аналітику, або використовувати інші засоби для підвищення функціональності програми. На прикладі крокоміру можна додати аналітику «за день» або «за тренування»: відобразити зміни активності, спробувати обчислити пройдену відстань чи витрачені калорії, для візуалізації можна побудувати відповідні графіки. На прикладі «вимірювача перевантажень в авто» можна спробувати додатково визначити швидкість та пройдену відстань, спробувати відмалювати карту пройденого маршруту, якщо рух кільцевий – можна вести відлік часу проходження кола, або окремих його ділянок. Для

визначення швидкості/дистанції можна спробувати інтегрування або через визначення місцезнаходження на основі геолокації.

Мій варіант завдання:

**Віртуальний будівельний рівень**

**Базова функціональність (10/20):**

- Використання вбудованого акселерометра для визначення нахилу пристрою.
- Розрахунок кута нахилу по двох осях — X та Y.
- Виведення числових значень кутів на екран у градусах.

**Розширена функціональність (20/20):**

1. **Візуалізація рівня:**
    - На екрані відображено віртуальний рівень з анімованою "бульбашкою".
    - Бульбашка рухається відповідно до положення пристрою в просторі (імітація класичного будівельного рівня).
2. **Анімація:**
    - Використано animateFloatAsState для плавного переміщення бульбашки при зміні кута нахилу.

**Hard Task:**

1. **Логування:**
    - Кут нахилу по осі X записується у список кожні 100 мс.
    - Ведеться історія значень останніх ~10 секунд (макс. 100 записів).
2. **Графік:**
    - Побудовано реальний графік зміни кута по осі X за останні 10 секунд.

- Графік реалізовано з нуля за допомогою Canvas (без сторонніх бібліотек).

Код:

```
package com.example.lab5android


import android.app.Activity
import android.content.Context
import android.hardware.Sensor
import android.hardware.SensorEvent
import android.hardware.SensorEventListener
import android.hardware.SensorManager
import android.os.Bundle
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.compose.animation.core.animateFloatAsState
import androidx.compose.foundation.Canvas
import androidx.compose.foundation.background
import androidx.compose.foundation.layout.*
import androidx.compose.material3.MaterialTheme
import androidx.compose.material3.Text
import androidx.compose.runtime.*
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.graphics.Color
import androidx.compose.ui.unit.dp
import androidx.compose.ui.unit.sp
import androidx.compose.ui.text.font.FontWeight
```

```
import kotlin.math.atan2
import kotlin.math.roundToInt
import kotlinx.coroutines.delay
import androidx.compose.ui.geometry.Offset


class MainActivity : ComponentActivity() {
   override fun onCreate(savedInstanceState: Bundle?) {
       super.onCreate(savedInstanceState)
       setContent {
           MaterialTheme {
               LevelScreen(this)
           }
       }
   }
}


@Composable
fun LevelScreen(activity: Activity) {
   val sensorManager = remember {
           activity.getSystemService(Context.SENSOR_SERVICE) as
SensorManager
   }
   val accelerometer = remember {

sensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER)
   }


   var angleX by remember { mutableStateOf(0f) }
```

```
    var angleY by remember { mutableStateOf(0f) }


    val angleXLog = remember { mutableStateListOf<Float>() }


    val maxLogSize = 10000


    DisposableEffect(Unit) {
        val listener = object : SensorEventListener {
            override fun onSensorChanged(event: SensorEvent?) {
                event?.let {
                    val ax = event.values[0]
                    val ay = event.values[1]
                    val az = event.values[2]


                        val newAngleX = (atan2(ax, az) * (180 /
Math.PI)).toFloat()
                        val newAngleY = (atan2(ay, az) * (180 /
Math.PI)).toFloat()


                    angleX = newAngleX
                    angleY = newAngleY
                }
            }


            override fun onAccuracyChanged(sensor: Sensor?,
accuracy: Int) {}
        }
        sensorManager.registerListener(listener, accelerometer,
SensorManager.SENSOR_DELAY_UI)
```

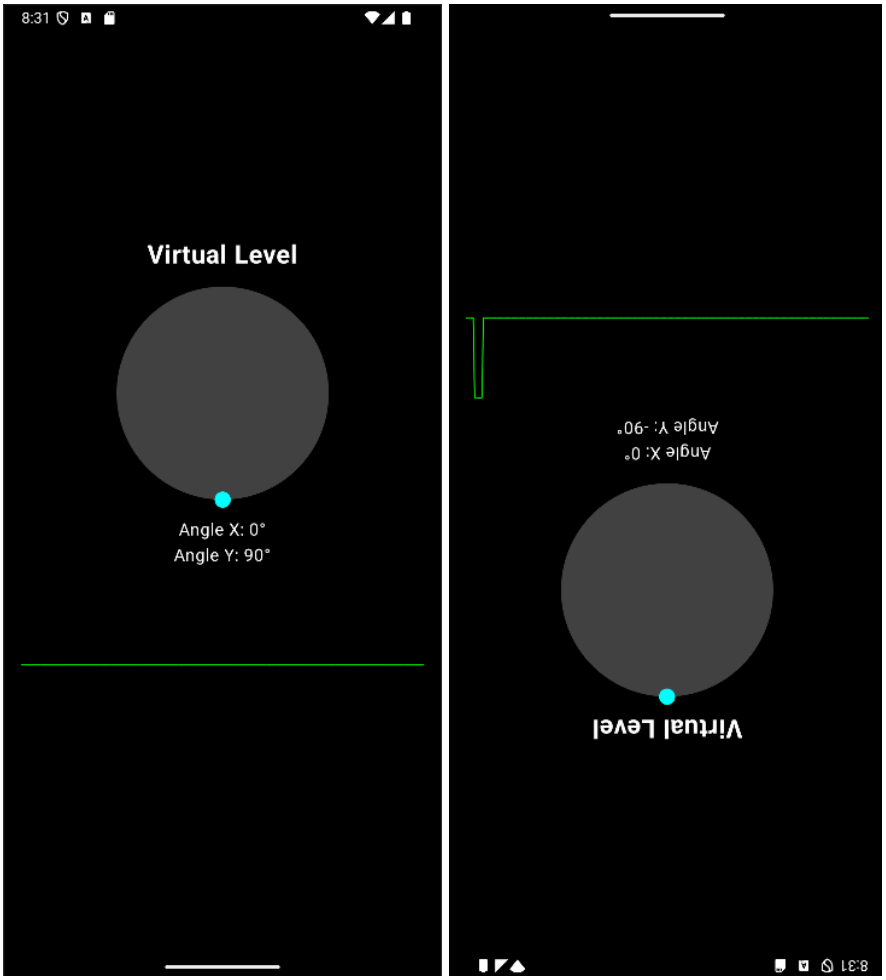
```
        onDispose {
            sensorManager.unregisterListener(listener)
        }
    }

    LaunchedEffect(Unit) {
        while (true) {
            angleXLog.add(angleX)
            if (angleXLog.size > maxLogSize) {
                angleXLog.removeAt(0)
            }
            delay(100)
        }
    }

    val animatedOffsetX by animateFloatAsState(targetValue = angleX)
    val animatedOffsetY by animateFloatAsState(targetValue = angleY)

    Box(
        modifier = Modifier
            .fillMaxSize()
            .background(Color.Black),
        contentAlignment = Alignment.Center
    ) {
                                Column(horizontalAlignment = Alignment.CenterHorizontally) {
```

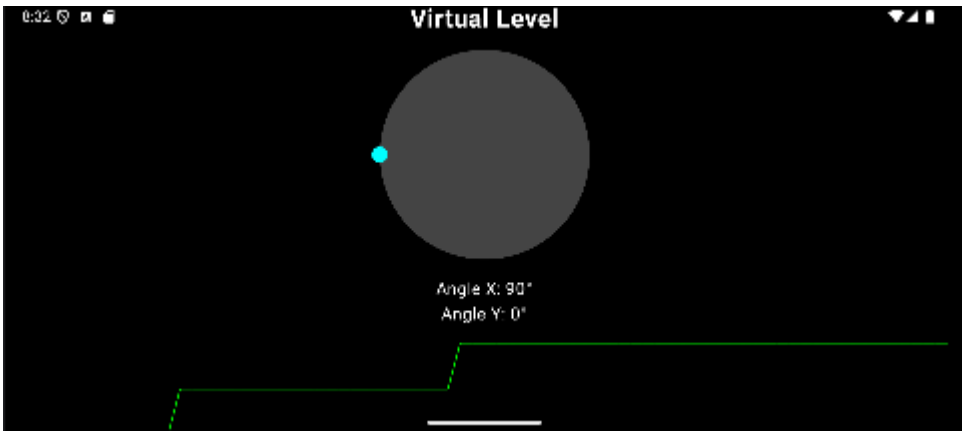
```
            Text(
                text = "Virtual Level",
                color = Color.White,
                fontSize = 24.sp,
                fontWeight = FontWeight.Bold
            )
            Spacer(modifier = Modifier.height(16.dp))
            Canvas(modifier = Modifier.size(200.dp)) {
                val radius = size.minDimension / 2
                val center = center

                drawCircle(
                    color = Color.DarkGray,
                    radius = radius,
                    center = center
                )
                val offsetX = (animatedOffsetX / 90f) * radius
                val offsetY = (animatedOffsetY / 90f) * radius
                drawCircle(
                    color = Color.Cyan,
                    radius = 20f,
                        center = center.copy(x = center.x -
offsetX, y = center.y + offsetY)
                )
            }
            Spacer(modifier = Modifier.height(16.dp))
               Text("Angle X: ${angleX.roundToInt()}°", color =
Color.White)
```

```
            Text("Angle Y: ${angleY.roundToInt()}°", color =
Color.White)

            Spacer(modifier = Modifier.height(16.dp))

            Graph(angleXLog)

        }

    }

}


@Composable

fun Graph(data: List<Float>) {

    Canvas(modifier = Modifier

        .fillMaxWidth()

        .height(150.dp)

        .padding(horizontal = 16.dp)) {


        if (data.isEmpty()) return@Canvas


        val maxValue = 90f

        val minValue = -90f

        val range = maxValue - minValue


                val stepX = size.width / (data.size -
1).coerceAtLeast(1)


        for (i in 0 until data.size - 1) {

            val x1 = i * stepX

                val y1 = size.height - ((data[i] - minValue) /
range) * size.height

            val x2 = (i + 1) * stepX
```

```
            val y2 = size.height - ((data[i + 1] - minValue) /
range) * size.height

            drawLine(

                color = Color.Green,

                start = Offset(x1, y1),

                end = Offset(x2, y2),

                strokeWidth = 3f

            )

        }

    }

}
```

Вигляд програми:






У емуляторі складно передати всю суть програми, але сподіваюсь головний функціонал розкрити вийшло