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

з лабораторної роботи №5 з дисципліни

«Розробка мобільних застосувань під Android»

«Дослідження роботи з вбудованими датчиками»

Виконав ІП-22 Нижник Дмитро Сергійович _____

(шифр, прізвище, ім'я, по батькові)

Перевірив Орленко Сергій Петрович _____

(прізвище, ім'я, по батькові)

Київ 2025

Завдання лабораторної роботи

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).

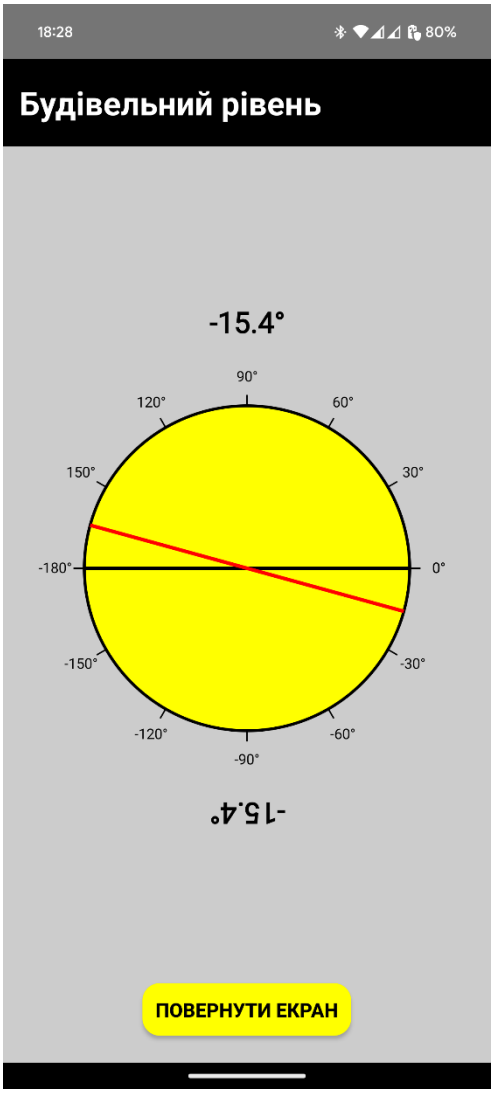
Для виконання ЛР обрано тему «будівельний рівень»

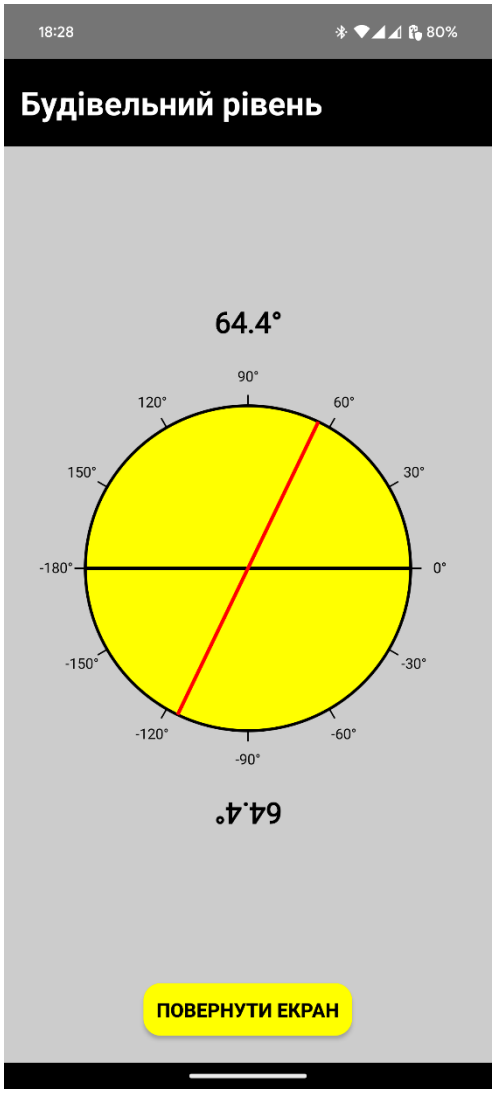
Хід виконання роботи

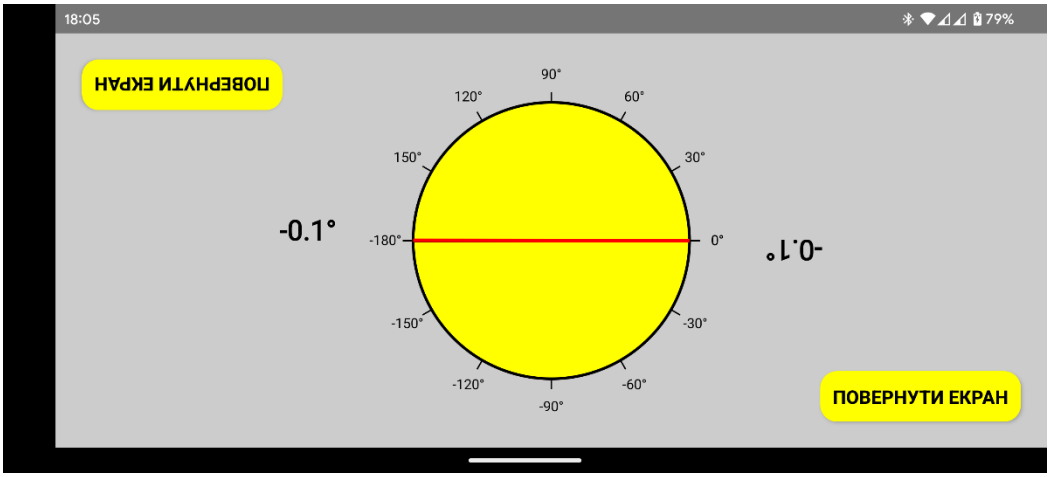
У процесі виконання лабораторної роботи було розроблено класи MainActivity та LevelView. Клас MainActivity реалізує інтерфейс SensorEventListener для обробки даних із вбудованого акселерометра. Зокрема, перевизначено метод onSensorChanged, який здійснює розрахунок кута нахилу пристрою. Крім того, клас містить логіку зміни орієнтації екрана за допомогою відповідних кнопок, а також забезпечує передачу значення кута візуальному компоненту.

Клас LevelView відповідає за графічне відображення рівня. У методі onDraw реалізовано відображення кута нахилу у вигляді горизонтальної лінії , що обертається в залежності від куту нахилу, а також текстових значень кута на екрані. Метод drawCircleBackground будує кругову шкалу з поділками та підписами, які позначають значення кута від -180° до 180°, створюючи таким чином інтерфейс цифрового будівельного рівня.

Приклад роботи застосунку







Код програми

MainActivity.kt

```
package com.androidlabs.lab_5

import android.content.Context
import android.content.pm.ActivityInfo
import android.hardware.Sensor
import android.hardware.SensorEvent
import android.hardware.SensorEventListener
import android.hardware.SensorManager
import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle
import android.widget.Button
import androidx.xr.runtime.math.toDegrees
import kotlin.math.atan2

class MainActivity : AppCompatActivity(), SensorEventListener {

    private lateinit var sensorManager: SensorManager
    private var accelerometer: Sensor? = null
    private lateinit var levelView: LevelView

    private var orientationIndex = 0
    private val orientations = arrayOf(
        ActivityInfo.SCREEN_ORIENTATION_PORTRAIT,
        ActivityInfo.SCREEN_ORIENTATION_LANDSCAPE
    )

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
        supportActionBar?.hide()

        if (savedInstanceState != null) {
            orientationIndex = savedInstanceState.getInt("orientationIndex", 0)
            requestedOrientation = orientations[orientationIndex]
        } else {
            requestedOrientation = ActivityInfo.SCREEN_ORIENTATION_NOSENSOR
        }

        val rotateButton = findViewById<Button?>(R.id.rotateButton)
        if (rotateButton != null) {
            rotateButton.setOnClickListener { cycleOrientation() }
        } else {
            val rotateButtonTopLeft: Button =
findViewById(R.id.rotateButtonTopLeft)
            val rotateButtonBottomRight: Button =
findViewById(R.id.rotateButtonBottomRight)
            rotateButtonTopLeft.setOnClickListener { cycleOrientation() }
            rotateButtonBottomRight.setOnClickListener { cycleOrientation() }
        }

        levelView = findViewById(R.id.levelView)
        sensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager
        accelerometer = sensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER)
    }
```

```
    override fun onSaveInstanceState(outState: Bundle) {
        super.onSaveInstanceState(outState)
        outState.putInt("orientationIndex", orientationIndex)
    }

    override fun onResume() {
        super.onResume()
        accelerometer?.also { sensor ->
            sensorManager.registerListener(this, sensor,
SensorManager.SENSOR_DELAY_UI)
        }
    }

    override fun onPause() {
        super.onPause()
        sensorManager.unregisterListener(this)
    }

    override fun onSensorChanged(event: SensorEvent?) {
        event?.let {
            val rawX = it.values[0]
            val rawY = it.values[1]

            var angle = 0f
            when (orientations[orientationIndex]) {
                ActivityInfo.SCREEN_ORIENTATION_PORTRAIT -> {
                    angle = toDegrees(atan2(rawX.toDouble(),
rawY.toDouble()).toFloat())
                }
                ActivityInfo.SCREEN_ORIENTATION_LANDSCAPE -> {
                    angle = toDegrees(atan2(rawY.toDouble(), -
rawX.toDouble()).toFloat())
                    angle = normalizeAngle(angle - 180f)
                }
            }
            levelView.tiltAngle = angle
        }
    }

    override fun onAccuracyChanged(sensor: Sensor?, accuracy: Int) { }

    private fun cycleOrientation() {
        orientationIndex = (orientationIndex + 1) % orientations.size
        requestedOrientation = orientations[orientationIndex]
    }

    private fun normalizeAngle(angle: Float): Float {
        var normalized = ((angle % 360) + 360) % 360
        if (normalized > 180f) normalized -= 360f
        if (normalized == 180f || normalized == -180f) normalized = 0f
        return normalized
    }

}
```

LevelView.kt

```
package com.androidlabs.lab_3
package com.androidlabs.lab_5

import kotlin.math.cos
import kotlin.math.min
import kotlin.math.sin
import android.view.View
import android.graphics.Color
import android.graphics.Paint
import android.content.Context
import android.graphics.Canvas
import android.util.AttributeSet

class LevelView(context: Context, attrs: AttributeSet?) : View(context, attrs) {

    var tiltAngle: Float = 0f
        set(value) {
            field = value
            invalidate()
        }

    private val centerLinePaint = Paint().apply {
        color = Color.BLACK
        strokeWidth = 10f
        isAntiAlias = true
    }

    private val horizonLinePaint = Paint().apply {
        color = Color.RED
        strokeWidth = 10f
        isAntiAlias = true
    }

    private val angleTextPaint = Paint().apply {
        color = Color.BLACK
        textSize = 80f
        isAntiAlias = true
        isFakeBoldText = true
        textAlign = Paint.Align.CENTER
    }

    override fun onDraw(canvas: Canvas) {
        super.onDraw(canvas)

        val centerX = width / 2f
        var centerY = height / 2f
        val circleRadius = min(width, height) / 3f

        if (width > height) {
            val margin = 300f
            val angleText = String.format("%.1f°", tiltAngle)

            val rightX = centerX - circleRadius - margin
            val rightY = centerY
            canvas.drawText(angleText, rightX, rightY, angleTextPaint)

            val leftX = centerX + circleRadius + margin
```

```
            val leftY = centerY
            canvas.save()
            canvas.rotate(180f, leftX, leftY)
            canvas.drawText(angleText, leftX, leftY, angleTextPaint)
            canvas.restore()
        } else {
            val margin = 200f
            val angleText = String.format("%.1f°", tiltAngle)
            centerY -= 100f
            canvas.drawText(angleText, centerX, centerY - circleRadius - margin,
angleTextPaint)
            canvas.save()
            canvas.rotate(180f, centerX, centerY + circleRadius + margin)
            canvas.drawText(angleText, centerX, centerY + circleRadius + margin,
angleTextPaint)
            canvas.restore()
        }

        drawCircleBackground(canvas, centerX, centerY, circleRadius)

        canvas.drawLine(centerX - circleRadius, centerY, centerX + circleRadius,
centerY, centerLinePaint)
        canvas.save()
        canvas.rotate(-tiltAngle, centerX, centerY)
        canvas.drawLine(centerX - circleRadius, centerY, centerX + circleRadius,
centerY, horizonLinePaint)
        canvas.restore()
    }

    private fun drawCircleBackground(canvas: Canvas, centerX: Float, centerY:
Float, radius: Float) {
        val fillPaint = Paint().apply {
            color = Color.YELLOW
            style = Paint.Style.FILL
            isAntiAlias = true
        }

        val borderPaint = Paint().apply {
            color = Color.BLACK
            style = Paint.Style.STROKE
            strokeWidth = 8f
            isAntiAlias = true
        }

        canvas.drawCircle(centerX, centerY, radius, fillPaint)
        canvas.drawCircle(centerX, centerY, radius, borderPaint)

        val tickLength = 30f

        val tickPaint = Paint().apply {
            color = Color.BLACK
            strokeWidth = 4f
            isAntiAlias = true
        }

        val tickTextPaint = Paint().apply {
            color = Color.BLACK
            textSize = 40f
            isAntiAlias = true
```

```
            textAlign = Paint.Align.CENTER
        }

        val textMargin = 10f

        for (angle in 0 until 180 step 30) {
            val rad = Math.toRadians(angle.toDouble())
            val startX = centerX + radius * cos(rad).toFloat()
            val startY = centerY + radius * sin(rad).toFloat()
            val endX = centerX + (radius + tickLength) * cos(rad).toFloat()
            val endY = centerY + (radius + tickLength) * sin(rad).toFloat()

            canvas.drawLine(startX, startY, endX, endY, tickPaint)

            val textX = centerX + (radius + tickLength + textMargin +
tickTextPaint.textSize) * cos(rad).toFloat()
            val textY = centerY - (radius + tickLength + textMargin +
tickTextPaint.textSize) * sin(rad).toFloat() + tickTextPaint.textSize / 3f

            canvas.drawText("$angle°", textX, textY, tickTextPaint)
        }
        for (angle in 0 downTo -180 step 30) {
            val rad = Math.toRadians(angle.toDouble())
            val startX = centerX + radius * cos(rad).toFloat()
            val startY = centerY + radius * sin(rad).toFloat()
            val endX = centerX + (radius + tickLength) * cos(rad).toFloat()
            val endY = centerY + (radius + tickLength) * sin(rad).toFloat()

            canvas.drawLine(startX, startY, endX, endY, tickPaint)

            val textX = centerX + (radius + tickLength + textMargin +
tickTextPaint.textSize) * cos(rad).toFloat()
            val textY = centerY - (radius + tickLength + textMargin +
tickTextPaint.textSize) * sin(rad).toFloat() + tickTextPaint.textSize / 3f

            canvas.drawText("$angle°", textX, textY, tickTextPaint)
        }
    }

}
```