# ДОДАТОК А

## Лістинг програмного коду

```
cmake_minimum_required(VERSION 3.10)
project(MineDetectorRobot)
set(CMAKE_CXX_STANDARD 17)
# Знайти необхідні пакети
find_package(wiringPi REQUIRED)
find_package( REQUIRED) # або інша бібліотека машинного навчання
# Включити директорії
include_directories(${WIRINGPI_INCLUDE_DIRS})
include_directories(${_INCLUDE_DIRS})

# Вказати вихідний виконуваний файл і його джерела
add_executable(MineDetector main.cpp SensorManager.cpp RobotController.cpp
NeuralNetworkInterface.cpp)
# Лінкування бібліотек з виконуваним файлом
target_link_libraries(MineDetector ${WIRINGPI_LIBRARIES} ${_LIBRARIES})
#include <iostream>
#include <wiringPi.h>
#include "SensorManager.h"
#include "RobotController.h"
#include "NeuralNetworkInterface.h"
#include "MapManager.h"
#include "Manipulator.h"
// GPIO pins for alert system
const int ALERT_LED_PIN = 5;
const int ALERT_BEEPER_PIN = 6;
// Exploration parameters
const double GRID_SIZE = 1.0; // The size of each grid square in meters
const double ROBOT_SPEED = 0.5; // Speed of the robot in meters per second
const int TURN_ANGLE = 90; // Angle to turn at each grid boundary
void initializeAlertSystem() {
   pinMode(ALERT_LED_PIN, OUTPUT);
   pinMode(ALERT_BEEPER_PIN, OUTPUT);}
void notifyOperator() {
   digitalWrite(ALERT_LED_PIN, HIGH);
   delay(500);
   digitalWrite(ALERT_LED_PIN, LOW);
   digitalWrite(ALERT_BEEPER_PIN, HIGH);
   delay(500);
   digitalWrite(ALERT_BEEPER_PIN, LOW);}
void decideNextMovement(bool& moveForward, bool& turnRight, bool& turnLeft, bool&
atBoundary) {
   static int gridCount = 0;
   static bool movingForward = true;
   if (gridCount >= GRID_SIZE / ROBOT_SPEED) { // Reached end of current grid line
         gridCount = 0;
```

```
            atBoundary = true;
            movingForward = !movingForward; // Change direction for next line
            if (movingForward) {
            turnRight = true;} else {
            turnLeft = true;}
        } else {
            moveForward = true;
            atBoundary = false;}
        gridCount++;}
int main() {
    try {
            wiringPiSetup();
            SensorManager::initialize();
            RobotController::initialize();
            NeuralNetworkInterface::initialize();
            Manipulator manipulator;
            manipulator.initialize();
            MapManager mapManager;
            mapManager.initialize();
            initializeAlertSystem();
            bool moveForward = false, turnRight = false, turnLeft = false, atBoundary = false;
            while (true) {
            SensorData sensorData = SensorManager::readSensors();
            mapManager.updateMap(sensorData);
            bool mineDetected = NeuralNetworkInterface::analyzeData(sensorData);
            if (mineDetected) {
                    RobotController::stop();
                    notifyOperator();
                    std::cout << "Mine detected! Movement stopped." << std::endl;
                manipulator.moveTo(sensorData.gpsData.latitude, sensorData.gpsData.longitude, 0);
                    if (manipulator.checkForMetal()) {
                  mapManager.markPointAsMine(sensorData.gpsData.latitude,
sensorData.gpsData.longitude, 0);}}
            decideNextMovement(moveForward, turnRight, turnLeft, atBoundary);
            if (moveForward) {
               RobotController::moveForwardIncrementally(ROBOT_SPEED);
            } else if (turnRight) {
               RobotController::turnRightIncrementally(TURN_ANGLE);
                    turnRight = false; // Reset the flag after turning
            } else if (turnLeft) {
               RobotController::turnLeftIncrementally(TURN_ANGLE);
                    turnLeft = false; // Reset the flag after turning}
            delay(1000 / ROBOT_SPEED); // Adjust delay based on robot speed}
    } catch (const std::exception& e) {
            std::cerr << "Initialization error: " << e.what() << std::endl;
            return -1;}
    return 0;}
#include "Manipulator.h"
#include <wiringPi.h>
#include <softPwm.h>
// GPIO pins for the servos and metal detector sensor
const int SERVO_BASE = 11;
```

```
const int SERVO_SHOULDER = 12;
const int SERVO_ELBOW = 13;
const int SERVO_WRIST = 14;
const int SERVO_GRIPPER = 15;
const int METAL_DETECTOR_SENSOR = 16;
Manipulator::Manipulator() {
   // Constructor implementation (if needed)}
Manipulator::~Manipulator() {
   // Destructor implementation (if needed)}
void Manipulator::initialize() {
   wiringPiSetup();  // Initialize wiringPi
   // Initialize servo motor control pins
   softPwmCreate(SERVO_BASE, 0, 100);
   softPwmCreate(SERVO_SHOULDER, 0, 100);
   softPwmCreate(SERVO_ELBOW, 0, 100);
   softPwmCreate(SERVO_WRIST, 0, 100);
   softPwmCreate(SERVO_GRIPPER, 0, 100);
   // Initialize the metal detector sensor pin
   pinMode(METAL_DETECTOR_SENSOR, INPUT);}
void Manipulator::moveTo(double x, double y, double z) {
   int baseAngle = calculateBaseAngle(x, y);
   int shoulderAngle = calculateShoulderAngle(x, y, z);
   int elbowAngle = calculateElbowAngle(x, y, z);
   int wristAngle = calculateWristAngle(x, y, z);
   softPwmWrite(SERVO_BASE, baseAngle);
   softPwmWrite(SERVO_SHOULDER, shoulderAngle);
   softPwmWrite(SERVO_ELBOW, elbowAngle);
   softPwmWrite(SERVO_WRIST, wristAngle);}
void Manipulator::grabObject() {
   softPwmWrite(SERVO_GRIPPER, 100);}
void Manipulator::releaseObject() {
   softPwmWrite(SERVO_GRIPPER, 0);}
bool Manipulator::checkForMetal() {
   return digitalRead(METAL_DETECTOR_SENSOR) == HIGH;}
int Manipulator::calculateBaseAngle(double x, double y) {
   return atan2(y, x)  180 / M_PI; // convert from radians to degrees}
int Manipulator::calculateShoulderAngle(double x, double y, double z) {
   return atan2(z, sqrt(xx + yy))  180 / M_PI;}
int Manipulator::calculateElbowAngle(double x, double y, double z) {
   // Simplified calculation; to be replaced with actual kinematic calculation
   return 45; // Placeholder value}
int Manipulator::calculateWristAngle(double x, double y, double z) {
   // Simplified calculation; to be replaced with actual kinematic calculation
   return 45; // Placeholder value}
#ifndef MANIPULATOR_H
#define MANIPULATOR_H
class Manipulator {
public:
   Manipulator();
   ~Manipulator();
   void initialize();
   void moveTo(double x, double y, double z);
```

```
    void grabObject();
    void releaseObject();
    bool checkForMetal(); // Function to check for metal using the metal detector
private:
    // Member variables for actuators, sensors, etc.
    // Example: int servoMotorPin;
    // Add other necessary private methods and attributes};
#endif // MANIPULATOR_H
#include "MapManager.h"
MapManager::MapManager() {
    // Initialize the RTAB-Map parameters
    rtabmap::ParametersMap parameters;
    // Set various parameters. For example:
    //  parameters.insert(rtabmap::ParametersPair(rtabmap::Parameters::kRtabmapDetectionRate(),
"1"));
    rtabmap_.init(parameters);}
MapManager::~MapManager() {
    rtabmap_.close();}
void MapManager::processSensorData(const SensorData& data) {
    rtabmap::SensorData rtabmapData = convertToRtabmapSensorData(data);
    rtabmap_.process(rtabmapData);
    // if metal is detected, mark the current location as a mine
    if (data.metalDetected) {
        rtabmap::Transform currentPose = rtabmap_.getPose();
        if(!currentPose.isNull()) {
        markPointAsMine(currentPose.x(), currentPose.y(), currentPose.z());}}}
rtabmap::SensorData MapManager::convertToRtabmapSensorData(const SensorData& data) {
    // Creating a dummy image as RTAB-Map expects visual/laser data for mapping
    cv::Mat dummyImage(1, 1, CV_8UC1, cv::Scalar(0));
    // Creating a Transform object from GPS and IMU data
    // Assuming GPS provides latitude, longitude, and altitude
    // and IMU provides orientation (roll, pitch, yaw)
    rtabmap::Transform pose = rtabmap::Transform::fromGPS(data.gpsData.latitude,
data.gpsData.longitude, data.gpsData.altitude,
data.imuData.roll, data.imuData.pitch,
data.imuData.yaw);
    // Constructing the RTAB-Map sensor data object
    return rtabmap::SensorData(dummyImage, pose, data.distance, data.temperature);}
void MapManager::markPointAsMine(double x, double y, double z) {
    MapPoint newMinePoint;
    newMinePoint.x = x;
    newMinePoint.y = y;
    newMinePoint.z = z;
    newMinePoint.isMine = true;
    int newMineId = minePoints_.size() + 1;
    minePoints_[newMineId] = newMinePoint;}
void MapManager::saveMap(const std::string& filePath) {
    rtabmap_.write(filePath);
    // Additional code to handle the saving process, including mine locations}
std::vector<MapPoint> MapManager::getMines() const {
    std::vector<MapPoint> mines;
    for (const auto& kv : minePoints_) {
```

```
            mines.push_back(kv.second);}
    return mines;}
#ifndef MAPMANAGER_H
#define MAPMANAGER_H
#include "SensorManager.h"
#include <rtabmap/core/Rtabmap.h>
#include <rtabmap/utilite/UEventsManager.h>
#include <map>
#include <vector>
struct MapPoint {
    double x, y, z;
    bool isMine;};
class MapManager {
public:
    MapManager();
    ~MapManager();
    void processSensorData(const SensorData& data);
    void markPointAsMine(double x, double y, double z);
    void saveMap(const std::string& filePath);
    std::vector<MapPoint> getMines() const;
private:
    rtabmap::Rtabmap rtabmap_;
    std::map<int, MapPoint> minePoints_; // Stores the mines' locations
    void updateMapWithSensorData(const SensorData& data);
    // Convert SensorData to RTAB-Map compatible format
    rtabmap::SensorData convertToRtabmapSensorData(const SensorData& data);};
#endif // MAPMANAGER_H
#include "NeuralNetworkInterface.h"
#include "SensorManager.h"
// Припустимо, що ми використовуємо  Lite
#include "/lite/interpreter.h"
#include "/lite/model.h"
#include "/lite/kernels/register.h"
std::unique_ptr<tflite::Interpreter> interpreter;
void NeuralNetworkInterface::initialize() {
    // Завантаження моделі  Lite
    auto model = tflite::FlatBufferModel::BuildFromFile("model.tflite");
    tflite::ops::builtin::BuiltinOpResolver resolver;
    tflite::InterpreterBuilder builder(model, resolver);
    builder(&interpreter);
    interpreter->AllocateTensors();}
bool NeuralNetworkInterface::analyzeData(const SensorData& data) {
    // Підготовка вхідних даних для нейронної мережі
    float input = interpreter->typed_input_tensor<float>(0);
    // Заповнення вхідних даних (припустимо, що вони одновимірні)
    input[0] = data.distance;
    input[1] = data.temperature;
    input[2] = data.metalDetected ? 1.0 : 0.0;
    // Виконання висновку
    interpreter->Invoke();
    // Отримання вихідних даних від нейронної мережі
    float output = interpreter->typed_output_tensor<float>(0);
```

```
    return output[0] > 0.5; // Припустимо, що вихід - це ймовірність наявності міни}
#ifndef NEURALNETWORKINTERFACE_H
#define NEURALNETWORKINTERFACE_H
#include "SensorManager.h"
class NeuralNetworkInterface {
public:
    static void initialize();
    static bool analyzeData(const SensorData& data);};
#endif // NEURALNETWORKINTERFACE_H
#include "RobotController.h"
#include <wiringPi.h>
#include <chrono>
#include <thread>
// GPIO pins for motor control (these values should be set according to your hardware configuration)
const int MOTOR_LEFT_FORWARD = 1;
const int MOTOR_LEFT_BACKWARD = 2;
const int MOTOR_RIGHT_FORWARD = 3;
const int MOTOR_RIGHT_BACKWARD = 4;
void RobotController::initialize() {
    wiringPiSetup();  // Initialize wiringPi
    pinMode(MOTOR_LEFT_FORWARD, OUTPUT);
    pinMode(MOTOR_LEFT_BACKWARD, OUTPUT);
    pinMode(MOTOR_RIGHT_FORWARD, OUTPUT);
    pinMode(MOTOR_RIGHT_BACKWARD, OUTPUT);
    stop();  // Ensure motors are stopped initially}
void RobotController::moveForwardIncrementally(double distance) {
    // Assuming distance is in meters and robot moves at a speed of 0.5 m/s
    // Calculate the time to move the given distance
    int moveTimeMs = static_cast<int>((distance / 0.5)  1000);
    digitalWrite(MOTOR_LEFT_FORWARD, HIGH);
    digitalWrite(MOTOR_RIGHT_FORWARD, HIGH);
    std::this_thread::sleep_for(std::chrono::milliseconds(moveTimeMs));
    stop();}
void RobotController::moveBackwardIncrementally(double distance) {
    // Similar to moveForwardIncrementally
    int moveTimeMs = static_cast<int>((distance / 0.5)  1000);
    digitalWrite(MOTOR_LEFT_BACKWARD, HIGH);
    digitalWrite(MOTOR_RIGHT_BACKWARD, HIGH);
    std::this_thread::sleep_for(std::chrono::milliseconds(moveTimeMs));
    stop();}
void RobotController::turnLeftIncrementally(double angle) {
    // Assuming the robot turns at 90 degrees/sec
    // Calculate the time to turn the given angle
    int turnTimeMs = static_cast<int>((angle / 90)  1000);
    digitalWrite(MOTOR_LEFT_BACKWARD, HIGH);
    digitalWrite(MOTOR_RIGHT_FORWARD, HIGH);
    std::this_thread::sleep_for(std::chrono::milliseconds(turnTimeMs));
    stop();}
void RobotController::turnRightIncrementally(double angle) {
    // Similar to turnLeftIncrementally
    int turnTimeMs = static_cast<int>((angle / 90)  1000);
```

```
    digitalWrite(MOTOR_LEFT_FORWARD, HIGH);
    digitalWrite(MOTOR_RIGHT_BACKWARD, HIGH);
    std::this_thread::sleep_for(std::chrono::milliseconds(turnTimeMs));
    stop();}
void RobotController::stop() {
    digitalWrite(MOTOR_LEFT_FORWARD, LOW);
    digitalWrite(MOTOR_LEFT_BACKWARD, LOW);
    digitalWrite(MOTOR_RIGHT_FORWARD, LOW);
    digitalWrite(MOTOR_RIGHT_BACKWARD, LOW);}
#ifndef ROBOTCONTROLLER_H
#define ROBOTCONTROLLER_H
class RobotController {
public:
    static void initialize();
    static void moveForwardIncrementally(double distance);
    static void moveBackwardIncrementally(double distance);
    static void turnLeftIncrementally(double angle);
    static void turnRightIncrementally(double angle);
    static void stop();
    // Additional methods for continuous movement can be added as needed};
#endif // ROBOTCONTROLLER_H
#include "MLX90640_API.h"
#include "SensorManager.h"
#include "TinyGPS++.h"  // Library for Neo-6M GPS Module
#include "MPU6050_tockn.h"  // Library for MPU6050 IMU Sensor
#include <wiringPi.h>
#include <Wire.h>  // I2C library for communication
#include <chrono>
#include <mcp3008.h>
const int MLX90640_ADDRESS = 0x33; // Стандартна I2C адреса для MLX90640
TinyGPSPlus gps;  // Object for GPS Module
MPU6050 mpu6050(Wire);  // Object for MPU6050
// Ультразвуковий сенсор HC-SR04
const int TRIG_PIN = 23;
const int ECHO_PIN = 24;
// АЦП MCP3008 для металодетектора
const int SPI_CHANNEL = 0;
const int METAL_DETECTOR_CHANNEL = 0;
const int MCP3008_BASE = 100;
std::atomic<bool> SensorManager::keepReading = false;
std::thread SensorManager::readingThread;
std::function<void(const SensorData&)> SensorManager::updateCallback;
// Ініціалізація GPIO та інших ресурсів
void SensorManager::initialize() {
    wiringPiSetup();
    pinMode(TRIG_PIN, OUTPUT);
    pinMode(ECHO_PIN, INPUT);
    mcp3008Setup(MCP3008_BASE, SPI_CHANNEL);
    MLX90640_Setup(MLX90640_ADDRESS);
    // GPS module initialization (assumes a serial connection)
    Serial.begin(9600);  // the default baud rate for Neo-6M GPS Module
    // MPU6050 initialization
```

```
    Wire.begin();
    mpu6050.begin();
    mpu6050.calcGyroOffsets(true);  // calibrate MPU6050}
GPSData SensorManager::getGPSData() {
    GPSData data;
    while (Serial.available() > 0) {
        if (gps.encode(Serial.read())) {
        if (gps.location.isValid()) {
                data.latitude = gps.location.lat();
                data.longitude = gps.location.lng();
                data.altitude = gps.altitude.meters();}}}
    return data;}
IMUData SensorManager::getIMUData() {
    mpu6050.update();
    IMUData data;
    data.roll = mpu6050.getRoll();
    data.pitch = mpu6050.getPitch();
    data.yaw = mpu6050.getYaw();
    return data;}
// Читання даних з ультразвукового сенсора
double SensorManager::getUltrasonicDistance() {
    digitalWrite(TRIG_PIN, LOW);
    delayMicroseconds(2);
    digitalWrite(TRIG_PIN, HIGH);
    delayMicroseconds(10);
    digitalWrite(TRIG_PIN, LOW);
    while (digitalRead(ECHO_PIN) == LOW);
    auto startTime = std::chrono::high_resolution_clock::now();
    while (digitalRead(ECHO_PIN) == HIGH);
    auto endTime = std::chrono::high_resolution_clock::now();
    auto duration = std::chrono::duration_cast<std::chrono::microseconds>(endTime -
startTime).count();
    double distance = duration  0.034 / 2;
    return distance;}
// Читання даних з металодетектора
bool SensorManager::getMetalDetectorReading() {
    int value = analogRead(MCP3008_BASE + METAL_DETECTOR_CHANNEL);
    return value > 512;  // Припущення, що значення вище 512 означає виявлення металу}
double SensorManager::getThermalReading() {
    float emissivity = 0.95;
    uint16_t frame[834];
    MLX90640_GetFrameData(MLX90640_ADDRESS, frame);
    float temperatures[768]; // MLX90640 має матрицю 32x24 пікселів
    MLX90640_CalculateTo(MLX90640_ADDRESS, emissivity, frame, temperatures);
    double averageTemp = 0;
    for (int i = 0; i < 768; ++i) {
        averageTemp += temperatures[i];}
    averageTemp /= 768;
    return averageTemp;}
SensorData SensorManager::readSensors() {
    SensorData data;
    data.distance = getUltrasonicDistance();
```

```
    data.temperature = getThermalReading();
    data.metalDetected = getMetalDetectorReading();
    return data;}
void SensorManager::startContinuousReading() {
    keepReading = true;
    readingThread = std::thread(continuousReadingThread);}
void SensorManager::stopContinuousReading() {
    keepReading = false;
    if (readingThread.joinable()) {
        readingThread.join();}}
void SensorManager::setUpdateCallback(std::function<void(const SensorData&)> callback) {
    updateCallback = callback;}
void SensorManager::continuousReadingThread() {
    while (keepReading) {
        SensorData data = readSensors();
        if (updateCallback) {
        updateCallback(data);}
      std::this_thread::sleep_for(std::chrono::milliseconds(100)); // Adjust delay as needed}}
#ifndef SENSORMANAGER_H
#define SENSORMANAGER_H
#include <vector>
#include <thread>
#include <atomic>
#include <functional>
struct SensorData {
    // ... Existing sensor data structure ...};
// Assuming a simple struct for GPS data
struct GPSData {
    double latitude;
    double longitude;
    double altitude;};
// Assuming a simple struct for IMU data
struct IMUData {
    double roll;
    double pitch;
    double yaw;};
struct SensorData {
    double distance;
    double temperature;
    bool metalDetected;
    GPSData gpsData;
    IMUData imuData;};
class SensorManager {
public:
    static void initialize();
    static SensorData readSensors();
    static void startContinuousReading();
    static void stopContinuousReading();
    static void setUpdateCallback(std::function<void(const SensorData&)> callback);
private:
    double getUltrasonicDistance();
    bool getMetalDetectorReading();
```

```
    double getThermalReading();
    GPSData getGPSData();
    IMUData getIMUData();
    static std::atomic<bool> keepReading;
    static std::thread readingThread;
    static void continuousReadingThread();
    static std::function<void(const SensorData&)> updateCallback;};
#endif                                  //                                  SENSORMANAGER_H
```

**ДОДАТОК Б**

ДЕМОНСТРАЦІЙНИЙ МАТЕРІАЛ

| Формат | Позиція. | ПОЗНАЧЕННЯ | НАЙМЕНУВАННЯ | Кільк. | Примітки |
|---|---|---|---|---|---|
| | | | | | |
| А4 | 1 | ГЮІК 464430.016 ПЗ | Текстові документи | | |
| | | | Пояснювальна записка | 140 | с. |
| | | | | | |
| | | | | | |
| | | | Інші документи | | |
| | 2 | | Презентаційний матеріал | 12 | с. |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |

| Змін. | Арк | Номер докум. | Підпис | Дата | ГЮІК 464430.016 ВД | | | |
|---|---|---|---|---|---|---|---|---|
| Розроб. | | Заїкін В. О. | | | Удосконалення модуля конструкції робототехнічної системи для пошуку та ідентифікації вибухонебезпечних предметів комбінованим методом<br>Відомість кваліфікаційної роботи | Літера | Арку | Аркуш |
| Перевір. | | Янушкевич Д. А. | | | | Н | 1 | 1 |
| | | | | | | Кафедра КІТАР ХНУРЕ | | |
| Н.контр. | | Бронніков А.І. | | | | | | |
| Затв. | | Невлюдов І. Ш. | | | | | | |