Ministry of Education and Research of the Republic of Moldova
Technical University of Moldova
Department of Software and Automation Engineering

REPORT

Laboratory work No. 3.2

Discipline: Embedded Systems

Student: Revenco Victor, FAF - 221

Checked: asist. univ., Martiniuc A.

Chișinău 2025

Analysis of the Situation in the Field

1. Description of the Technologies Used and Application Context

This project implements a modular system for acquiring distance measurements from an ultrasonic sensor, processing the signals, filtering the data using salt and pepper filter + weighted average filter, and displaying the data on Serial terminal. The system uses FreeRTOS for task scheduling and STDIO for formatted output.

Hardware Components

- **Microcontroller**
- **Jumper Wires**
- **USB Power Supply**
- **Ultrasonic Sensor**

Software Components

- **PlatformIO with Visual Studio Code**: Integrated Development Environment (IDE)
- **C++ for Embedded Systems**: Programming language used for implementation
- **STDIO Functions (printf)**: For system reporting and monitoring
- **Serial Monitor**: For displaying system status

2. System Architecture Explanation and Solution Justification

The application is organized into several modules:

- **Main Module**: Sets up the system, creates FreeRTOS tasks, and initializes modules
- **Ultrasonic Sensor Module**: Handles initialization and reading from the ultrasonic sensor
- **Signal Proccesing Module**: Applies salt&pepper + weighted average filter
- **Serial I/O Module**: Configures STDIO for serial communication

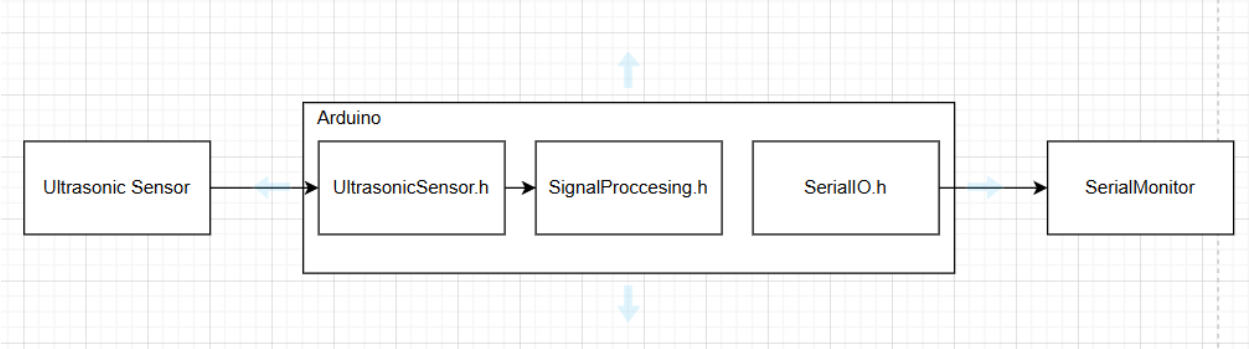
3. Case Study

The ultrasonic sensor can be used to measure the distance between the vehicle and nearby objects, like walls or other vehicles. The project's digital filters can help remove noise from sensor data, ensuring that the distance readings are accurate even in noisy environments (e.g., crowded parking lots).

4. Design

1. Architectural Sketch and Component Interconnection

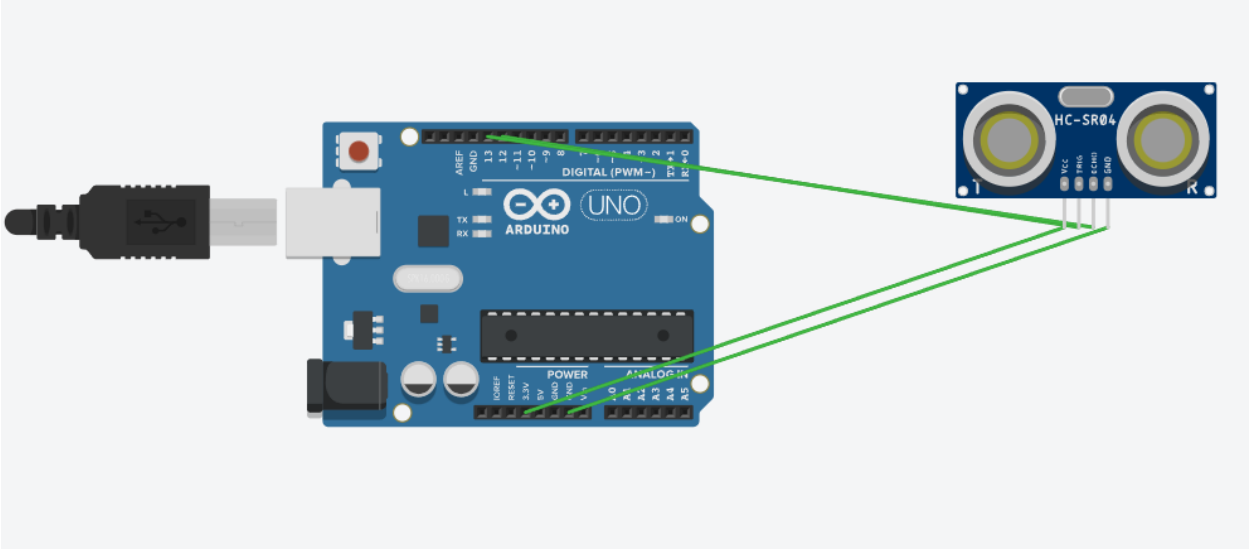
- Ultrasonic sensor connected to MCU
- Microcontroller executing sequential task scheduling
- Serial Monitor for system status output



Component Description and Their Roles:

- **Microcontroller**: Core processing unit executing tasks
- **Ultrasonic Sensor**: Registers the distance to any object infront of it

The sketch shows how the components interact.



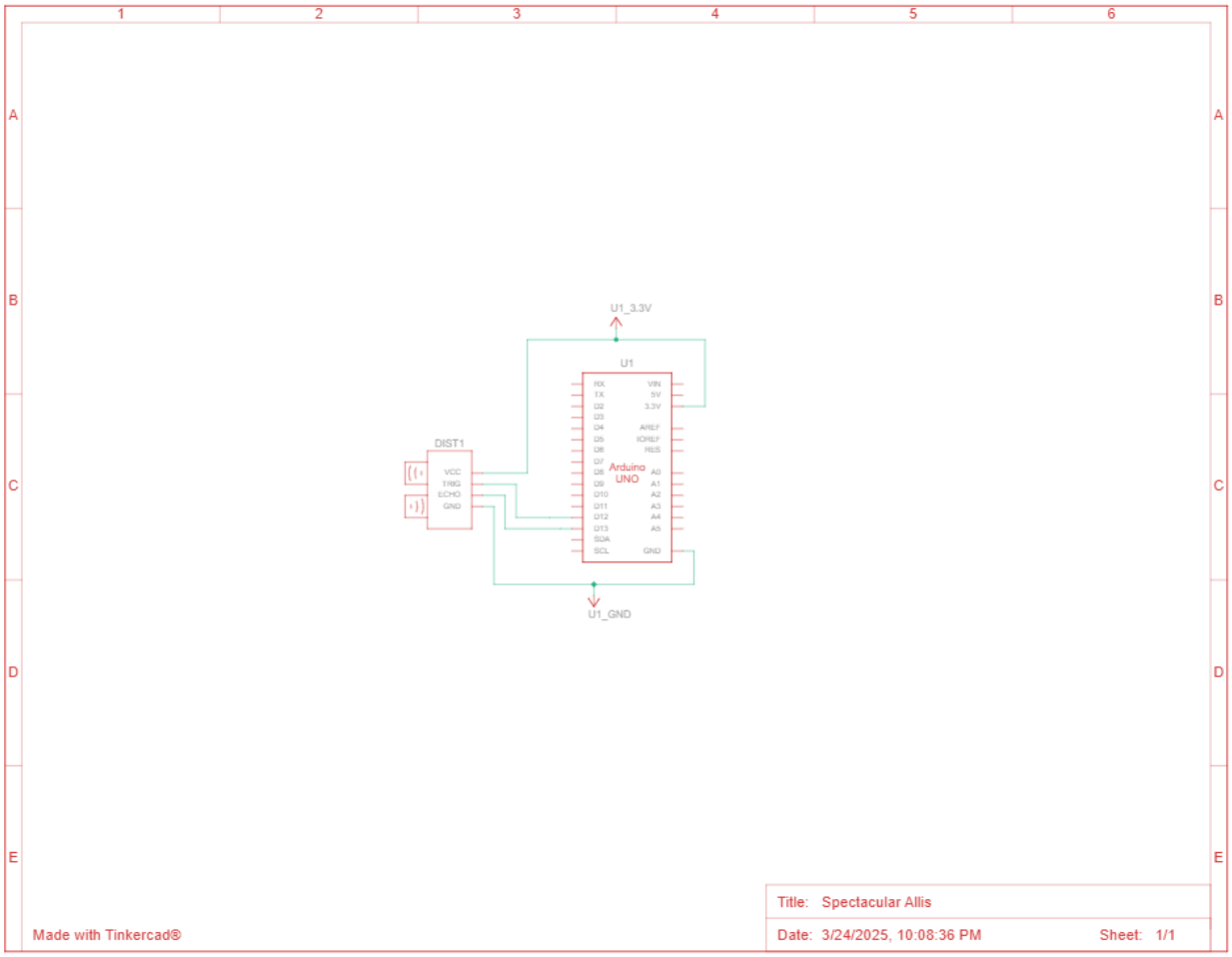


Figure 1, 2 Electrical schematic

This image shows an **Arduino Uno** connected to a ultrasonic sensor.

Here's a breakdown of the components and their connections:

1. Microcontroller (Arduino Uno)

- The **Arduino Uno** is the main processing unit, providing power and controlling the circuit.

2. Ultrasonic Sensor

- The sensor is used to measure the distance to any object its pointed at
- The VCC pin is connected to 3.5V

- The TRIG pin is connected to pin 12
- The ECHO pin is connected to pin 13
- The GND is connected to any GND pin on the MCU

2. Schematic diagrams

To understand the system's behavior, a Flowchart and a Finite State Machine (FSM) are used.

Flowchart – Serial Command Processing

The FSM includes the following states:

1. Normal – default state when distance is above threshold
2. Alert – entered when distance below threshold

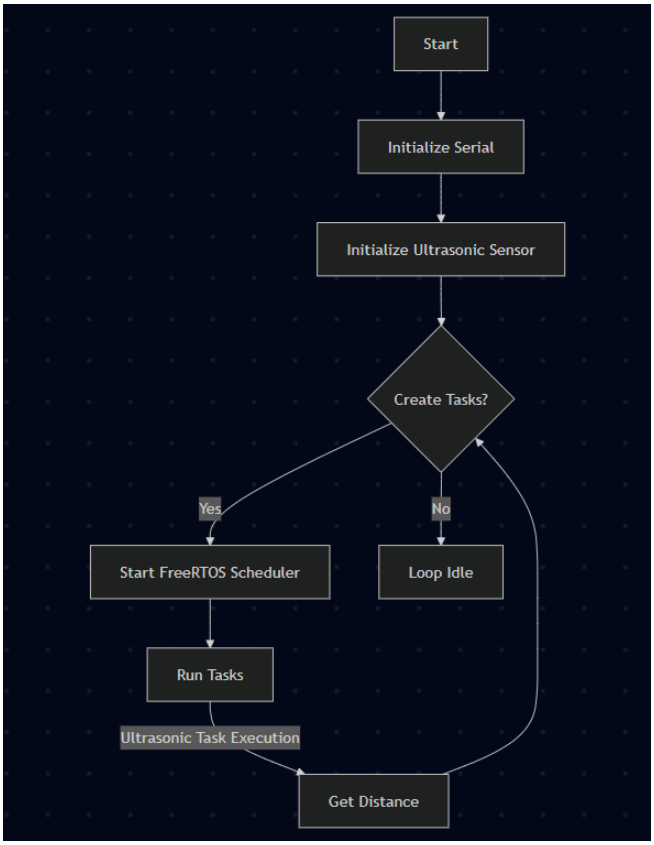


Figure 3 System

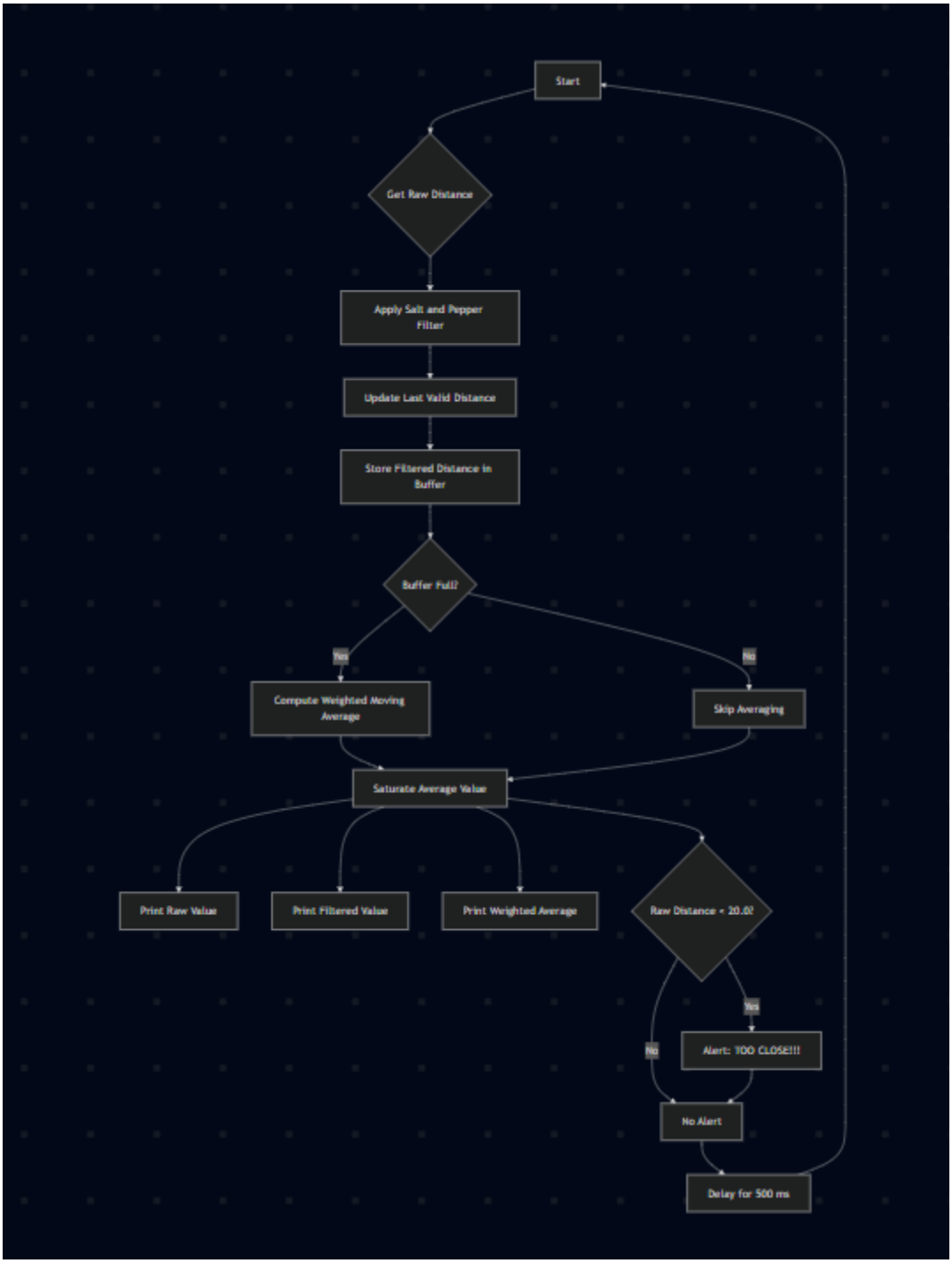


Figure 4 Ultrasonic task

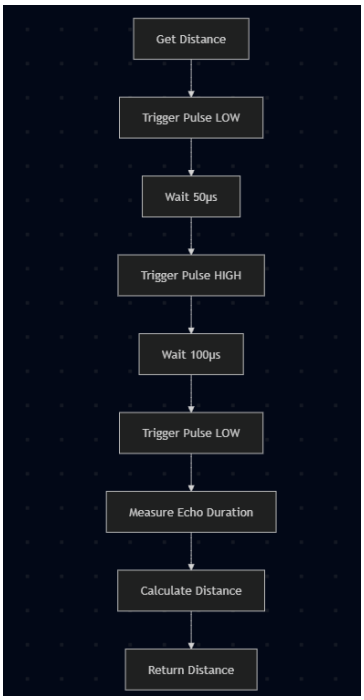


Figure 5 Distance measurement

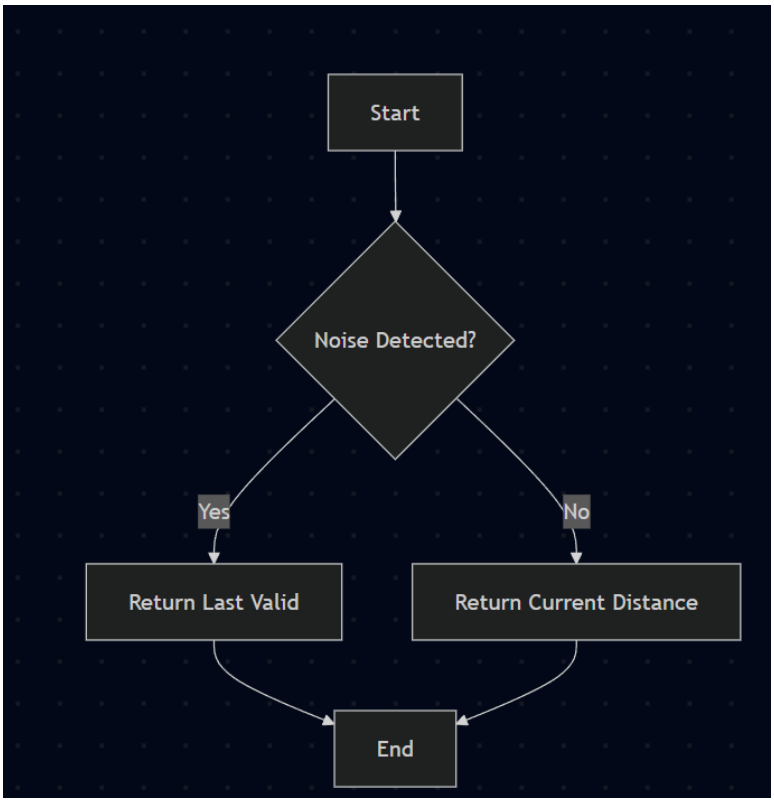


Figure 6 Salt&pepper filter

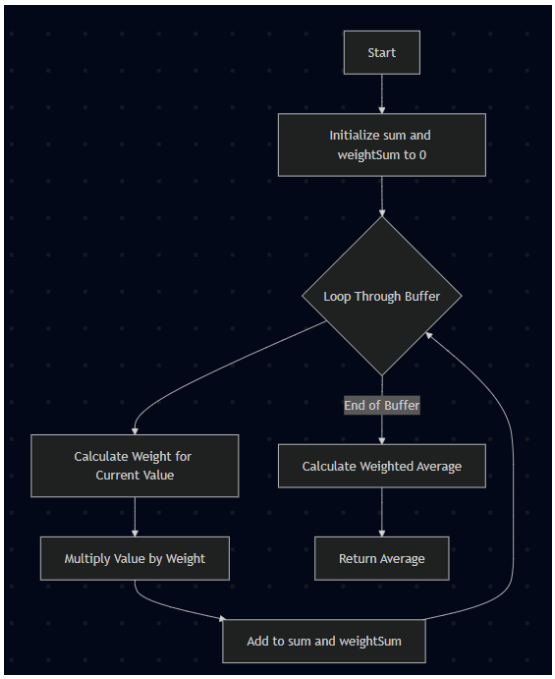


Figure 7 WeightedMovingAverage filter

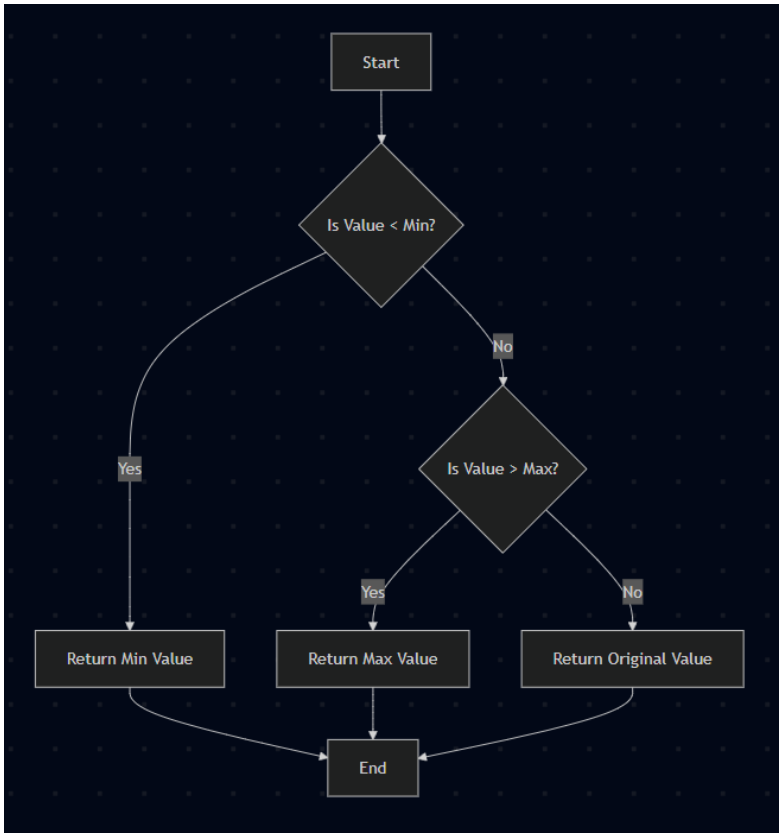


Figure 8 SaturateValue

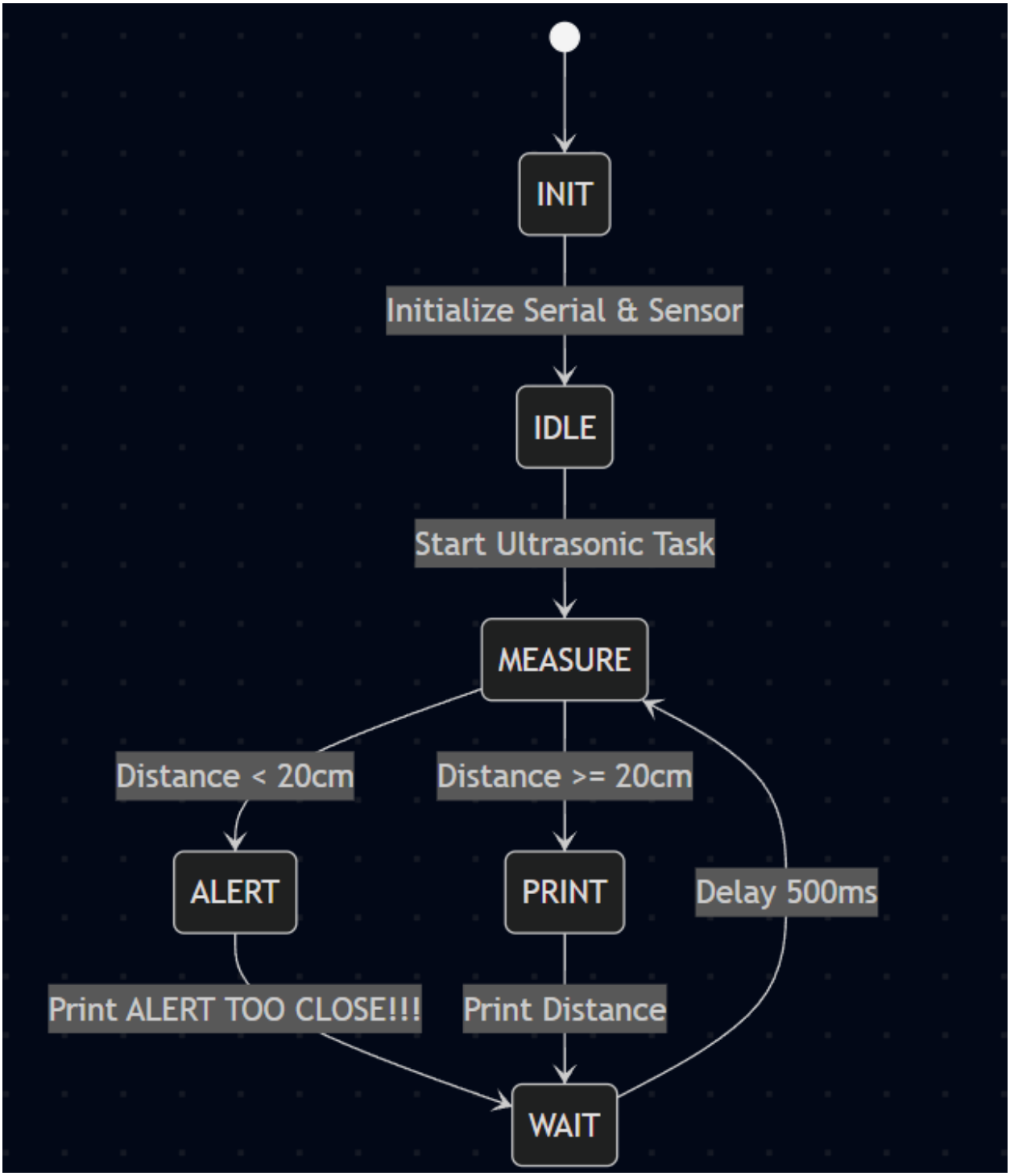


Figure 9 FSM diagram

3. Modular implementation

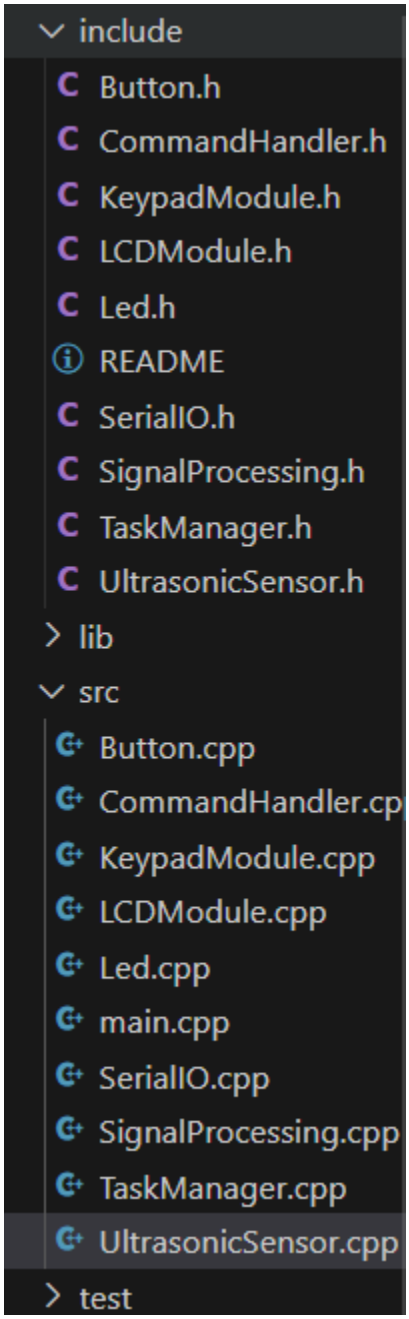


Figure 10 Project organization

In the SerialIO.h we have the following methods for reading and writing the text:

1. serial_putc(char c, FILE *stream) – it uses Serial.write to write the text

2. serial_getc(FILE *stream) – it uses Serial.read serial is not available
3. printDistance(float distance) – it prints the distance registered by sensor

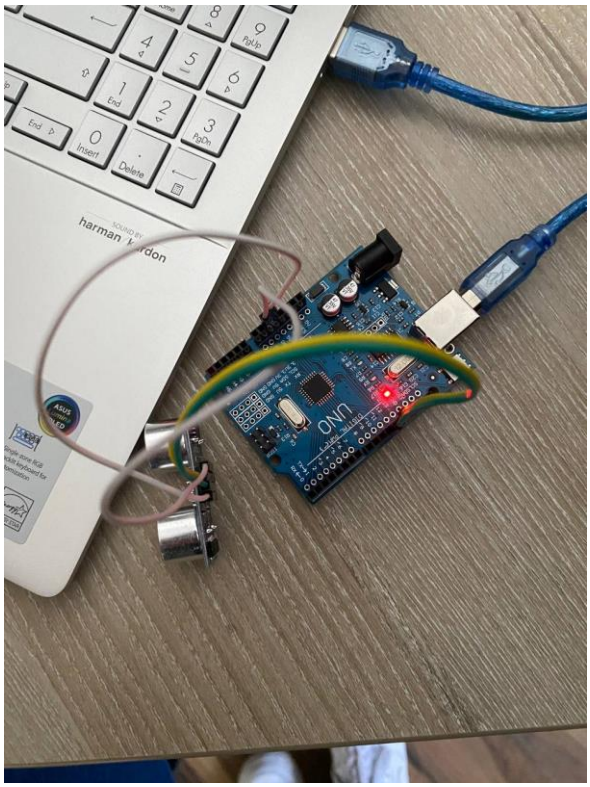
In the UltrasonicSensor.h we have the following methods:

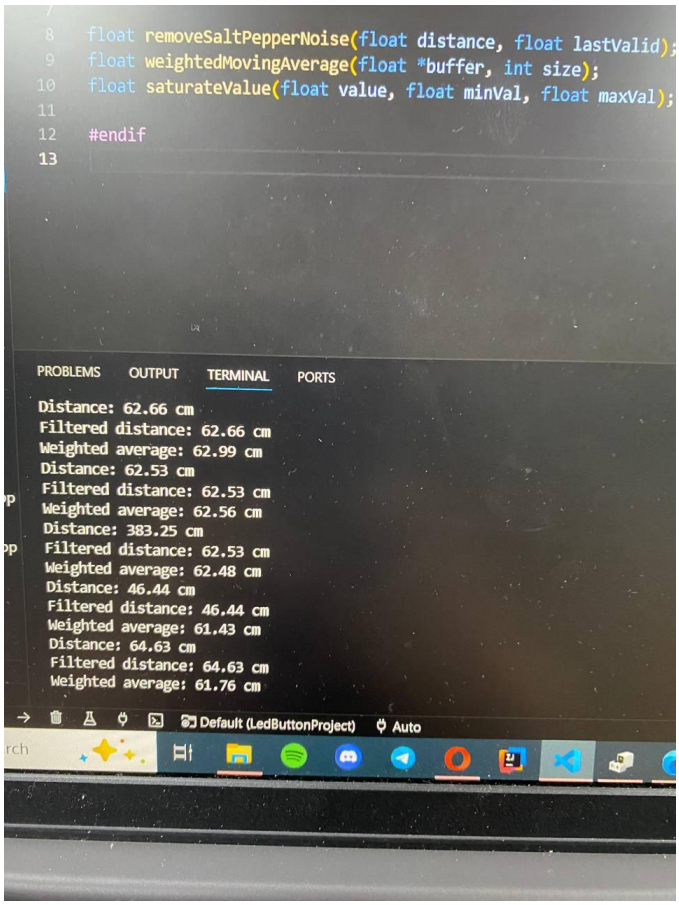
1. ultrasonicInit() – for initialization
2. getDistance() – to read the distance from the object the sensor is pointed at

In the SignalProcessing.h we have the following methods:

1. removeSaltPepperNoise(float distance, float lastValid) – remove spikes in data
2. weightedMovingAverage(float *buffer, int size) – smooth the signal by using average value
3. saturateValue(float value, float minVal, float maxVal) – limit range of possible values

Results





Conclusions

In this laboratory exercise, we successfully implemented an ultrasonic sensor system using FreeRTOS to measure distances in real-time. The system was designed to continuously monitor the distance measured by the sensor, with the ability to trigger alerts when the distance falls below a predefined threshold (20 cm).

This lab helped in understanding the application of real-time operating systems in embedded systems and reinforced key concepts such as multitasking, sensor integration, and conditional logic in a system design. The project successfully demonstrated the efficiency of using FreeRTOS in resource-constrained environments like Arduino boards.

Bibliography

1. Official Arduino Documentation
    - Arduino Reference – Serial Communication https://www.arduino.cc/reference/en/#communication

- Arduino Mega 1280 Pinout & Datasheet https://docs.arduino.cc/hardware/mega-1280

2. PlatformIO Official Documentation
   - PlatformIO for Arduino Development https://docs.platformio.org/en/latest/platforms/atmelavr.html
3. TUM Courses
   - Introducere în Sistemele Embedded și Programarea Microcontrolerelor
   - Principiile comunicației seriale și utilizarea interfeței UART

Appendix

1. **GitHub**: https://github.com/Kipitokisk/SI_Lab

```
#include <Arduino.h>
#include "Arduino_FreeRTOS.h"
#include "task.h"
#include "UltrasonicSensor.h"
#include "SerialIO.h"
#include "SignalProcessing.h"

void ultrasonicTask(void *pvParameters) {
  TickType_t xLastWakeTime = xTaskGetTickCount();
  const int bufferSize = 5;
  float readings[bufferSize] = {0};
  int index = 0;
  float lastValid = 0;

  for (;;) {
      float rawDistance = getDistance();
      float filteredDistance = removeSaltPepperNoise(rawDistance, lastValid);
      lastValid = filteredDistance;
      readings[index] = filteredDistance;
      index = (index + 1) % bufferSize;

      float avgDistance = weightedMovingAverage(readings, bufferSize);
      avgDistance = saturateValue(avgDistance, MIN_VALID_DISTANCE,
MAX_VALID_DISTANCE);

      printf("Raw Value: %.2f cm\n", rawDistance);
      printf("Salt & Pepper Filtered Value: %.2f cm\n", filteredDistance);
      printf("Weighted Average: %.2f cm\n", avgDistance);
```

```
      if (rawDistance < 20.0) {
          printf("ALERT: TOO CLOSE!!!\n");
      }

      vTaskDelayUntil(&xLastWakeTime, pdMS_TO_TICKS(500));
  }
}



void setup() {
    serialInit();
    ultrasonicInit();
    xTaskCreate(ultrasonicTask, "Ultrasonic", 128, NULL, 1, NULL);
}

void loop() {
}


#include "UltrasonicSensor.h"

void ultrasonicInit() {
    pinMode(TRIG_PIN, OUTPUT);
    pinMode(ECHO_PIN, INPUT);
}

float getDistance() {
    digitalWrite(TRIG_PIN, LOW);
    delayMicroseconds(50);
    digitalWrite(TRIG_PIN, HIGH);
    delayMicroseconds(100);
    digitalWrite(TRIG_PIN, LOW);

    long duration = pulseIn(ECHO_PIN, HIGH);
    return duration * 0.034 / 2;
}


#include "SignalProcessing.h"

float removeSaltPepperNoise(float distance, float lastValid) {
    if ((distance - lastValid > NOISE_THRESHOLD) || (lastValid - distance >
NOISE_THRESHOLD)) {
        return lastValid;
```

```
    }
    return distance;
}

float weightedMovingAverage(float *buffer, int size) {
    float sum = 0;
    float weightSum = 0;
    for (int i = 0; i < size; i++) {
        float weight = i + 1;
        sum += buffer[i] * weight;
        weightSum += weight;
    }
    return sum / weightSum;
}

float saturateValue(float value, float minVal, float maxVal) {
    if (value < minVal) return minVal;
    if (value > maxVal) return maxVal;
    return value;
}


#include "SerialIO.h"

int serial_putchar(char c, FILE* f) {
    Serial.write(c);
    return c;
}

int serial_getchar(FILE* f) {
    while (!Serial.available());
    return Serial.read();
}

FILE serial_stdout;

void serialInit() {
    Serial.begin(115200);
    while (!Serial);

    fdev_setup_stream(&serial_stdout, serial_putchar, serial_getchar,
_FDEV_SETUP_WRITE);
    stdout = &serial_stdout;
    stdin = &serial_stdout;
}
```

```
void printDistance(float distance) {
    char buffer[10];
    dtostrf(distance, 5, 2, buffer);
    printf("Distance: %s cm\n", buffer);
}
```