Spring
2025

Project Report (LFR)

SABIR JAN [FA20-BCE-075]
MUHAMMAD UBAIDULLAH [FA23-BCE-083]

SUBMITTED TO : | Dr. Ifranullah

Project Report (LFR)

Task # 1 :

Make a LFR that follow the line (black & 2.5cm) and execute a rescue mission .

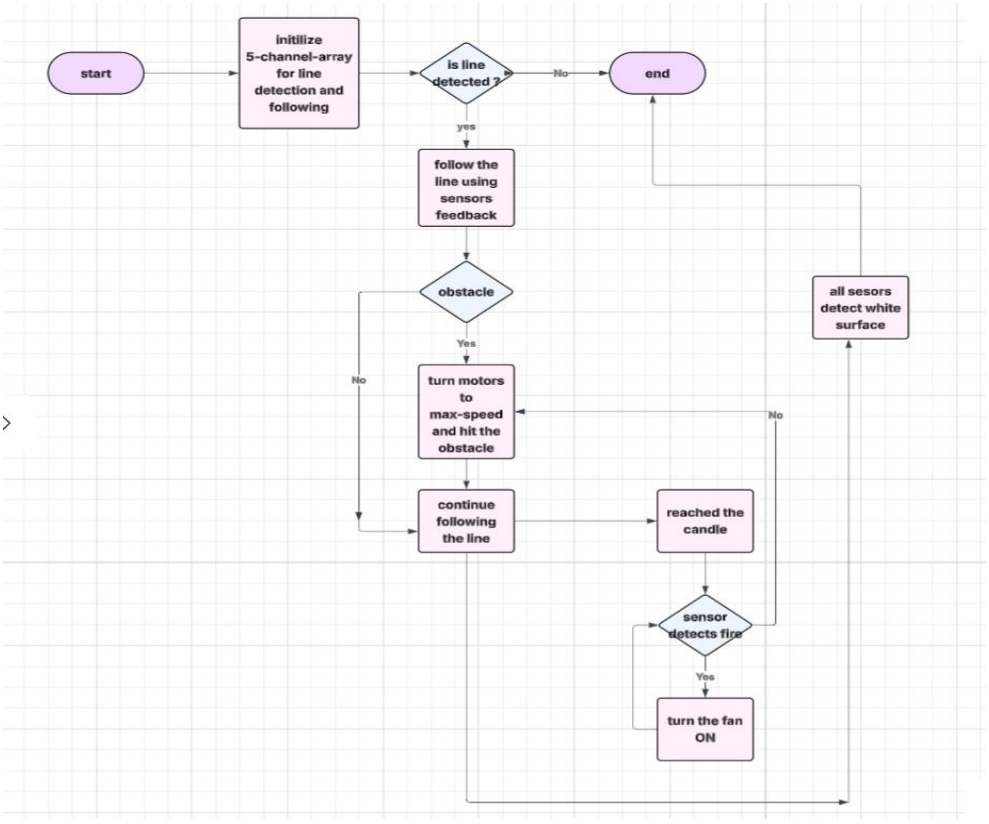
Mission :

- Robot should follow the black line and should reach a certain point (where candles are located) and should turn on the fan to blow off the fire .
- And in its journey if it come across something (obstacle) it should hit it and through it away .

Components list :

1. DC motors (4)
2. Fan
3. Ultrasonic sensors
4. Flame sensor
5. 5 channel array (TCRT5000)
6. L298
7. Boost converter
8. Batteries (3) + holder
9. On/off button
10. NPN transistor
11. Jumper wires
12. Arduino NANO

Flowchart :



Code :

```
#define F_CPU 16000000UL
#include <avr/io.h>
#include <util/delay.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>

// UART
#define BAUD 9600
#define MYUBRR F_CPU/16/BAUD-1

// Constants
#define NUM_SENSORS 5
#define OBSTACLE_DISTANCE_CM 10
#define BASE_SPEED 100
#define MAX_SPEED 180
#define Kp 10
#define Kd 0

// Motor Pins
#define LEFT_PWM OCR0A  // PD6
#define RIGHT_PWM OCR0B // PD5
#define LEFT_IN1 PD2
#define LEFT_IN2 PD3
#define RIGHT_IN1 PD4
#define RIGHT_IN2 PD7

// Fan (connected to D10)
#define FAN_DDR DDRB
#define FAN_PORT PORTB
#define FAN_PIN PB2   // D10
```

```
// Ultrasonic
#define US_TRIG_PORT PORTB
#define US_TRIG_DDR DDRB
#define US_TRIG_PIN PB1   // D9

#define US_ECHO_PIN PB0   // D8
#define US_ECHO_DDR DDRB
#define US_ECHO_PIN_REG PINB

// UART Functions
void USART_Init(unsigned int ubrr) {
    UBRR0H = (unsigned char)(ubrr >> 8);
    UBRR0L = (unsigned char)ubrr;
    UCSR0B = (1 << TXEN0);
    UCSR0C = (1 << UCSZ01) | (1 << UCSZ00);
}

void USART_Transmit(unsigned char data) {
    while (!(UCSR0A & (1 << UDRE0)));
    UDR0 = data;
}

void USART_Print(const char* str) {
    while (*str) USART_Transmit(*str++);
}

void USART_Println(const char* str) {
    USART_Print(str);
    USART_Transmit('\n');
}
```

```
// PWM / Motor Control
void PWM_Init() {
    DDRD |= (1 << PD6) | (1 << PD5);
    TCCR0A |= (1 << COM0A1) | (1 << COM0B1) | (1 << WGM00) | (1 << WGM01);
    TCCR0B |= (1 << CS01);
}

void setLeftMotor(int speed) {
    if (speed >= 0) {
        PORTD |= (1 << LEFT_IN1);
        PORTD &= ~(1 << LEFT_IN2);
        } else {
        PORTD &= ~(1 << LEFT_IN1);
        PORTD |= (1 << LEFT_IN2);
        speed = -speed;
    }
    if (speed > MAX_SPEED) speed = MAX_SPEED;
    LEFT_PWM = speed;
}

void setRightMotor(int speed) {
    if (speed >= 0) {
        PORTD |= (1 << RIGHT_IN1);
        PORTD &= ~(1 << RIGHT_IN2);
        } else {
        PORTD &= ~(1 << RIGHT_IN1);
        PORTD |= (1 << RIGHT_IN2);
        speed = -speed;
    }
    if (speed > MAX_SPEED) speed = MAX_SPEED;
    RIGHT_PWM = speed;
}
```

```
void setMotorSpeed(int left, int right) {
    setLeftMotor(left);
    setRightMotor(right);
}

// Ultrasonic Sensor
uint16_t measureDistanceCM() {
    US_TRIG_PORT &= ~(1 << US_TRIG_PIN);
    _delay_us(2);
    US_TRIG_PORT |= (1 << US_TRIG_PIN);
    _delay_us(10);
    US_TRIG_PORT &= ~(1 << US_TRIG_PIN);

    uint32_t count = 0;

    while (!(US_ECHO_PIN_REG & (1 << US_ECHO_PIN))) {
        count++;
        if (count > 60000) return 999; // timeout
    }

    TCNT1 = 0;
    TCCR1B = (1 << CS11);
    while (US_ECHO_PIN_REG & (1 << US_ECHO_PIN));
    TCCR1B = 0;

    uint16_t duration = TCNT1;
    uint16_t distance = duration / 58;
    return distance;
}
```

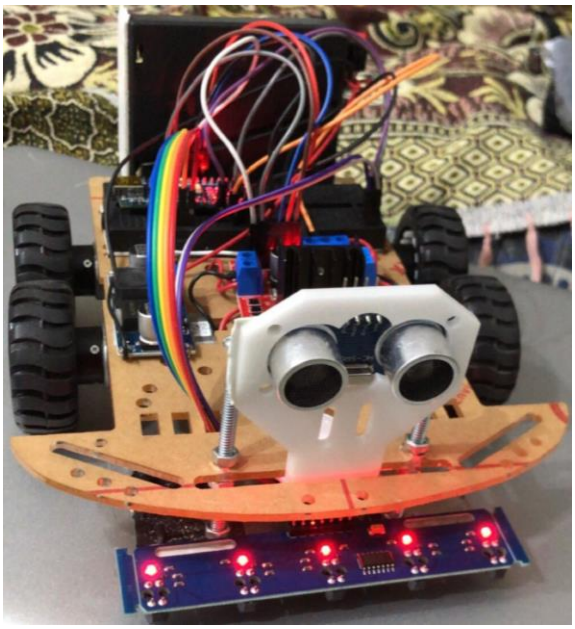
```
// Main Program
int main(void) {
    USART_Init(MYUBRR);
    PWM_Init();

    // Direction pins as output
    DDRD |= (1 << LEFT_IN1) | (1 << LEFT_IN2) | (1 << RIGHT_IN1) | (1 << RIGHT_IN2);

    // Fan pin output
    FAN_DDR |= (1 << FAN_PIN);

    // Ultrasonic pins
    US_TRIG_DDR |= (1 << US_TRIG_PIN);   // Trigger output
    US_ECHO_DDR &= ~(1 << US_ECHO_PIN);  // Echo input

    // Digital line sensor input pins (PD0 to PD4)
    DDRD &= ~((1 << PD0) | (1 << PD1) | (1 << PD2) | (1 << PD3) | (1 << PD4));

    int16_t lastError = 0;

    while (1) {
        uint16_t distance = measureDistanceCM();

        if (distance <= OBSTACLE_DISTANCE_CM) {
            // Instead of stopping, hit the obstacle!
            setMotorSpeed(MAX_SPEED, MAX_SPEED);
            USART_Println("?? Obstacle Detected! CHARGE FORWARD");
            _delay_ms(500); // Hit duration
            continue;
        }
```

```
// Read digital line sensors
uint8_t sensorBinary[NUM_SENSORS];
sensorBinary[0] = !(PIND & (1 << PD0));
sensorBinary[1] = !(PIND & (1 << PD1));
sensorBinary[2] = !(PIND & (1 << PD2));
sensorBinary[3] = !(PIND & (1 << PD3));
sensorBinary[4] = !(PIND & (1 << PD4));

uint8_t sumBinary = 0;
for (int i = 0; i < NUM_SENSORS; i++) sumBinary += sensorBinary[i];

if (sumBinary == 0) {
    setMotorSpeed(0, 0);
    USART_Println("STOP: No Line");
    lastError = 0;
    _delay_ms(50);
    continue;
}

if (sensorBinary[0]) {
    setMotorSpeed(-BASE_SPEED, BASE_SPEED);
    USART_Println("Turn: SHARP LEFT");
    lastError = -100;
    continue;
}
if (sensorBinary[4]) {
    setMotorSpeed(BASE_SPEED, -BASE_SPEED);
    USART_Println("Turn: SHARP RIGHT");
    lastError = 100;
    continue;
}
```

```
if (sensorBinary[1]) {
    setMotorSpeed(0, BASE_SPEED);
    USART_Println("Turn: SOFT LEFT");
    lastError = -50;
    continue;
}
if (sensorBinary[3]) {
    setMotorSpeed(BASE_SPEED, 0);
    USART_Println("Turn: SOFT RIGHT");
    lastError = 50;
    continue;
}
if (sensorBinary[2]) {
    setMotorSpeed(200, 200);
    USART_Println("STRAIGHT");
    lastError = 0;
    continue;
}

// Optional PD Control (for fine adjustments)
int8_t weights[NUM_SENSORS] = {-2, -1, 0, 1, 2};
int32_t weighted_sum = 0;
int16_t count = 0;

for (int i = 0; i < NUM_SENSORS; i++) {
    if (sensorBinary[i]) {
        weighted_sum += weights[i];
        count++;
    }
```

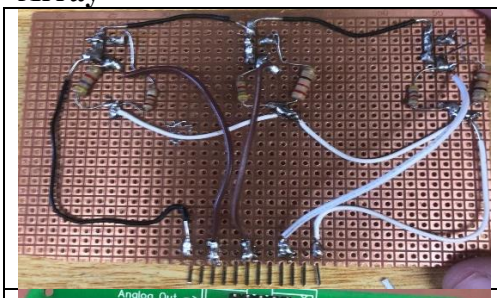
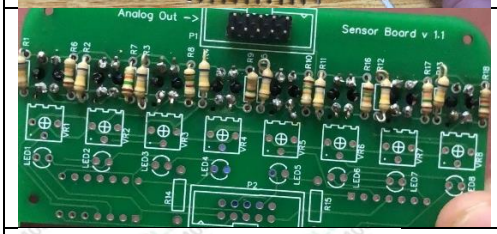
```
    int16_t error = (count > 0) ? (weighted_sum / count) : 0;
    int16_t correction = Kp * error + Kd * (error - lastError);
    lastError = error;

    int16_t left = BASE_SPEED + correction;
    int16_t right = BASE_SPEED - correction;
    setMotorSpeed(left, right);

    char buf[32];
    sprintf(buf, "PD Error: %d", error);
    USART_Println(buf);

    _delay_ms(100);
}
```

Pics of the LFR :



Problems we faced and their solutions :

1. **Array**

	First we wanted to use our own made 3 channel array but it was broken in my laptop bag.
	Then we made this PCB using TCRT5000 but, • The readings of the sensors were wrong • We changed the sensors two times but still wrong readings
	So we left with no choice cause no time was left so we bought this 5-channel array from the market • But later it came to our knowledge that it was digital sensor and due to money issues we used this.

2. **Centered condition of LFR**
   In the center condition the both side motors should move with the base speed but, In our LFR

   Problem :

- When the center sensor detects the line only the right motors of the LFR move forward but left ones don't move.

   Solution :

- We consulted it with our classmates and some seniors but they also couldn't find out the solution .
- First we thought problem was related to the base speed but when we changed it, the problem still remained.
- We also consulted with ChatGPT but it also couldn't figure out the problem.
- So we left it as it remained ----------

Critical Analysis :

This LFR project integrates motor control, flame detection, and obstacle interaction using AVR microcontroller logic. A notable design decision is the **removal of ADC functionality**, replaced by digital-only sensor logic. While this simplifies hardware interfacing and reduces analog overhead, it limits precision—especially in flame detection, which now relies on binary thresholds rather than variable intensity measurement.

Another key feature is the **ultrasonic obstacle detection mechanism**. Initially designed to halt the system when an object is detected, the logic was revised to **drive forward intentionally**, allowing the robot to "push through" obstacles. This aggressive behavior suits specific applications like fire-fighting robots in cluttered environments but raises ethical and mechanical concerns in general contexts due to the potential for unintended damage or safety hazards.

Motor control using PWM and basic PD (Proportional-Derivative) error correction demonstrates a solid foundation in autonomous navigation. However, the lack of integral control (Ki) could result in long-term drift or steady-state errors in line following.

The system's architecture is modular and well-structured, but its reliance on fixed thresholds (e.g., flame detection or line sensor ADC levels) could be improved using adaptive calibration or sensor fusion. Overall, the code balances simplicity and functionality but has room for refinement in safety, flexibility, and control algorithms.