Spring 2025

Yeditepe University
Mechanical Engineering Department

ME 456

Mechatronics

Workshop #2

PWM Speed Control and Ultrasonic Radar

FINAL REPORT

Instructor

- Assoc. Dr. Koray Kadir Şafak

Lab Partners

- Abdullah Serdar Özdemir 20200705049
- Ömer Dolaş 20200705001
- Abdulkadir Salman 20200705027
- Furkan Yılmaztekin 20180705040

Due Date: 02/04/2025 Group Number: 5

Table Of Content

1. PWM Speed Control

- **Objective:** Our main goal in this task is to control the motors on the "MBot" platform, which we have actively used during the course, through different "PWM" signals. In this context, specific feedbacks in the lab manual were taken as reference and a code was prepared in the light of this feedback. The relevant feedbacks are detailed in Table-1.

Step	Duration	Left motor	Right motor
1	1 seconds	20% full speed forward	20% full speed forward
2	1 seconds	40% full speed forward	40% full speed forward
3	1 seconds	60% full speed forward	60% full speed forward
4	1 seconds	80% full speed forward	80% full speed forward
5	1 seconds	100% full speed forward	100% full speed forward
6	1 seconds	Stop	Stop
7	1 seconds	20% full speed reverse	20% full speed reverse
8	1 seconds	40% full speed reverse	40% full speed reverse
9	1 seconds	60% full speed reverse	60% full speed reverse
10	1 seconds	80% full speed reverse	80% full speed reverse
11	1 seconds	100% full speed reverse	100% full speed reverse
12	1 seconds	Stop	Stop

Table-1: PWM Control Feedbacks

- **Code and Explanation:** A code has been prepared for the process in the relevant table to be carried out optimally. The code and the explanations of the infrastructures in this code are detailed in the following section.

```
#include <MeMCore.h>

MeDCMotor motorLeft(M1);
MeDCMotor motorRight(M2);
```

⇑ *Library and Motor Definitions*

The MeMCore.h library is used for motor control. Left and right motors are connected to M1 and M2 ports.

```
// Motorları Çalıştır Aga
void moveMotors(int speed, int duration, bool forward) {
    if (forward) {
        motorLeft.run(-speed); // mBot'ta ters yön için negatif
        motorRight.run(speed);
    } else {
        motorLeft.run(speed);
        motorRight.run(-speed);
    }
    delay(duration);
}

// Motorları Stop Et Aga
void stopMotors(int duration) {
    motorLeft.stop();
    motorRight.stop();
    delay(duration);
}
```

⇑ *Motor Control Functions*

Moves the motors at a given speed and duration. The bool forward variable controls forward or backward movement.

```
void setup() {
    Serial.begin(115200);
}

void loop() {
    int speeds[5] = {80, 120, 160, 200, 255}; // Minimum hız 80'den başlıyor

    // Git Oğlum (5 adım)
    for (int i = 0; i < 5; i++) {
        moveMotors(speeds[i], 1000, true);
    }

    // Dur Oğlum
    stopMotors(1000);

    // Geri Gel Kuçu Kuçu (5 adım)
    for (int i = 0; i < 5; i++) {
        moveMotors(speeds[i], 1000, false);
    }

    // Dur Oğlum
    stopMotors(1000);

    while (true); // Dewamke
}
```

⇑ *Main Loop*

Motor speeds are defined as an array to gradually increase speed.
Moves forward by increasing speed step by step.
Pauses for 1 second.
Moves backward with the same speed increments in reverse.
After completing the loop, the program stops and does not restart.

- **Discussion:** The most critical situation in this task is to control the PWM values between 0-255 within a certain percentage range and to move the robot in light of this control. In the first stage, it was decided that the starting value for the PWM value changing with a 20% increase would be “50”. In this context, it was observed that the robot did not show any movement in the first 20% value and that the PWM value started to move at 40%, that is, “100”. In this context, it was determined that starting the starting value of the PWM signal with “80” rather than “50” was more efficient. However, this situation prevented the PWM signals from increasing with a full 20% interval and caused the signal increase to continue as follows; “80, 120, 160, 200, 255”. As can be seen, the PWM signal, which increased with a “40” interval in the first stage, increased to “55” in the last set and did not provide a linear feedback. However, since the minimum PWM signal value required for the system to activate is "80", this situation has become tolerable.

- **Conclusion:** It was determined that there was noise coming from the motors at the “50” PWM signal value, but no movement was observed in the robot. In this context, it was thought that the first problem that was considered was due to the motors not being able to

provide the desired torque at the “50” PWM signal value in the feedback generated by taking the weight of the robot as reference. For this reason, it was concluded that the torque value needed for the robot to transition from static to dynamic was generated at the “80” PWM value. It was determined that some changes could be applied to the system in order to solve this problem more effectively and increase the PWM values with a proportional (linear) percentage value. These changes are as follows,

A) Battery Voltage Increase

The increase in battery voltage will provide a linear increase in the feedback in terms of the torque that the motor will create at the same PWM value. In this context, the torque created by a system operating with 2S (7.4V) will differ from the torque created by a system operating with 3S (11.1V).

B) Motor Swap

A motor with higher reduction and torque values can be used at the same voltage and PWM value. In this context, the transition of the system from static to dynamic can be achieved at a PWM value of "50".

2. Proportional Speed Control

- **Objective:** For this task, the distance data obtained via the ultrasonic sensor on the robot was taken as reference. Motor control was provided by taking into account the distance data coming from the sensor and the reference distance (50 [cm]) that we added to the code. The main purpose is to ensure that the robot positions itself exactly 50 [cm] away by taking the obstacle in front of it as reference and to use the proportional speed infrastructure while doing this positioning. For example, if the obstacle is 100 cm away and the robot needs to stop exactly 50 cm away, it needs to run the 50 cm running distance at [10X] PWM speed and this PWM value should approach 0 as the distance gets shorter. If the robot is 60 cm away and the reference distance is 50 cm, the robot needs to run the 10 cm distance it needs to run at [2X] PWM speed and this PWM (speed) value should approach 0 as it gets closer to the target. In this context, the required proportional equation and error formula were prepared by taking the following equations as reference.

$$error = ref - dist$$

$$speed = K_p * error$$

- **Code and Explanation:** A code has been prepared for the process in the relevant equations to be carried out optimally. The code and the explanations of the infrastructures in this code are detailed in the following section.

```
#include <MeMCore.h>

MeUltrasonicSensor ultrasonic(PORT_4);  // Sensör portu
MeDCMotor motorLeft(M1);
MeDCMotor motorRight(M2);
```

⇑ *Library and Motor Definitions*

The MeMCore. h library is used to control mBot's motor and sensor components.
Left and right motors are connected to M1 and M2 ports
The HC − SR04 sensor is assigned to PORT_4.

```
const float referenceDistance = 50.0; // Hedef mesafe (cm)
const int minSpeed = 80;  // Minimum motor hızı (PWM)
const int maxSpeed = 255; // Maksimum motor hızı (PWM)
const float Kp = 15.0; // Oransal kontrol katsayısı
```

⇑ *Constant PID Parameters*

The target distance is set to 50 cm..
The PWM signal range determines the controllable speed limits.
The Kp coefficient adjusts the P control.

```
// Mesafeyi Oku Kardeş
float getDistance() {
    float distance = ultrasonic.distanceCm();
    if (distance < 2 || distance > 200) return -1; // Geçersiz okuma
    return distance;
}
```

⇑ *Distance Measurement Function*

The HC − SR04 sensor measures distance.
Invalid readings (less than 2 cm or greater than 200 cm) are filtered.

```
// Hızı Set et ve Çalıştır
void setMotorSpeed(int speed) {
    if (speed > 0) {  // İleri hareket
        motorLeft.run(speed);
        motorRight.run(-speed);
    } else {  // Geri hareket
        motorLeft.run(speed);
        motorRight.run(-speed);
    }
}

// Aga Dur
void stopMotors() {
    motorLeft.stop();
    motorRight.stop();
}
```

⇑ *Motor Control Functions*

PWM values are applied to control forward or backward movement.
Completely stops the motors.

```
void setup() {
    Serial.begin(115200);
    Serial.println("Proportional Position Control Başlatıldı...");
}
```

```
void loop() {
    float distance = getDistance();

    if (distance != -1) {
        float error = referenceDistance - distance;
        int speed = Kp * error; // Oransal hız hesaplama

        // Hızı sınırlama (min ve max arasında tut)
        if (speed > maxSpeed) speed = maxSpeed;
        if (speed < -maxSpeed) speed = -maxSpeed;
        if (abs(speed) < minSpeed) speed = 0; // Çok küçük hızlarda dur

        Serial.print("Mesafe: ");
        Serial.print(distance);
        Serial.print(" cm, Hata: ");
        Serial.print(error);
        Serial.print(", Hız: ");
        Serial.println(speed);

        if (speed == 0) {
            stopMotors();
        } else {
            setMotorSpeed(speed);
        }
    }

    delay(100); // Stabil okuma için kısa gecikme
}
```

⇑ *Main Loop*

Error is calculated (reference distance − current distance).
The error is multiplied by Kp to compute the PWM Speed value
Speed limitations are applied (kept with min − max range.).
Displays real − time distance, error, and speed values on the serial monitor(Not able to print these over the MBlock editor
If the speed is 0, the motors stop; otherwise, they move at the calculated speed.

- **Discussion:** The basic feedback needed for this task is for the robot to stop when it sees the target exactly 50 cm away and to optimize its own speed by taking the difference as a reference. In this context, the "Kp" value was used to manipulate the PWM signals. The change of this value between 5-25 triggered the aggressiveness of the robot's movement. In the robot configuration with a value of 5 Kp, the system reacted late and could not perceive the distance correctly. On the other hand, when the Kp value was increased to 20-25, it was observed that instantaneous oscillations occurred and the robot exhibited sudden movements by going back and forth. In this context, it was determined that the most optimal value was 15 Kp and it was observed that the system worked properly.

- **Conclusion:** In order to get the most out of the Kp value in the system and create a robot platform with sensitive feedback, the following optimizations can be evaluated. These are as follows,

  A) Sensor Swap: Use of an ultrasonic sensor that provides more precise and faster measurement (CAN communication protocol may be preferred.)

B) Microcontroller Swap: Selecting a more powerful equivalent of the microcontroller used to calculate the error rate and control the motor speed accordingly (Raspberry Pi or Nvidia Jetson Nano instead of Arduino)

3. Arduino-Based Mobile Robot Radar

- **Objective:** The main purpose of this task is to scan the environment via the ultrasonic sensor on the robot and create a live 2D mapping system in the light of the feedback obtained from this scanning data. A SLAM infrastructure was created at the initial level and two different code editors were used to optimize the system. Using the Arduino IDE editor, the robot was rotated 360 degrees around its own axis in multiple steps and the obtained data was observed via the Serial Monitor. The MATLAB program was used to perform the mapping live and create a sketch in the 2D environment and the instant distance data from the robot was written live to the map infrastructure using various trigonometric functions.

    - **Code and Explanation:**

The Arduino (robot control) code developed for the relevant task is explained in the following section.

```
#include <MeMCore.h>

MeUltrasonicSensor ultrasonic(PORT_3);
MeDCMotor motorLeft(M1);
MeDCMotor motorRight(M2);

int stepAngle = 6;
int totalSteps = 60;
int currentStep = 0;
```

⇑ *Declarations (Global Definitions)*

includes the Makeblock library for mBot hardware.
defines ultrasonic sensor on port 3
defines left and right motors on M1 and M2
sets rotation angle per step and total steps $(60 \times 6° = 360°)$

```
void setup() {
  Serial.begin(9600);
  stopMotors();
  delay(1000); // sistem otursun
}
```

⇑ *Setup () Function*

starts serial communication.
stops motors initially
waits 1 second for system stabilization

```
void loop() {
  if (currentStep < totalSteps) {
```

```
    long distance = ultrasonic.distanceCm();
    if (distance > 150 || distance <= 0) {
      Serial.print("Angle: ");
      Serial.print(currentStep * stepAngle);
      Serial.println(", Distance: NA");
    } else {
      Serial.print("Angle: ");
      Serial.print(currentStep * stepAngle);
      Serial.print(", Distance: ");
      Serial.print(distance);
      Serial.println(" cm");
    }

    delay(800); // bekle kardeş
    turnLeft(); // sabit 5 derece döndür
    stopMotors();
    delay(600); // dönüş sonrası sabitle

    currentStep++;
  } else {
    stopMotors();
    while (true); // işlem tamam
  }
}
```

⇑ *Loop () Function*

executes 60 total measurement and rotation steps
each step reads distance from the sensor
if the value is out of range, "NA" is printed
otherwise, the current angle and distance are printed
robot rotates after each measurement
stops completely when 360° scan is complete

```
void turnLeft() {
  int rotateTime = 75; // bu değer kafi
  motorLeft.run(120);
  motorRight.run(120);
  delay(rotateTime);
}
```

⇑ *Turn Left() Function*

rotates the robot by driving both motors forward
using 75 milliseconds produces approximately a 6° turn
rotation direction is counter – clockwise (to the left)

```
void stopMotors() {
  motorLeft.run(0);
  motorRight.run(0);
}
```

⇑ *Stop Motors() Function*

stops both motors by setting their speed to 0
ensures the robot halts before or after rotating

Likewise, the MATLAB code used to plot the incoming data is as follows.

```
clear; clc;

% COM portu bul babacım
port = "COM5";        % Robotun bağlı olduğu PORT
baud = 9600;

% Serial port elemanı oluşturalım kardeş
s = serialport(port, baud);
configureTerminator(s, "LF");  % satır sonu karakteri
```

⇑ *Initialization & Serial Setup*

clears workspace and command window
sets the correct COM port (COM5 for USB dongle/mBot)
opens serial communication at 9600 baud
sets newline character (\n) as line terminator for reading full lines

```
% Verileri bu matriste tutacazzz
data = [];

% Haritamızı güzelleştirelim
figure;
hold on;
grid on;
xlabel('X (cm)');
ylabel('Y (cm)');
title('mBot SLAM Haritası');
xlim([-200 200]);
ylim([-200 200]);
```

⇑ *Plot Setup*

initializes an empty matrix to store angle, distance, x, y
opens a figure window for plotting
sets axes and labels for a 2D environment
ensures points remain visible and plotted correctly

```
while true
    try
        line = readline(s);
        disp(line);  % Terminale bas
```

⇑ *Main Loop for Real – Time Serial Reading & Plotting*

enters an infinite loop
reads one line of serial data each time
prints it in MATLAB console for debugging

```
        % veri örneği: "Angle: 36, Distance: 84 cm"
        tokens = regexp(line, 'Angle:\s*(\d+),\s*Distance:\s*(\d+)', 'tokens');

        if ~isempty(tokens)
            angle = str2double(tokens{1}{1});
            dist = str2double(tokens{1}{2});

            % X-Y hesapla gülüm
            x = dist * cosd(angle);
            y = dist * sind(angle);
```

⇑ *Line Parsing (Angle & Distance)*

uses regular expression to extract angle and distance values
if the format matches, values are converted from string to numeric
supports lines like: "Angle: 36, Distance: 84 cm"

```
            % Çiz
            data(end+1, :) = [angle dist x y];
            scatter(x, y, 50, 'filled');
            drawnow;
        end
```

⇑ *X − Y Coordinate Calculation & Live Plotting*

calculates Cartesian coordinates from polar data
cosd/sind used for degrees (not radians)
appends the data to the data matrix
draws the new point on the scatter plot in real − time

```
    catch e
        disp("Hata oluştu:");
        disp(e.message);
        break;
    end
end
```

⇑ *Error Handling*

catches errors (e.g., device disconnected)
displays the error message
exits the loop safely without crashing MATLAB

- **Test Setup and Results:** The codes explained above were loaded into the robot and the robot was positioned in a simple cage. The live plot of the robot that made 60 turns and the test setup are detailed in the images below. The test process is explained in detail in the test video sent in addition to this report.

Figure-1: Test Setup of the Arduino-Based Mobile Radar Project

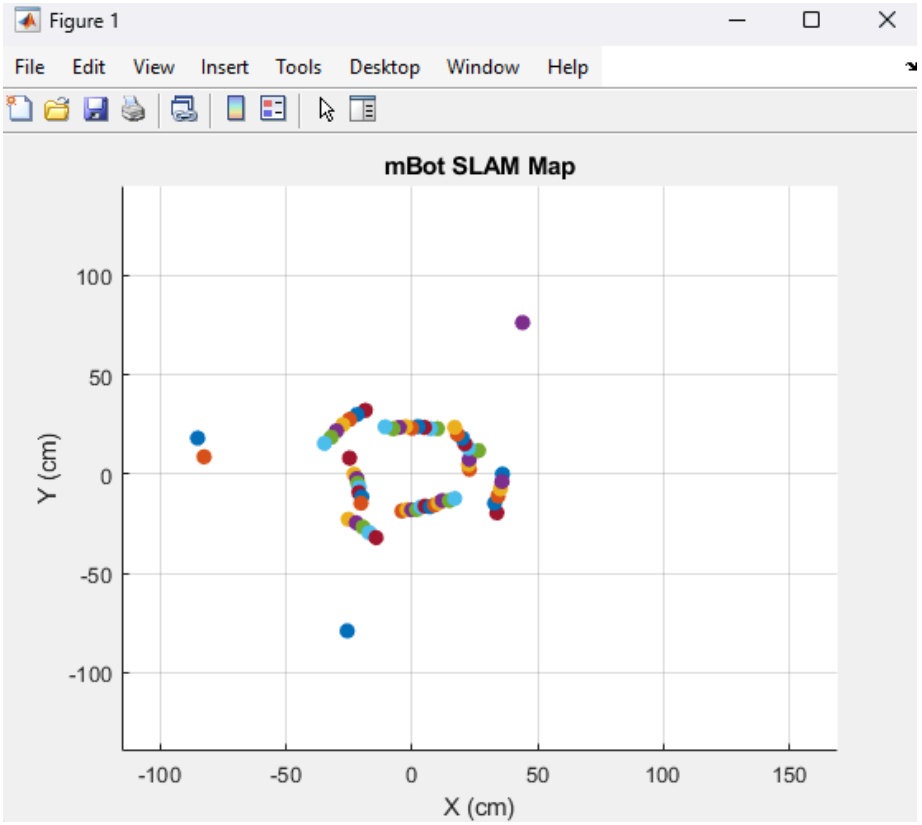

Figure-2: MATLAB Slam Map

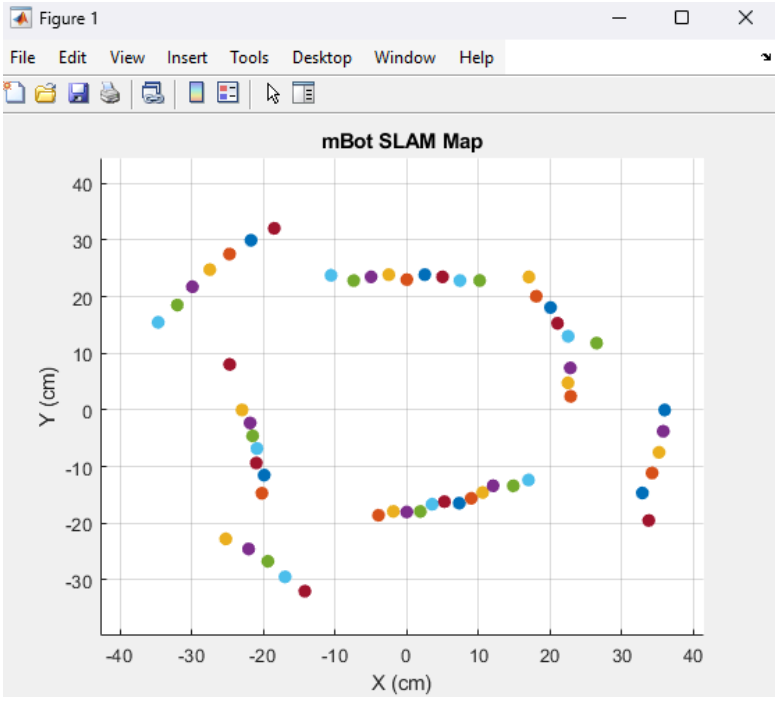

Figure-2: MATLAB Slam Map (Zoomed)

- **Discussion:** There are 3 main problems with this task. The first of these was that the robot could not complete a fixed 6-degree turn without error. As a result, shifts and deficiencies in the mapping occurred. Another error was that shifts occurred on a rotation basis due to the fact that the communication in the system was provided by cable. The robot occasionally got stuck due to the tension created by the cable. The third problem was that the distance sensor did not read the correct data and therefore, incorrect points were observed in the mapping. Apart from all these, the system provided the expected feedback and produced a 2-dimensional map live by scanning the environment.

- **Conclusion:** Some improvements that need to be implemented to increase the performance of the system and obtain more optimal feedback in mapping are listed. These are as follows;

    A) Compass Sensor Adding: A compass sensor can be added to clearly define the angle range in which the system is located and eliminate erroneous data collection situations. This may seem a bit absurd due to the nature of the SLAM system, but it can be used to optimize the system's feedback on a rotational basis.

    B) Wireless Communication: NRF modules can be used to prevent the system from getting stuck on the cable and to create a more agile infrastructure. The Wireless 2.4G module in the mBot does not work on the Arduino IDE.

    C) Distance Sensor Replacement: As a result of the tests, it was determined that this sensor did not collect accurate data due to the vibration of the robot and various external factors. In this context, a higher quality sensor, the TF Mini Plus Lidar module, can be used. A more efficient data transfer can be achieved with the RS485 communication protocol.