ME2400 Course Project

Krishna	Avani	Srinidhi	Twesha	Aakash	Akash
ME23B103	ME23B128	ME23B115	ME23B118	ME23B095	ME23B125

May 2025

1 AIM AND OBJECTIVES

1.1 Aim

The aim of the project is to design and build a compact autonomous robot that can transport a wheelchair to the back of a car by maintaining a fixed distance from the car's side, making a right turn when the car end, and stopping accurately at the center of the rear side using a PID controller for precise wall-following, without relying on camera-based vision systems.

1.2 Objectives

- Wall-Following: Maintain a steady distance (x cm) from the right side of the vehicle using an ultrasonic sensor.
- MATLAB simulation: Simulate the transfer function and understand the response of the system
- PID Control: Implement and tune a PID controller to correct deviations and ensure smooth, responsive tracking
- Precise Stopping: Move parallel to the rear of the car and stop near the center point (20 cm from the edges).
- The robot should fit in a box $10 \times 10 \times 10$ cm^3.

2 CHASSIS DESIGN AND COMPONENTS

- **Design Platform**
  - The robot chassis was designed using **SolidWorks**.
- **Structural Overview**
  - The chassis consists of two primary components:
    - Upper layer
    - Bottom layer
  - The **battery** is positioned centrally between these two layers to maintain a balanced **center of gravity (CG)**, which is essential for stable locomotion.
  - The layers feature **interlocking slots** that enable the upper layer to securely cap onto the bottom layer without the need for additional fasteners, ensuring structural integrity.
- **Component Integration**
  - Custom-designed slots and cutouts are incorporated into the chassis to accommodate critical components, including:
    - **Battery**
      - 3 x 3.7V lithium-ion batteries
    - **Motors**
      - 2 x TT gear motors
      - 2 x wheels
      - 1 x castor wheel
    - **Microcontroller and Driver**
      - Arduino UNO
      - L298N motor driver
    - **Sensors**
      - 2 x HC-SR04 ultrasonic sensors
    - **Wiring**
      - Jumper wires

  - Additional mounting holes are provided for:
    - Securing the *Arduino* board
    - Attaching the *motor driver*
    - *Routing wires* to ensure a clean and organized internal layout

- **Dimensional Constraints and Specifications**
  - The overall chassis design adheres to the project constraint of:

$$10 \times 10 \times 10 \text{ cm}^3$$

  - **Bottom Layer**:
    - Thickness: 3 mm
    - Width: 70 mm
    - Length: 98 mm
    - The width is intentionally reduced to account for the lateral extension of the:
      - *Wheels:* 10 mm
      - *Shaft:* 4 mm
    - Total lateral extension from components contributes to the robot's final width.
  - **Top Layer**:
    - Thickness: 3 mm
    - Width: 90 mm
    - Length: 98 mm
    - The increased width provides:
      - Better coverage of internal components
      - Additional mounting space for extra modules

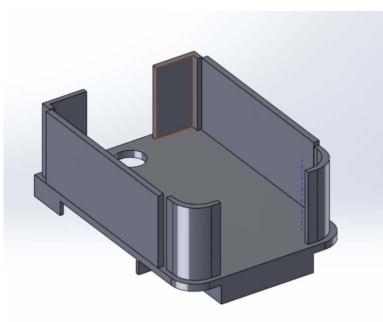

Figure 1: Bottom Part

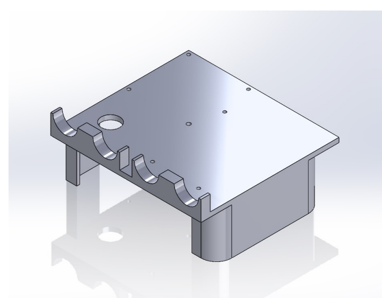

Figure 2: Top Part

3 CONTROLLER DESIGN AND TUNING METHOD

3.1 MATLAB Model

To obtain the transfer function of the system we will use an open-loop approach.

$$T(s) = \frac{Y(s)}{R(s)} = \frac{G_2(s)}{1 + G_2(s)H(s)}$$

Here, we have written the transfer function without the PID controller since that is what we need to derive. $G_2(s)$ is the plant which mainly consists of the motor, where we take the transfer function of the microprocessors and ultrasonic sensor as 1 for modeling.

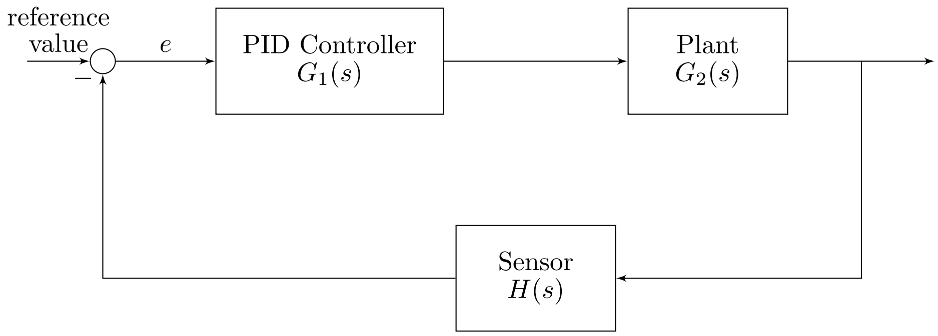

Figure 3: Block diagram of PID control system

$H(s) = 1$

To approximate $G_2(s)$ we aimed to apply a step response to the system, as constant speed to the motors towards the wall and then use this input/output data to fit a linear transfer function model.

Disclaimer:
We were not able to obtain this live input/output data due our experience with an unpowered Arduino UNO board hampering with our laptop display. We could not unfortunately risk our computer systems to obtain these values from a powered board while the bot was moving. So below we will explain how we could have used MATLAB to simulate our model's transfer function if we were able to extract the data.

We will use the data to be a smooth exponential curve for demonstration purposes and show how we can use that data to fit a linear model and obtain the mathematical transfer function.

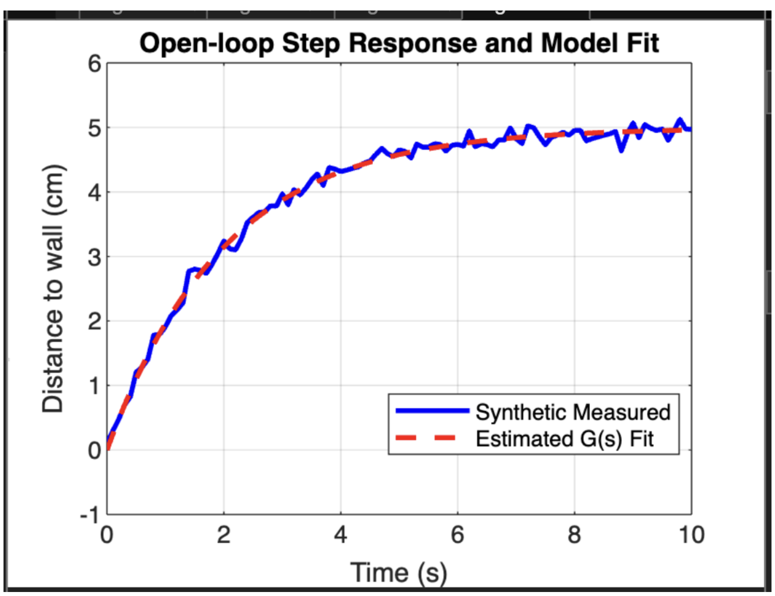

Figure 4: Hypothetical input vs output plot

```
K   = 0.25;     % synthetic plant gain
tau = 2;        % synthetic plant time constant
U         = 20;       % Step input = difference between the 2 wheel
    speeds
Ts        = 0.1;      % Sampling interval
```

```
t          = 0:Ts:10; % Time array

%Here y should be the distance measured from the sensors but we are
    assuming ideal first-order step response
y_meas = K*U*(1 - exp(-t/tau));

% Measurement Noise to show how the real data would look like:
y_meas = y_meas + 0.1*randn(size(y_meas));

% Fitting a 1-pole, 0-zero transfer function
% Linearizing model
u = U * ones(size(t));
data = iddata(y_meas(:), u(:), Ts);
sysG = c2d(tfest(data, 1, 0), Ts);
```

```
Command Window
sysG =

  From input "u1" to output "y1":
    0.1235
  ----------
  s + 0.4942
```

Figure 5: $G_1(s)$ Open loop Plant Transfer System

```
% Tune on open-loop plant:
[C_pid,info] = pidtune(sysG,'PID');

C_pid

%Closed-loop:
sysCL = feedback(C_pid*sysG, 1);

%plotting the step response correction for the system
figure;
step(sysCL*U)
```

```
Command Window
C_pid =

             1
  Kp + Ki * ---
             s

  with Kp = 5.72, Ki = 5.07

Continuous-time PI controller in parallel form.
Model Properties
```

Figure 6: $G_2(s)$ PID Control Transfer function

Important points to note in the above modeling process

- We utilized the MATLAB's *System Identification* tool which allows us to fit a transfer function model to our system characteristics.
- The model was tuned on a discretized system by using the function `c2d()` with a sampling time as 0.1sec.

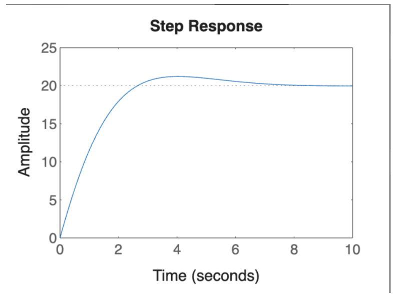

Figure 7: Response after PID

- The system was modeled linearly for simplicity. But the TTM-95RPM motor transfer function is a second order as specified in the datasheet.
- The transfer function T(s) is not for the plant and not for the control loop.

Description of the Tuning method

Through this method we can find the first initialization for PID values and save ourselves a random search. But due to the aforementioned issues with our PID issues we had to proceed with the latter method for tuning. Through the principles of PID and observations we made the following choices for Kp, Kd, Ki

- If the overshoot was too high : Increased Kd
- If the settling time too long : Increased Kp
- If no. of oscillating rapidly : Increased Kd
- If steady state error is observed : Increased Ki

3.2 Controller Design and Arduino Program

Code Listing 1: Embedded Arduino Code

```
  // setting the pin connections
// === Motor Pins ===
const int enL = 9;
const int in1L = 7;
const int in2L = 6;

const int enR = 3;
const int in1R = 4;
const int in2R = 5;

// === Ultrasonic Sensor ===
const int trigPin = 10;
const int echoPin = 11;

// === setting Constants ===
const int BASE_SPEED = 100;
const int MIN_SPEED = 65;
const int SPIN_SPEED = 110;

const float DESIRED_DISTANCE = 10.0;
const float ERROR_MARGIN = 2.0;
const float WALL_LOST_THRESHOLD = 50.0;

const unsigned long SPIN_DURATION = 260;
```

```
const unsigned long ADVANCE_DURATION = 210;
const unsigned long STOP_AFTER_TIME = 500;
const unsigned long STARTUP_DELAY = 400;
const unsigned long STRAIGHT_TIME = 220;  // 0.1s straight after turn

// === PID Constants ===
float kp = 3.8; //3.6
float ki = 0.4;
float kd = 1.6;

const unsigned long SAMPLE_TIME = 50;
const float DERIVATIVE_FILTER_ALPHA = 0.6; //0.8

// === State Variables ===
float error = 0, previousError = 0, integral = 0, derivative = 0,
    filteredDerivative = 0;
unsigned long lastTime = 0;
unsigned long spinStartTime = 0;
unsigned long postSpinStartTime = 0;
unsigned long startupTime = 0;
unsigned long postTurnStraightStartTime = 0;

int spinStage = 0;  // 0 = idle, 1 = spinning, 2 = advancing
bool firstSpinCompleted = false;
bool hasStoppedAfterFirstSpin = false;
bool programEnded = false;
bool inPostTurnStraight = false;

void setup() {
  Serial.begin(9600);

  pinMode(trigPin, OUTPUT);
  pinMode(echoPin, INPUT);

  pinMode(enL, OUTPUT); pinMode(in1L, OUTPUT); pinMode(in2L, OUTPUT);
  pinMode(enR, OUTPUT); pinMode(in1R, OUTPUT); pinMode(in2R, OUTPUT);

  lastTime = millis();
  startupTime = millis();
}
//== calculating distance from sensors and sampling out reading to ignore
    noise readings==
float readDistance() {
  float sum = 0;
  int samples = 5;
  for (int i = 0; i < samples; i++) {
    digitalWrite(trigPin, LOW); delayMicroseconds(2);
    digitalWrite(trigPin, HIGH); delayMicroseconds(10);
    digitalWrite(trigPin, LOW);
    long duration = pulseIn(echoPin, HIGH, 20000);
    if (duration == 0) duration = 3000;
    float d = duration * 0.0343 / 2.0;
    sum += d;
    delay(5);
  }
  return sum / samples;
}

void setLeftMotor(int speed, bool forward) {
  speed = constrain(speed, 0, 255);
  analogWrite(enL, speed);
  digitalWrite(in1L, forward ? HIGH : LOW);
```

```
  digitalWrite(in2L, forward ? LOW : HIGH);
}

void setRightMotor(int speed, bool forward) {
  speed = constrain(speed, 0, 255);
  analogWrite(enR, speed);
  digitalWrite(in1R, forward ? HIGH : LOW);
  digitalWrite(in2R, forward ? LOW : HIGH);
}

void spinRight() {
  setLeftMotor(SPIN_SPEED, true);
  setRightMotor(SPIN_SPEED, false);
}

void moveForward() {
  setLeftMotor(BASE_SPEED, true);
  setRightMotor(BASE_SPEED, true);
}

void stopMotors() {
  setLeftMotor(0, true);
  setRightMotor(0, true);
}
// == we set 3 different stages for the bot spin state to take a right turn
    when no wall is detected
//== PID follower when the wall is present before taking a turn
//== Straight follower after taking a turn it goes straight and stops at 20cms
void loop() {
  if (programEnded) return;

  float distance = readDistance();
  unsigned long currentTime = millis();
  float deltaTime = (currentTime - lastTime) / 1000.0;

  // === Stop after time post-spin ===
  if (firstSpinCompleted && !hasStoppedAfterFirstSpin &&
      currentTime - postSpinStartTime >= STOP_AFTER_TIME) {
    stopMotors();
    hasStoppedAfterFirstSpin = true;
    programEnded = true;
    Serial.println("Traveled 21cm after turn and Stopping permanently.");
    return;
  }

  // === Spin handling ===
  if (spinStage == 1) {
    if (currentTime - spinStartTime < SPIN_DURATION) {
      spinRight();
      Serial.println("Spinning 90 degrees right");
      return;
    } else {
      spinStage = 2;
      spinStartTime = currentTime;
      Serial.println("Spin done. Moving forward briefly");
      return;
    }
  }

  if (spinStage == 2) {
    if (currentTime - spinStartTime < ADVANCE_DURATION) {
      moveForward();
```

```
      return;
    } else {
      spinStage = 0;
      if (!firstSpinCompleted) {
        firstSpinCompleted = true;
        postSpinStartTime = millis();

        // Begin straight motion before PID
        inPostTurnStraight = true;
        postTurnStraightStartTime = currentTime;
        Serial.println(" Advance complete. Moving straight for 0.1s");
        return;
      }
    }
  }

  // === Straight move after turn ===
  if (inPostTurnStraight) {
    if (currentTime - postTurnStraightStartTime < STRAIGHT_TIME) {
      moveForward();
      return;
    } else {
      inPostTurnStraight = false;
      Serial.println("Straight motion done. Resuming PID wall following");
    }
  }

  // === Initiate spin if wall lost ===
  if (spinStage == 0 && !firstSpinCompleted &&
      (millis() - startupTime > STARTUP_DELAY) &&
      distance > WALL_LOST_THRESHOLD && distance < 300.0) {
    spinStage = 1;
    spinStartTime = currentTime;
    Serial.println(" Wall lost. Starting spin...");
    return;
  }

  // === PID Wall Following ===
  if (currentTime - lastTime >= SAMPLE_TIME && !hasStoppedAfterFirstSpin) {
    error = distance - DESIRED_DISTANCE;

    integral += error * deltaTime;
    integral = constrain(integral, -50, 50);

    derivative = (error - previousError) / deltaTime;
    filteredDerivative = DERIVATIVE_FILTER_ALPHA * filteredDerivative +
                         (1 - DERIVATIVE_FILTER_ALPHA) * derivative;

    float output = kp * error + ki * integral + kd * filteredDerivative;
    output = constrain(output, -20, 20);
//== making sure that the bot doesn't speed up when it's moving away
    float absError = abs(error);
    float speedScale = constrain(1.0 - (absError / 10.0), 0.3, 1.0);
    float baseSpeed = BASE_SPEED * speedScale;

    int leftSpeed = baseSpeed + output;
    int rightSpeed = baseSpeed - output;

    leftSpeed = max(leftSpeed, MIN_SPEED);
    rightSpeed = max(rightSpeed, MIN_SPEED);

    setLeftMotor(leftSpeed, true);
```

```
    setRightMotor(rightSpeed, true);

    Serial.print("Distance: "); Serial.print(distance);
    Serial.print(" | Error: "); Serial.print(error);
    Serial.print(" | L: "); Serial.print(leftSpeed);
    Serial.print(" | R: "); Serial.println(rightSpeed);

    previousError = error;
    lastTime = currentTime;
  }}
```

4 CONCLUSION

- **Hardware Safety Lessons:** Faced issues from uploading code while the circuit was powered, resulting in damaged Arduinos. This reinforced the importance of safe hardware debugging procedures.
- **Chassis Limitations and Redesign:** The initial chassis lacked space and proper mounting options. Switching to a custom 3D-printed design enabled compact integration of electronics within the 10×10×10 cm^3 constraint.
- **Control System Drawback:** After a 90° turn, the robot lacked PID correction, leading to crashes due to inconsistent wall-following. Hard-coded stopping distance further affected reliability across varying conditions.
- **Lessons in Control Engineering:** We learned to tune PID controllers based on real-time behavior and used MATLAB simulations to predict system performance before hardware deployment.
- **Performance Highlights:**
  - Implemented dynamic speed adjustment to maintain stability under varying path conditions.
  - Achieved accurate and responsive PID tuning through iterative testing.
  - Employed dual ultrasonic sensors with averaging to reduce noise and improve wall detection.
- **Overall Takeaway:** The project deepened our understanding of robotics through iterative design, control system tuning, and precise system integration under physical constraints.

Bill (Only one kit was used)

5 TEAM CONTRIBUTIONS AND PROJECT VISUALS

Chassis Design was handled by Akash and Aakash. **Arduino Programming** was carried out by Srnidhi and Krishna. **Report Writing and MATLAB Simulations** were managed by Twesha and Avani.

Videos

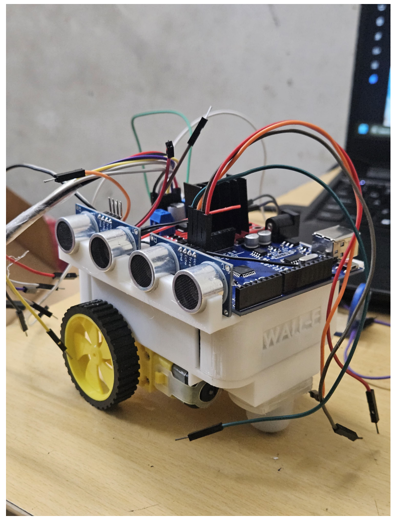

Figure 8: Wall- E

Figure 9: Team Picture

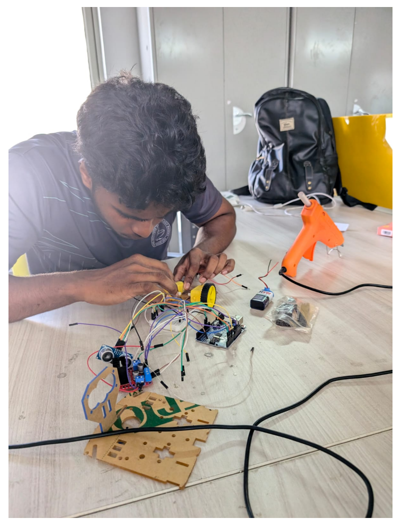

Figure 10: Team Picture

Figure 11: Team Picture

Figure 12: Team Picture