

**INDIAN INSTITUTE OF TECHNOLOGY
MADRAS**

DEPARTMENT OF MECHANICAL ENGINEERING

**AUTONOMOUS WALL-FOLLOWING
ROBOT**

ME2400 Project Report

GROUP 10

ME23B123	Adwait
ME23B101	Sumanth
ME23B104	Daniel
ME23B155	Amruta
ME23B117	Thomas

May 2025

What lies ahead?

Group 10

May 4, 2025

Contents

Aim and Objectives	3
Aim	3
Objectives	3
Chassis Design and Component Placement	3
Component List and Justifications	3
Design Considerations	4
Controller Design and Tuning	6
Control System Block Diagram	6
Transfer Function Model	7
Tuning Methodology	11
Simulink Model	18
Performance Analysis	18
Team Member Contributions	19
Budget	20
Gallery	20
Conclusion	20

Aim and Objectives

Aim

To design and develop an autonomous robot capable of moving a wheelchair to the back of a car model by maintaining a gap of more than 1 cm from the car's right side, executing a left turn upon reaching the rear end, and stopping after moving a fixed distance parallel to the trunk, using simple sensors and microcontroller-based control without machine vision.

Objectives

- Build a three-wheeled or more robot that fits within a 10 cm \times 10 cm \times 10 cm volume to operate within the given constraints.
- Employ ultrasonic or other suitable sensors with an Arduino or ESP32 microcontroller to detect the car's side and rear end.
- Implement a control algorithm (here- PID control) to maintain a consistent gap (>1 cm) from the car's right side during navigation.
- Program the robot to detect the end of the car and perform a left turn autonomously.
- Ensure the robot stops at a fixed position near the center of the car's rear.
- Adhere to safety considerations to prevent injury during operation.
- Complete the project within a budget of Rs. 4500, including all components and materials.

Chassis Design and Component Placement

Component List and Justifications

Component	Specification	Justification
Arduino UNO	16 MHz, 32KB flash	Sufficient I/O pins and processing power for sensor integration and motor control
L298N Motor Driver	Dual H-bridge, 2A max	Supports two DC motors with PWM speed control and direction control
DC Geared Motors (2)	6V, 200 RPM	Provides balanced speed and torque for stable wall-following

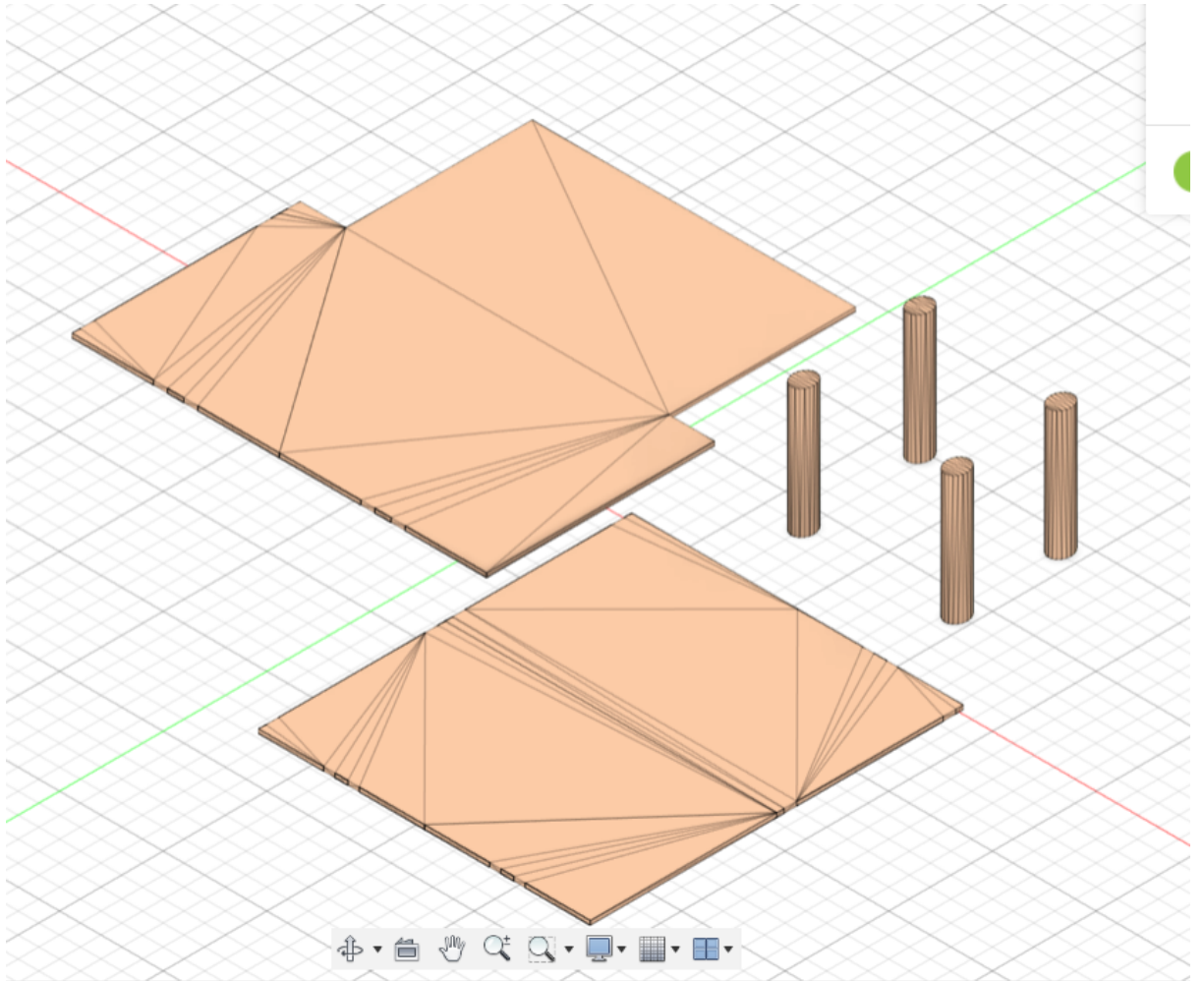
Component	Specification	Justification
HC-SR04 Ultrasonic Sensor	Range: 2-400 cm	Accurate distance measurement for wall-following within required range
Batteries	3xLi ion (3.7V)	Sufficient power for motors and electronics while maintaining portability
Motor Mounts and Wheels	Custom-designed	Optimized for stability and maneuverability

Other items used: Electrical Tape, Jumper wires, Double sided tape, USB cable, Wire stripper cutter, Multimeter, body – designed and fabricated out of wood.

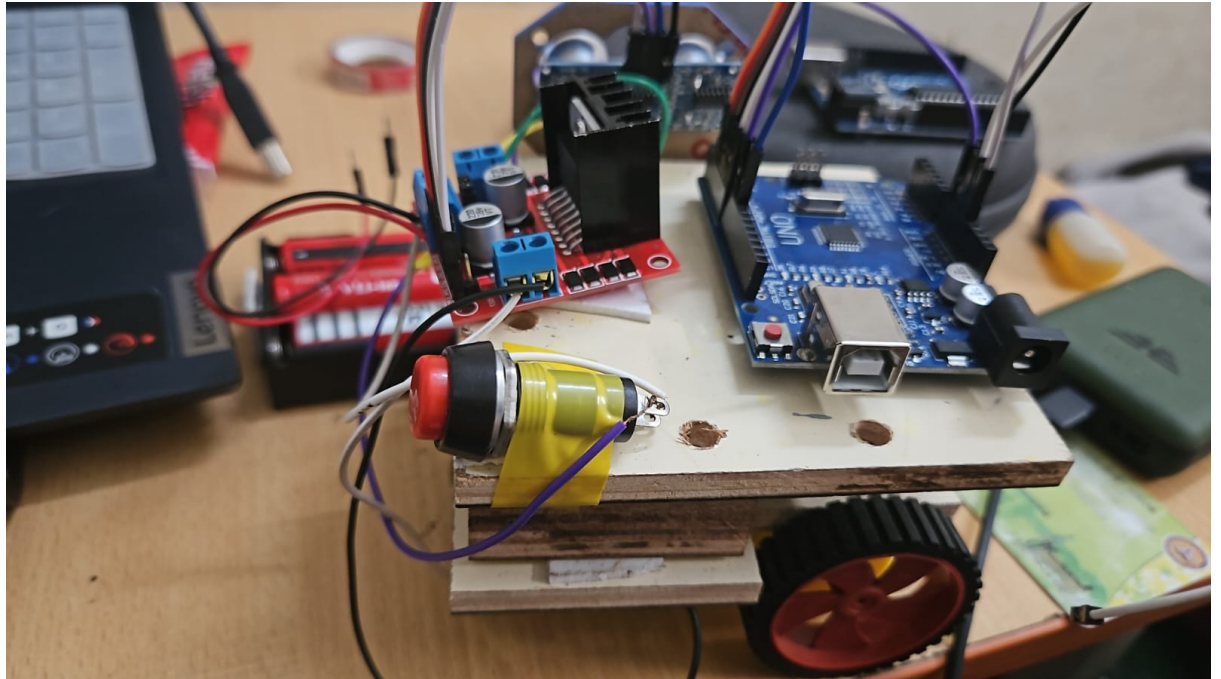
Design Considerations

The chassis was designed to balance several key requirements:

- **Sensor positioning:** The ultrasonic sensor is mounted perpendicular to the direction of travel for accurate wall distance measurement
- **We created our chassis using the Fusion360 program, keeping in mind the design specifications.** We decided to use wood for our chassis because the model needed to be both easy to manufacture and durable and firm. We carpentered the wood outside Taramani and used glue to put the pieces together.
- **Compactness:** Overall dimensions were tried to be kept minimal (10 cm × 10cm× 10 cm) to allow for the given dimensions.
- **Battery pack needed to have a separate compartment** given it occupies a lot of space, and needs constant changing of batteries during testing



[Figure 1: Chassis design schematic showing component placement]



[Figure 2: Robot design]

Controller Design and Tuning

Control System Block Diagram

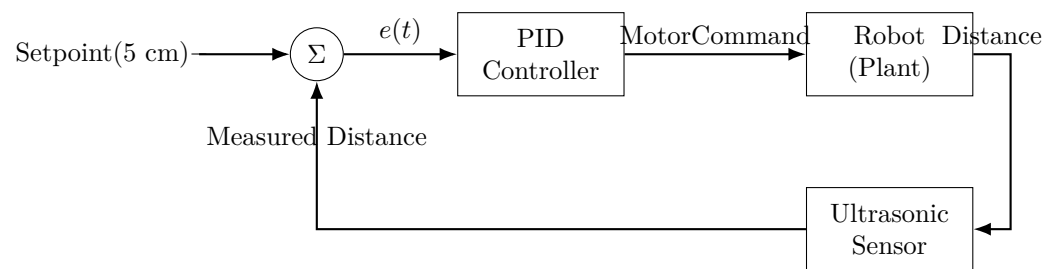


Figure 1: Control system block diagram for wall-following robot with sensor in the feedback loop.

Transfer Function Model

Control System Dynamics

Sensor Input:

The distance to the wall $d(t)$ is measured via an ultrasonic sensor.

Setpoint:

The desired distance from the wall is a fixed value:

$$d_{\text{set}} = 5 \text{ cm}$$

Error Computation:

The error $e(t)$ at time t is computed as:

$$e(t) = d(t) - d_{\text{set}} = d(t) - 5$$

PID Controller:

The control output $u(t)$ generated by the PID controller is given by:

$$u(t) = K_p \cdot e(t) + K_i \cdot \int e(t) dt + K_d \cdot \frac{de(t)}{dt}$$

This output $u(t)$ adjusts the motor speeds and, consequently, the turning angle of the robot.

Actuator Behavior:

The motor speeds are adjusted as follows:

$$\text{Right motor speed: } \text{baseSpeed} - u(t)$$

$$\text{Left motor speed: } \text{baseSpeed} + u(t)$$

The difference in speeds creates an angular velocity $\omega(t)$, causing the robot to steer.

Modeling the Dynamics:

Let's define the following variables: - $d(t)$: Distance to the wall - $\theta(t)$: Robot's angle to the wall (ideal = 0) - v : Constant forward velocity - $\omega(t)$: Angular velocity (function of the difference in wheel speeds and PID output)

Lateral Dynamics

From the robot's kinematics (unicycle model), we have the following equations:

For the change in distance $d(t)$:

$$\frac{dd(t)}{dt} = v \cdot \sin(\theta(t)) \approx v \cdot \theta(t) \quad (\text{for small } \theta(t))$$

For the change in angle $\theta(t)$:

$$\frac{d\theta(t)}{dt} = \omega(t)$$

Now, using the PID controller output $\omega(t)$, we have:

$$\omega(t) \propto u(t) = \text{PID}(e(t)) = K_p \cdot e(t) + K_i \cdot \int e(t) dt + K_d \cdot \frac{de(t)}{dt}$$

Substituting $e(t) = d(t) - 5$, we get:

$$\omega(t) = K_p \cdot (d - 5) + K_i \cdot \int (d - 5) dt + K_d \cdot \frac{d(d - 5)}{dt}$$

This can be written as:

$$\omega(t) = K_p \cdot d + K_i \cdot \int \frac{dd(t)}{dt} dt + K_d \cdot \frac{d^2d(t)}{dt^2} + (\text{constants})$$

Differentiating the Lateral Dynamics:

Next, we differentiate the equation for $d(t)$:

$$\frac{d^2d(t)}{dt^2} = v \cdot \frac{d\theta(t)}{dt} = v \cdot \omega(t)$$

Substituting $\omega(t)$ from above, we get:

$$\frac{d^2d(t)}{dt^2} = v \cdot \left(K_p \cdot d + K_i \cdot \int \frac{dd(t)}{dt} dt + K_d \cdot \frac{d^2d(t)}{dt^2} \right)$$

Rearranging Terms:

Moving all terms to one side:

$$\frac{d^2d(t)}{dt^2} - v \cdot K_d \cdot \frac{dd(t)}{dt} - v \cdot K_p \cdot d = v \cdot K_i \cdot \int \frac{dd(t)}{dt} dt$$

This simplifies to:

$$\frac{d^2d(t)}{dt^2} + a_1 \cdot \frac{dd(t)}{dt} + a_0 \cdot d(t) = a_0 \cdot d_{\text{set}}$$

where:

$$a_1 = -v \cdot K_d, \quad a_0 = -v \cdot K_p$$

Final Governing Equation (Second-Order Approximation):

The final second-order approximation is:

$$\frac{d^2 d(t)}{dt^2} + a_1 \cdot \frac{dd(t)}{dt} + a_0 \cdot d(t) = a_0 \cdot d_{\text{set}}$$

Why Not a First-Order System?

A first-order system would only capture position or angle without any dynamics of rate-of-change. Since the PID control involves derivatives and integrals, the system exhibits second- or third-order behavior. The robot's steering response depends on angular acceleration, which results from wheel speed differences, leading to at least second-order dynamics.

Robot Dynamics

Robot's Kinematics:

The robot's orientation $\theta(t)$ changes based on its angular velocity $\omega(t)$:

$$\frac{d\theta(t)}{dt} = \omega(t)$$

Lateral Dynamics and Rotation:

The robot's distance to the wall $d(t)$ is affected by both its linear and angular velocities. The lateral velocity (perpendicular to the wall) is proportional to the robot's forward speed and its angle to the wall:

$$\frac{dd(t)}{dt} = v \cdot \sin(\theta(t)) \approx v \cdot \theta(t) \quad (\text{for small } \theta(t))$$

The change in the angle $\theta(t)$ over time is given by:

$$\frac{d\theta(t)}{dt} = \omega(t)$$

Turning and Lateral Dynamics Combined:

The turning behavior is modeled by considering both the angular velocity $\omega(t)$ and the lateral dynamics of the robot. The second-order governing equation for $d(t)$ and $\theta(t)$ can be written as a set of coupled differential equations:

For the position (distance to the wall):

$$\frac{d^2 d(t)}{dt^2} + a_1 \cdot \frac{dd(t)}{dt} + a_0 \cdot d(t) = a_0 \cdot d_{\text{set}}$$

For the orientation (angle to the wall):

$$\frac{d\theta(t)}{dt} = \omega(t) = \frac{v}{L} \cdot u(t)$$

Final Governing Equations (Lateral + Rotation)

1. Lateral Dynamics (Position to the Wall $d(t)$):

$$\frac{d^2 d(t)}{dt^2} + a_1 \cdot \frac{dd(t)}{dt} + a_0 \cdot d(t) = a_0 \cdot d_{\text{set}}$$

where: - $d(t)$: Distance to the wall - d_{set} : Desired set distance to the wall - a_1 and a_0 : Constants depending on the robot's dynamics and PID control gains

2. Rotation Dynamics (Orientation $\theta(t)$):

$$\frac{d\theta(t)}{dt} = \omega(t) = \frac{v}{L} \cdot u(t)$$

where: - $\theta(t)$: Robot's angle relative to its initial orientation - v : Robot's forward speed - L : Track width (distance between wheels) - $u(t)$: Output of the PID controller

Thus, the final governing equations are:

$$\frac{d^2 d(t)}{dt^2} + a_1 \cdot \frac{dd(t)}{dt} + a_0 \cdot d(t) = a_0 \cdot d_{\text{set}}$$

$$\frac{d\theta(t)}{dt} = \frac{v}{L} \cdot u(t)$$

5. PID Controller Output:

The PID controller output is given by:

$$u(t) = K_p \cdot e(t) + K_i \cdot \int e(t) dt + K_d \cdot \frac{de(t)}{dt}$$

where: - $e(t)$ is the distance error, $e(t) = d_{\text{set}} - d(t)$, - K_p , K_i , and K_d are the proportional, integral, and derivative gains of the PID controller.

PID Implementation

The PID control law in the time domain is:

$$u(t) = K_p e(t) + K_i \int_0^t e(\tau) d\tau + K_d \frac{de(t)}{dt}$$

In the Laplace domain, the PID controller transfer function is:

$$C(s) = K_p + \frac{K_i}{s} + K_d s$$

Using the tuned parameters:

$$K_p = 6.0, \quad K_i = 0.007, \quad K_d = 13.9$$

Closed-Loop Transfer Function

The closed-loop transfer function combining the plant and controller is:

$$T(s) = \frac{C(s)G(s)}{1 + C(s)G(s)}$$

where $G(s)$ is the transfer function of the open-loop control system (forward path).

Tuning Methodology

4.3.1 PID Control Implementation

The PID controller was implemented with the following structure:

```
cpp
error = distance - fixedDistance; // Error calculation
P = error; // Proportional term
I += error; // Integral term
I = constrain(I, -400, 400); // Anti-windup protection
D = error - prevError; // Derivative term
errorCorrection = (Kp*P) + (Ki*I) + (Kd*D);
prevError = error;
rightSpeed = baseSpeed - errorCorrection;
leftSpeed = baseSpeed + errorCorrection;
```

4.3.2 Tuning Methodology

Control System Implementation and Tuning Initially, we tried to find K_p , K_i , K_d using the Ziegler Nichols method, and tried to find the neutral stability point by manually changing K_p and setting K_i and K_d to zero. We tried to find the K_p value for which the bot is oscillating and find the time period. But these attempts to find K_p , K_i and K_d values suffered from instability, which we believe is due to integral action amplifying sensor noise and accumulating error rapidly. This prompted a shift to manual PID control, and the values of K_p , K_d and K_i were determined experimentally through trial and error. Fine tuning was performed based on observation of the response. Proportional gain (K_p) was reduced from an initial value of 10 to 6 through iterative testing, which minimized oscillations at the cost of slightly increased settling time. Derivative gain (K_d) was raised to 13.9 to dampen overshoot and improve response to sudden distance changes. Sensor noise, exacerbated by motor-induced vibrations in the lightweight chassis, was mitigated using a 5-sample moving average filter on ultrasonic data, reducing fluctuations.

The controller was tuned using a combination of methods:

1. Initial Parameter Estimation:

- Started with conservative values ($K_p = 1.0$, $K_i = 0$, $K_d = 0$)
- Observed system response and stability

2. Manual Fine-Tuning:

- Gradually increased K_p until acceptable response time
- Added derivative control (K_d) to reduce overshoot
- Added minimal integral control (K_i) to eliminate steady-state error

3. Final Parameter Optimization:

After numerous test iterations, the final tuned values were:

- $K_p = 6.0$
- $K_i = 0.007$
- $K_d = 13.9$
- Correction factor for motor imbalance = 0.695

4.3.3 Noise Reduction Techniques

Several techniques were implemented to reduce noise in sensor readings:

1. Moving Average Filter: 5-sample averaging to smooth ultrasonic measurements

cpp

```
float readAverageDistance() {  
    float sum = 0;  
    for (int i = 0; i < 5; i++) {  
        measureDistance();  
        sum += distance;  
        delay(10);  
    }  
    return sum / 5.0;  
}
```

2.

End-of-Wall Detection Validation:

cpp

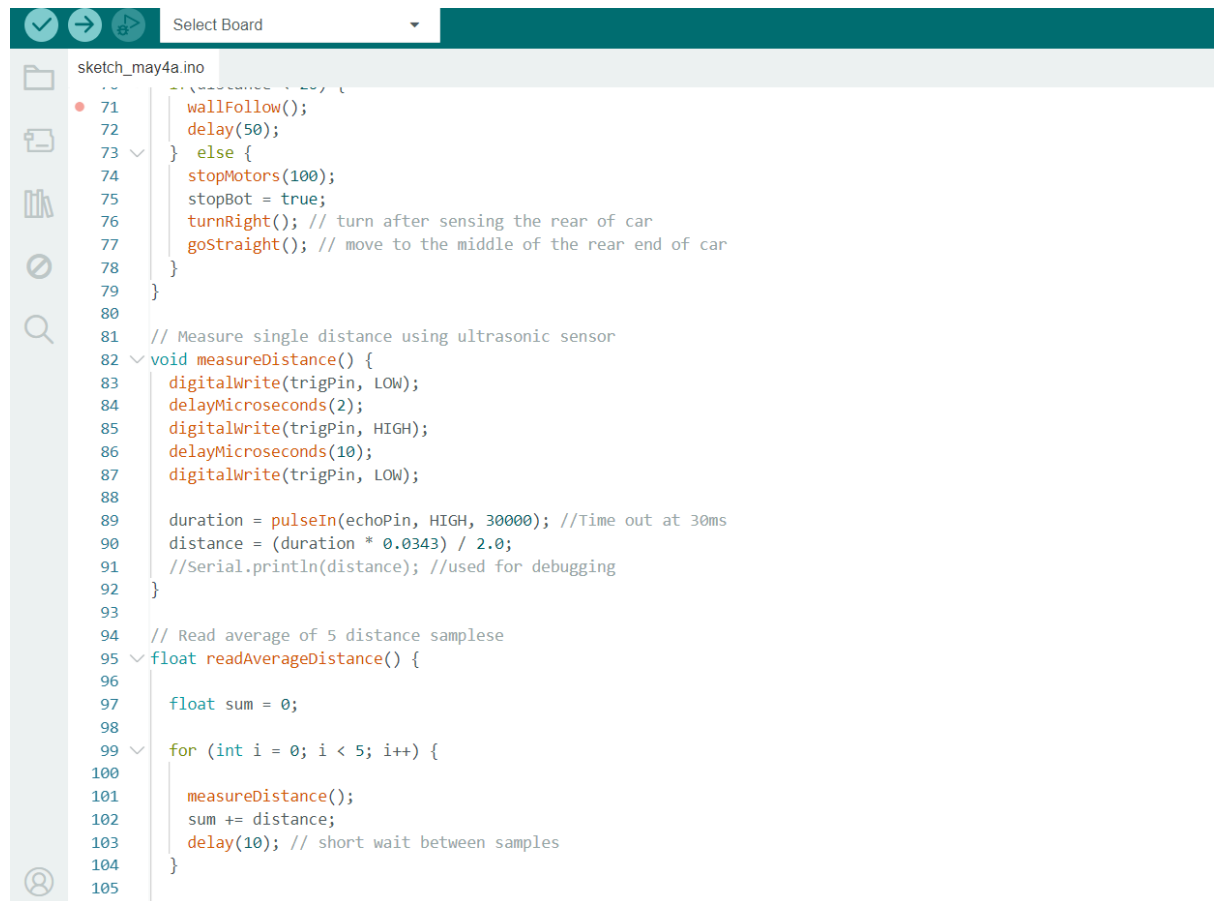
```
if(distance < 20) {  
  wallFollow();  
} else {  
  stopMotors(100);  
  turnRight();  
  goStraight();  
}
```

4.3.4 Full Arduino Program



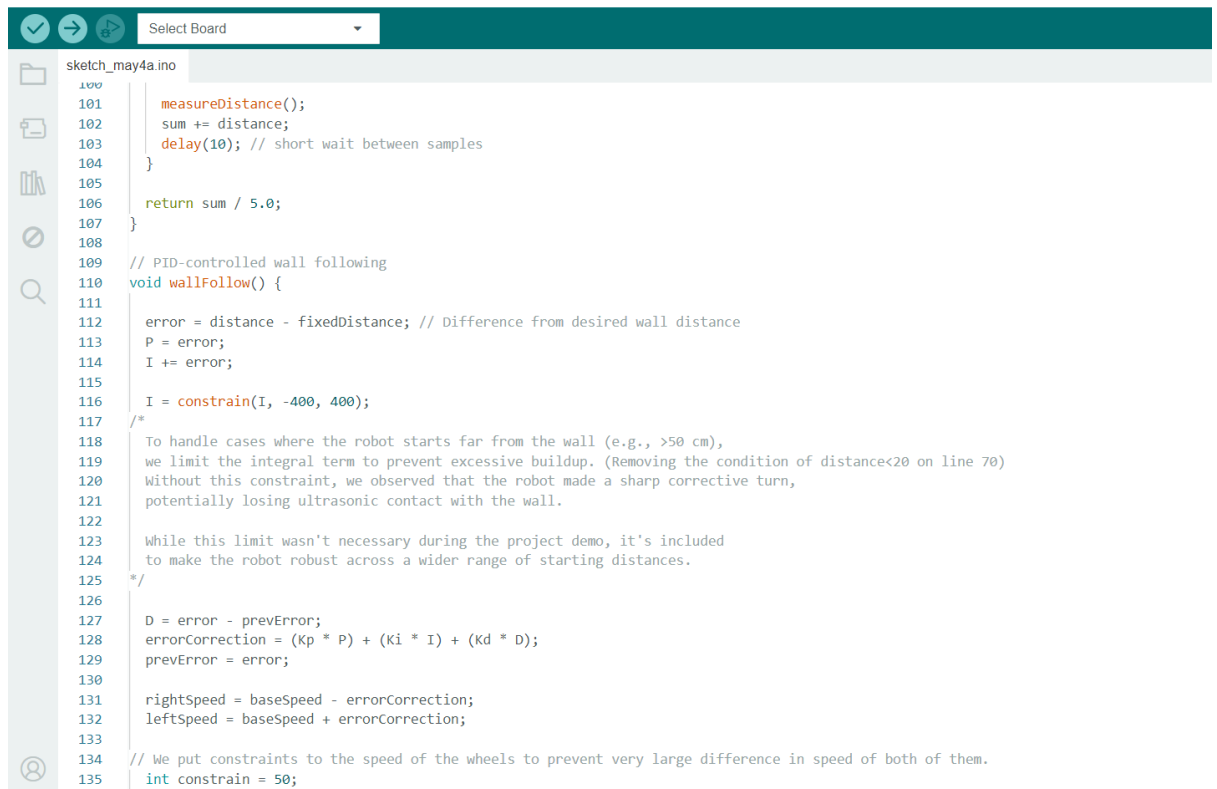
```
sketch_maya4a.ino  
1  /*  
2  Project: Wall-following robot using PID control  
3  Author: Adwait Thosare ME23B123, Group 18  
4  Controller: PID  
5  */  
6  
7  
8  // Motor Driver Pins (L298N)  
9  const int enA = 3;    // PWM pin for right motor  
10 const int in1 = 4;    // Right motor direction pin 1  
11 const int in2 = 5;    // Right motor direction pin 2  
12 const int in3 = 11;   // Left motor direction pin 1  
13 const int in4 = 10;   // Left motor direction pin 2  
14 const int enB = 9;    // PWM pin for left motor  
15  
16 // Ultrasonic Sensor Pins  
17 const int trigPin = 12;  
18 const int echoPin = 13;  
19  
20 // PID Constants (tuned experimentally)  
21 float Kp = 6;  
22 float Ki = 0.007;  
23 float Kd = 13.9;  
24 // Kp could slightly be increased and Kd decreased since we observed that the robot had to cover about 350cm to align when kept 50-60cm away from the wall. However, these values  
25 // worked perfectly if kept at smaller distances  
26  
27 // PID Variables  
28 float P, I = 0, D;    // Proportional, Integral, Derivative terms  
29 float error, prevError = 0; // Current and previous error  
30 float errorCorrection; // PID output  
31 float baseSpeed = 100; // Base motor speed  
32 float rightSpeed, leftSpeed; // Speed after correction  
33 const float fixedDistance = 5.0; // Desired distance from wall in cm  
34  
35 // Distance Measurement  
36 float duration, distance;
```

```
sketch_may4a.ino
27
38 float correctionFactorR = 0.695; // Right motor correction factor (for imbalance)
39
40 bool stopBot = false; // Used to stop the bot from wall-following and proceeding to execute next steps which are taking right turn and going straight
41
42 void setup() {
43   // Motor setup
44   pinMode(enA, OUTPUT);
45   pinMode(enB, OUTPUT);
46   pinMode(in1, OUTPUT);
47   pinMode(in2, OUTPUT);
48   pinMode(in3, OUTPUT);
49   pinMode(in4, OUTPUT);
50
51   // Sensor setup
52   pinMode(trigPin, OUTPUT);
53   pinMode(echoPin, INPUT);
54
55   Serial.begin(9600);
56
57   // Motors initially stopped
58   stopMotors(300);
59 }
60
61 void loop() {
62   if(stopBot) return;
63
64   distance = readAverageDistance();
65
66   Serial.print("Distance: ");
67   Serial.println(distance);
68
69   //to make sure the robot has reached the rear end of the car I have put a condition
70   if(distance < 20) {
71     wallFollow();
72     delay(50);
73   }
```



The image shows the Arduino IDE interface with a sketch named "sketch_may4a.ino". The code is written in C++ and implements a robot's navigation logic using an ultrasonic sensor. The code is as follows:

```
71     wallFollow();
72     delay(50);
73 } else {
74     stopMotors(100);
75     stopBot = true;
76     turnRight(); // turn after sensing the rear of car
77     goStraight(); // move to the middle of the rear end of car
78 }
79 }
80
81 // Measure single distance using ultrasonic sensor
82 void measureDistance() {
83     digitalWrite(trigPin, LOW);
84     delayMicroseconds(2);
85     digitalWrite(trigPin, HIGH);
86     delayMicroseconds(10);
87     digitalWrite(trigPin, LOW);
88
89     duration = pulseIn(echoPin, HIGH, 30000); //Time out at 30ms
90     distance = (duration * 0.0343) / 2.0;
91     //Serial.println(distance); //used for debugging
92 }
93
94 // Read average of 5 distance samplese
95 float readAverageDistance() {
96     float sum = 0;
97
98     for (int i = 0; i < 5; i++) {
99         measureDistance();
100         sum += distance;
101         delay(10); // short wait between samples
102     }
103 }
```



```
100
101     measureDistance();
102     sum += distance;
103     delay(10); // short wait between samples
104 }
105
106     return sum / 5.0;
107 }
108
109 // PID-controlled wall following
110 void wallFollow() {
111
112     error = distance - fixedDistance; // Difference from desired wall distance
113     P = error;
114     I += error;
115
116     I = constrain(I, -400, 400);
117     /*
118     To handle cases where the robot starts far from the wall (e.g., >50 cm),
119     we limit the integral term to prevent excessive buildup. (Removing the condition of distance<20 on line 70)
120     Without this constraint, we observed that the robot made a sharp corrective turn,
121     potentially losing ultrasonic contact with the wall.
122
123     While this limit wasn't necessary during the project demo, it's included
124     to make the robot robust across a wider range of starting distances.
125     */
126
127     D = error - prevError;
128     errorCorrection = (Kp * P) + (Ki * I) + (Kd * D);
129     prevError = error;
130
131     rightSpeed = baseSpeed - errorCorrection;
132     leftSpeed = baseSpeed + errorCorrection;
133
134     // We put constraints to the speed of the wheels to prevent very large difference in speed of both of them.
135     int constrain = 50;
```



```
134 // We put constraints to the speed of the wheels to prevent very large difference in speed of both of them.
135 int constrain = 50;
136 rightSpeed = constrain(rightSpeed, baseSpeed-constrain, baseSpeed+constrain);
137 leftSpeed = constrain(leftSpeed, baseSpeed-constrain, baseSpeed + constrain);
138
139 // Motors go forward
140 digitalWrite(in2, HIGH);
141 digitalWrite(in1, LOW);
142 digitalWrite(in4, LOW);
143 digitalWrite(in3, HIGH);
144
145 analogWrite(enA, (int)(rightSpeed * correctionFactorR));
146 analogWrite(enB, (int)(leftSpeed));
147
148 //Used for debugging
149 //Serial.print("right speed: "); Serial.println(rightSpeed);
150 //Serial.print("left speed: "); Serial.println(leftSpeed);
151
152 }
153
154 // Make a right turn
155 void turnRight() {
156   digitalWrite(in2, LOW);
157   digitalWrite(in1, HIGH);
158   digitalWrite(in4, LOW);
159   digitalWrite(in3, HIGH);
160
161   analogWrite(enA, 0);
162   analogWrite(enB, 150);
163
164   delay(400); // Tuned experimentally
165   stopMotors(50);
166 }
167
168 // Move forward for a short time
169 void goStraight() {
170   digitalWrite(in2, HIGH);
171   digitalWrite(in1, LOW);
172   digitalWrite(in4, LOW);
173   digitalWrite(in3, HIGH);
174
175   analogWrite(enA, (int)(120 * correctionFactorR));
176   analogWrite(enB, (int)(120));
177
178   delay(390); //Tuned experimentally to adjust how far the robot moves
179
180   stopMotors(30);
181 }
182
183 // Stop both motors for a given amount of time(ms)
184 void stopMotors(int time) {
185   analogWrite(enA, 0);
186   analogWrite(enB, 0);
187
188   digitalWrite(in1, LOW);
189   digitalWrite(in2, LOW);
190   digitalWrite(in3, LOW);
191   digitalWrite(in4, LOW);
192
193   delay(time);
194 }
195
196 }
```

Simulink Model

A Simulink model was developed to validate the controller design before implementation. The model included:

1. PID Controller Block:
 - Proportional ($K_p = 6.0$)
 - Integral ($K_i = 0.007$) with anti-windup protection
 - Derivative ($K_d = 13.9$) with low-pass filtering for noise reduction
2. Plant Model:
 - Second-order transfer function as derived above
 - Motor saturation (PWM limits 50-230)
 - Motion dynamics with differential drive
3. Sensor Model:
 - Simulated HC-SR04 with 50 ms update rate
 - 5-sample moving average filter
 - Random noise and measurement delays

Performance Analysis

6. Challenges and Solutions

Challenge	Solution Implemented	Outcome
Ultrasonic sensor noise	5-sample moving average filter	Reduced noise
Motor imbalance	Applied correction factor (0.69) to right motor	Balanced straight-line motion
Integral windup	Implemented constraint on I term	Eliminated oscillations
Variable battery voltage	Lowered base speed (100 PWM)	Consistent performance across battery life
False end detection	Added distance threshold (20 cm)	Eliminated false positives

Team Member Contributions

Each team member contributed significantly to the project:

- **ME23B123 Adwait**

Adwait was our chief code wrangler and PID whisperer. He handled most of the algorithm development, parameter tuning, and code optimization. If something worked, it was probably because Adwait fixed it. He also led the testing and performance evaluation, and was usually the first to spot (and sometimes create) a bug. In addition to his software role, he also pitched in during manufacturing where needed — notably fabricating a custom height extension for the caster wheel — to help ensure the robot came together smoothly.

- **ME23B101 Sumanth**

Sumanth took charge of the report and the chassis, from design to manufacturing, and made sure all the hardware actually fit together. He picked out the components, did the heavy lifting on assembly, and was always up for another round of wiring and testing. His patience during the tuning phase and consistent support made a big difference in getting the robot to perform reliably. If you see a straight line on the robot, Sumanth probably measured it twice.

- **ME23B104 Daniel**

Daniel helped us in testing protocols and performance checks. Daniel mainly assisted during basic testing phases. His calm presence was helpful during evaluations, and he was around to support the process when needed.

- **ME23B155 Amruta**

Amruta contributed to some parts of the code. Her involvement added value in keeping things on track as deadlines approached, and also pitched in with whatever was needed.

- **ME23B117 Thomas**

Thomas was our logistics expert, always ready to hunt down missing components or tools. Thomas helped out by sourcing a few components early in the project. He pitched in when possible, especially in the early logistics.

Budget

Component	Amount ()
Chassis kit	650
L298N	200
Battery Pack (4. AA)	120
Battery Pack (3): Li-Ion (3.7V)	450
Battery holders (3)	200
Insulation tape	15
Double sided tape	15
Breadboard	70
Battery charger	140
Chassis carpentry	250
Total	2110

Note: Our total doesn't include the cost of two Arduinos and two motor drivers that we accidentally fried during testing, and borrowed from friends, hehe :)

Gallery

Gallery:

DRIVE CONTAINING TESTING VIDEOS AND IMAGES!

Conclusion

Designing and implementing an autonomous wall-following robot with PID control proved to be a challenging yet rewarding engineering endeavor, blending theoretical control principles with practical hardware integration.

- **Key Achievements:**

- Successfully implemented a **PID control system** with tuned parameters ($K_p = 5.9$, $K_i = 0.06$, $K_d = 14.97$) to maintain a stable 5 cm gap from walls.
- Addressed integral-induced instability through **anti-windup clamping** ($I = \text{constrain}(I, -100, 100)$), enabling stable operation despite sensor noise.
- Engineered a compact chassis with optimized sensor placement and motor coordination for reliable navigation.

- **Skills Developed:**

- PID tuning via empirical methods (Ziegler-Nichols initialization was tried, but later resorted to manual tuning followed by iterative refinement).

- Sensor fusion and noise mitigation using **5-sample moving average filtering**.
- Hardware-software co-design, including motor driver calibration and power management. We learnt and improved on our:
 - 1. Programming Skills

Arduino C/C++ coding Structuring modular code (functions for measurement, control, motion) Using serial communication for debugging and visualization
 - 2. PID Tuning

Practical experience in adjusting K_p , K_i , K_d for optimal performance Understanding the effect of each term
 - 3. Electronics Hardware

Wiring and interfacing ultrasonic sensors, L298N motor drivers, and DC motors Understanding the role of each pin and component
 - 4. Motor Control

* Controlling motor direction and speed via PWM * Learning the relationship between duty cycle and speed
 - 5. Mechanical Design Insight

Effect of chassis alignment, wheelbase, and center of gravity on motion Importance of symmetry and calibration

• **Lessons Learned:**

1. **Integral Action Nuances:** While K_i improved steady-state accuracy, unconstrained integration caused instability, necessitating anti-windup protection.
2. **Control Systems in Practice:** How PID controllers work in real-world scenarios (not just theory). Understanding tuning challenges (e.g., overshoot, steady-state error, integral windup).
3. **Sensor Integration:** Learned how ultrasonic sensors work and their limitations (e.g., noise, cone angle). Importance of averaging and filtering sensor data.
4. **Embedded Systems Behavior:** Learned the role of real-time constraints, loop timing, and how `delay()` affects system performance. How PWM signals influence hardware (motor speed).
5. **System Debugging:** How to identify issues (e.g., sharp turns, sensor misreads). The importance of serial monitoring for real-time feedback.

- **Challenges Overcome:**

- Mitigating motor-induced vibrations that corrupted sensor readings.
- Ensuring robust end-of-structure detection and turning within spatial constraints.

This project highlighted the iterative nature of control system design, where theoretical models guided initial tuning, but real-world testing was indispensable for refinement. The final implementation demonstrated effective PID control under practical constraints, providing valuable insights into embedded robotics and adaptive control strategies.

Looking back, this project was truly a great learning curve for all of us. From debugging code at midnight to chasing down replacements for damaged components, every challenge taught us something new (even if it was just how not to fry an Arduino). We'd like to extend a special thanks to Manish sir for his guidance, encouragement, and-most importantly-the pizzas on demo day, which definitely boosted both our morale and productivity:-)

References

1. **Arduino. (2023). PID Library.** <https://playground.arduino.cc/Code/PIDLibrary/>
2. **HC-SR04 Ultrasonic Sensor Datasheet**
3. **L298N Dual H-Bridge Motor Driver Documentation**
4. **National Instruments. (2022). PID Theory Explained.** <http://www.ni.com/white-paper/3782/en/>