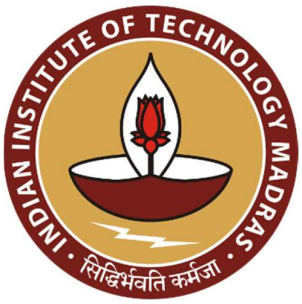


INDIAN INSTITUTE OF TECHNOLOGY, MADRAS

Automated Guided Vehicle Project Report

Instructor: CHARCHIT KUMAR

Group Members

ME23B162 - B. Rahul

ME23B163 - Ch. Arun Kumar

ME23B171 - J. Tej Pavan

ME23B175 - K. Yagna Sreenivas

ME23B191 - R. Vinay

INTRODUCTION

- Automated Guided Vehicles (AGVs) are essential in modern industrial and service automation, enabling efficient and autonomous material transportation.
- A fundamental type of AGV is the line follower, which follows a predefined path or track, typically marked by a line of contrasting colour.
- The objective of this project was to design and develop a mini-AGV capable of autonomously navigating a curved track using line following techniques.
- The AGV was required to follow the track as quickly and accurately as possible while maintaining stability and minimizing deviation from the centreline.
- The development approach integrated mechanical design, electronic systems, and control algorithms to create a compact and reliable vehicle.
- Infrared (IR) sensors were utilized for track sensing, and a PID (Proportional-Integral-Derivative) control strategy was implemented to dynamically adjust steering and ensure accurate tracking.
- This report systematically presents the planning, conceptualization, mechanical and electronic design, software development, system integration, testing procedures, and performance evaluation of the line follower AGV.

PLANNING AND LITERATURE REVIEW

The initial phase of the project involved structured research and planning to design a three-wheeled AGV capable of autonomously following a curved track. To achieve this, we studied various aspects of line-following robots, including control algorithms, sensor types, actuator mechanisms, and chassis design considerations. After gaining a foundational understanding of these systems, we began selecting individual components that would best meet our design and performance goals.

- **Microcontroller:**
  We chose the **Robocraze Improved Version Nano V3 Board** (Arduino Nano compatible) due to its compact size, sufficient I/O pins, and ease of programming.
- **Sensors:**
  An **Analog IR Sensor Array** was selected to allow multi-point detection of the track position with high sensitivity.
- **Motors:**
  TowerPro SG90 Continuous Rotation Servo Motors were used for their superior control over speed and direction compared to standard DC motors.
- **Caster Wheel:**
  A **360° ball caster** was chosen for stability and minimal directional resistance.
- **Power System:**
  The AGV was powered using **INVENTO 5V 1500mAh Ni-MH Rechargeable Batteries** to ensure reliable and portable operation.

During control system planning, we initially considered using only Proportional (P) control. However, after further analysis, we decided to implement a full **PID controller** to improve tracking accuracy, responsiveness, and reduce steady-state errors. The PID algorithm was chosen to dynamically adjust the steering based on real-time and past errors, providing smooth movement even on curved paths.

Research into existing line-following robots helped us understand key aspects such as optimal sensor placement, chassis light-weighting, and efficient power distribution, all of which were incorporated into the final conceptual design. This comprehensive planning and literature review phase laid a strong foundation for the successful development of the mini-AGV

In the following section, we will examine the benefits of each of these components in detail.

Robocraze Improved Version Nano V3 Board:

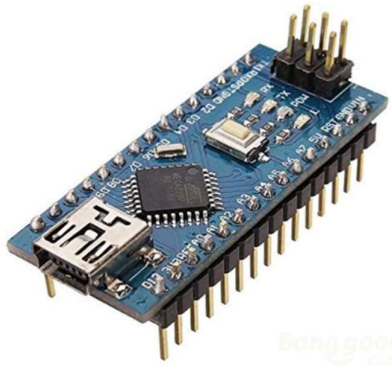


Fig: Arduino Nano

- A compact, breadboard-friendly microcontroller board based on the **ATmega328P** microchip.
- Fully compatible with Arduino Nano V3.0.
- Comes with pre-soldered header pins and a mini-USB connector, simplifying immediate use.
- Improved build quality and stability compared to generic Nano boards.

Advantages:

- **Compact Size:** Small form factor (~45mm × 18mm) makes it ideal for space-constrained projects like mini-AGVs.
- **Breadboard Friendly:** Fits directly into standard breadboards for easy prototyping.
- **Pre-Soldered Pins:** Comes with pre-attached header pins, saving setup time.

- **Ease of Programming:** Easily programmable through Arduino IDE using Mini-USB cable.
- **Sufficient I/O Pins:** Provides 14 digital I/O and 8 analog inputs, supporting multiple components.
- **Reliable Microcontroller:** Based on ATmega328P, ensuring stable and well-documented performance.
- **Built-in Voltage Regulator:** Accepts 7–12V input safely via Vin pin.
- **Cost-Effective:** Offers strong features at a budget-friendly price.
- **Lightweight:** Reduces total system weight, suitable for mobile applications.

Minor Disadvantages:

- **Limited Processing Power:** Operates at 16 MHz, unsuitable for very complex tasks.
- **Lower Current Handling:** Needs external drivers for powering heavy loads.
- **No Native Debugging:** Lacks built-in hardware debugging features.

Analog IR Sensor Array:

Fig: IR sensor before modification

- **Component Overview:**
  We used an **Analog Output IR Sensor Array** for detecting the line on the track. This array enabled the AGV to sense multiple positions simultaneously, improving the accuracy of line tracking.
- **Functionality:**
  The sensor array consists of multiple IR emitter-receiver pairs that detect the

contrast between the black line and the background by measuring reflected infrared light intensity.

- **Advantages of Using an Array:**
  Unlike a single sensor, the array provided a wide coverage area, allowing the controller to detect slight deviations from the track early and respond appropriately.
  Analog Output Feature:
  The analog output provided graded readings (not just binary black/white), enabling smoother PID control by interpreting how far off-centre the AGV was.
  Placement:
  The sensor array was mounted at the front-bottom of the chassis to maintain close proximity to the track and provide real-time feedback for steering corrections.
  Integration with Microcontroller:
  Each sensor's output was connected to the analog input pins of the Arduino Nano, allowing the microcontroller to continuously monitor track position data.
  Sensitivity and Tuning:
  Threshold values for sensor readings were determined experimentally to accommodate varying ambient light conditions and surface textures.
  Role in PID Control:
  The sensor readings were processed to calculate the real-time error, which served as the input for the PID controller to adjust motor speeds and correct the vehicle's heading.
- **Limitations:**
  Care had to be taken during sensor calibration, as excessive ambient light or dirty sensors could lead to false readings.

<u>TowerPro SG90 Continuous Rotation Servo Motors:</u>

Fig: Servo motor

- **Component Overview:**
  We used **TowerPro SG90 Continuous Rotation Servo Motors** for driving the two primary wheels of the AGV, providing both propulsion and steering.
- **Continuous Rotation Feature:**
  Unlike standard servos, these motors can rotate continuously in either direction, making them suitable for wheel-based vehicle movement.
- **Advantages over DC Motors:**
  Servo motors offer finer speed control, easier directional control, and built-in position/speed feedback, unlike conventional DC motors which require external motor drivers and encoders for similar control.
    - **Stability and Precision:**
      Due to internal feedback mechanisms, the servo motors provided **greater stability and smoother motion**, essential for following curved paths without sudden jerks or oscillations.
    - **Simplified Wiring:**
      Servo motors require only three wires (power, ground, and PWM signal), significantly simplifying the wiring compared to a DC motor setup with separate H-bridge circuits.
    - **Ease of Control:**
      The motors were directly controlled through PWM signals from the Arduino Nano, allowing easy and accurate speed adjustments through simple code modifications.
    - **Compact Size and Lightweight:**
      The SG90 servos are compact and lightweight, making them ideal for small, mobile robots where weight and space constraints are critical.
    - **Performance in PID Control:**
      The precision of the servos enhanced the effectiveness of the PID control algorithm by allowing smooth proportional corrections based on the error calculated by the sensor array.
- **Limitations:**
  Servo motors have relatively lower torque and speed compared to high-powered DC motors, which must be considered when designing for heavier loads or higher-speed applications.

INVENTO 5V 1500mAh Ni-MH Rechargeable Batteries:

Fig: Ni-MH Battery

- **Component Overview:**
  Used INVENTO 5V 1500mAh Ni-MH rechargeable batteries as the main power source for the AGV.
- **Voltage and Current:**
  Provided 5V output, ideal for directly powering the Arduino Nano, servo motors, and IR sensor array without extra voltage regulation.
- **Rechargeability:**
  Rechargeable via a 5V 2A DC adaptor charger, reducing long-term costs and promoting sustainable use.
- **Compactness:**
  Small and lightweight enough to fit neatly into the chassis without affecting the AGV's weight much.
- **Power Stability:**
  Delivered stable voltage output for consistent motor speeds and sensor readings during operation.
  Integration:
  Mounted securely on the chassis using adhesive pads for stability during movement.
- **Advantages:**
  Safe chemistry with no memory effect.
  Adequate runtime for track completion and testing cycles.
  Direct compatibility with Arduino and servo operating voltages.

- **Limitations:**
  Gradual voltage drop over long runs may slightly affect speed.
  Slightly heavier compared to Li-Po batteries but acceptable for this project

Wheels:

HASTHIP® 360° Ball Caster Wheel:

Fig: Ball caster wheel

- Component Overview: Used a HASTHIP® 360° swivel ball caster as the third passive support wheel for the AGV.
- Function: Provided stability and allowed free directional movement without adding significant rolling resistance.
- Placement: Installed at the rear of the chassis to balance the two driven wheels and maintain vehicle orientation.
- Advantages:
  Reduced friction during turning.
  Allowed smooth curved path following without drag.
  Lightweight and compact, easy to integrate into the design.
- Limitations:
  Required a smooth surface for optimal rolling; performance could be affected by surface irregularities.

Wheel Attachments: Wheels for SG90/MG90/FS90R Servo Motors:

- Component Overview:
  Wheels designed specifically for SG90, MG90, and FS90R servo motors were used for drive transmission.
- **Compatibility:** Securely attached to the servo shafts, preventing slippage during movement and turns.
  Advantages:
- Lightweight construction helped reduce overall AGV weight.
- Rubberized tires provided better traction on the track.

Fig: Wheels

- Accurate fitting ensured reliable straight and turning motions.
- **Performance Contribution:** Helped maintain consistent speed and steering accuracy, critical for precise line following under PID control.
  Limitations:
- Minor wobble could occur at very high speeds if the wheels were not perfectly aligned during mounting.

Conceptualization:

Chassis:

Initially, while planning the mechanical design of the AGV, our idea was to simply use a normal rectangular plank as the chassis base. The primary goal at that stage was to quickly create a stable platform where the components — motors, sensors, microcontroller, and battery — could be mounted with minimal effort.

However, after deeper consideration, we realized that weight distribution and total mass would play a crucial role in the vehicle's performance. A heavier chassis would increase the load on the servo motors, reduce agility, and potentially slow down the vehicle, particularly on curves where quick response was essential. This insight prompted us to rethink the design approach more carefully.

Recognizing the importance of keeping the AGV lightweight while maintaining structural strength, we decided to move beyond a simple rectangular plank. We began working on a detailed CAD (Computer-Aided Design) model for a custom chassis. The aim was to design a structure that would:

- Accommodate all components efficiently.
- Maintain rigidity where necessary.
- Remove unnecessary material to reduce weight.

The final design consisted of a **lightweight rectangular plate** with **multiple scooped-out slots** strategically placed to minimize material usage without compromising the chassis's strength and durability. Critical areas, such as the motor mount points and sensor attachment zones, were kept solid for stability, while non-load-bearing sections were hollowed out.

After finalizing the CAD model, the chassis was **3D printed** using a durable yet lightweight material (PLA). 3D printing allowed us to precisely realize the designed dimensions and slot placements, ensuring that all electronic and mechanical components could be mounted securely and cleanly.

The key features of the chassis design included:

- **Dimensioning:** 180 mm in length and 120 mm in width, providing ample space for component arrangement.
- **Mounting Slots:** Precisely positioned for servo motors, IR sensor array, Arduino Nano board, and the rechargeable battery pack.
- **Material Optimization:** Hollow sections and optimized thicknesses to lower the overall weight, enhancing the AGV's speed and manoeuvrability.

Chassis CAD Design:

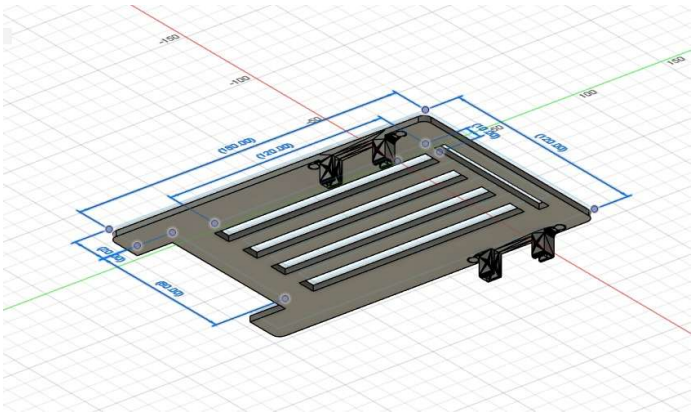


Fig: Chassis side view

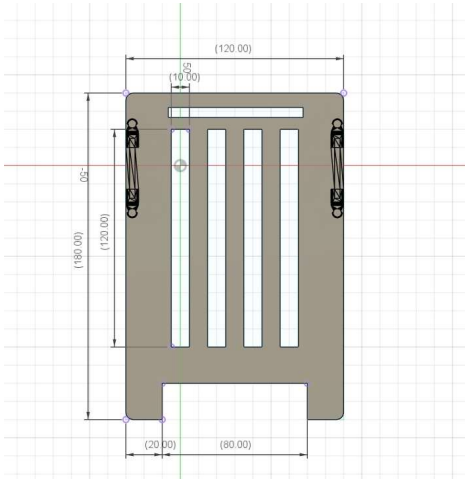


Fig: Chassis top view

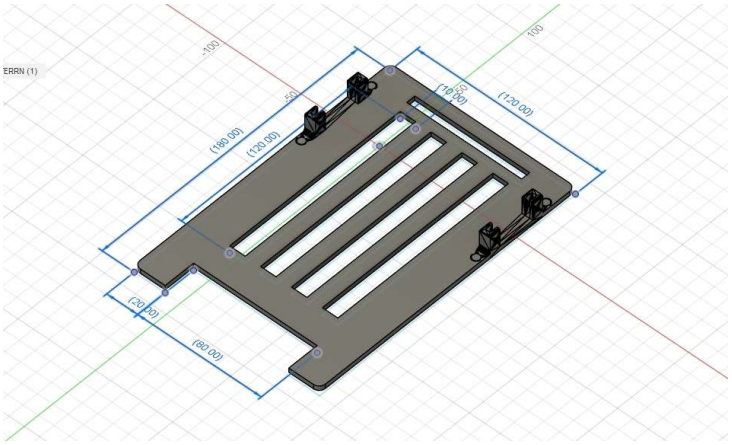


Fig: Chassis bottom view

The **two-wheel drive and caster wheel configuration** was maintained throughout the design, fulfilling the three-wheel AGV requirement stated in the project guidelines. The IR sensor array was mounted at the front bottom of the chassis to closely monitor the track, while the Arduino Nano was centrally located to simplify wiring and balance.

This evolutionary design process, from a basic rectangular plank to a carefully engineered, 3D-printed lightweight chassis, played a key role in improving the AGV's overall performance and set the stage for successful prototyping and testing.

SYSYTEM INTEGRATION

MECHANICAL INTEGRATION:

- The mechanical integration began with positioning all core components on the custom 3D-printed chassis.
- The **Arduino Nano board** was mounted on the upper rear side of the chassis, providing easy access to all I/O pins and minimizing wire length to other components.
- Two **servo motors** were secured on the underside of the chassis using small screws, positioned symmetrically on either side to serve as the primary drive wheels.
- **Servo-compatible wheels** were directly attached to the servo shafts and fastened tightly with screws, ensuring no slippage during motion.
- At the front of the chassis, a **long IR sensor array** was mounted using custom supports, ensuring proximity to the ground for accurate line detection.
- The centre of the chassis housed two 5V rechargeable batteries, held firmly in place with double-sided tape to maintain balance, reduce vibrations, and keep the center of gravity low.
  A **ball caster** was attached to the rear-bottom side of the chassis to provide stability and unrestricted turning, completing the three-wheel AGV configuration.
- All components were integrated such that weight was evenly distributed, minimizing unwanted tilting or wobbling during motion.

ELECTRICAL INTEGRATION:

- Electrical connections were established primarily using jumper wires routed across the chassis, minimizing overlap and reducing signal interference.
- A mini breadboard was used for centralizing all connections, especially for distributing power and ground lines to multiple components like servos and sensors.
- Power was split into two distinct sources:
    1. A **9V battery** supplied power to the Arduino Nano and the IR sensor array, ensuring stable logic-level voltage for control and sensing.
    2. Two **5V rechargeable batteries** were used exclusively to power the servo motors, preventing voltage drops caused by motor loads from affecting sensor accuracy or Arduino performance
- The ground of both power sources was commonly tied to maintain a shared reference across all components, avoiding signal instability.
- Wires were color-coded where possible to simplify debugging and reduce wiring errors.
- Connections to the IR sensor array were made through analog pins (A0–A6), while the servo control signals were connected to digital PWM pins on the Arduino.
- The overall wiring was planned to be compact and neat, as reflected in the **Tinkercad circuit diagram**, ensuring both functionality and serviceability.

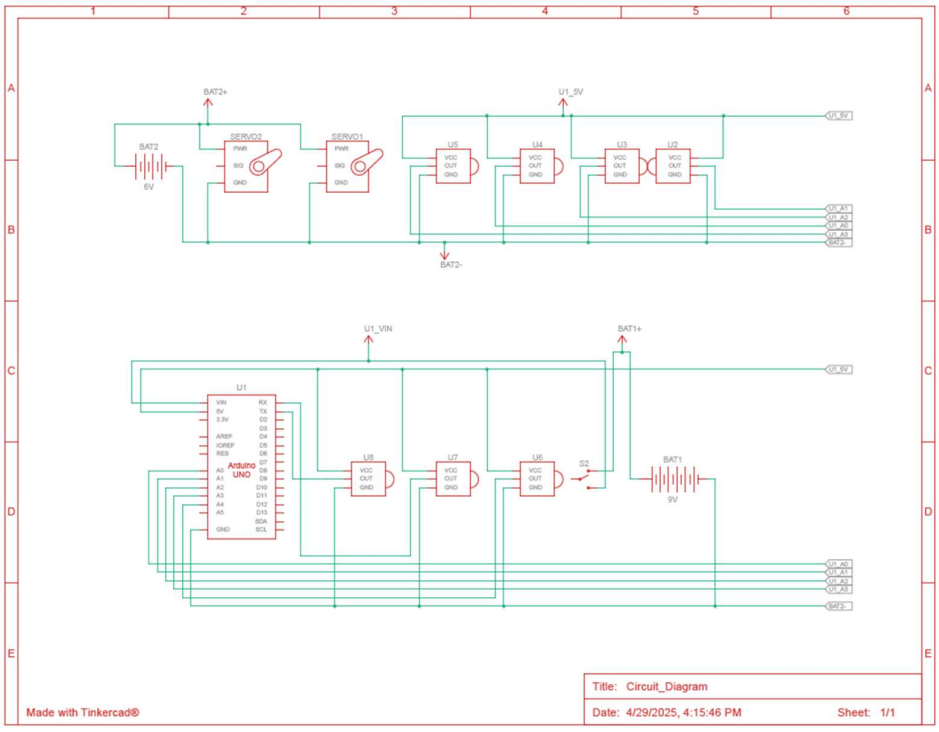


Fig: Tinkercad circuit diagram

SOFTWARE INTEGRATION:

1. **Library Inclusion and Servo Object Creation**
   - The code begins with including the **Servo library**, which allows control over continuous rotation servo motors.
   - Two servo objects are declared — one for the left motor and one for the right motor.

```
1   #include <Servo.h>
2   Servo Servoleft;
3   Servo Servoright;
```

2. **Pin Assignments and PID Variable Declarations**
   - Assigns the digital pins connected to the left and right servos.
   - Defines an array arr[] that holds analog pin numbers connected to the 7 IR sensors.
   - PID constants K_p, K_i, and K_d are declared and tuned for optimal line-following performance.
   - Initializes variables needed for storing error terms and the correction factor.

```
4   const int servoleftPin = 4;
5   const int servorightPin = 6;
6   int arr[7] = {A0, A1, A2, A3, A4, A5, A6};
7   int preverror = 0;
8   float kp = 14;
9   float ki = 0.00;
10  float kd = 8;
11  float correction = 0;
12  int P = 0, I = 0, D = 0;
```

3. **Setup Function**
   - Begins serial communication for debugging (optional).
   - Attaches both servo motors to their respective pins.
   - Initializes starting PWM angles for both servos.
   - Sets all IR sensor pins as input using a loop.

```
13  void setup() {
14    Serial.begin(9600);
15    Servoleft.attach(servoleftPin);
16    Servoright.attach(servorightPin);
17    Servoleft.write(55);
18    Servoright.write(125);
19    for (int i = 0; i < 7; i++) {
20      pinMode(arr[i], INPUT);
21    }
22  }
```

4. **Sensor Reading and Line Detection Logic**
   - Reads analog values from the 7 IR sensors and populates a 1 if the sensor detects the line.
   - Edge sensors (0 and 6) use a greater-than condition for white-line detection; middle sensors use less-than for black-line detection.
   - A boolean flag outofline is used to detect if all sensors miss the line (line loss condition).

```
23  bool outofline = true;
24  int ir[7] = {0, 0, 0, 0, 0, 0, 0};
25  for (int i = 1; i < 6; i++) {
26    if (analogRead(arr[i]) < 300) {
27      ir[i] = 1;
28      outofline = false;
29    }
30  }
31  if (analogRead(arr[0]) > 300) {
32    ir[0] = 1;
33    outofline = false;
34  }
35  if (analogRead(arr[6]) > 300) {
36    ir[6] = 1;
37    outofline = false;
38  }
```

5. **Error Assignment Logic**
   - Calculates the positional error based on the active sensor(s).
   - The error scale is centered at 0 (sensor 3), negative to the left and positive to the right.
   - This error serves as input to the PID controller.

- when the middle sensor (A3) is active, the robot is perfectly aligned with the track, so the error is 0.
- If sensor 4 (right of center) is active, the robot has drifted slightly right, so error is +2.
- If both 3 and 4 are active, it's a mild right drift, so error is averaged to +1.
- If sensor 2 (left of center) is active, the robot has drifted slightly left, so error is -2.
- If both 2 and 3 are active, it's a mild left drift, so error = -1.
- Sensor 1 is further left than sensor 2. If it's active, the robot is further off the line, so error = -4.
- If both 1 and 2 are active, the error is moderated to -3.
- Sensor 0 is extreme left. If it's active, the robot is way off to the left, so the error is large and negative: -7.
- If both 0 and 1 are active, it's slightly better, so the error is set to -5.5.

```
39   int error = 0;
40   if (ir[3]) error = 0;
41   if (ir[4]) error = 2;
42   if (ir[4] && ir[3]) error = 1;
43   if (ir[2]) error = -2;
44   if (ir[2] && ir[3]) error = -1;
45   if (ir[1]) error = -4;
46   if (ir[1] && ir[2]) error = -3;
47   if (ir[0]) error = -7;
48   if (ir[0] && ir[1]) error = -5.5;
49   if (ir[5]) error = 4;
50   if (ir[5] && ir[4]) error = 3;
51   if (ir[6]) error = 7;
52   if (ir[6] && ir[5]) error = 5.5;
```

6. PID Calculation

➢ Calculates the three terms of the PID controller:

- Proportional (P) — based on the current error
- Integral (I) — sum of past errors (disabled with $K_i = 0$)
- Derivative (D) — rate of change of error

➢ The final correction value is computed and saved for actuation

```
53   I += error;
54   D = preverror - error;
55   P = error;
56   correction = kp * P + ki * I + kd * D;
57   preverror = error;
```

- Proportional control(P)

$$P = K_p \times e(t)$$

  - Where e(t) is the current error
  - K_p is the proportional gain
- Integral Control (I):

$$I = K_i \cdot \int_0^t e(t)\ dt$$

  - This term is the accumulation of past errors
  - K_i is the integral gain
- Derivative Control (D):

$$D = K_d \cdot \frac{d(e(t))}{dt}$$

  - This measures the rate of change of the error
  - K_d is the derivative gain

7. Motor Speed Mapping and Line Recovery

- The correction value adjusts the left and right servo speeds.
- If the line is lost, default values are applied to help the AGV reacquire the line.

```
58   int leftspeed = 180 - correction;
59   int rightspeed = 0 - correction;
60   if (outofline) {
61     leftspeed = 60;
62     rightspeed = 120;
63   }
```

When the robot is centered on the line:

- error = 0 → correction = 0
- leftspeed = 180, rightspeed = 0
- This means:
  - Left wheel spins forward
  - Right wheel spins backward (since it's 0, and servo rotation is reversed)
- The AGV moves straight forward

When the robot drifts left (negative error):

Example: error = -3, so correction = negative

- leftspeed = 180 - (negative) → remains same
- rightspeed = 0 - (negative) → decreases

  Result:

  - Left motor remains at same speed
  - Right motor slows down
- The robot turns right to return to the center

When the robot drifts right (positive error):

Example: error = +3, so correction = positive

- leftspeed = 180 - (positive) → decreases
- rightspeed = 0 - (positive) → remains same

  Result:

  - Left motor slows down
  - Right motor remains at same speed
- The robot turns left to return to the center

If line is lost completely (outofline = true)

- The code assigns default speeds:

  leftspeed = 60

  rightspeed = 120

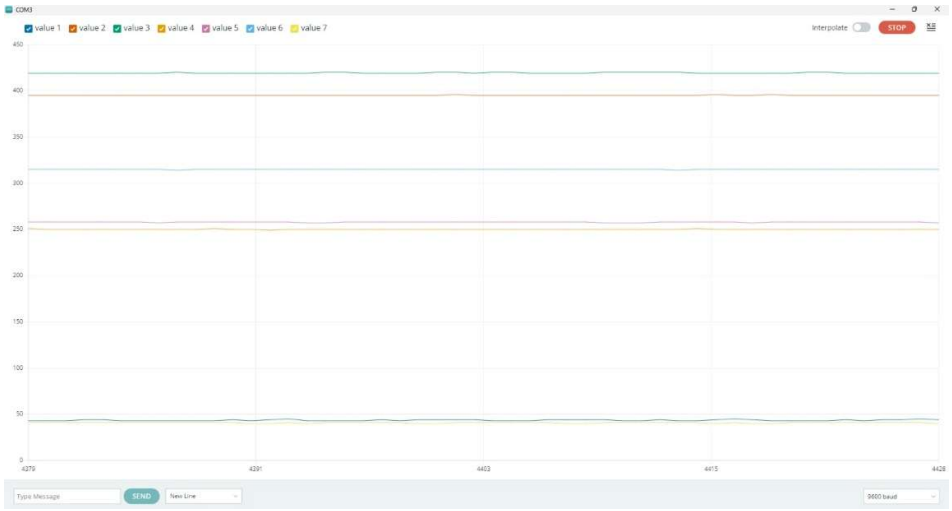
This makes the robot turn in backward motion until it finds the line again.

8. **Final Servo Commands**
   - Sends the calculated PWM signals to both servo motors to steer and propel the AGV.

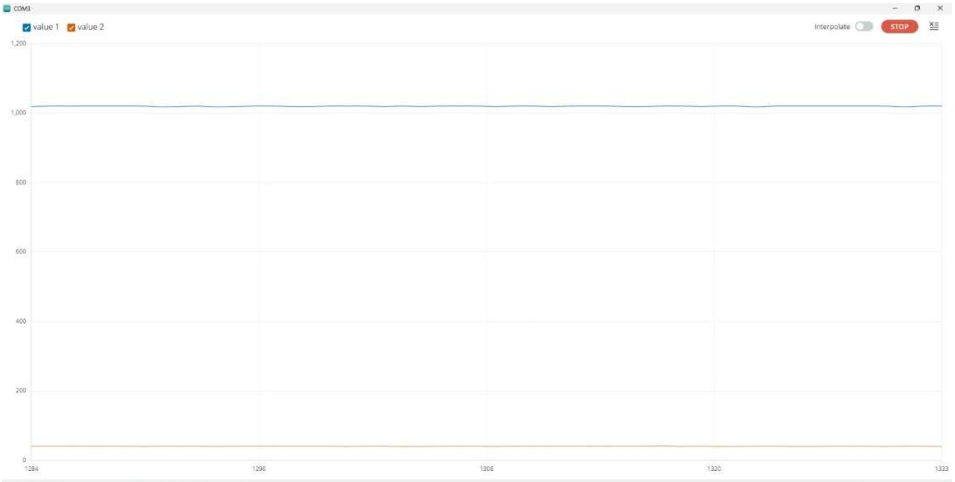
```
65    Servoleft.write(leftspeed);
66    Servoright.write(rightspeed);
```

9.IR testing Serial Plotter

(i)



(ii)



(iii)

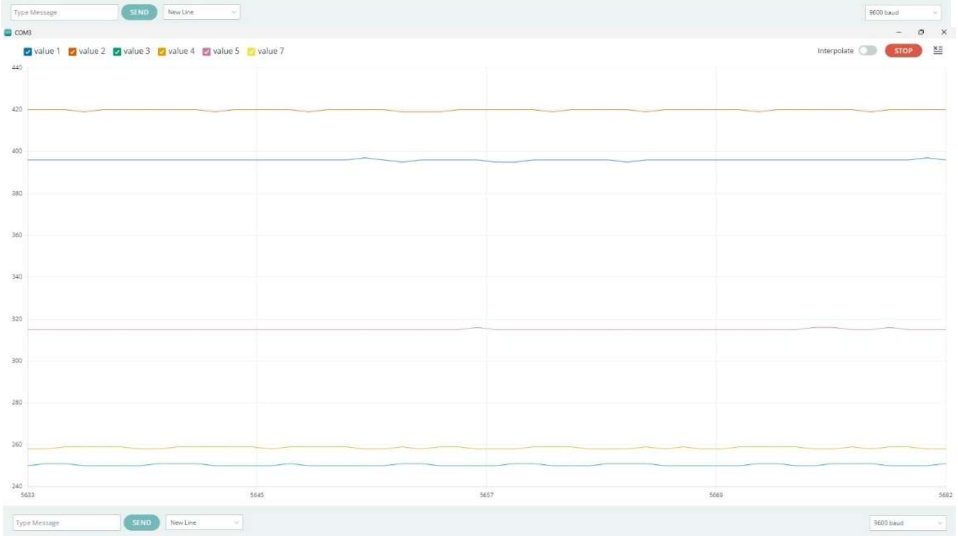


Fig: Serial plot of outputs of IR sensors (i) all 7 sensors (ii) digital output sensors only (iii) analog output sensors only

CHALLENGES AND MODIFICATIONS

1. Chassis Design and Material Selection:

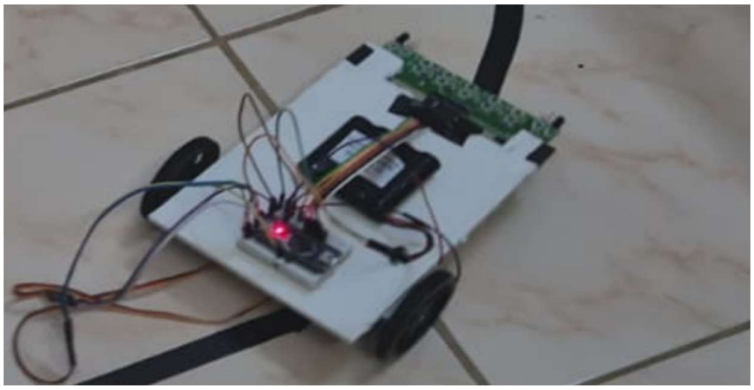


Fig: Initial Prototype of AGV

The first challenge arose during the design of the chassis. Our initial choice was **sunboard**, a lightweight and easy-to-cut material commonly used for prototyping. While sunboard allowed for quick shaping and assembly, it lacked the necessary strength and rigidity to support repeated movement, mechanical vibrations, and component mounting. Over time, it began to flex and deform, especially under the weight of servo motors and batteries, compromising the structural integrity of the vehicle.

To resolve this, we transitioned to a more durable and customizable solution: **3D printing using PLA (Polylactic Acid)**. The first iteration of the 3D-printed chassis was a rectangular plate designed for simplicity and stability. However, once assembled, it became evident that there was excessive material in non-critical areas, which increased weight unnecessarily and limited component access.

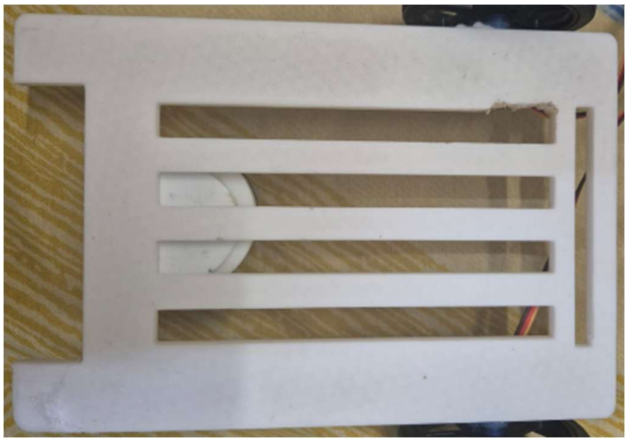


Fig: 3D Printed base

We then iteratively refined the design: removing unneeded sections, optimizing the layout, and adding features such as **mounting extensions for the servo motors**. One of the more subtle challenges was height mismatch between the front-mounted ball caster and the rear servo-mounted wheels. The ball caster sat lower than the wheels, causing imbalance and reduced traction. To solve this, we added **custom circular risers** around the caster to raise it to an appropriate level, allowing smooth and even motion across the surface. These small but critical adjustments made the chassis both lighter and more functionally optimized for the line-following task.

2. IR Sensor Configuration and Line Detection Limitations:

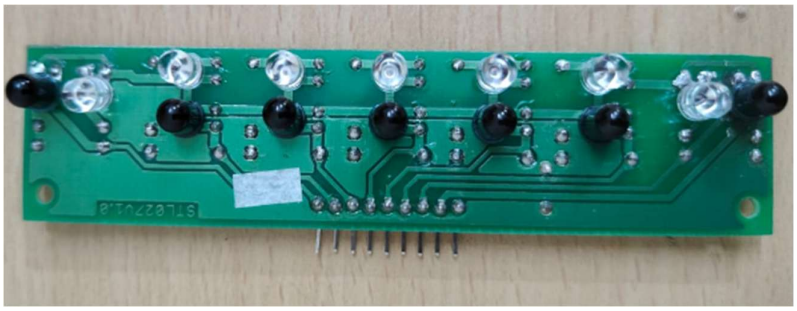


Fig: IR Sensor after modification

Initially, we used a 5-line IR sensor array to detect the black line on a white surface. This array provided direct, centralized detection suitable for linear movement. However, during testing, the robot consistently lost track of the line on sharp turns or irregular paths. The fixed, narrow alignment of the array did not allow sufficient lateral detection, which is essential for early curve recognition.

We attempted to resolve this by using **two additional sensors** that were physically attached to the sides of the main module but not wired for active use. We soldered these two sensors independently and attempted to angle them outward to widen the detection field. Unfortunately, this modification did not produce stable or reliable outputs, likely due to signal integrity issues or poor compatibility with the original board's architecture.

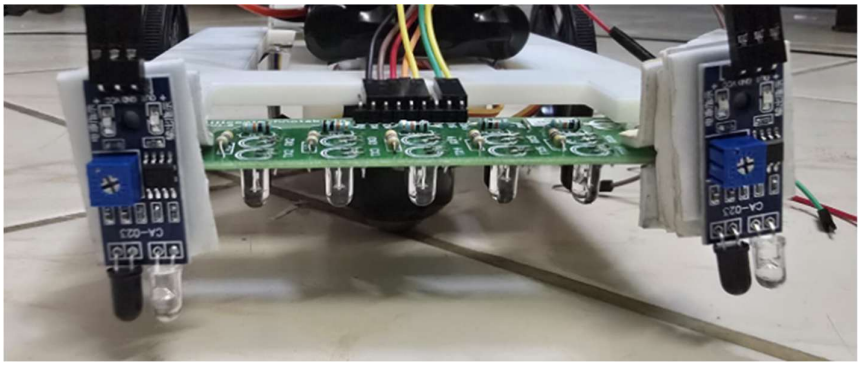


Fig: Final IR sensor used in AGV

Eventually, we abandoned this approach and opted for a more effective solution: we purchased **two standalone digital IR sensors**, each equipped with a potentiometer for sensitivity tuning. These were mounted at custom angles on the front corners of the chassis using double-sided tape, which allowed flexibility in positioning. After calibrating their sensitivity and fine-tuning the angles through multiple trials, these sensors provided consistent detection even during aggressive turns.

The final configuration used **7 IR sensors in total**—5 in the original array for straight-line tracking, and 2 externally mounted digital sensors to detect deviations early during cornering. This hybrid sensor layout greatly improved stability and ensured the robot could handle a wider variety of paths and curve angles.

3. Sensor Calibration and PID Control Tuning:

Even after finalizing the physical layout of the IR sensors, a major challenge remained in terms of **calibrating the sensor data** and implementing an effective control system. The two types of sensors in our configuration—analog and digital—required different approaches.

The central **5-line IR sensor array** outputs analog signals from each of its five sensors. These analog values represent varying levels of reflectivity, typically ranging from 0 to 1023. However, the actual output depends heavily on surface conditions, ambient lighting, and power supply fluctuations. To make use of these values, we had to manually define threshold ranges in the Arduino code to determine whether each sensor was detecting a black line or a white background. Finding the correct thresholds took multiple rounds of testing under different conditions to ensure consistent detection.

In contrast, the **two side-mounted IR sensors** added to improve turn detection were **digital IR modules**. These have built-in comparators and adjustable potentiometers, allowing the detection threshold to be set manually on the sensor itself. They output a binary signal (HIGH or LOW) to the Arduino based on whether the surface beneath them is above or below the configured reflectivity threshold. This simplified software integration, no analog reading or software-level thresholding was needed, but required precise physical placement and sensitivity tuning to ensure they responded appropriately during curves or off-track scenarios.

Once sensor inputs were reliably handled, we implemented a **PID (Proportional-Integral-Derivative) control algorithm** to adjust the robot's motion based on error in line position. The robot's servo motors were mapped using angle values from 0 to 180, where 0 represented maximum forward speed, 180 full reverses, and 90 was the neutral stop position. This servo mapping introduced an added constraint: any

PID correction that pushed the motor input beyond ±90 from neutral could inadvertently reverse the motor's direction, causing instability or confusion in directional control.

Our first tuning attempt used non-zero values for all three PID constants **K_p**, **K_i**, and **K_d**. While this allowed initial corrections, the **integral term (K_i)** caused the system to accumulate error over time. This led to persistent overcorrection and oscillatory behaviour, especially after the robot had travelled some distance, even if it was back on track. Upon analysing this behaviour, we determined that the integral action was counterproductive for our application.

We decided to set **$K_i = 0$** and focused on tuning **K_p** and **K_d** through trial and error. The proportional term was responsible for the robot's responsiveness to line deviation, while the derivative term smoothed out sudden changes, helping prevent sharp jerks or instability. After extensive testing on both straight and curved paths, we arrived at a combination of K_p and K_d values that provided **smooth, accurate, and responsive line-following behaviour**, without overshooting or instability.

4. Line Overshoot and Fail-Safe Recovery:

Even with the tuned PID and sensor setup, real-world conditions revealed a critical flaw, **line overshoot**. Occasionally, the robot would miss a turn or curve so sharply that all sensors lost track of the line. In such situations, the robot continued to move forward blindly, as the code lacked a condition to respond to complete loss of detection.

To address this, we implemented a **fail-safe recovery mechanism** in the Arduino code. The logic continuously monitored whether none of the 7 sensors were detecting the line. If this situation persisted for more than a short duration, the robot was commanded to **reverse slowly**, scanning for the line in reverse until one or more sensors reacquired it. Once detected, the robot would resume normal line-following behaviour.

This simple yet powerful addition significantly improved reliability and allowed the vehicle to **self-correct** during navigation mishaps, especially on sharp corners or intersections where overshooting was more likely.

The implementation of this logic can be verified from the code provided in the Software Integration section.

5. Traction and Wheel Grip Optimization:

Finally, a hardware challenge arose with the **servo-mounted drive wheels**. While the motors themselves were functioning correctly, the plastic wheels they were connected to lacked sufficient friction on smooth surfaces, especially during acceleration or turning. This caused the wheels to **slip or skid**, leading to erratic movement or failure to follow commands accurately.

Our first attempt to resolve this involved **wrapping rubber bands** around the wheel treads. While this temporarily improved traction, the results were inconsistent, one wheel gripped more than the other, resulting in asymmetric behaviour and occasional veering.

To create a more balanced and lasting solution, we applied a thin, uniform **layer of gum-based adhesive** to the wheel surfaces. Once dried, this coating provided a moderate but consistent grip, enough to prevent slippage while maintaining smooth motion. This **low-cost, effective fix** solved the traction issue and contributed to improved stability and turning accuracy during operation.

Demonstration

Fig: Test Track

Fig: Prototype Testing

Final Output:

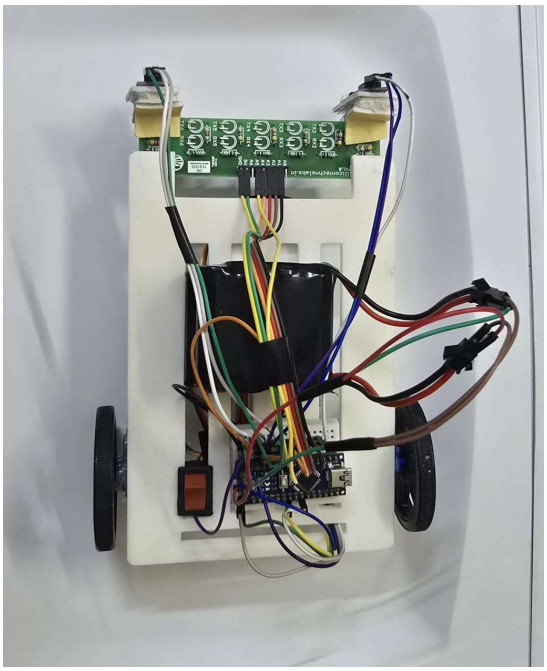


Fig: Top View

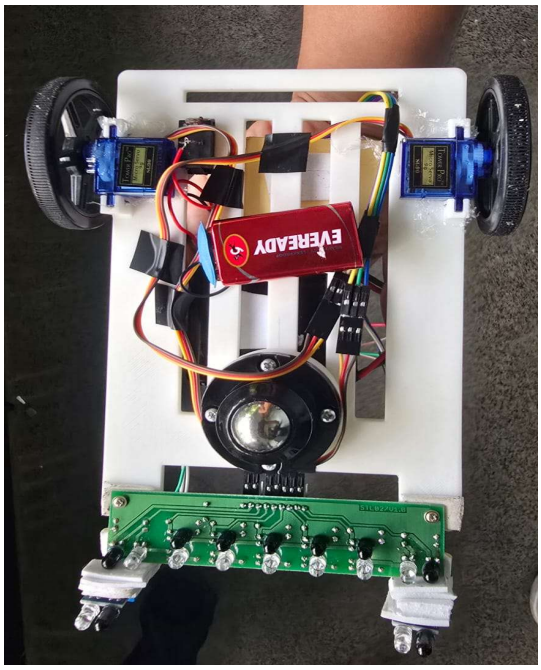


Fig: Bottom View

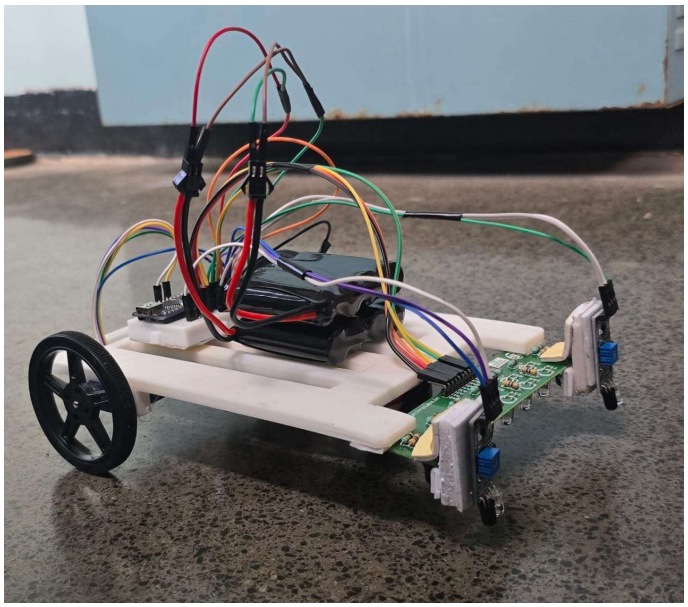


Fig: Side view

Conclusion

The development of our AGV-based line follower involved a systematic approach of design, testing, and iterative improvements across both hardware and software domains. Starting from a lightweight but fragile sunboard chassis, we progressed to a more durable and adaptable 3D-printed PLA structure, incorporating practical modifications for sensor mounting and mechanical balance. Through various stages of experimentation, we overcame key challenges in sensor accuracy, signal interpretation, PID tuning, and traction control. The integration of both analog and digital IR sensors, combined with fallback logic for line overshoot scenarios, significantly improved the system's robustness and reliability. Careful tuning of PID parameters, with the strategic elimination of the integral term, resulted in smooth and responsive line following behaviour. Ultimately, the final prototype demonstrated consistent performance on custom-designed tracks with various curves and junctions, validating the effectiveness of the design decisions and control strategies implemented. This project not only enhanced our understanding of embedded systems and feedback control but also provided practical insights into iterative engineering and system optimization.