GHULAM ISHAQ KHAN INSTITUTE OF ENGINEERING SCIENCES AND TECHNOLOGY

Project Documentation: LINE FOLLOWING ROBOT

Written By: TEAM TECHNO

Table of Contents

Line Following Robot – Documentation

1. Components Used

Component	Quantity	Description
Arduino Uno	1	Microcontroller board to control logic
IR Sensor Module	5	Used to detect black and white surfaces
L298N Motor Driver Module	1	Drives the motors based on signals from Arduino
Yellow Gear Motors	2	DC motors with a gearbox for torque
Wheels	2	Attached to motors for movement
Ball Caster Wheel	1	Acts as support for the front of the chassis
Chassis Plate	1	Base platform to mount all components
3.7V Li-ion Cells	3	Power supply (~11.1V total)
Battery Holder (3-cell)	1	Holds the Li-ion cells securely
Jumper Wires & Connectors	-	For all electrical connections

2. Mechanical Assembly

2.1 Prepare the Chassis

1. General Design

- The chassis is not just a frame — it directly affects **control response, speed, stability**, and **turning accuracy**.
- It should be **strong, lightweight, symmetric**, and large enough to hold **all essential components**.
- A well-planned chassis can **minimize vibrations**, improve sensor readings, and ensure smooth movement on competitive tracks.

2. Dimensions & Form Factor

- **Recommended Dimensions** (for standard LFR builds):
    - **Length:** 15–18 cm
    - **Width:** 12–15 cm
- **Shorter chassis** → quicker turns, better for tight curves.
- **Wider chassis** → better stability, ideal for proper IR sensor spacing and mounting components.

Box or U-shaped profiles are recommended for structural rigidity and better component arrangement.

3. Weight Distribution & Center of Gravity (CG)

- **Mount heavy components low and centered** (e.g., battery, motor driver).
- **CG must be close to the drive wheels**, not the caster, to ensure proper traction and responsiveness.
- Keeping the **load centered** provides **maximum wheel grip** and avoids skidding.
- **Avoid placing components on overhanging edges** to maintain stability during high-speed turns

---

2.2 Attach the Motors

Installation Location and Positioning

Chassis Mounting Requirements: Motors must be securely mounted beneath the chassis at the rear section where maximum vehicle weight is concentrated. This strategic positioning enhances balance, improves traction, and ensures optimal power distribution during operation.

Alignment Specifications: Motor shafts must be positioned parallel to the ground and perpendicular to the chassis to guarantee consistent and smooth movement. Both motors must maintain identical orientation with symmetrical wheel positioning and equal distance from the vehicle centerline to prevent drift or uneven motion.

Mechanical Fastening: Motors should be mounted using appropriately sized screws through pre-drilled holes in the chassis. Avoid adhesives such as glue, which can degrade over time and introduce unwanted vibrations that compromise performance and stability.

Operational Implementation

Control System Integration: Each rear wheel must be controlled by an individual motor to enable precise speed and direction adjustments, significantly enhancing overall maneuverability and control accuracy.

Signal Processing: Implement PWM (Pulse Width Modulation) duty cycle control instead of pure analog signals to maximize wheel traction and improve motor efficiency. This approach provides better torque control and reduces power consumption.

Driver Module Connection: Connect motors to the L298N motor driver module using high-quality jumper wires. Ensure secure connections between motor driver input terminals and Arduino digital pins for reliable signal transmission.

Installation Sequence and Timing

Assembly Order: Motors must be installed and secured to the chassis before connecting the motor driver and other electronic components. This sequence prevents wiring complications and allows for easier troubleshooting during installation.

Pre-Installation Testing: Conduct individual motor testing using a dedicated battery source before final chassis integration to verify proper operation and performance characteristics.

Electrical Connections and Specifications

Terminal Configuration: Each motor features two terminals without predetermined polarity markings. Designate one terminal as positive and the other as negative, maintaining consistency across all motors in the system.

Connection Standards:

- Motor terminals connect directly to L298N motor driver output terminals
- Implement precise soldering techniques for robust connections and uninterrupted current supply
- Use appropriate wire gauge (typically 18-22 AWG) based on current requirements
- Apply heat shrink tubing or electrical tape over all solder joints for protection

Performance Requirements: All motors must demonstrate identical performance characteristics including uniform RPM output, matching torque specifications, and consistent current draw at equivalent voltage levels.

Troubleshooting and Debugging

Direction Correction: If the robot fails to move in the intended direction, swap motor connections on the motor driver module. This simple reversal corrects polarity issues without rewiring.

Power Verification: Ensure both motors receive identical power levels and control signals from the motor driver. Use multimeter measurements to verify voltage consistency across motor terminals.

Performance Monitoring: Measure current and voltage readings for each motor to minimize back EMF effects and resistance-related issues that could impact performance.

Quality Assurance Protocol

Individual Testing Procedure: Test each motor independently using battery power to verify:

- Directional rotation (clockwise and counterclockwise)
- No-load and stall current measurements
- RPM consistency at various voltage levels
- Smooth operation without mechanical binding

System Integration Verification: After installation, confirm that both motors operate with identical RPM and torque characteristics under the same voltage conditions to ensure balanced vehicle movement.

Additional Technical Considerations

Gear System Requirements: Ensure all motors are equipped with appropriate gear reduction systems to provide necessary torque for effective speed control and load handling capabilities.

Electrical Protection: Implement overcurrent protection through fuses or circuit breakers rated appropriately for the motor specifications to prevent damage from electrical faults.

Maintenance Accessibility: Position motors and connections to allow easy access for future maintenance, testing, and potential replacement without requiring complete system disassembly.

This comprehensive approach ensures reliable, consistent, and maintainable motor operation while maximizing vehicle performance and operational longevity.

2.3 Mount the Ball Caster

Strategic Positioning and Design Rationale

Optimal Placement: The ball caster must be mounted at the front center of the chassis to achieve optimal vehicle dynamics. While rear placement might theoretically improve turning responsiveness by allowing front wheels to directly influence steering, front-center mounting provides superior stability and prevents the robot from tipping during rapid directional changes.

Alternative Consideration: Rear chassis mounting could enhance turning capability as the drive wheels would be positioned at the front, providing direct steering influence while the caster acts as a pivot anchor. However, this configuration may compromise straight-line stability and increase the risk of forward tipping.

Installation Requirements and Specifications

Mounting Position: The ball caster must be precisely centered on the front edge of the chassis to ensure balanced weight distribution and prevent rotational bias during movement.

Height Adjustability: Design the mounting system with adjustable height capability to accommodate various arena surfaces and competition requirements. This adaptability ensures consistent ground contact across different operational environments.

Load Distribution: Minimize the load placed upon the caster wheel by ensuring proper weight balance between front caster and rear drive motors, preventing excessive wear and maintaining smooth operation.

Mechanical Characteristics and Performance

Rotational Freedom: The ball caster must rotate freely in all 360 degrees without restriction to facilitate smooth omnidirectional movement and efficient turning capabilities. Any binding or resistance will negatively impact vehicle maneuverability.

Material Selection: Ball-type caster wheels are preferred due to their superior characteristics:

- Lightweight construction reducing overall vehicle mass
- Smooth 360-degree rotation capability
- Simplified mounting requirements
- Reduced friction coefficients compared to traditional wheel casters
- Enhanced durability and wear resistance

Installation Sequence and Assembly

Assembly Timing: Install the ball caster after chassis preparation and rear motor attachment are complete. This sequence prevents interference during motor installation and allows for proper weight distribution assessment.

Mounting Method: Secure the ball caster using appropriately sized screws for maximum reliability. While hot glue gun application may be mentioned as an alternative, mechanical fasteners provide superior long-term stability and serviceability.

Structural Integrity: Ensure the selected ball caster possesses adequate load-bearing capacity to handle the complete robot weight without deformation or failure during operation.

Operational Considerations

Sensor Integration: Calculate and adjust caster height and position to minimize interference with sensor readings and prevent light reflection issues that could affect line-following accuracy.

Performance Optimization: Apply appropriate lubricants sparingly to maintain smooth operation and prevent jamming, particularly in dusty or debris-laden environments.

Maintenance and Troubleshooting

Alignment Verification: If the robot exhibits forward or backward tilting tendencies, immediately check ball caster alignment and mounting security. Improper installation can significantly impact vehicle stability and performance.
Obstruction Detection: Regularly inspect the ball mechanism for hidden obstructions, debris accumulation, or mechanical binding that could impede rotation. Clear any foreign material immediately.

Lubrication Protocol: Apply minimal amounts of appropriate lubricant (silicone-based preferred) when rotation becomes stiff or inconsistent. Excessive lubrication can attract debris and worsen performance.

Electrical and Control Considerations

No Electrical Requirements: The ball caster operates purely mechanically and requires no electrical connections, power supply, or control signals, simplifying overall system design and reducing potential failure points.

System Integration: Ensure caster operation complements the motor control system without introducing unwanted drag or resistance that could affect programmed movements.

Quality Assurance and Testing

Functional Verification: After installation, conduct comprehensive testing to verify:

- Unrestricted 360-degree rotation
- Smooth rolling motion without binding
- Proper ground contact pressure
- Absence of wobbling or instability

- Consistent performance across various surface types

Load Testing: Verify the caster can support the robot's operational weight without deformation, ensuring long-term reliability and consistent performance throughout competitive use.

Advanced Considerations

Surface Adaptability: Consider caster ball material selection based on anticipated operating surfaces (smooth floors, textured surfaces, outdoor terrain) to optimize traction and durability.

Vibration Dampening: Ensure the mounting system provides adequate vibration isolation to prevent sensor interference and maintain stable robot operation during high-speed maneuvers.

This comprehensive approach ensures optimal ball caster performance while maximizing robot maneuverability, stability, and operational reliability across diverse competitive environments.

2.4 Fix the Wheels

Strategic Positioning and Mounting Requirements

Chassis Placement: Wheels must be mounted at the rear end of the chassis, directly attached to the motor shafts for optimal power transmission and control. This rear-wheel drive configuration provides superior traction and stability during line-following operations.

Symmetrical Alignment: Both left and right wheels must be positioned with perfect axial symmetry to ensure straight-line movement without drift or orientation deviation. Any misalignment will result in uncontrolled veering that compromises navigation accuracy.

Wheelbase Optimization: Maximize the distance between wheels according to Line Following Robot (LFR) specifications to facilitate sharp, fast turns while maintaining stability. Wider wheelbase configurations improve turning responsiveness and reduce the turning radius required for directional changes.

Mechanical Installation Standards

Shaft Attachment: Wheels must be securely attached to motor shafts using mechanical fasteners rather than adhesives. Mechanical connections provide superior alignment precision, enhanced safety margins, and improved serviceability compared to glue-based mounting systems.

Wobble Elimination: Ensure complete absence of wheel wobble during installation, as loose wheel mounting creates erratic movement patterns that severely compromise robot performance—effectively creating a "drunk robot" behavior that prevents accurate line following.

Rotational Freedom: Verify that wheels spin freely without friction interference from chassis components, wiring harnesses, or other mechanical obstructions. Any drag or binding will reduce efficiency and affect movement precision.

Wheel Specifications and Selection Criteria

Size Uniformity: Both wheels must be identical in size and type to ensure balanced movement characteristics. Mismatched wheels create differential rolling resistance that leads to unwanted turning tendencies and navigation errors.

Diameter Considerations:

- **Larger wheels**: Provide increased speed but reduced torque output and may elevate the robot's center of mass, decreasing overall stability
- **Smaller wheels**: Offer superior control for tight turns and better torque utilization, making them preferable for line-following applications where **control takes precedence over speed**

Radius vs. Contact Area Balance: Maintain small wheel radius while maximizing surface contact area with the ground to optimize traction without sacrificing maneuverability or increasing rotational inertia.

Material Selection and Performance Characteristics

Rubber Composition: Utilize medium-soft rubber material wheels to achieve optimal balance between traction and controlled slipping. This material provides sufficient grip for reliable movement while allowing necessary slip during turning maneuvers.

Surface Interface: Ensure wheels have appropriate diameter and superior ground grip to prevent slipping during acceleration, deceleration, and turning operations. Inadequate traction compromises positioning accuracy and response time.

Traction Enhancement: If wheel slipping occurs during operation, upgrade to high-grip rubber materials or consider tread pattern modifications to improve traction characteristics without significantly increasing rolling resistance.

Structural and Load-Bearing Requirements

Weight Support Capacity: Wheels must possess adequate structural strength to support the complete robot weight without deformation, flexing, or failure during operation. Insufficient load capacity leads to premature wear and performance degradation.

Quality Standards: Never compromise on wheel quality, as wheels play a crucial role in line-following robot performance. Invest in high-quality components that maintain dimensional stability and consistent performance throughout competitive use.

Center of Mass Considerations: Avoid oversized wheels that elevate the robot's center of mass, as this configuration reduces stability and increases the risk of tipping during rapid directional changes or when traversing uneven surfaces.

Installation Verification and Testing

Alignment Inspection: Conduct thorough alignment verification to ensure both wheels track parallel paths during straight-line movement. Use precision measurement tools to confirm symmetrical installation.

Performance Testing: Test wheel rotation under various load conditions to verify:
- Smooth rotation without binding or resistance
- Consistent grip characteristics across different surface types
- Absence of vibration or irregular motion patterns
- Proper torque transmission from motor to wheel

Advanced Configuration Considerations

Differential Performance: Ensure both wheels provide identical rolling resistance and traction characteristics to prevent unintended differential movement that could affect line-following accuracy.

Surface Adaptability: Consider wheel tread patterns and rubber compounds based on anticipated competition surface characteristics (smooth floors, textured surfaces, dusty conditions) to optimize performance.

Maintenance Accessibility: Position wheels and mounting hardware for easy inspection, maintenance, and replacement without requiring extensive disassembly of other robot components.

Troubleshooting and Optimization

Slip Detection: Monitor wheel performance for signs of slipping, which indicates insufficient traction or excessive applied torque. Address slip issues through material upgrades or surface preparation rather than accepting reduced performance.

Wear Pattern Analysis: Regularly inspect wheel wear patterns to identify alignment issues, surface irregularities, or load distribution problems that could affect long-term performance and reliability.

Dynamic Balance: Verify that wheel mounting maintains dynamic balance during high-speed operation to prevent vibration transmission that could interfere with sensor accuracy or mechanical stability.

This comprehensive approach ensures optimal wheel performance while maximizing robot speed, control, and reliability across diverse competitive line-following scenarios.

2.5 Add the Battery Holder

Strategic Positioning and Weight Distribution

Central Chassis Placement: The battery holder must be mounted centrally on the chassis, positioned near the robot's center of mass to maintain optimal weight distribution and ensure smooth turning capabilities. This central positioning prevents front or rear weight bias that could compromise stability and maneuverability.

Proximity to Critical Components: Install the battery holder close to the motor driver and center of gravity for enhanced balance and reduced wire lengths. Avoid placement at extreme front or rear positions, as this disrupts weight distribution and negatively impacts robot performance.

Upper Surface Mounting: Position the battery holder on the upper surface of the chassis to provide easy access for battery replacement and maintenance while maintaining proper weight distribution throughout the system.

Mechanical Installation Requirements

Secure Attachment Methods: Firmly attach the battery holder to prevent any movement that could affect robot stability during operation. Use mechanical fasteners such as screws through pre-drilled holes at the holder corners and corresponding chassis mounting points.

Removable Installation Design: Create a removable mounting system by drilling holes in both the battery holder corners and chassis, allowing easy detachment for battery charging and maintenance without permanent modifications to the robot structure.

Installation Sequence: Install the battery holder only after mechanical components (motors, caster wheel, and chassis) are securely attached and properly aligned. This sequence ensures optimal weight distribution calculation and prevents interference during primary component installation.

Battery Selection and Capacity Planning

Holder Type Selection: Choose any battery holder type based on project requirements and preferences, considering the number of cell slots needed for desired voltage and capacity specifications.

Cell Configuration: Select holders with optimal cell section quantities to achieve required voltage output. Calculate total voltage as the sum of all installed cell voltages (e.g., 3 × 4V AA = 12V total).

Battery Type Considerations:

- Fresh AA batteries: Suitable for basic applications but may experience voltage drops under load
- Li-ion batteries: Provide consistent voltage output and higher capacity for demanding applications
- Rechargeable options: Offer long-term cost efficiency and environmental benefits

Electrical Integration and Safety

Power Switch Implementation: Install a dedicated switch between the battery holder and motor driver to enable convenient power on/off control without battery removal. This feature enhances operational safety and extends battery life.

Voltage Regulation: Implement voltage regulator circuits to stabilize power supply and ensure consistent robot operation regardless of battery discharge levels. This prevents performance degradation as batteries deplete.

Arduino Power Connection: Connect the battery system to Arduino Uno's 12V input terminal, ensuring proper voltage compatibility and adequate current supply for all connected components.

Wiring Standards and Connectivity

Polarity Verification: Use jumper wires to connect battery holder to motor driver and Arduino power inputs, strictly observing correct polarity to prevent component damage. Mark positive and negative connections clearly for future reference.

Connection Security: Verify all connections are secure with no loose wires that could cause short circuits, power interruptions, or safety hazards. Use appropriate wire gauge for expected current loads and implement strain relief where necessary.

Wire Management: Route power wires carefully to avoid interference with moving components, sensors, or mechanical assemblies. Use cable ties or wire management systems to maintain neat, professional installations.

Quality Assurance and Testing Protocol

Pre-Installation Verification: Ensure all batteries are fully charged before installation in the holder. Verify that the holder delivers full voltage equivalent to the combined voltage of all installed cells.
Voltage Testing: Measure output voltage under various load conditions to confirm adequate power delivery and identify potential voltage drop issues before system integration.

Connection Integrity: Test all electrical connections for continuity, proper polarity, and absence of short circuits using appropriate test equipment.

Troubleshooting and Maintenance

Performance Issues and Solutions:

Problem	Cause	Solution
Weak robot performance	Low battery voltage	Replace or recharge batteries
Voltage drops under load	Insufficient battery capacity	Use fresh AA or properly rated Li-ion batteries
Holder overheating	Short circuit conditions	Immediately check all connections and wiring
Loose battery fit	Inadequate holder retention	Wrap batteries with thin tape for secure fit
Power disconnection	Loose battery contacts	Use fasteners to secure batteries and ensure tight connections

Advanced Configuration Considerations

Battery Monitoring: Consider implementing battery voltage monitoring systems to provide early warning of low charge conditions and prevent unexpected shutdowns during operation.

Backup Power: For critical applications, design redundant power systems or emergency backup batteries to maintain operation during primary battery failure or replacement.

Thermal Management: Ensure adequate ventilation around the battery holder to prevent overheating, particularly when using high-capacity batteries or operating in demanding conditions.

Load Distribution: Calculate total system current draw and verify battery capacity can sustain operation for required duration while maintaining adequate voltage levels throughout the discharge cycle.

Safety and Compliance

Short Circuit Protection: Implement appropriate fuses or circuit breakers to protect against overcurrent conditions that could damage batteries, wiring, or connected components.

Battery Chemistry Considerations: Follow manufacturer guidelines for specific battery types, including charging procedures, storage requirements, and disposal protocols for environmental compliance.

Maintenance Schedule: Establish regular inspection intervals for battery condition, connection integrity, and holder mechanical security to ensure continued reliable operation.

This comprehensive approach ensures reliable, safe, and maintainable battery power systems while maximizing robot performance and operational duration across diverse competitive scenarios.

---

3. Electronics Assembly

3.1 Mount the Arduino Uno

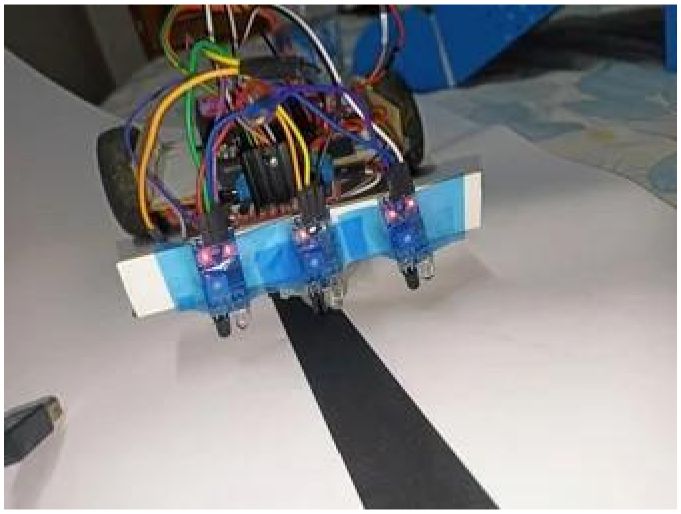
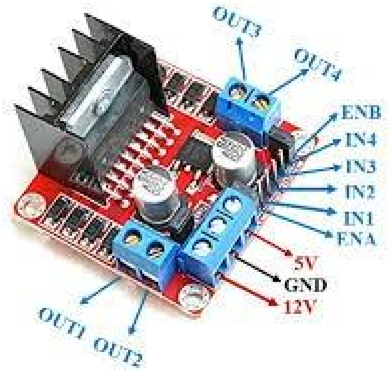
- 

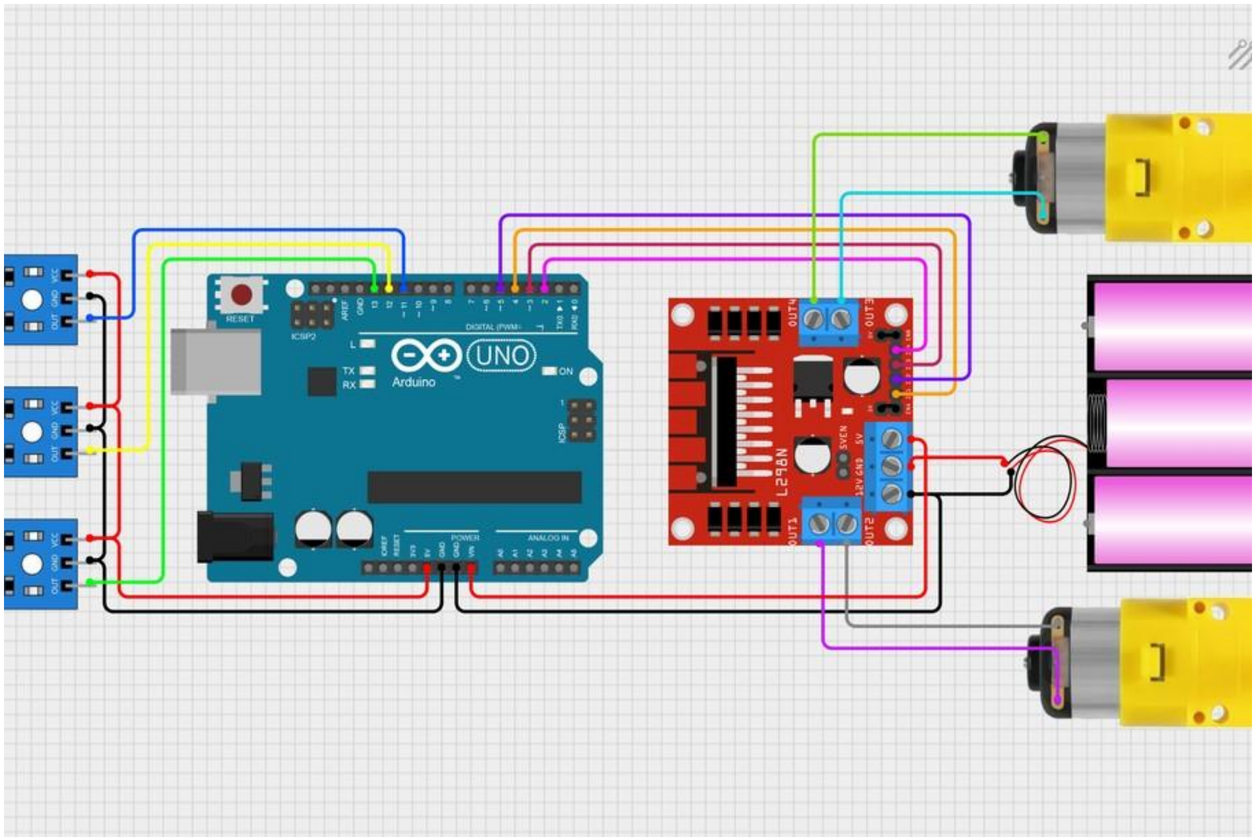
  Figure: Example wiring for a 3-sensor line-follower robot. Three IR reflectance modules (left, mid, right) each tie VCC→5V and GND→GND on the Uno, with their digital OUT pins going to Arduino inputs (e.g. D13, D12, D11).. An L298N motor driver is powered by a 6–12V battery pack (VCC) and shares ground with the Arduino; its OUT1–OUT4 terminals attach to the two DC motors, and its IN1–IN4 inputs are driven by Arduino pins (for example D2–D5). The driver's enable pins (ENA/ENB) should be pulled HIGH (jumpered to 5V or driven by PWM) to activate the motorsi. Note: many L298N boards do *not* supply a regulated 5V output, so the Arduino often must use its own 5V regulator (Vin or USB) rather than the driver's 5V

- **IN1 (LM1):** Arduino D2
- **IN2 (LM2):** Arduino D3
- **IN3 (RM1):** Arduino D4

- **IN4 (RM2):** Arduino D5



-



Mounting the Arduino Uno

- **Placement:** Mount the Uno near the chassis center for balance and stability. Keep it away from high-vibration parts (motors) and excessive heat (battery). Orient it so USB or power jacks remain accessible. On metal chassis use plastic or nylon standoffs (≈M2 or #4-40) to insulate the board from the chassis.

- **Securing:** Fasten the Uno firmly – use screws and nuts (or nylon bolts) through its mounting holes when possible. If there are no mounting holes, double-sided foam tape or nylon cable ties can hold it securely (foam also damps vibration). For quick builds a blob of hot glue works, but avoid gluing any components (sensors, shields) that may need repositioning.

- **Weight distribution:** Install the Uno so its weight (≈25g) contributes to an even center of gravity. For a typical 2WD robot, this often means the Uno sits centrally above or between the motors. Avoid cantilevering it far at one end of the chassis, which can tip the robot.

IR Sensor Array (3–5 sensors)

- **Sensor modules:** Use IR reflectance modules (e.g. TCRT5000 or CNY70 board) which have three pins: VCC (to 5V), GND, and digital OUT. Mount them near the front edge of the chassis, pointing down at the floor. For 3 sensors, place one at center and the others symmetrically spaced on left/right. Five-sensor robots often space them ~10–20mm apart to cover wider linesinstructables.com.

- **Height & alignment:** Adjust each sensor about 3–8 mm above the ground (≈5 mm is typical) to reliably distinguish black vs. whitecircuitdigest.com. Ensure all sensors lie in a straight line across the chassis and are oriented parallel to the floor. Test different heights: too high (>10mm) may miss the line, too low (<2mm) may falsely trigger.

- **Wiring:** Daisy-chain all sensor GND pins to Arduino GND. Tie all sensor VCC pins to the Arduino's 5V output. Run each OUT pin to a separate Arduino digital input (choose any unused D-pin). For example, a 3-sensor bot might use D13=left, D12=middle, D11=rightinstructables.com. If your modules have an onboard potentiometer, **calibrate each sensor** by placing it over a white surface and turning the pot until the module's LED just turns off, then verifying the LED lights solidly over the black linecircuitdigest.com.

- **Signal behavior:** Most IR reflectance modules output **HIGH** when over a light surface and **LOW** over a dark linekevsrobots.com. In software you can invert this logic or use it directly: e.g. `if(digitalRead(sensorPin)==LOW)` might mean "line detected."
  Always test each sensor individually: read its pin on white vs. black and confirm it toggles as expected.

Motor Driver and Motors

- **Driver choice:** Use an H-bridge driver (e.g. L298N or L293D) since Arduino pins cannot directly drive motors. The driver's outputs connect to the motors, while its inputs connect to the Arduino. Mount the driver board close to the motors if possible (shorter motor leads reduce noise).

- **Connections:** Wire each motor coil to one pair of driver outputs: typically motor1→OUT1/OUT2 and motor2→OUT3/OUT4 on L298N. On L293D chips, outputs are labeled 1Y,2Y and 3Y,4Y.

Connect the driver's INx pins to your chosen Arduino digital pins. For example, one common mapping is:

- ○ Left motor forward/back inputs (IN1, IN2) ← Arduino D2, D3
- ○ Right motor inputs (IN3, IN4) ← Arduino D4, D5instructables.com
  Set the driver's enable pins (ENA for IN1/2, ENB for IN3/4) HIGH (to 5V) if using no speed control, or to Arduino PWM pins (e.g. D6, D7) if you want to modulate speed. (On many L298N modules, a jumper can tie ENA/ENB to 5V.)instructables.com

- **Power to driver:** Feed the motor supply (battery+) to the driver's VCC/Vs pin (6–12V range is typical)circuitdigest.com. The driver's ground (GND) **must** connect to the Arduino GND to establish a common referenceinstructables.com. On L298N modules, if they have a 5V regulator (sometimes there are jumpers), the 5V output can power the Arduino, but be cautious: some cheap L298N boards do *not* regulate reliablyinstructables.com. Many builders instead power the Arduino separately (via USB or Vin) and only use the battery for the motors.

- **Motor wiring:** Connect the motor wires (often red/black) to the driver's output terminals. If motors spin in the wrong direction, simply swap the two wires for that motor or swap which IN pins drive it in code. Verify each motor spins by hand-test: set one IN high, the other low, and check the motor turns in one direction; reverse them to turn backward.

- **Driver enable pins (L293D):** If using an L293D chip (16-pin), remember to tie pin1 and pin9 (the enable pins) HIGH (connect to 5V) so the motor outputs are enabledcircuitdigest.com. Without this, the motors will not move.

Power Supply and Battery Pack

- Make sure the voltage suits your motors' ratings. The Arduino's Vin pin can accept 7–12V to feed its regulator, or power via USB (5V) if the driver's regulator is stable.

- **Wiring the battery:** Mount the battery holder securely on the chassis (use screws or strong double-sided tape). Connect the battery **positive (+)** through a power switch to the motor driver's VCC input. Connect the battery **negative (–)** to the driver's ground terminal. Also tie that negative to the Arduino GND (common ground). For example, a typical connection is: battery+ → switch → L298N VCC; battery– → L298N GND and Arduino GNDinstructables.com. This way the battery drives the motors (and any 5V regulator on the driver) while the Arduino shares ground.

- **Voltage regulation:** If your battery exceeds 12V, use a DC regulator or drop resistor to protect the Arduino (which accepts ≤12V on Vin). Do **not** power motors from the Arduino 5V pin – it cannot supply motor currentforum.arduino.cc. Likewise, avoid powering the Arduino by injecting voltage into its 5V pin (bypasses its regulator). Stick to Vin or USB.

Wiring and Grounding Best Practices

- **Wire routing:** Bundle each pair of power wires together (e.g. battery+ and battery–) and each signal-return pair together. Route motor power wires separately from sensor wires to minimize electromagnetic interferenceforum.arduino.cc. Avoid forming large loops: keep wires as direct as possible. Secure groups of wires with zip-ties or adhesive clamps
  to keep the layout tidy and strain-free.

- **Common ground:** Tie all grounds (battery–, motor driver GND, Arduino GND, sensor GND) at a single point. This prevents ground loops and ensures all parts share the same reference voltage. For example, connect the Arduino GND and driver GND to the battery negative at one junction.

- **Decoupling:** Place a large electrolytic capacitor (e.g. 470µF) across the driver's power input to smooth motor noise. Also put 0.1µF ceramic caps close to the Arduino's Vcc/GND pins. Optionally, ferrite beads on motor leads can reduce interference.

- **Avoid loose connectors:** Use snug dupont wires or soldered connections. Loose or thin jumpers can intermittently break contactforum.arduino.cc. If using breadboards, know they are unreliable for high-current connections – soldered joints or terminal blocks are better for motors.

- **Unpowered wiring: Never connect or disconnect wires while the system is powered** – this can short something or send spikes back into the boardforum.arduino.cc. Always turn off power (or unplug USB) before reworking wiring.

Programming (Non-PID Logic)

- **Sensor reads:** In setup(), use pinMode(sensorPin, INPUT); for each IR sensor pin. In the loop, use digitalRead(pin) to get its state. By convention, you may consider LOW (0) as "on black line" and HIGH (1) as "on white surface." For example: bool onLine = (digitalRead(sensorPin) == LOW);.

- **Control logic:** A simple "bang-bang" controller uses if/else statements on the sensor bits. For instance, with three sensors L/M/R:

  ○ If L=0 and R=0 (both edge sensors see line), go *forward*.

  ○ If L=1 (no line) but R=0, then the line is on the right – turn **right**.

  ○ If R=1 and L=0, turn **left**.

  ○ If all three see white (no line), you've lost the line – stop or reverse.

  - If all three see black (wide black area), treat as an intersection or stop pointinstructables.com.

    You can implement this with nested if() checks or a switch on the combined sensor bits. Document your truth table in comments for clarity.

- **Motor commands:** Tie motor driver inputs high/low in code to drive the motors. For example, to go forward: set left motor IN1=HIGH, IN2=LOW; right motor IN3=LOW, IN4=HIGH (actual pins depend on your wiring). To turn, stop one motor or reverse it. Always call digitalWrite(enablePin, HIGH) (or analogWrite for PWM) in setup() to enable the outputs.

- **Calibration in code:** Since sensors output digital HIGH/LOW, no complex math is needed. Simply ensure your code's HIGH/LOW interpretation matches reality: e.g. test a sensor over white ground and check if digitalRead returns HIGHkevsrobots.com.
  Adjust your logic if it's inverted (some modules invert output polarity).

- **Non-PID note:** This guide assumes simple on/off control. There's no PID loop; decisions are made purely on the current sensor pattern. Your turns will therefore be immediate and possibly abrupt, which is fine for basic line-following.

Serial Monitor & Debugging

- **Using Serial Monitor:** In setup(), call Serial.begin(9600);. Then inside loop(), add Serial.print() statements to output sensor states and any debug info. For example:

```
Serial.print("L="); Serial.print(digitalRead(pinLeft));

Serial.print(" M="); Serial.print(digitalRead(pinMid));

Serial.print(" R="); Serial.println(digitalRead(pinRight));
```

- Open the Arduino IDE's Serial Monitor to see these values live. This helps verify each sensor's behavior and the code's decision pathinstructables.com.

- **Sensor testing:** Place the robot stationary and manually move it so only the line passes under sensors. Observe the serial output or blink an LED on each sensor pin to ensure they flip HIGH/LOW at the right moment. Calibrate their pots if neededcircuitdigest.com. Check that when on white the reading is one state, and on black the opposite.

- **Motor checks:** If a motor doesn't turn, disconnect it and test the driver output with a multimeter: applying HIGH/LOW to its input pins should toggle the voltage on the output terminals. Also ensure the driver ENA/ENB are pulled HIGH and that the Arduino pins driving INx are actually toggling as per your code. You can write a simple sketch to spin one motor continuously to isolate driver vs. code issues.

- **Power checks:** Measure the battery voltage under load. A fresh pack should read in its normal range (≈7.4–9V for a 6×AA pack)circuitdigest.com. If the Arduino resets or motors stall, the battery may be sagging. Verify 5V is present on the Arduino's 5V pin when it's powered. Use a multimeter or add a voltage divider on an analog pin to read battery voltage if needed.

Do's and Don'ts – Tips & Warnings

- **Do** wire power and ground pairs together (twist or shorten return path)forum.arduino.cc. This minimizes EMI and voltage drops.

- **Do** double-check all GND connections: the Arduino, sensor array, motor driver, and battery negative **must** share a common ground.

- **Do** tie unused digital inputs to a defined state. If you don't use an IR input, set it to OUTPUT or enable the internal pull-up to avoid "floating" noise.

- **Don't** power any motor directly from the Arduino's 5V pinforum.arduino.cc. That pin cannot supply enough current and may damage the board (motors draw amps!). Always power motors from the external battery via the driver.

- **Don't** feed the Arduino's 5V pin from a battery or motor supply unless it's a regulated 5V source; prefer using Vin or USB. (Feeding 5V directly bypasses the onboard regulator and can allow spikes back into the board.)forum.arduino.cc

- **Don't** leave signal jumpers loose. A loose wire can make the robot behave erratically. Ensure all connectors are pushed fully onto the headersforum.arduino.cc. For long-term builds, soldering critical wires is best.

- **Don't** rewire while the robot is powered. Always shut off power when making changesforum.arduino.cc.

- **Don't** rely on breadboard for final wiring – it's fine for prototyping but poor for motors. Use solder or screw terminals for the motor driver and battery connections.

- **Do** tidy up wiring after each test. Label or color-code your wires, and secure them so that turning or moving the motors won't pull on connections.

3.2 Connect the Motor Driver (L298N)

● L298N Driver

● This guide explains how to use an L298N dual H-bridge motor driver to control two DC motors in a simple line-following robot (non-PID). In this setup, the ENA and ENB enable pins are tied high via onboard jumpers.

<u>LFR without ENA & ENB pins:</u>

Module Placement on the Chassis

- **Choose a stable location:** Mount the L298N on a flat part of the chassis, ideally near the center or close to the battery. This keeps motor wires reasonably short and balances weight. Avoid placing it too close to sensitive sensors to minimize electrical noise interference.

- **Orientation:** Position the board so its heat sink (and mounting holes) face up or outwards for cooling and easy access. The L298N is metal-backed and can dissipate heat better when unobstructed.

- **Physical mounting:** If your L298N board has mounting holes (often one on each side), use screws and plastic or metal standoffs to secure it firmly. If no holes are available, use strong double-sided tape or a hot-glue pad. Ensure the board is insulated from any metal surfaces of the chassis to prevent short circuits (e.g. place it on an insulating spacer or attach insulating washers under the screws).

Power Supply Wiring (Battery to L298N)

- **Battery positive to VCC:** Connect the positive terminal of your battery pack to the L298N's VCC (motor power) input screw terminal. For example, if using a 7.4–12V battery (2S Li-ion or 6–10 NiMH cells), wire its + terminal to the "+12V" or "VCC" label on the driver.

- **Battery negative to GND:** Connect the negative terminal of the battery to the L298N's GND terminal. This provides the return path for motor current. Use a heavy-gauge wire (at least 18 AWG for high-current motors) to handle motor currents without large voltage drops.

- **Onboard 5V regulator:** If your battery voltage is between about 7V and 12V, leave the onboard regulator jumper in place. The L298N will then generate a 5V output on its "5V" screw terminal that can power the Arduino's 5V input. If you use this 5V output, still wire the Arduino's 5V pin to this terminal. (Do **not** apply battery +V directly to the Arduino 5V pin!) If your battery pack is **below** ~7V or **above** ~12V, remove the regulator jumper and supply a stable 5V to the L298N's 5V pin from an external source (e.g. the Arduino's 5V regulator or a separate 5V regulator).

- **Decoupling capacitor:** It's good practice to place a large electrolytic capacitor (e.g. 470 µF–1000 µF, 25V) across the L298N's VCC and GND terminals. This helps smooth voltage spikes when the motors start or change direction.

Arduino Control Connections (IN1–IN4)

- **Common ground:** First, connect the Arduino's GND pin to the L298N's GND terminal (the same ground used by the battery). This common reference is essential for the control signals to work.

- **IN1–IN4 to Arduino outputs:** Choose four digital output pins on the Arduino Uno (e.g. 2, 3, 4, 5). Connect them as follows:

    - L298N **IN1** ← Arduino digital pin (e.g. D2)

    - L298N **IN2** ← Arduino digital pin (e.g. D3)

    - L298N **IN3** ← Arduino digital pin (e.g. D4)

    - L298N **IN4** ← Arduino digital pin (e.g. D5)
      These inputs control motor direction: for Motor A (OUT1/OUT2) use IN1/IN2; for Motor B (OUT3/OUT4) use IN3/IN4.

- **Enable pins (ENA/ENB):** With the jumpers installed, ENA and ENB are tied to +5V, enabling each motor at full speed. You do not need to wire these to the Arduino. If later you want speed control, you would remove the jumpers and connect ENA/ENB to PWM-capable Arduino pins.

- **Signal wiring tips:** Use short jumper wires for the IN pins and keep them away from motor power wires to reduce noise coupling. Secure the wires so they don't vibrate loose.

Motor Output Connections (OUT1–OUT4)

- **Motor A:** Take your first DC motor (e.g. left wheel motor) and connect its two leads to **OUT1** and **OUT2** on the L298N (which form the terminals of H-bridge A). It does not matter which lead goes to which pin initially; swapping them simply reverses the motor's direction.

- **Motor B:** Connect the second motor (e.g. right wheel motor) to **OUT3** and **OUT4** (H-bridge B). Again, wire orientation only affects spin direction, which can be fixed in software or by swapping wires.

- **Wire gauge:** Use appropriate wire thickness (e.g. 20 AWG or thicker) for motor leads. Motor currents can be significant, so thicker wires reduce voltage drop.

- **No direct Arduino connection:** Motors connect only to the driver outputs, not to the Arduino.

Grounding and Power Distribution Best Practices

- **Common ground point:** Tie all grounds together at one point: battery negative, L298N GND, Arduino GND (and any sensor grounds). Ideally form a star ground with short leads meeting at the L298N GND terminal or a small bus. This reduces ground loop issues.

- **Separate power paths:** Keep motor power (battery to L298N to motors) separate from low-power sensor logic. If possible, route power and ground for the Arduino/sensors on separate wires from the battery/regulator.

- **Use proper wire sizes:** Use heavier gauge wire for battery and motor power lines. Smaller gauge wire can be used for Arduino and sensor power (5V) or control signals. This prevents voltage sag on the main supply lines.

- **Noise suppression:** Attach small (e.g. 0.1 µF ceramic) capacitors across each motor's terminals (wired directly at the motor) to suppress electrical noise. This helps prevent EMI from affecting the Arduino.

- **Voltage monitoring:** Consider adding a voltage divider to an Arduino analog pin or a battery fuel gauge, so you can monitor battery voltage and know when it's low.

Mounting and Securing the L298N Module

- **Use standoffs or screws:** As mentioned, use the module's mounting holes if available. Typical L298N boards have holes near the heat sink. Fasten with screws and plastic or metal standoffs to keep the board elevated off any conductive surfaces.

- **Insulation:** If the underside of the L298N has exposed solder points or metal, insulate it from the chassis with nylon washers or adhesive plastic tape. This prevents accidental shorts if the chassis is metal.

- **Vibration resistance:** Secure the module so it does not wiggle. Vibration can loosen screw terminals and connectors. For extra stability, anchor any loose wires with zip ties or clips so they cannot tug on the driver.

- **Heat dissipation:** Ensure the heatsink and metal parts have airflow. Avoid covering the module with additional layers. If the robot enclosure is tight, allow some space around the L298N for heat to escape.

- **Dust protection:** In dusty or damp environments, consider a protective cover that still allows air flow. Keep the connector heads clear of debris.

Wire Management Tips

- **Neat routing:** Route wires along the chassis beams or edges using cable ties or clips. Keep motor wires grouped and Arduino/control wires grouped separately. This reduces electromagnetic interference between high-current and low-power lines.

- **Twisted pairs:** For each motor, twist the two power wires together. This further reduces electromagnetic interference.

- **Label wires:** If possible, label or color-code wires (for example: red=VCC, black=GND, yellow/green=control signals). This makes troubleshooting much easier.

- **Avoid sharp bends:** Leave gentle curves instead of tight kinks in wires. This avoids stress that can eventually break the wire insulation or conductor.

- **Strain relief:** Provide slack and secure tie-down points at connectors so that plugging and unplugging or bumping doesn't pull on solder joints.

Debugging Common Issues

Follow these systematic steps to diagnose problems:

- **Motors Not Spinning at All:**

    1. **Power check:** Ensure the battery is charged and connected correctly (battery + to VCC, – to GND). Measure the voltage at the L298N VCC and GND with a multimeter.

    2. **Enable jumpers:** Verify that the ENA and ENB jumpers are in place (these tie enables high). Without them, motors will not run.

3. **Arduino outputs:** Use a simple test program to set IN pins HIGH/LOW manually. For example, set IN1 HIGH and IN2 LOW and see if Motor A spins.

4. **Common ground:** Confirm the Arduino GND is connected to the driver GND. A missing ground is a very common issue.

5. **Motor wires:** Check that the motor leads are firmly screwed into OUT terminals and not shorted or broken. Test the motors by directly powering them from the battery (briefly and carefully!) to ensure they work.

6. **Overcurrent shutdown:** If the L298N senses too much current, it may shut down. Disconnect one motor and test the other individually. If one motor is seized or shorted, that can halt both channels on some modules.

- **Motors Run in Wrong Direction:**

1. **Swap motor connections:** On the L298N, swap the two wires of the affected motor (swap OUT1/OUT2 or OUT3/OUT4). This reverses the "forward" direction.

2. **Invert control logic:** In your Arduino code, invert which pin you set HIGH versus LOW for forward/backward. (e.g. change from IN1 HIGH, IN2 LOW to vice versa.)

3. **Test individually:** Drive each motor separately through the L298N to confirm its behavior. This isolates wiring from coding issues.

4. **Crossed wires:** Make sure you haven't inadvertently mixed up which IN pins go to which motor. Label or trace each connection to be sure.

- **Voltage Drops or Brown-Outs:**

1. **Check battery health:** A weak or discharged battery can drop voltage under load. Measure battery voltage with motors stopped and while one motor is running. Significant sag indicates the battery or its wiring is insufficient.

2. **Thick power wires:** Use thicker wires from battery to L298N (and between L298N and motors) to reduce IR losses.

3. **Separate power for Arduino:** If Arduino resets when motors start, try using the L298N's 5V to power the Arduino (if not already) or add a separate regulator for the Arduino. Ensure grounds remain common.

4. **Capacitors:** Add or increase capacitance across the motor supply to buffer transients.

5. **No long wires:** Keep battery wiring short to avoid inductance and resistance causing dips.

- **Driver Overheating:**

    1. **Feel the heat sink:** It’s normal for the L298N heatsink to get warm, but if it becomes very hot (too hot to touch), current is too high.

    2. **Reduce load:** Use smaller/less current-hungry motors or lighter weights. Check motor current draw; typical DC toy motors should not draw more than ~1–2 A each on stall if possible.

    3. **Heat sink and fan:** Ensure the heat sink is properly attached. In extreme cases, add a small fan or a larger heat sink to assist cooling.

    4. **Duty cycle:** Avoid running the motors at maximum continuously if not needed; intermittent operation gives time to cool.

    5. **Check for shorts:** Ensure no wires are shorted and no unintended high current path exists. A short can rapidly overheat the chip.

- **Interference and Noise Issues:**

    1. **Signal noise:** If the Arduino behaves erratically when motors run, ensure you have decoupling caps on the motors and that sensor wires are not placed right next to motor wires.

    2. **Resetting or glitches:** If the Arduino resets under motor load, double-check your power supply stability and add an electrolytic capacitor (like 100–470 µF) at the Arduino’s 5V input.

    3. **Sensor errors:** If line sensors give false readings near motors, add shielding or reroute their wires away from the motors and driver. Ferrite beads on motor leads can also help.

Safety Considerations

- **Correct polarity:** Always confirm battery positive goes to VCC and negative to GND. A reversed connection can instantly damage the L298N and possibly the Arduino.

- **Fuses or protection:** To protect against shorts or battery failures, use a suitable fuse or PTC resettable fuse on the main battery line.

- **Voltage limits:** Do not exceed the L298N's rating. The L298N can handle motor voltages up to ~35 V, but avoid going that high (stay below 12–18 V for reliability and safe regulator use). Ensure the battery voltage matches the motor ratings.

- **Isolation:** Keep fingers and metal objects clear of exposed terminals when powered. The L298N can draw large currents briefly.

- **Heat management:** Never touch the heat sink while testing under load; it can burn you. Provide adequate cooling.

- **Disconnecting power:** Always disconnect the battery before making wiring changes. Connect positive and negative in the proper sequence (ground first, then positive) to minimize sparks.

- **Environment:** Operate the robot in a dry, clean area. Avoid water or conductive debris that could short the driver board.

Integration in the Robot System

The L298N serves as the interface between the Arduino brain (which reads sensors and decides movement) and the motors (which drive the wheels). In a typical line-following robot:

1. **Sensors:** A line sensor array or individual IR sensors are mounted at the front. These sensors detect the line (usually a dark strip on a lighter surface). Arduino's 5V powers them and sends signals to Arduino input pins. They share the common ground of the system.

2. **Arduino processing:** The Arduino reads the sensor inputs and, based on its line-following logic (e.g., simple threshold checks or more advanced control), determines how the robot should move (forward, turn left, turn right, etc.). Because this is a non-PID setup, the logic might be as simple as: "If the left sensor sees white and the right sees
black, turn right" and so on.

3. **Control signals:** The Arduino then sets the L298N inputs accordingly. For example:

   - **Move forward:** Set IN1=HIGH, IN2=LOW (Motor A forward) and IN3=HIGH, IN4=LOW (Motor B forward). Both motors spin forward at full speed.

   - **Turn left:** For instance, stop or reverse the left motor while the right motor runs forward. One way: IN1=LOW, IN2=LOW (left motor off) and IN3=HIGH, IN4=LOW (right motor forward).

   - **Turn right:** Opposite: left forward, right off.

○ **Pivot (in-place turn):** Reverse one motor and forward the other (IN1=HIGH, IN2=LOW and IN3=LOW, IN4=HIGH) to spin the robot around its center.

4. **Power delivery:** When an INx pin is HIGH (and the opposite INy is LOW), the L298N connects the motor to the battery in a given polarity, driving current through the motor at full battery voltage. Since ENA/ENB are tied HIGH, every time the Arduino activates a motor channel, it delivers full voltage/power. To stop a motor, the Arduino sets both IN pins the same (either both LOW or both HIGH), which disables that motor output.

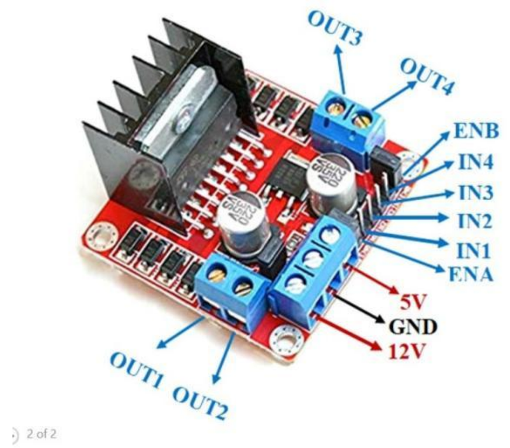
5. **Common grounds:** The sensors, Arduino, and L298N all share the same ground, ensuring the sensor and motor driver signals have the same reference. The Arduino is typically powered from the L298N's 5V regulator output or from the battery via its own regulator, tying into the same ground.

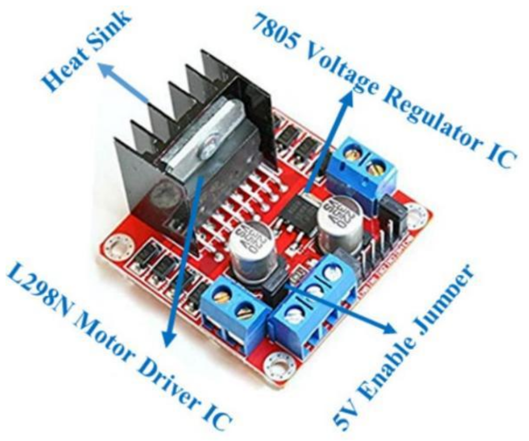
In summary, the L298N module is the "power bridge" of the robot. The Arduino handles low-power signals (from sensors) and sends simple HIGH/LOW commands to the L298N inputs. The L298N then takes power from the battery and switches it to the motors accordingly. By following the wiring and mounting guidelines above, and by carefully debugging any issues, you can ensure that the motors respond predictably to the Arduino's commands, allowing the robot to follow lines as intended.

LFR with ENA & ENB pins:

❖ **BASIC OVERVIEW OF MOTOR DRIVER L298n:**

This L298N Motor Driver Module is a high power motor driver module for driving DC and Stepper Motors. This module consists of an L298 motor driver IC and a 78M05 5V regulator. L298N Module can control up to 4 DC motors, or 2 DC motors with directional and speed control.

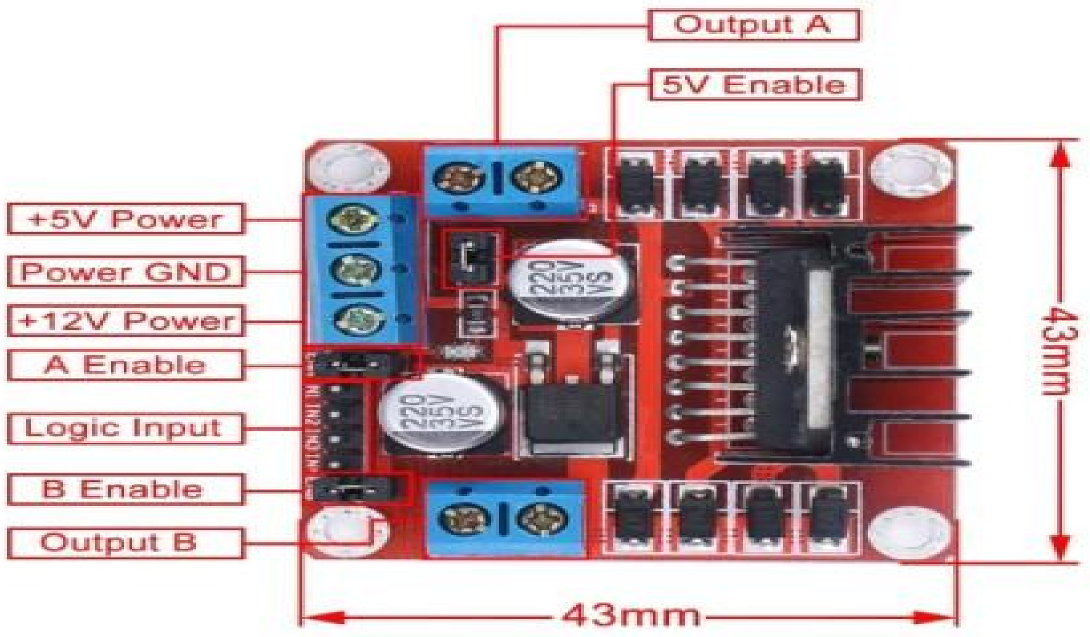




❖ **PIN CONFIGURATION:**

IN1 & IN2	Motor A input pins. Used to control the spinning direction of Motor A
IN3 & IN4	Motor B input pins. Used to control the spinning direction of Motor B
ENA	Enables PWM signal for Motor A
ENB	Enables PWM signal for Motor B
OUT1 & OUT2	Output pins of Motor A

OUT3 & OUT4	Output pins of Motor B
12V	12V input from DC power Source
5V	Supplies power for the switching logic circuitry inside L298N IC
GND	Ground pin



- ❖ This dual bidirectional motor driver, is based on the very popular L298 Dual H-Bridge Motor Driver Integrated Circuit. The circuit will allow you to easily and independently control two motors of up to 2A each in both directions.It is ideal for robotic applications and well suited for connection to a microcontroller requiring just a couple of control lines per motor. It can also be interfaced with simple manual switches, TTL logic gates, relays, etc. This board equipped with power LED indicators, on-board +5V regulator and protection diodes
- ❖ It's very sensitive component .
- ❖ Kindly check it before using in the robot .

3.3 Install the IR Sensors

Setting Up a 5×TCRT5000 Sensor

This guide provides a step-by-step walkthrough for mounting and wiring five TCRT5000 infrared line sensors on an Arduino Uno-based robot. It covers physical placement, wiring, alignment, calibration, testing, and troubleshooting to ensure reliable line detection (non-PID control). Use short paragraphs and bullet lists for clarity, and follow the expert tips to avoid common pitfalls.

Sensor Array Placement

- **Orientation**: Mount all five sensors in a straight line perpendicular to the robot's forward direction, near the front edge of the chassis. Center the middle sensor under the robot's centerline.
- **Spacing**: Space sensors evenly (about 1–2 cm apart). This covers the typical line width (e.g. a 2–3 cm black tape on white floor). Too-close spacing may overlap detection zones; too far apart may skip narrow curves. Adjust based on your track's width.
- **Height above floor**: Keep each sensor about **2–5 mm above** the ground (the shorter range of TCRT5000). Use a spacer or shims to set a uniform height so all sensors see the same floor. Too high – they won't detect the line; too low – they might scratch or trigger on dust.
- **Mounting tips**: Secure sensors firmly (screws, tape, or hot glue) to prevent movement or tilt. Each sensor's IR emitter and phototransistor should point straight down at the surface. Avoid tilting them inward/outward. Ensure no bright reflective surfaces (like shiny screws or chrome parts) are directly under the sensors.

Wiring Connections

- **Power rails (VCC/GND)**: Tie all sensor module VCC pins to the Arduino's **5V** pin (for example, using a common 5V rail on a breadboard). Connect all sensor ground (GND) pins to the Arduino's GND. This parallel wiring shares power among the 5 modules. Use thick, short wires (red for VCC, black for GND) and twist them if possible to reduce noise.
- **Output pins**: Connect each sensor's digital output pin to a separate Arduino digital input. For example, use D2, D3, D4, D5, D6 for sensors 1–5 respectively. Label each wire so you know which sensor is which in code. If the modules have on-board indicator LEDs, note their labeling (commonly "OUT"). Ensure no two sensors share the same Arduino pin.
- **Current-limiting**: Most TCRT5000 modules include a built-in resistor for the IR LED. If you are using bare sensors (without a board), place ~330Ω resistors in series with each IR LED to prevent over-current. If modules have trim-pots (sensitivity adjusters), no extra resistors are needed.
- **Common ground**: Ensure the Arduino ground and any motor driver or battery ground are connected. This common reference prevents erratic readings.
- **Example connection scheme**:
  - Sensor 1 VCC → Arduino 5V, GND → Arduino GND, OUT → D2
  - Sensor 2 VCC → 5V, GND → GND, OUT → D3

- ○ Sensor 3 VCC → 5V, GND → GND, OUT → D4
- ○ Sensor 4 VCC → 5V, GND → GND, OUT → D5
- ○ Sensor 5 VCC → 5V, GND → GND, OUT → D6

Sensor Alignment and Spacing

- **Height adjustment**: Use rulers or calibration blocks to set sensor height around 3–5 mm. Verify all sensors are level and the same height by placing a sheet of paper under them. They should barely hover above the paper.

- **Spacing accuracy**: For a standard line (~2 cm wide), 1–2 cm spacing is ideal. If your track is wider, increase spacing; for narrow lines, move them closer. Keep the arrangement symmetric so edge sensors are equidistant from the center.

- **Lateral alignment**: The center sensor should be exactly at the robot's midpoint. The other four should be offset left and right equally. This symmetry helps the robot detect deviations left or right accurately.

- **Avoid interference**: Make sure the IR LEDs and phototransistors from neighboring sensors do not overlap fields. If two sensors are too close, one's IR emitter might affect the other's phototransistor. If interference occurs, add small barriers or increase spacing slightly.

- **Mount rigidly**: Once aligned, lock sensors in place (use brackets or hot glue) so they can't wobble. Even a small tilt change can affect detection accuracy. Test by hand: gently rock the sensor mounting and ensure outputs stay stable.

Wiring and Reliability Best Practices

- **Use solid connections**: Solder wire leads to sensor pins or use well-seated header pins. Loose jumper wires often cause flakiness. If using a breadboard, ensure wires are pushed fully and use short lengths.

- **Cable management**: Bundle sensor wires together (zip ties or spiral wrap) and route them away from motors and battery cables. This reduces electromagnetic noise coupling. Keep power (5V/GND) wires separate from signal wires as much as possible.

- **Twist power pairs**: If running VCC and GND wires together, twist them to form a mini power bus; this helps reject interference.

- **Decoupling capacitors**: Place a 0.1 µF ceramic capacitor across VCC and GND near each sensor board or cluster to smooth out voltage spikes. This is especially useful if motors share the supply.
- **Ground plane**: Whenever possible, use a ground plane or wide copper traces (on a custom PCB) for the GND connection. This lowers resistance and noise on the reference ground.
- **Avoid reflections**: If your robot chassis is glossy or metallic, paint the sensor area matte black or attach a non-reflective plate under the sensors. This prevents false readings from chassis reflections.
- **Protection**: Do not short VCC to GND. Double-check connections before powering. Use a multimeter continuity check for wiring errors. If using LEDs, consider current-limiting resistors if not already on the board.

Troubleshooting Guide

1. **No readings or all zeros/ones**:
    - **Wiring mistakes**: Verify each sensor's output is wired to the correct Arduino pin. Check that VCC is 5V, not 3.3V, unless using a 3.3V board (TCRT5000 works at 5V). Ensure sensor GND is actually ground.
    - **Broken sensor**: Test each sensor individually by hooking it up and observing its LED/state over black/white. Replace any dead IR LEDs or phototransistors.
    - **Code issues**: Confirm your sketch's pin assignments match your wiring. Use `pinMode(pin, INPUT)` (no pull-up needed, sensors drive the pin high/low).
      Serial-print individual pin values to confirm.
2. **Erratic flickering**:
    - **Height issues**: If sensors are too high, small variations cause rapid toggling.
      Lower them. If too low, dust or surface bumps can trigger noise; raise slightly.
    - **Ambient light**: Strong sunlight or LEDs (e.g. on the floor) can flood the phototransistors. Test the robot in different lighting or cover the sensors (opaque shroud) to see if readings stabilize. Adjust trim-pots to require a stronger reflection threshold.
    - **Electrical noise**: Motors can induce spikes. Add decoupling caps (0.1 µF) on VCC/GND near sensors. Twist signal wires with GND, or add small pull-down resistors (10–100 kΩ) from each output to ground if needed (to avoid floating states when sensor is inactive).

3. **False positives (detecting line when none)**:
    - **Calibration**: If a sensor lights up on plain floor, tighten its sensitivity. The black surface of your track may reflect some IR (especially if it's not very dark), so adjust each sensor until it barely registers white as "line".
    - **Sensor damage**: A cracked lens or dirt can cause stray reflections. Clean each sensor lens. Check for solder blobs or scratches.
    - **Glowing eyes**: Sometimes, one sensor's IR emitter can reflect into an adjacent sensor. If this happens, cover sides between sensors with opaque tape to isolate them.
4. **One sensor dead or stuck**:
    - **Pin test**: Swap its output wire with a neighboring sensor's output in code (e.g. read pin 2 on 3 and vice versa). If the problem moves, it's wiring or code. If it stays with the sensor, the module is bad.
    - **LED check**: Does its indicator LED ever light? If not, the IR LED or photodiode might be open/shorted. Replace or repair the module.
5. **Motor interference**:
    - If sensor readouts change when motors run, the 5V line might be sagging. Add a separate 5V regulator for electronics or ensure the motor driver's power supply shares ground but has isolated supply. Add decoupling on the motor supply.

Interpreting Sensor Data in Code

- **Signal logic**: Typically, digitalRead() returns **LOW (0)** when the sensor detects a dark line (no IR reflected) and **HIGH (1)** on a light/white background (IR reflected).
  (Some modules can be inverted; always test with a black/white sample.)
- **Binary pattern**: Treat the five sensor readings as a bit pattern [S1 S2 S3 S4 S5]. For example, 00100 (only center sensor LOW) means the line is directly under the center – go straight. A pattern 10000 (leftmost LOW) means the line is far to the left – turn sharply left. Conversely, 00001 (rightmost LOW) means turn right. Patterns like 01100
  (left-middle and center LOW) mean the line is slightly left – veer left gently. Design your code to respond to each combination.
- **Decision logic**: Write if/else or switch statements based on the sensor array:

  - **Center-focused**: If the center sensor (S3) reads low, drive forward.
  - **Left side**: If one of the left sensors (S1 or S2) reads low (and not the center), steer left. The further left the LOW bit, the sharper the turn.
  - **Right side**: If one of the right sensors (S5 or S4) reads low (and not the center), steer right.
  - **Multiple detections**: If multiple adjacent sensors are LOW (e.g. 01110), it could be a wide line or intersection. You can treat this as on line (straight) or implement special logic (slow down, look for line fork).
  - **No detection (11111)**: The robot lost the line. A simple strategy is to keep the last turn direction (continue veering) or stop and scan (rotate) until a sensor finds the line again.
- **Mapping to movement**: Use the sensor pattern to set motor speeds. For example, if turning left, slow or stop the left motor and run the right motor, and vice versa. Gradual errors (center + one side sensor) should result in gentle corrections; extreme errors (only an edge sensor) require sharper turns. Ensure your code continuously reads the sensors and updates motor commands rapidly (e.g. in each loop).

Power and Safety Considerations

- **Current draw**: Each TCRT5000 module's IR LED draws about 20 mA. Five sensors draw ~100 mA total. The Arduino Uno's 5V regulator (or USB 5V) can easily supply this extra load along with its own microcontroller needs. However, if you add more devices (motors, servos) on the 5V rail, ensure you do not exceed the regulator's capacity (~500 mA from USB power).
- **Voltage**: Power sensors with the Arduino's 5V pin or a stable 5V supply. Do **not** exceed 5V on VCC or connect sensors directly to high voltages. The Arduino will output TTL 5V signals, so all sensor outputs are safe to read directly on digital pins.
- **Grounding**: Tie all grounds (Arduino, sensors, motor driver, battery pack) together. A floating ground will cause random readings.
- **Decoupling**: As mentioned, use capacitors on the 5V line to prevent noise from motors or switching regulators upsetting the sensors. A common practice is a 10–100 µF electrolytic capacitor near the 5V source, plus 0.1 µF ceramics near each sensor module.
- **Overheating**: If you place resistors on the IR LEDs manually, pick values (220–330Ω) so the LEDs don't overheat. 5V with 330Ω yields ~10 mA which is safe and still effective.

- **Enclosure safety**: If your robot has a metal chassis, ensure no short circuits when mounting the sensors. Use insulating spacers or tape where needed. Double-check polarity of wires: swapping VCC/GND will burn out the sensor LED.
-

Through my observations, I conclude the following:

$$Robot\ Stability \propto \frac{1}{Distance\ between\ sensors}$$

$$Rigidity\ in\ movement \propto \frac{1}{Distance\ between\ sensors}$$

3.4 Power Distribution

Component	Voltage Needed	Notes
Arduino Uno	5V or 7-12V via Vin	Avoid giving direct 7.4V to 5V pin!
L298N Motor Driver	6V – 12V	Can take full 7.4V directly
TCRT5000 Sensors	5V	Must use **regulated** 5V only

Connections Overview:

1. Power Input to L298N

- **Connect battery's +ve (7.4V)** → L298N's **+12V terminal**
- **Battery -ve (GND)** → L298N's **GND terminal**

2. L298N to Motors

- Connect **Motor A and Motor B outputs** to your two DC motors (polarity can be adjusted later).

3. L298N to Arduino (5V Power)

- **L298N has a 5V regulator** (only works if input > 6V).
- **Connect L298N's 5V OUT → Arduino’s Vin or 5V pin**
  If you’re using Arduino’s **5V pin**, do not plug USB power at the same time to avoid backpowering conflict.

4. GND Sharing

- Make sure **all GNDs are connected together**:
  - Battery GND
  - L298N GND
  - Arduino GND
  - IR Sensors GND

5. Power to IR Sensors

- Connect **regulated 5V from Arduino** to all 5x TCRT5000 VCC pins
- IR sensors draw very little current (~10-20mA each) – safe to use Arduino’s onboard 5V.

Issue	Possible Cause	Fix
Arduino resets randomly	Voltage dip on motor start	Add capacitor (470µF+) across Vin & GND
IR sensors flicker or behave weirdly	Ground not shared properly	Ensure **common GND**
Motors weak or not running	Undervoltage or low battery charge	Check battery with multimeter
Robot behaves weird with USB + Battery both plugged	Voltage conflict	NEVER power via USB and external 5V at same time

Use an **on/off toggle switch** between battery +ve and L298N Vin.

Use a **fuse or polyfuse (1A-2A)** in case of accidental short circuits.

Mount battery low and centered for stability.

Use thick wires (18–22 AWG) for motor power, thin wires (24–26 AWG) for sensors.

---

4. Programming

4.1 Arduino Code

```
//int S_A = 10;  //speed motor a

int M_A1 = 8; //motor a = + int

M_A2 = 9; //motor a = - int

M_B1 = 11; //motor b = - int

M_B2 = 10; //motor b = +

//int S_B = 11;  //speed motor b


int R_S = 2; //sensor  R

int S_S = 3; //sensor S    int

L_S = 4; //sensor L /*int RR_S

= 3 ; //sensor RR int LL_S = 7 ;

//sensor LL*/


void setup()

{

pinMode(M_B1, OUTPUT); pinMode(M_B2,

OUTPUT); pinMode(M_A1, OUTPUT);
```

```
pinMode(M_A2, OUTPUT); //pinMode(S_B,

OUTPUT);

//pinMode(S_A, OUTPUT);


pinMode(L_S, INPUT); pinMode(S_S,

INPUT); pinMode(R_S, INPUT);


//analogWrite(S_A, 150);

//analogWrite(S_B, 150);

//delay(200);

Serial.begin(9600);

}

void loop()

{

if ((digitalRead(L_S) == 1)&&(digitalRead(S_S) == 0)&&(digitalRead(R_S) == 1)){forward();} else if

((digitalRead(L_S) == 1)&&(digitalRead(S_S) == 1)&&(digitalRead(R_S) == 1)){Stop();}

else if ((digitalRead(L_S) == 1)&&(digitalRead(S_S) == 1)&&(digitalRead(R_S) ==
0)){turnLeft();}

else if ((digitalRead(L_S) == 1)&&(digitalRead(S_S) ==0)&&(digitalRead(R_S) == 0))
{turnLeft();}

else if ((digitalRead(L_S) == 0)&&(digitalRead(S_S) ==1)&&(digitalRead(R_S) == 0)) {turnLeft();}


else if ((digitalRead(L_S) == 0)&&(digitalRead(S_S) == 1)&&(digitalRead(R_S) ==
1)){turnRight();}

else if ((digitalRead(L_S) == 0)&&(digitalRead(S_S) == 0)&&(digitalRead(R_S) ==
1)){turnRight();}
```

```
else if ((digitalRead(L_S) == 0)&&(digitalRead(S_S) == 0)&&(digitalRead(R_S) == 0)){turnLeft();}


/*if ((digitalRead(LL_S) == 1)&& (digitalRead(L_S) == 1)&&(digitalRead(S_S) ==
0)&&(digitalRead(R_S) == 1)&&(digitalRead(RR_S) == 1)){forward();}


else if ((digitalRead(LL_S) == 1)&&(digitalRead(L_S) == 1)&&(digitalRead(S_S) ==
1)&&(digitalRead(R_S) == 0)&&(digitalRead(RR_S) == 1)){turnLeft();}

else if ((digitalRead(LL_S) == 1)&&(digitalRead(L_S) == 1)&&(digitalRead(S_S) ==
1)&&(digitalRead(R_S) == 1)&&(digitalRead(RR_S) == 0)){turnLeft();}

else if ((digitalRead(LL_S) == 1)&&(digitalRead(L_S) == 1)&&(digitalRead(S_S) ==
1)&&(digitalRead(R_S) == 0)&&(digitalRead(RR_S) == 0)){turnLeft();}

else if ((digitalRead(LL_S) == 1)&&(digitalRead(L_S) == 1)&&(digitalRead(S_S)
==0)&&(digitalRead(R_S) == 0)&&(digitalRead(RR_S) == 1)) {turnLeft();}


else if ((digitalRead(LL_S) == 1)&&(digitalRead(L_S) == 1)&&(digitalRead(S_S)
==0)&&(digitalRead(R_S) == 0)&&(digitalRead(RR_S) == 1)) {turnLeft();}

else if ((digitalRead(LL_S) == 0)&&(digitalRead(L_S) == 0)&&(digitalRead(S_S) ==
0)&&(digitalRead(R_S) == 1)&&(digitalRead(RR_S) == 1)){turnRT();}

else if ((digitalRead(LL_S) == 0)&&(digitalRead(L_S) == 1)&&(digitalRead(S_S) ==
1)&&(digitalRead(R_S) == 1)&&(digitalRead(RR_S) == 1)){turnRT();}

else if ((digitalRead(LL_S) == 0)&&(digitalRead(L_S) == 0)&&(digitalRead(S_S) ==
1)&&(digitalRead(R_S) == 1)&&(digitalRead(RR_S) == 1)){turnRT();}

else if ((digitalRead(LL_S) == 0)&&(digitalRead(L_S) == 0)&&(digitalRead(S_S) ==
0)&&(digitalRead(R_S) == 0)&&(digitalRead(RR_S) == 1)){turnRight();}

else if ((digitalRead(LL_S) == 1)&&(digitalRead(L_S) == 0)&&(digitalRead(S_S) ==
0)&&(digitalRead(R_S) == 0)&&(digitalRead(RR_S) == 0)){turnLeft();}




else if ((digitalRead(LL_S) == 1)&&(digitalRead(L_S) == 0)&&(digitalRead(S_S) ==
0)&&(digitalRead(R_S) == 0)&&(digitalRead(RR_S) == 1)){turnLeft();}
```

```
else if ((digitalRead(LL_S) == 1)&&(digitalRead(L_S) == 0)&&(digitalRead(S_S) ==
1)&&(digitalRead(R_S) == 1)&&(digitalRead(RR_S) == 1)){turnRight();}

else if ((digitalRead(LL_S) == 0)&&(digitalRead(L_S) == 0)&&(digitalRead(S_S) ==
1)&&(digitalRead(R_S) == 1)&&(digitalRead(RR_S) == 1)){turnRight();}

else if ((digitalRead(LL_S) == 1)&&(digitalRead(L_S) == 0)&&(digitalRead(S_S) ==
0)&&(digitalRead(R_S) == 1)&&(digitalRead(RR_S) == 1)){turnRight();}




else if ((digitalRead(LL_S) == 1)&&(digitalRead(L_S) == 1)&&(digitalRead(S_S) ==
0)&&(digitalRead(R_S) == 0)&&(digitalRead(RR_S) == 0)){turnLT();}

else if ((digitalRead(LL_S) == 0)&&(digitalRead(L_S) == 0)&&(digitalRead(S_S) ==
0)&&(digitalRead(R_S) == 1)&&(digitalRead(RR_S) == 1)){turnRight();}

else if ((digitalRead(LL_S) == 0)&&(digitalRead(L_S) == 0)&&(digitalRead(S_S) ==
0)&&(digitalRead(R_S) == 0)&&(digitalRead(RR_S) == 0)){turnLeft();}

else if ((digitalRead(LL_S) == 1)&&(digitalRead(L_S) == 1)&&(digitalRead(S_S) ==
1)&&(digitalRead(R_S) == 1)&&(digitalRead(RR_S) == 1)){turnLeft();}

*/



else if ((digitalRead(L_S) == 0)&&(digitalRead(S_S) == 0)&&(digitalRead(R_S) ==
0)){turnLeft();}

else if ((digitalRead(L_S) == 0)&&(digitalRead(S_S) ==1)&&(digitalRead(R_S) == 0)) {turnLeft();}



}



void forward(){ analogWrite(M_A1,

0); analogWrite(M_A2, 255);

analogWrite(M_B1, 255);

analogWrite(M_B2, 0);
```

```
}



void turnRight(){ analogWrite(M_A1,
0); analogWrite(M_A2, 0);
analogWrite(M_B1, 255);
analogWrite(M_B2, 0);
}
void turnRT(){
analogWrite(M_A1, 0); analogWrite(M_A2, 0);
analogWrite(M_B1, 255); analogWrite(M_B2,
0);
}
void turnLT(){ analogWrite(M_A1,
0); analogWrite(M_A2, 255);
analogWrite(M_B1, 0);
analogWrite(M_B2, 0);
}

void turnLeft(){ analogWrite(M_A1,
0); analogWrite(M_A2, 255
);
analogWrite(M_B1, 0); analogWrite(M_B2, 0);
}
```

void Stop(){ digitalWrite(M_A1,

LOW); digitalWrite(M_A2, LOW);

digitalWrite(M_B1, LOW);

digitalWrite(M_B2, LOW);

delay(500); turnLeft;

}

4.2 Logic Explanation

What This Robot Does

Imagine you have a toy car with eyes (sensors) that can see a black line on the floor. The car wants to follow this line like it's driving on a road.

The Robot's "Eyes" (Sensors)

Your robot has 3 special eyes:

- **Left eye** (L_S): Looks to the left
- **Middle eye** (S_S): Looks straight ahead
- **Right eye** (R_S): Looks to the right

These eyes can tell if they see the black line (1) or just the white floor (0).

The Robot's "Legs" (Motors)

The robot has two motors - think of them like legs:

● **Motor A**: Controls the left wheel ●
Motor B: Controls the right wheel

How It Thinks (The Logic)

The robot looks with all 3 eyes and decides what to do:
When it sees the line perfectly:

- Left eye: sees white floor (1)
- Middle eye: sees black line (0)
- Right eye: sees white floor (1)
- **Decision**: "Great! Go straight forward!"

When the line goes to the right:

- The robot sees more black on the left side
- **Decision**: "I need to turn right to get back on the line!"

When the line goes to the left:

- The robot sees more black on the right side
- **Decision**: "I need to turn left to get back on the line!"

When it can't find the line at all:

- All eyes see white or it's confused
- **Decision**: "I'll turn left and look for the line!"

When all eyes see the black line:

- This might mean "STOP!" or the end of the path
- **Decision**: "Stop for a moment, then turn left to keep exploring!"

The Actions

- **Forward**: Both wheels spin forward - the robot goes straight
- **Turn Left**: Only the right wheel spins - robot turns left
- **Turn Right**: Only the left wheel spins - robot turns right
- **Stop**: Both wheels stop completely

It's like the robot is constantly asking itself: "Where's my line? Oh there it is! Let me steer back to it!" over and over again, many times per second!

---

5. Calibration

Now need to calibrate using potetentiometer on IR sensors. For black line the light of IR sensor must be OFF while for white line it should be ON

TCRT5000 IR Sensor Potentiometer Calibration Guide

Understanding the Potentiometer Function

The blue/white potentiometer on TCRT5000 sensors controls the **threshold voltage** for the digital output (D0 pin):

- **Clockwise rotation** = Higher threshold (less sensitive)
- **Counterclockwise rotation** = Lower threshold (more sensitive)
- The onboard LED indicates when the threshold is crossed

Physical Calibration Process

Step 1: Identify Components

- **Potentiometer**: Small blue or white screw-adjustable component
- **LED Indicator**: Usually red, shows digital output state
- **D0 Pin**: Digital output that changes between HIGH/LOW **Step 2: Manual**

Calibration Method

Equipment Needed:

- Small screwdriver (Phillips or flathead depending on potentiometer type)
- Test surface with black line on white background
- Good lighting conditions

Calibration Procedure:

1. **Power On the Sensor**
   - Connect VCC to 5V, GND to ground
   - Observe the onboard LED status
2. **Start with Black Line**

- Place sensor directly over the **black line**
  ○ Turn potentiometer **clockwise** (increase threshold)
  ○ Keep turning until the LED turns **OFF**
  ○ This means the sensor now detects "black"

3. **Test on White Surface**
   - Move sensor to **white surface**
     ○ The LED should turn **ON** (detecting "white")
     ○ If it doesn't turn ON, turn potentiometer slightly **counterclockwise**
4. **Fine-Tune the Sensitivity**
   - Move sensor back and forth between black line and white surface ○ Adjust potentiometer until you get reliable switching:
     - **Black line** = LED OFF
     - **White surface** = LED ON **Step 3:**

Precision Calibration

For Optimal Performance:

1. **Find the Edge**
   - Position sensor exactly at the **edge** between black line and white surface
     ○ Slowly adjust potentiometer until LED just starts to flicker
     ○ This is your optimal sensitivity point
2. **Test Different Heights**
   - Lift sensor slightly (2-3mm) and test again
     ○ Adjust if sensitivity changes with height variations
     ○ Ensure consistent detection at your robot's operating height
3. **Environmental Testing**
   - Test under different lighting conditions
     ○ Fluorescent lights, sunlight, shadows can affect readings
     ○ Re-adjust if necessary for your competition environment

Step 4: Multi-Sensor Calibration

For Multiple Sensors (Left, Center, Right):

1. **Calibrate Each Individually**
   - Each sensor may have slightly different characteristics

    - Use the same black line and white surface for consistency
2. **Synchronization Check**
    - Place all sensors on white surface - all LEDs should be ON
    - Place all sensors on black line - all LEDs should be OFF
    - If not synchronized, adjust individual potentiometers
3. **Line Edge Testing**
    - Position robot so center sensor is on black line
    - Left and right sensors should be on white surface
    - Verify: Center LED OFF, Side LEDs ON

Common Calibration Issues and Solutions

Problem	Cause	Solution
LED always ON	Threshold too low	Turn potentiometer clockwise
LED always OFF	Threshold too high	Turn potentiometer counterclockwise
Inconsistent switching	Ambient light interference	Recalibrate in actual operating conditions
Different sensor responses	Manufacturing variations	Calibrate each sensor individually
Height sensitivity	Fixed threshold doesn't adapt	Find compromise height and stick to it

Pro Tips for Perfect Calibration

1. **Consistent Surface**: Always use the same test surface material that matches your competition track
2. **Stable Mounting**: Ensure sensors don't move during calibration
3. **Room Lighting**: Calibrate under similar lighting to your operating environment
4. **Multiple Tests**: Move the robot along the line to test consistency
5. **Mark Settings**: Once calibrated, mark potentiometer positions with a pen for future reference

Final Verification

- **Straight Line**: Robot should follow straight black lines smoothly
- **Curves**: Test on curved sections of track
- **Intersections**: Verify behavior at T-junctions or crossings

- **Ambient Changes**: Test under different lighting conditions