Perfboard Layout and Setup Guide for PID Line Follower Robot

Author: Manus AI

Date: January 2025

Version: 1.0

Table of Contents

Perfboard Layout Overview

The perfboard layout for your PID line follower robot has been designed to accommodate all essential components while maintaining clean wiring and easy access for debugging. The rectangular perfboard should be approximately 10cm x 8cm to provide adequate space for all components.

Component List:

- Arduino Nano microcontroller
- L298N motor driver module
- RLS-08 sensor array
- Step-down voltage regulator module (Buck converter)
- Two N20 geared motors
- Calibration push button
- Status LED indicator

- Various connecting wires and headers

The layout prioritizes signal integrity, power distribution efficiency, and mechanical stability while keeping the overall footprint compact for optimal robot performance.

Component Placement

Arduino Nano Positioning

The Arduino Nano is positioned centrally on the perfboard to minimize wire lengths to all peripheral components. This central placement provides easy access to both digital and analog pins while maintaining balanced weight distribution.

L298N Motor Driver Placement

The L298N motor driver is positioned to the right of the Arduino Nano, allowing direct connection to the motor control pins while keeping the high-current motor connections away from sensitive analog circuits.

RLS-08 Sensor Array Connection

The RLS-08 sensor array is connected via an 8-wire ribbon cable or individual wires to the Arduino's analog pins A0 through A7. The sensor array itself will be mounted at the front of the robot chassis, below the perfboard.

Step-Down Module Positioning

The step-down voltage regulator is positioned to handle the main power input and provide regulated 5V power to the Arduino and sensor array. This module should be placed where it can efficiently distribute power to all components.

Additional Components

The calibration button and status LED are positioned for easy access during operation and debugging.

Wiring Connections

Accurate wiring is critical for the proper functioning of your line follower robot. Below is a detailed breakdown of the connections between each component.

Arduino Nano to L298N Motor Driver

These connections control the direction and speed of the N20 motors. The L298N module requires digital signals for direction control and PWM signals for speed control.

Arduino Nano Pin	L298N Pin	Function	Notes
Digital Pin 9	ENA	Left Motor Enable	Connects to PWM pin for speed control of Left Motor (Motor A)
Digital Pin 8	IN1	Left Motor Direction	Controls direction of Left Motor (Motor A)
Digital Pin 7	IN2	Left Motor Direction	Controls direction of Left Motor (Motor A)
Digital Pin 10	ENB	Right Motor Enable	Connects to PWM pin for speed control of Right Motor (Motor B)
Digital Pin 11	IN3	Right Motor Direction	Controls direction of Right Motor (Motor B)
Digital Pin 12	IN4	Right Motor Direction	Controls direction of Right Motor (Motor B)
GND	GND	Common Ground	Connects Arduino ground to L298N ground

Arduino Nano to RLS-08 Sensor Array

The RLS-08 sensor array provides analog readings from its 8 infrared sensors. These are connected to the analog input pins of the Arduino Nano.

Arduino Nano Pin	RLS-08 Pin	Function	Notes
Analog Pin A0	SENSOR_0	Leftmost Sensor	Reads analog value from sensor 0
Analog Pin A1	SENSOR_1	Sensor 1	Reads analog value from sensor 1
Analog Pin A2	SENSOR_2	Sensor 2	Reads analog value from sensor 2

Analog Pin A3	SENSOR_3	Sensor 3	Reads analog value from sensor 3
Analog Pin A4	SENSOR_4	Sensor 4	Reads analog value from sensor 4
Analog Pin A5	SENSOR_5	Sensor 5	Reads analog value from sensor 5
Analog Pin A6	SENSOR_6	Sensor 6	Reads analog value from sensor 6
Analog Pin A7	SENSOR_7	Rightmost Sensor	Reads analog value from sensor 7
5V	VCC	Power Supply	Provides 5V power to the sensor array
GND	GND	Common Ground	Connects Arduino ground to RLS-08 ground

Arduino Nano to Calibration Button and LED

These are simple digital connections for user interaction and status indication.

Arduino Nano Pin	Component	Function	Notes
Digital Pin 2	Button	Calibration Trigger	Connect one side of the button to D2, other to GND
Digital Pin 13	LED	Status Indicator	Connect LED anode to D13, cathode to GND via a current-limiting resistor (e.g., 220 Ohm)

L298N Motor Driver to N20 Motors

Connect the N20 motors to the output terminals of the L298N. The polarity of the motor connection will determine its physical forward direction.

L298N Pin	N20 Motor	Function	Notes
OUT1	Left Motor	Motor A Connection	Connect one terminal of the Left N20 Motor

OUT2	Left Motor	Motor A Connection	Connect the other terminal of the Left N20 Motor
OUT3	Right Motor	Motor B Connection	Connect one terminal of the Right N20 Motor
OUT4	Right Motor	Motor B Connection	Connect the other terminal of the Right N20 Motor

Power Distribution

Proper power management is crucial for stable robot operation, especially with motors that draw significant current. A step-down voltage regulator (buck converter) is used to provide a stable 5V supply for the logic components.

Main Power Input

Your main power source (e.g., a 7.4V LiPo battery or 6xAA battery pack) will connect to the L298N motor driver and the step-down module.

- **L298N VCC (Motor Power):** Connect the positive terminal of your main battery (e.g., 7.4V) to the VCC (or +12V) terminal of the L298N. This supplies power directly to the motors.
- **L298N GND:** Connect the negative terminal of your main battery to the GND terminal of the L298N.

Step-Down Voltage Regulator

This module converts the higher battery voltage to a stable 5V for the Arduino Nano and RLS-08 sensor array.

- **Input to Step-Down Module:** Connect the positive and negative terminals of your main battery to the input (IN+ and IN-) terminals of the step-down module.
- **Output from Step-Down Module:** The output (OUT+ and OUT-) of the step-down module should be set to 5V. This 5V output will power the Arduino Nano and the RLS-08 sensor array.

Powering Arduino Nano

- Connect the 5V output from the step-down module to the `5V` pin of the Arduino Nano. Alternatively, you can connect it to the `VIN` pin if your step-down module outputs

slightly above 5V (e.g., 7V-12V), but connecting to the 5V pin directly is more efficient if the voltage is precisely 5V.

- Connect the GND output from the step-down module to a `GND` pin on the Arduino Nano.

Powering RLS-08 Sensor Array

- Connect the 5V output from the step-down module to the `VCC` pin of the RLS-08 sensor array.
- Connect the GND output from the step-down module to the `GND` pin of the RLS-08 sensor array.

Common Ground

Ensure all GND pins from the Arduino Nano, L298N, RLS-08, and the step-down module are connected together to form a common ground reference for the entire circuit. This is crucial for stable operation and accurate sensor readings.

Assembly Instructions

1. **Plan Component Placement:** Before soldering, lay out all components on your perfboard according to the diagram and the placement guidelines above. Use a marker to mark the holes where component pins will go.
2. **Solder Components:** Begin by soldering the smaller components (resistors, LEDs) and then move to larger components (Arduino Nano, L298N, step-down module). Ensure all solder joints are clean and secure.
3. **Wire Connections:** Use solid core wires for short, direct connections and stranded wires for connections that might need flexibility (e.g., to motors if they are mounted off-board). Follow the wiring tables carefully. Keep wires as short as possible to reduce noise and improve signal integrity.
4. **Test Continuity:** After soldering a section, use a multimeter to check for continuity and ensure there are no short circuits between adjacent traces or pins.
5. **Mount Motors and Sensors:** Securely mount the N20 motors to your robot chassis. The RLS-08 sensor array should be mounted at the front of the robot, approximately 5-10mm above the track surface, ensuring all 8 sensors can detect the line.

Sensor Calibration Guide

Accurate sensor calibration is paramount for the line follower robot to reliably detect the line and calculate the error. This process determines the minimum (white surface) and

maximum (black line) analog readings for each sensor, allowing the robot to adapt to different track colors and lighting conditions.

Prerequisites:

- Your Arduino Nano is powered and connected to your computer via USB.
- The `line_follower_code.ino` sketch is uploaded to your Arduino Nano.
- The Arduino IDE Serial Monitor is open and set to `9600 baud`.
- Your robot is placed on a section of the line follower track that includes both the black line and the white background.

Calibration Steps:

1. **Initiate Calibration:**
   - Upon startup, the robot will typically enter a calibration mode, indicated by the status LED (connected to Digital Pin 13) blinking rapidly. This signifies that the robot is waiting for you to begin the calibration process.
   - Alternatively, you can manually trigger calibration at any time by typing `calibrate` into the Serial Monitor input field and pressing Enter.
2. **Perform the Sweep:**
   - Once calibration is initiated (LED blinking), press and hold the dedicated calibration button (connected to Arduino Digital Pin 2).
   - While holding the button, slowly move your robot back and forth across the black line and white background. Ensure that *all 8 sensors* pass over both the black line and the white surface multiple times during this process. This allows each sensor to record its minimum (when over white) and maximum (when over black) values.
   - Continue this sweeping motion for approximately 10-15 seconds. The longer and more thoroughly you sweep, the more accurate your calibration will be.
3. **Complete Calibration:**
   - Release the calibration button.
   - The status LED should stop blinking and either turn off or remain solid, indicating that calibration is complete.
   - Check the Arduino Serial Monitor. You should see a message confirming "Calibration complete!" followed by the recorded minimum and maximum values for each of your 8 sensors. These values are stored in the Arduino's EEPROM, so they persist even after power cycling.

Important Considerations:

- **Lighting Conditions:** Perform calibration in the same lighting conditions where the robot will operate. Significant changes in ambient light can affect sensor readings.
- **Track Material:** If you change tracks or track materials, it is highly recommended to re-calibrate your sensors.
- **Sensor Height:** Ensure your sensor array is mounted at a consistent height (typically 5-10mm) above the track surface for optimal performance.
- **Verification:** After calibration, you can use the `debug` command in the Serial Monitor to observe the `calibratedSensorValues` in real-time. When a sensor is over white, its calibrated value should be close to 0. When it's over black, it should be close to 1000 (or your defined `MAX_SENSOR_VALUE`).

PID Tuning Guide

PID (Proportional-Integral-Derivative) control is the heart of a high-performance line follower. Proper tuning of the Kp, Ki, and Kd gains is crucial for achieving speed, accuracy, and stability. This guide outlines a systematic approach to tuning your robot.

Understanding PID Terms:

- **Proportional (Kp):** Responds to the current error. A larger Kp means a stronger correction for a given error. Too high Kp leads to oscillation.
- **Integral (Ki):** Addresses accumulated error over time. It helps eliminate steady-state errors (robot consistently drifting to one side on a straight line). Too high Ki can lead to integral windup and oscillation.
- **Derivative (Kd):** Responds to the rate of change of the error. It predicts future error and dampens oscillations, providing a smoother response. Too high Kd can make the robot jittery or overreact to noise.

Prerequisites:

- Your Arduino Nano is powered and connected to your computer via USB.
- The `line_follower_code.ino` sketch is uploaded to your Arduino Nano.
- The Arduino IDE Serial Monitor is open and set to `9600 baud`.
- Sensors are calibrated (refer to the Sensor Calibration Guide above).
- Your robot is placed on a line follower track.

Systematic Tuning Steps (Ziegler-Nichols Method Adaptation):

Step 1: Tune Kp (Proportional Gain) First

1. Set `Ki` and `Kd` to `0` (using `ki0.0` and `kd0.0` commands in Serial Monitor).
2. Start with a small `Kp` value (e.g., `kp0.5`).
3. Place the robot on the line and observe its behavior. It will likely move slowly and drift off the line.
4. Gradually increase `Kp` (e.g., `kp1.0`, `kp1.5`, `kp2.0`, etc.). After each increase, test the robot.
5. Continue increasing `Kp` until the robot starts to oscillate around the line. It should follow the line, but with noticeable wobbling. This is your starting point for oscillation.
6. Once you find the `Kp` value that causes consistent oscillation, reduce it slightly (e.g., by 10-20%) to get a stable but slightly wobbly response. This is your initial `Kp`.

Step 2: Tune Kd (Derivative Gain) Second

1. Keep the `Kp` value from Step 1.
2. Start with a small `Kd` value (e.g., `kd0.1`).
3. Gradually increase `Kd` (e.g., `kd0.5`, `kd1.0`, `kd1.5`, etc.). After each increase, test the robot.
4. `Kd` will help to dampen the oscillations caused by `Kp`. You should see the robot's wobbling decrease and its movement become smoother.
5. Be careful not to increase `Kd` too much, as it can make the robot overly stiff, jittery, or prone to overshooting on turns.
6. Find the `Kd` value that provides the smoothest, most stable line following without excessive jitter or slow response.

Step 3: Tune Ki (Integral Gain) Last

1. Keep the `Kp` and `Kd` values from the previous steps.
2. Start with a very small `Ki` value (e.g., `ki0.001`).
3. `Ki` is used to eliminate steady-state errors, which means if your robot consistently drifts slightly to one side on a long straight line, `Ki` will correct it.
4. Gradually increase `Ki` (e.g., `ki0.005`, `ki0.01`, `ki0.02`, etc.). Test after each increase.
5. If `Ki` is too high, it can cause slow, long-period oscillations or

or integral windup. Use `Ki` sparingly and only if necessary to correct persistent drift.

Step 4: Fine-Tuning and Optimization

1. Once you have initial `Kp`, `Ki`, and `Kd` values, test the robot on the entire track, including curves and intersections.

2. **Adjust `baseSpeed`:** Gradually increase the `baseSpeed` (e.g., `speed150`, `speed180`) and observe the robot's performance. As speed increases, you may need to slightly re-adjust `Kp` and `Kd`.
3. **Aggressive Turning:** For competitive performance, you might need to adjust the `turnSpeed` variable in the code (or add a serial command for it if not already present) to control how aggressively the robot turns on sharp corners.
4. **Iterate:** Tuning is an iterative process. Make small adjustments, test, and observe. It often takes many trials to find the optimal balance for your specific robot and track.

Common Tuning Issues and Solutions:

- **Robot Oscillates (Wobbles):**
  - `Kp` is too high. Reduce `Kp`.
  - `Kd` is too low. Increase `Kd`.
- **Robot is Sluggish/Slow to Respond:**
  - `Kp` is too low. Increase `Kp`.
- **Robot Consistently Drifts to One Side on Straight Lines:**
  - `Ki` is too low or zero. Increase `Ki` (use very small increments).
- **Robot Jitters/Overreacts to Small Changes:**
  - `Kd` is too high. Reduce `Kd`.
- **Robot Overshoots Turns:**
  - `Kp` might be too high, or `Kd` might be too low.
  - Consider adjusting `turnSpeed` if using aggressive turning strategies.

Using the Serial Monitor for Tuning:

As previously mentioned, you can use the following commands in the Arduino Serial Monitor to adjust parameters in real-time:

- `kpX.X` (e.g., `kp2.0`)
- `kiX.X` (e.g., `ki0.01`)
- `kdX.X` (e.g., `kd1.5`)
- `speedX` (e.g., `speed150`)
- `debug` (to view real-time sensor, error, and motor data)
- `calibrate` (to re-run sensor calibration)

Testing and Troubleshooting

Thorough testing is essential to ensure your line follower robot performs reliably. This section outlines key testing procedures and common troubleshooting tips.

Testing Procedures:

1. **Static Sensor Test:** Before placing the robot on the track, open the Serial Monitor and use the `debug` command. Manually move objects (e.g., your hand) over each sensor of the RLS-08 array. Observe the `calibratedSensorValues` to ensure each sensor responds correctly (values should change from near 0 to near 1000 when detecting black).
2. **Motor Direction Test:** Temporarily modify your code to run each motor independently at a low speed (e.g., `setLeftMotor(50);` and `setRightMotor(50);`). Verify that both motors spin in the desired forward direction. If not, reverse the motor connections to the L298N (swap OUT1/OUT2 or OUT3/OUT4).
3. **Basic Line Following Test:** Start with conservative PID values (e.g., `Kp=1.0`, `Ki=0.0`, `Kd=0.0`) and a low `baseSpeed` (e.g., `speed80`). Observe if the robot attempts to follow the line, even if it's wobbly or slow. This confirms basic functionality.
4. **Curve Test:** Test the robot on curves of varying radii. Observe if it can maintain the line without cutting corners or veering off. Adjust `Kp` and `Kd` as needed.
5. **Sharp Turn/Intersection Test:** For more advanced tracks, test on 90-degree turns or intersections. This is where aggressive turning strategies and well-tuned `Kd` become critical.
6. **Speed Test:** Gradually increase `baseSpeed` while monitoring stability. The goal is to find the highest speed at which the robot can reliably complete the track without losing the line.
7. **Line Loss Recovery Test:** If your track has intentional gaps or line breaks, test the robot's ability to re-acquire the line after losing it. The `line_follower_code.ino` includes basic line loss recovery logic.

Common Troubleshooting Tips:

- **No Power/Robot Dead:**
  - Check battery connections and charge level.
  - Verify step-down module output is 5V.
  - Ensure all GND connections are common.
  - Check L298N power connections (VCC, GND).
- **Motors Not Spinning:**

- Check motor connections to L298N OUT pins.
- Verify `ENA` and `ENB` pins are connected to Arduino PWM pins and receiving a PWM signal (not 0).
- Ensure `IN` pins (IN1-IN4) are correctly set HIGH/LOW for direction.
- L298N might be in a fault state (check its datasheet for error indicators).

- **Incorrect Motor Direction:**
  - Swap the motor wires connected to the L298N OUT pins for the affected motor.
- **Sensors Not Reading Correctly:**
  - Re-run sensor calibration. Ensure thorough sweeping over black and white.
  - Check wiring from RLS-08 to Arduino Analog pins.
  - Verify RLS-08 is receiving 5V power and has a common ground.
  - Clean sensor lenses if dirty.
- **Robot Doesn't Follow Line / Random Movement:**
  - Most likely a PID tuning issue. Revisit the PID Tuning Guide.
  - Ensure sensor calibration is accurate.
  - Check for loose wires or intermittent connections.
- **Arduino Not Responding to Serial Commands:**
  - Verify baud rate in Serial Monitor is `9600`.
  - Ensure correct COM port is selected.
  - Check USB cable connection.
  - Confirm `Serial.begin(9600);` is in `setup()` and `serialTuning()` is called in `loop()`.

Future Improvements and Modifications

This PID line follower robot provides a strong foundation for competitive performance. Here are some areas for future improvements and modifications to further enhance its capabilities:

1. **Advanced Sensor Fusion:** Integrate additional sensors like an IMU (Inertial Measurement Unit - accelerometer and gyroscope) to provide more robust position and orientation data, especially useful for high-speed turns and recovering from complex track features.
2. **Dynamic PID Tuning:** Implement algorithms that allow the PID gains to adapt dynamically based on the robot's speed, track curvature, or sensor readings. This can

provide more aggressive control on straightaways and smoother control on turns.

3. **Look-Ahead Control:** For very high-speed applications, consider adding a

a camera or additional IR sensors further ahead of the main array to anticipate upcoming turns and adjust the control strategy proactively.
4. **Wireless Communication:** Implement Bluetooth or Wi-Fi modules for wireless debugging, data logging, and real-time parameter adjustment without needing a USB connection.
5. **Motor Encoders:** Add encoders to the N20 motors to get precise feedback on wheel speed and distance traveled. This can be used for more accurate odometry, closed-loop speed control, and advanced turning maneuvers.
6. **User Interface:** Develop a simple LCD or OLED display to show real-time sensor readings, PID values, and robot status without relying solely on the Serial Monitor.
7. **Battery Management:** Implement a more sophisticated battery monitoring system to track battery voltage and provide warnings when power is low.
8. **Mechanical Design Optimization:** Further refine the robot chassis for optimal weight distribution, sensor placement, and motor mounting to maximize stability and speed.

By exploring these improvements, you can push the boundaries of your line follower robot and achieve even higher levels of performance in competitive environments.

References

[1] Arduino Nano Documentation: https://www.arduino.cc/en/Guide/ArduinoNano
[2] L298N Motor Driver Datasheet: https://www.sparkfun.com/datasheets/Robotics/L298_H_Bridge.pdf
[3] RLS-08 Sensor Array Information (example): https://robodo.in/product/rls-08-analog-digital-line-sensor-array/
[4] PID Control Theory: https://en.wikipedia.org/wiki/PID_controller
[5] N20 Geared Motor Specifications (example): https://www.pololu.com/product/3043