Line Follower Robot

Abstract

This project focuses on the design, analysis, and hardware implementation of a closed-loop control system for a differential-drive line follower robot. The primary objective was to bridge the gap between theoretical control systems and practical robotics by implementing a Proportional-Integral-Derivative (PID) controller. The system uses an array of infrared sensors to detect lateral deviation from a path and adjusts motor speeds to minimize tracking error.

The project methodology involved modeling the robot's dynamics using System Identification techniques in MATLAB, designing a discrete controller in the Z-domain, and deploying the solution on an Arduino UNO microcontroller. Real-time telemetry was established between the robot and a MATLAB Graphical User Interface (GUI) to visualize performance. The results demonstrate that the tuned PID controller successfully eliminates steady-state error and maintains stable path tracking, fulfilling the design requirements for autonomous navigation.

Introduction

Project Overview

Automated Guided Vehicles (AGVs) are a cornerstone of modern logistics and manufacturing. The line follower robot serves as a fundamental prototype for these systems, requiring precise steering control to follow a predefined path . This project integrates concepts from modern control theory, including system modeling, stability analysis, and digital implementation, to create a robust autonomous vehicle.

Control Objective

The control system's goal is to maintain the robot's center of mass directly over a black line on a white surface. This is achieved by minimizing the "tracking error," which is the difference between the desired position (center) and the measured position sensed by the hardware . The controller must ensure the robot reacts quickly to curves (fast rise time) without oscillating wildly (low overshoot) and follows straight lines perfectly (zero steady-state error).

System Description

Physical Architecture

The robot is built upon a four-wheel differential-drive chassis, where steering is accomplished by spinning the left and right wheels at different speeds. The core components include:

- **Microcontroller:** An Arduino UNO acts as the central processing unit, executing the control loop.
- **Sensors:** An infrared (IR) sensor array is mounted at the front of the chassis. These sensors distinguish between the reflective white surface and the non-reflective black line, providing digital inputs to the controller.
- **Actuators:** Four DC gear motors drive the wheels. These are powered and controlled via an L298N dual motor driver, which translates low-power logic signals from the Arduino into high-power current for the motors.
- **Power Supply:** Three 3.7V Li-ion batteries provide the necessary voltage for the motors and logic circuits.

Signal Flow

The system operates in a closed loop. The IR sensors read the current line position and send this data to the Arduino. The Arduino calculates the error and processes it through the PID algorithm. The resulting control signal determines the speed difference between the left and right motors, physically steering the robot back toward the line center.

Control System Methodology

System Identification (Modeling)

Instead of relying on theoretical mechanics alone, we derived the robot's transfer function experimentally using the MATLAB System Identification Toolbox. We applied step inputs to the motor speeds (commanding a sudden turn) and recorded the resulting lateral movement using the sensor array .

This input-output data allowed us to fit a second-order continuous transfer function that accurately represents how the robot responds to steering commands. This mathematical model served as the foundation for all subsequent control design.

Discretization

Since the Arduino is a digital computer, it cannot process continuous analog signals directly. We converted the continuous system model into a discrete digital model (Z-domain) using the Zero-Order Hold (ZOH) method . This step ensures that our simulation accurately reflects the discrete time steps (sampling time) of the physical microcontroller code.

PID Controller Design

Using the discrete model, we designed a PID controller using the MATLAB PID Tuner app. The controller consists of three terms:

1. **Proportional (P):** Reacts to the current error, providing immediate steering correction.
2. **Integral (I):** Sums up past errors to eliminate any long-term drift or steady-state error.
3. **Derivative (D):** Predicts future error based on the rate of change, adding damping to prevent oscillations.

The tuning process involved adjusting these three parameters to achieve a fast response time and minimal overshoot. The stability of the final design was verified by analyzing the poles of the closed-loop system, confirming they all lie within the stable region (inside the unit circle) .

Hardware Implementation

Sensor Logic and Error Calculation

The Arduino code reads the state of four IR sensors. To quantify the position, a "weighted error" method is used. Sensors further from the center are assigned higher weights , while sensors closer to the center are assigned lower weights . By summing these weighted values, the robot determines exactly how far left or right it has drifted from the line .

The Control Loop

The Arduino executes the control loop at a frequency of approximately 100 Hz. In each cycle, it performs the following steps:

1. **Read Sensors:** Acquire the current binary state of all IR sensors.
2. **Compute Error:** Calculate the weighted deviation from the setpoint.
3. **Calculate PID Output:** Compute the correction value by combining the proportional, integral, and derivative terms.
4. **Motor Mixing:** Add the correction value to the left motor speed and subtract it from the right motor speed (differential steering).
5. **Actuation:** Send the new speed commands to the L298N driver via Pulse Width Modulation (PWM).

Real-Time Telemetry & Safety

To validate performance, the robot transmits its current error and steering commands to a computer via USB serial communication. A custom MATLAB GUI plots this data in real-time, allowing for live monitoring. Additionally, safety logic was implemented: if the robot loses the line completely (all sensors read white), it enters a "recovery mode," spinning in the direction of the last known position until the line is re-acquired.

Results and Discussion

Transient Response Analysis

Simulation results in MATLAB indicated that the tuned PID controller achieves a settling time of less than one second. The addition of the Derivative term significantly reduced the oscillation compared to a simple Proportional controller. The Integral term proved essential for eliminating steady-state error, ensuring the robot does not drive parallel to the line but returns exactly to the center .

Stability Verification

The stability analysis performed in the Z-domain confirmed that the system is stable. The root locus plot showed that as the gain increases, the system poles move but remain stable up to a critical limit. Our chosen gains kept the system well within the stable margin, as verified by the `isstable()` command in MATLAB returning a true value .

Real-World Performance

During physical testing, the robot demonstrated smooth line following on both straight paths and curves. The "deadband" implemented in the code (ignoring very small errors) prevented motor jitter, while the differential steering allowed for sharp turns without losing traction. The visual feedback system (LEDs) accurately indicated when the system was stable versus when it was correcting a large error.

Conclusion

This project successfully demonstrated the end-to-end design of an autonomous robotic system. By starting with mathematical modeling in MATLAB and progressing to digital implementation on Arduino, we validated the practical application of modern control theory.

The final system met all design requirements: it tracks the line with high accuracy, remains stable under disturbance, and provides real-time data for analysis. The use of a discrete PID controller was proven to be highly effective for this application. Future improvements could include adaptive tuning, where the PID gains adjust automatically based on speed, or the implementation of Kalman filters to handle sensor noise more robustly.