Final Project Report: Autonomous Lane Keeping & Traffic Sign Recognition System

Course: Software Engineering for Autonomous Vehicles

Students: Masanbat Mulu & Yosef Chekol

Platform: TurtleBot3 (ROS Noetic & Gazebo)

Abstract

This project presents the design and implementation of an autonomous driving software stack for the TurtleBot3 platform. The system operates within the Gazebo physics simulation environment, relying exclusively on a monocular camera sensor to navigate a track. The core objectives include robust lane keeping under varying geometry and the detection of traffic regulations, specifically "Stop" signs. The solution integrates classical Computer Vision techniques for perception, a Proportional Controller for lateral stability, and a Finite State Machine (FSM) for high-level decision-making. The results demonstrate that low-cost monocular vision, when combined with efficient control algorithms, is sufficient for basic autonomous navigation tasks.

1. Introduction

1.1 Motivation

The automotive industry is undergoing a paradigm shift towards automation. While modern autonomous vehicles often rely on expensive sensor suites involving LiDAR, Radar, and stereo cameras, there is significant scientific interest in achieving navigation capabilities using low-cost sensors. Monocular vision systems (single camera) present a challenging yet cost-effective alternative, requiring sophisticated algorithms to extract depth and spatial information from 2D images.

1.2 Project Goals

The primary goal of this project was to develop a system capable of navigating a closed-loop track without human intervention. The specific operational requirements were:

- **Real-Time Perception:** Identifying drivable paths (lanes) and obstacles (signs) from a raw video feed.

- **Lateral Control:** Keeping the vehicle centered between yellow and white lane markings.
- **Longitudinal Control:** Managing velocity to ensure safety during turns and stops.
- **Traffic Rule Compliance:** Detecting a Stop sign, executing a complete halt for a safety duration (3 seconds), and resuming navigation.

2. Theoretical Background: Computer Vision

The perception module is the "eye" of the autonomous system. It acts as the bridge between the raw environment and the control logic.

2.1 Color Space Transformation (RGB vs. HSV)

A fundamental challenge in computer vision is lighting invariance. Standard digital cameras capture images in the **RGB (Red, Green, Blue)** color space. In RGB, chromatic information (color) is mixed with luminance (brightness). This means that a yellow lane line in a shadow has significantly different RGB values than the same line in sunlight, making thresholding difficult.

To solve this, the system converts frames to the **HSV (Hue, Saturation, Value)** color space.

- **Hue (H):** Represents the dominant wavelength (color perception).
- **Saturation (S):** Represents the purity of the color.
- **Value (V):** Represents the brightness.

By filtering primarily based on the 'Hue' channel, the system can robustly isolate yellow and white lane markings regardless of the lighting intensity in the simulation.

2.2 Region of Interest (ROI)

Processing the entire high-resolution image is computationally expensive and prone to noise. For example, walls or objects in the upper half of the frame might have colors similar to lane markings. To mitigate this, a **Region of Interest (ROI)** is defined. The algorithms crop the image to focus only on the bottom 45-55% of the frame. This mathematical operation reduces the search space matrix, effectively eliminating environmental distractions and improving the real-time performance of the node.

2.3 Image Moments and Centroid Calculation

Once the image is binarized (pixels converted to 0 for background and 1 for lane), the system needs to find the "center" of the lane. This is achieved using **Image Moments**.

The **Centroid (Center of Mass)**, denoted as (C_x, C_y), is derived from the zeroth and first moments. In this project, the calculated C_x of the white and yellow masks serves as the target heading point for the robot.

3. Theoretical Background: Control Systems

The control module serves as the "brain," converting perception data into motor commands.

3.1 Closed-Loop Feedback System

The navigation logic operates as a classic closed-loop feedback system.

- **Reference (Set Point):** The center of the image (screen width / 2), which represents the vehicle's forward axis.
- **Process Variable:** The calculated Lane Center (C_x) from the vision module.
- **Error (e):** The deviation between the Reference and the Process Variable.

Error = LaneCenter - ImageCenter

3.2 Proportional Controller (P-Controller)

To minimize the error (i.e., steer the robot back to the center), a P-Controller is implemented. The control law is:

$u(t) = -K_p * e(t)$

Where:

- $u(t)$ is the **Angular Velocity** (steering command).
- K_p is the **Proportional Gain**.
- $e(t)$ is the lateral error.

Logic:

- If $e(t) > 0$ (Lane is to the right), the robot turns right (negative angular z).
- If $e(t) < 0$ (Lane is to the left), the robot turns left.
- If $e(t) \approx 0$, the robot drives straight.

3.3 Stability and Tuning

A major challenge in P-controllers is oscillation. If K_p is too high, the system becomes under-damped, causing the robot to "zig-zag" or overshoot the lane center. If K_p is too low, the system is over-damped and fails to react to sharp turns in time. The gain was empirically tuned to $K_p \approx 0.0025\text{-}0.004$ to ensure smooth tracking.

4. System Architecture & Methodology

4.1 ROS Framework

The system is built upon the **Robot Operating System (ROS Noetic)**. It utilizes a distributed node architecture:

1. **Sensor Node:** The Gazebo simulation publishes RGB data to /camera/rgb/image_raw.
2. **Perception & Control Node:** A single integrated Python node (SmartDriver) subscribes to the image topic, processes the data using OpenCV, and publishes velocity commands.
3. **Actuator Node:** The TurtleBot3 differential drive controller listens to /cmd_vel and drives the wheels.

4.2 Finite State Machine (FSM)

To handle the complex logic of traffic signs, the software implements a **Finite State Machine (FSM)**. This allows the robot to switch behaviors dynamically.

The States:

1. **FOLLOW_LANE:** The default cruising state using the P-controller.
2. **STOP:** Triggered when the red pixel count exceeds a threshold. The robot sets linear velocity to 0 for a fixed duration ($t_{stop} = 3.0s$).
3. **OVERTAKE:** A specialized state designed to handle the post-stop scenario. Since the robot must pass the sign (which might still be in view), an "Offset" is applied to the target center. This causes the robot to steer slightly around the virtual obstacle before returning to the lane center.
4. **RETURN:** The logic that gradually decays the offset, smoothly merging the robot back into the optimal path.

5. Challenges and Solutions

5.1 The "Zig-Zag" Oscillation

Problem: During initial testing, the robot exhibited severe oscillation on straight paths. This is a common characteristic of Proportional controllers where the gain is too aggressive. **Solution:** The K_p gain was lowered, and a "smoothing factor" (α) was added to the error calculation: $Error_{new} = \alpha \cdot Error_{prev} + (1-\alpha) \cdot Error_{raw}$. This acted as a low-pass filter for the steering commands.

5.2 False Positives

Problem: The vision system occasionally identified walls or background objects as lane lines. **Solution:** The ROI was tightened to focus strictly on the immediate road surface. Additionally, a "minimum pixel count" threshold was introduced—if the detected lane blob is too small, it is considered noise and ignored.

5.3 Sharp Curve Handling

Problem: In tight corners, the lane lines would sometimes disappear from the camera view completely. **Solution:** A "Search Mode" was implemented. If zero lane pixels are detected, the robot rotates in place (high angular velocity, zero linear velocity) in the direction of the last known lane position until the path is re-acquired.

6. Conclusion

This project successfully demonstrated the viability of monocular vision for autonomous navigation. The final system is capable of:

Traversing a complex track with curves and straightaways without collision. .1

Reliably detecting traffic signs and executing safety maneuvers (3-second stop). .2

Recovering from signal loss using intelligent search algorithms. .3

While limited by the lack of depth perception compared to LiDAR-based systems, the combination of HSV color filtering, P-control, and State Machine logic proved to be a robust and cost-effective solution for the TurtleBot3 AutoRace challenge.

7. References

Project Presentation: "Autonomous Vehicle Final Project" - Masanbat Mulu & Yosef Chekol. .1

Bradski, G. (2000). "The OpenCV Library." Dr. Dobb's Journal of Software Tools. .2

Quigley, M., et al. (2009). "ROS: an open-source Robot Operating System." ICRA Workshop on Open Source Software. .3

Farag, W. (2020). "Complex Trajectory Tracking using PID Control for Autonomous Driving." International Journal of Intelligent Transportation Systems Research. .4