# ROS-Enabled Multi-Sensor Fusion for Autonomous Navigation of Turtlebot 3 Burger in Challenging Environments

**Edouard David**
**Master ISI**
**28712992**

**Mario Veshaj**
**Master ISI (ERASMUS)**
**21336837**

## Abstract

This report details the design, implementation, and evaluation of a multi-sensor fusion system, enabled by ROS, for the autonomous navigation of a Turtlebot 3 Burger in difficult environments. The system combines inputs from a camera and laser scanner to facilitate reliable lane following, obstacle avoidance, and tunnel crossing. The report also examines the performance of this architecture, focusing on the algorithmic decisions and their effects on navigation efficiency and reliability.

## 1 Introduction

This paper describes the creation of a ROS-based autonomous navigation system for the Turtlebot 3 Burger, using multi-sensor data fusion and advanced control strategies to navigate complex environments. It covers the ROS architecture, analyzes its components, and evaluates performance, emphasizing design choices and their effects on navigation tasks.

## 2 PRESENTATION OF THE ROS ARCHITECTURE

### 2.1 A short overview

In this subsection, we introduce the architecture used for robot navigation. The architecture consists of three main components:

1. **ImageProcessingNode:** Processes camera images to detect lane centroids and publishes them along with the target coordinate.

2. **LaserControlStrategy:** Subscribes to laser scan data, computes mean distances in different directions, and publishes them.

3. **LineFollowingNode:** Subscribes to lane centroid and laser scan data. It implements obstacle avoidance and line following strategies, switching to tunnel crossing mode when necessary.

Figure 1 illustrates the topology of the architecture, showing how nodes communicate with each other.

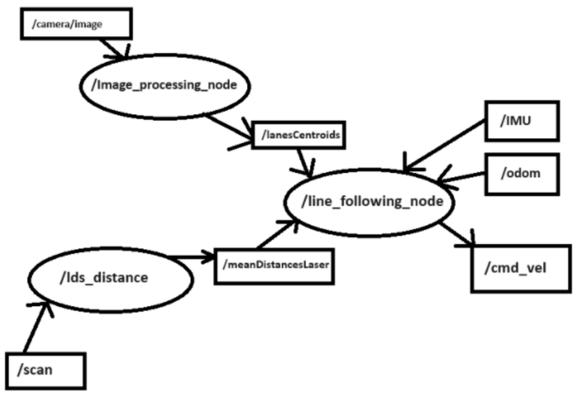

Figure 1: Architecture Topology

### 2.2 Algorithmic structure of the architecture

**ImageProcessingNode**: Processes images from the camera to detect the centroids of the lanes and publishes these centroids along with the target coordinate (middle of the image) to the **/lanesCentroids** topic.

**LaserControlStrategy**: Subscribes to laser scan data, calculates the mean distances in various directions (left forward, right forward, forward, left max, right max, left min, right min), and publishes these mean distances to the **/meanDistancesLaser** topic.

**LineFollowingNode**: Subscribes to both **/lanesCentroids** and **/meanDistancesLaser** topics.

Based on the centroids of the lanes and the mean distances obtained from laser scans, it decides on the control strategy to follow:

- If an obstacle is detected:
  - The node switches to the obstacle avoidance strategy, adjusting velocities based on the detected obstacles' positions.
  - It determines whether the obstacle is detected on the left or right and adjusts velocities accordingly.
  - If no obstacle is detected on either side, it continues on the current trajectory, using the IMU, it corrects its orientation so that it avoids obstacles while maintaining a consistent direction, adjusting velocities as needed.
- If no obstacle is detected:
  - The node switches to the line following strategy, which implements a PID controller to automatically follow the lines of the road.
  - It calculates the error between the current centroid position and the target centroid position.
  - Based on the error, it adjusts linear and angular velocities to keep the robot on track.
- Additionally, there's a condition to handle tunnel crossing:
  - When the real robot detects the blue strip marking the entrance of the tunnel and has initiated obstacle avoidance, it transitions into tunnel crossing mode. In the simulation, the robot employs odometry and the IMU to ensure it enters the tunnel's center upon reaching its entrance.
  - Once inside the tunnel, it switches to the tunnel crossing strategy, which uses a PD controller to navigate through the tunnel safely.

This algorithmic structure ensures that the robot can effectively navigate while following lanes, avoiding obstacles detected by the laser scan, maintaining a consistent direction with the IMU, and crossing tunnels when needed, with each node contributing its specific functionality.

## 3 System Dynamics Analysis

The system dynamics analysis investigates how the Proportional-Integral-Derivative (PID) controller's actions impact the line following strategy's behavior. It examines how the PID terms interact with the error, defined by the equation:

$$Error = Pixel_target - Pixel_centroid$$

This equation represents the disparity between the target pixel and the centroid pixel, serving as a fundamental metric for evaluating system performance.

### 3.1 Evolution of PID Terms and error

Figure 2 depicts the evolution of the error over time in relation to each PID term.

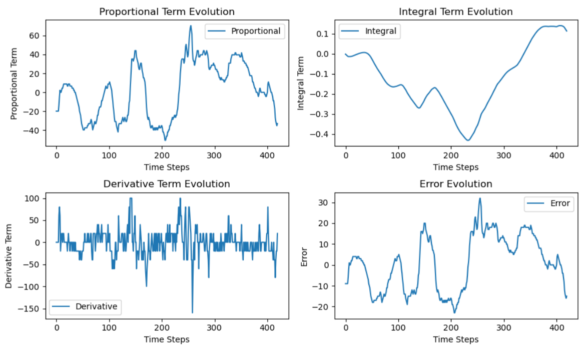

Figure 2: Evolution of each PID term in addition to the error evolution.

From the error evolution plot:

- **The proportional term:** follows the error closely, showing the same cyclical pattern with increasing and decreasing values, which is expected from the proportional term, two spies are worth noting, indicating when the robot enters and exists the roundabout.
- **The integral term:** initially decreases (goes negative), and then after the second spike in the error, it increases until stabilizing around 0.1. The integral term accumulates the error over time. When the error is consistently negative, the integral term will decrease. After a significant positive error spike, the accumulation trend shifts upwards, leading to stabilization once the error oscillations reduce.
- **The derivative term** shows a very oscillating behavior, resembling noise, with prominent

spikes at the same points as the error spikes. The derivative term represents the rate of change of the error. Small, rapid changes in error cause the derivative term to oscillate. The spikes in the error result in high derivative values due to sudden changes.

### 3.2 Histograms of PID Terms

Figure 3 illustrates the histograms of the PID terms (Proportional, Integral, and Derivative).

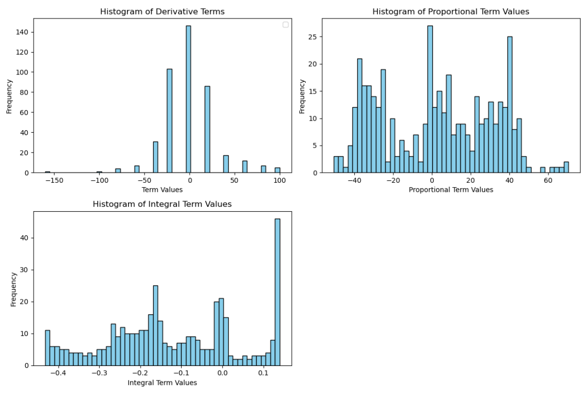

Figure 3: Evolution of each PID term in addition to the error evolution.

From the histograms, it is evident that:

- **Proportional Term Histogram:** Displays overlapping bell curves, indicating distinct operating modes or phases with varying error ranges, likely corresponding to different driving conditions or maneuvers.
- **Integral Term Histogram:** Appears generally flat, suggesting wide variation over time, with one notable spike possibly indicating a common operating point where error accumulation stabilizes.
- **Derivative Term Histogram:** Shows a bell curve centered around zero, indicating small and fluctuating error rate changes, typical in systems where positive and negative error changes balance each other out over time, such as when the robot adjusts its position to ensure alignment between targeted and computed coordinates.

### 3.3 Correlation Analysis

The correlation coefficients between each PID term and the error are summarized in Table 1.

The correlation analysis reveals:

| **PID Term** | **Correlation with Error** |
|---|---|
| Proportional | 1.0 |
| Integral | 0.09520627278961398 |
| Derivative | 0.06676387479298004 |

Table 1: Correlation Coefficients between PID Terms and Error

- **Proportional vs Error:** A correlation coefficient of 0.99999 shows a nearly perfect positive linear relationship, as the proportional term is directly derived from the error.
- **Integral vs Error:**The weak correlation indicates that the integral term, while influenced by the error, reflects the cumulative effect of past errors rather than the current error.
- **Derivative vs Error:** A correlation coefficient of 0.067 suggests a very weak positive linear relationship, meaning the derivative term tracks the rate of change of the error, not the error itself.

## 4 CONCLUSION AND PERSPECTIVES

This report outlines the development and deployment of a ROS-enabled multi-sensor fusion framework for autonomous navigation with the Turtlebot 3 Burger. Through the integration of diverse sensors and the application of advanced control methods, the robot adeptly navigates through challenging terrains, including lane tracking, obstacle evasion, and tunnel traversal. Key components such as the ImageProcessingNode, LaserControlStrategy, and LineFollowingNode play pivotal roles in achieving these tasks. However, significant latency issues were encountered during implementation. To address this challenge, a solution was proposed: the implementation of a phase-advance filter to reduce latency in real-time robot operations. This addition is expected to improve the responsiveness and efficiency of the navigation system, enhancing overall performance in dynamic environments.