Abhishek Sen : Documentation Report

Robust Navigation & Trajectory Control for Autonomous Mobile Robots

1. Overview

This project implements a complete navigation stack for a differential drive robot (TurtleBot3) operating in cluttered, unstructured environments. Unlike standard navigation stacks that rely on heavy global costmaps, this solution utilizes a lightweight, hybrid approach: combining **B-Spline Global Trajectory Generation** with a **Reactive Potential Field Controller**.

The system is capable of smoothing coarse global waypoints into a continuous path and robustly tracking that path while dynamically avoiding unmapped obstacles (walls, construction cones, barriers) using 2D LiDAR data.

2. Setup and Execution

2.1 Prerequisites

- **OS:** Ubuntu 22.04 LTS (Jammy Jellyfish)
- **Middleware:** ROS 2 Humble Hawksbill
- **Simulation:** Gazebo Classic 11
- **Dependencies:** `python3-scipy`, `python3-numpy`

2.2 Installation

1. **Clone the repository** into your ROS 2 workspace `src` directory:

2. **Install system dependencies:**
3. **Build the package:**
4. **Configure Environment:**

2.3 Execution Instructions

To replicate the full "Messy World" navigation scenario, use the following three terminals:

Terminal 1: Simulation Environment Launches Gazebo and spawns the robot along with randomized obstacles (cones, barriers).
Using the command : ros2 launch turtlebot3_gazebo empty_world.launch.py

Terminal 2: Visualization Launches RViz configured to show the calculated path and sensor data.

Using the command : ros2 run rviz2 rviz2

Terminal 3: Navigation Controller Starts the core node for path smoothing and control.

Using the command : ros2 launch turtlebot3_gazebo empty_world.launch.py

3. Design & Architecture

2.2 Algorithms and Architectural Decisions

The system is architected as a single, high-frequency ROS 2 node (RobustNavigator) operating at 20Hz. The logic is divided into three distinct pipeline stages:

A. Path Smoothing (Global Planner)

- **Problem:** Raw waypoints produce piecewise linear paths with sharp C^0 continuity corners, causing the robot to stop and rotate in place.
- **Solution:** Implemented **B-Spline Interpolation** using scipy.interpolate.splprep.
- **Design Choice:** A cubic spline (k=3) was chosen to ensure C^2 continuity (continuous velocity and acceleration). A smoothing factor (s=0.5) allows the trajectory to deviate slightly from waypoints to minimize curvature energy, resulting in more natural motion.

B. Trajectory Tracking (Controller)

- **Algorithm: Pure Pursuit** with Dynamic Lookahead.
- **Logic:** The controller calculates the curvature required to drive the robot from its current pose to a target point on the path.
- **Optimization - Sequential Search:** To handle looping paths (where Start == End), the lookahead search is restricted to a sliding window ahead of the robot's current index. This prevents the "shortcut problem" where the robot might mistakenly lock onto the finish line immediately after starting.
    - ***Control Law:*** **W = v_target, w = (2v *sin(a)) / (L_d)**

Where :

W = Angular Velocity , v_target = Linear Velocity , a =Heading Error (or Lookahead Angle) , L_d = Lookahead Distance.

C. Reactive Obstacle Avoidance (Local Planner)

- **Algorithm: Artificial Potential Fields (APF)**.
- **Logic:** The robot is treated as a particle moving through a force field.
    - **Attractive Force:** Pulls the robot along the Lookahead Vector toward the global path.
    - **Repulsive Force:** Generated by LiDAR points within a defined safety_dist (0.6m). The magnitude follows an inverse-square law (1/d^2) creating a "stiff" virtual spring against obstacles.
- **Result:** The final velocity vector is a weighted sum of these forces, allowing the robot to smoothly deviate from the path to avoid collision and naturally merge back when clear.

2.3 Extension to Real Hardware (TurtleBot3)

Migrating this node from Gazebo simulation to a physical TurtleBot3 requires three key modifications:

1. **State Estimation (Localization):**
    - *Simulation:* Relies on `/odom` (ground truth from wheel encoders).
    - *Real World:* Wheel encoders drift over time. I would integrate **sensor fusion** using an Extended Kalman Filter (`robot_localization` package) fusing IMU and Encoder data. For long-term navigation, I would implement **AMCL** (Monte Carlo Localization) to correct drift against a known map.

2. **Sensor Interface:**
    - The `scan_callback` assumes a 360-degree perfect sensor. On a real robot, I would implement a **median filter** on the LiDAR data to remove sensor noise and spurious readings before calculating repulsive forces.
3. **Safety & Fail-safes:**
    - Network latency can cause control loops to hang. I would implement a **dead-man switch** in the node: if `odom` or `scan` data is stale (>0.5s), the robot must command zero velocity immediately.

2.4 AI Tools Utilization

Generative AI tools (GPT-4) were utilized to accelerate the development workflow in the following capacities:

- **Boilerplate Generation:** Rapidly scaffolding the ROS 2 class structure, publishers, and subscribers to ensure adherence to ROS 2 object-oriented best practices.
- **Mathematical Verification:** Validating the vector transformation math required to convert Global Frame coordinates (Odom) into Robot Frame coordinates (Base Link) for the Pure Pursuit curvature calculation.
- **Debugging:** Assisting in identifying the logic flaw regarding circular paths, leading to the implementation of the "Sequential Index Search" to prevent premature goal completion.

2.5 Extra Credit: Obstacle Avoidance Implementation

The avoidance system is not a simple "stop and wait" logic; it is a **continuous active controller**.

Implementation Details:

1. **Safety Bubble:** A warning_dist (0.6m) and critical_dist (0.35m) are defined.
2. **Force Vectoring:**
    - The LiDAR scan is segmented into a frontal cone (-60° to +60°).
    - A repulsive vector F_rep is calculated by summing the inverse vectors of all points in this cone.
3. **Dynamic Speed Throttling:**
    - The linear velocity v is modulated by the magnitude of the repulsive force.
    - **v_cmd = v_target * (1.0 - |F_rep|)**

    - This ensures the robot naturally slows down when navigating narrow gaps (like the house doorways) to allow for tighter turning radii.
4. **Stuck Recovery:**
    - If the robot's Euclidean displacement is < 0.01m for 2 seconds while moving, a **Recovery Behavior** is triggered: the robot reverses linear velocity (-0.15 m/s) and rotates to dislodge itself from local minima.

RQT Graph

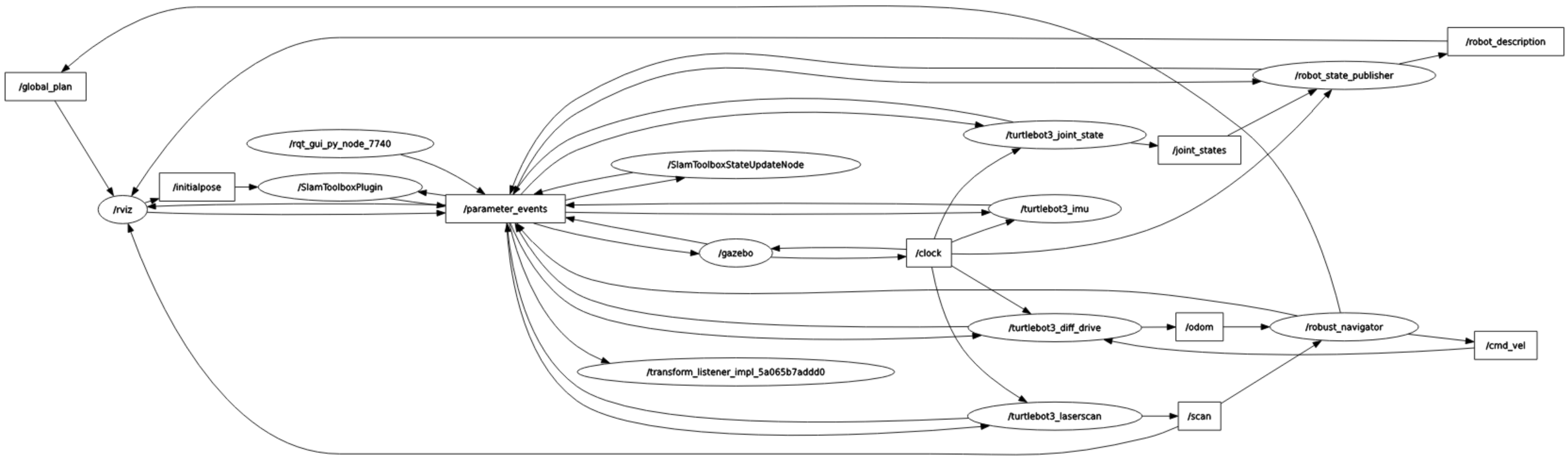

Fig :- This is the RQT graph