Path Smoothing and Trajectory Control in 2D Space.

Overview

This project implements:

- Path smoothing using cubic spline interpolation
- Time-parameterized trajectory generation
- Pure Pursuit trajectory tracking
- Simulation on TurtleBot3 (ROS2 + Gazebo)
- Performance evaluation using error metrics

The goal is to convert discrete 2D waypoints into a smooth trajectory and accurately track it using a differential drive robot.

Instructions

Basic setup:

```
source /opt/ros/humble/setup.bash
```

Repository

Navigate to your ros2 workspace and clone this repository inside the `src` folder:

```
mkdir -p ~/origin_ws/src
cd ~/origin_ws/src

git clone https://github.com/hanmol0312/trajectory_control.gi
t
```

```
# Build the workspace
cd ~/origin_ws
colcon build --symlink-install

# Source the workspace
source install/setup.bash
```

Changing paramater file and updating parameters:

```
cd ~/origin_ws/src/trajectory_control/config
nano controller_params.yaml


trajectory_controller:
  ros__parameters:

    # lookahead: 0.05
    # velocity: 0.2
    goal_position_tolerance: 0.1
    goal_heading_tolerance: 0.1
    lookahead: 0.2
    max_velocity: 0.4
    curvature_gain: 2.0
    k_theta: 0.8
    omega_max: 1.8


    # waypoints: [                                             #st
raight line
    #   0.0, 0.0,
    #   5.0, 0.0
    # ]

    # waypoints: [                                               #
```

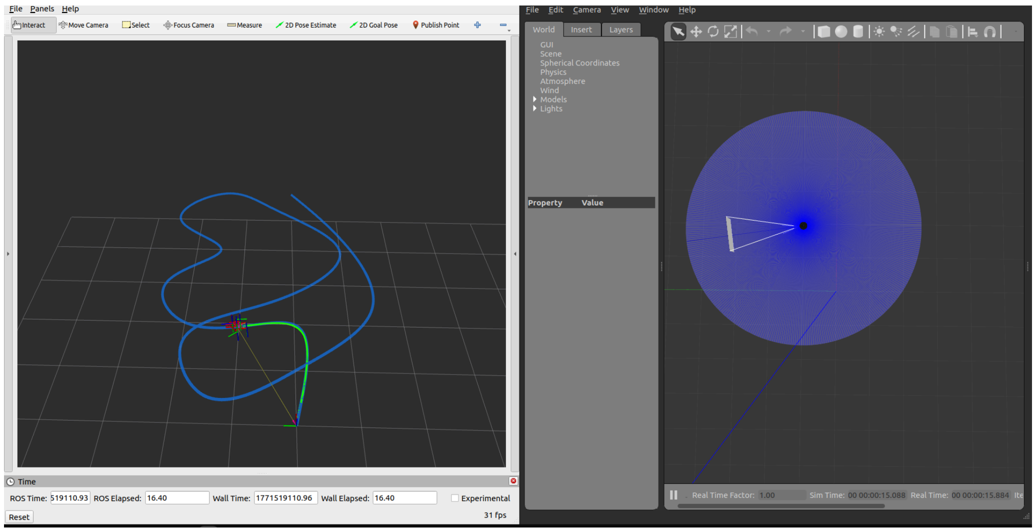
```
gentle_curve
    #   0.0, 0.0,
    #   2.0, 0.0,
    #   2.0, 2.0
    # ]

    # waypoints: [
# sharp turn
    #   0.0, 0.0,
    #   2.0, 0.0,
    #   2.0, 2.0
    # ]

    waypoints: [

      # --- Straight ---
      0.0, 0.0,
      2.0, 0.0,

      # --- 90 degree turn ---
      2.0, 2.0,

      # --- Zig-zag ---
      3.0, 2.5,
      4.0, 1.5,
      5.0, 2.5,
      6.0, 1.5,

      # --- S-curve ---
      5.5, 0.5,
      4.0, -1.0,
      2.5, 0.5,

      # --- Tight U-turn ---
      1.5, 2.0,
      0.5, 1.5,
      1.0, 0.0,

      # --- Final smooth return ---
      2.5, -1.5,
      4.5, -1.0,
      6.0, 0.0

    ]
```

Running the Launch File:

Run these commands in the terminal where you buit the code:

```
cd ~/origin_ws
source install/setup.bash
ros2 launch trajectory_control trajectory.launch.py
```

This will launch the rviz, gazebo showing the trajectory smoothened and the trajectory taken by the robot on the way.



ROS Time: 519110.93 ROS Elapsed: 16.40 Wall Time: 1771519110.96 Wall Elapsed: 16.40 ☐ Experimental

31 fps

Real Time Factor: 1.00 Sim Time: 00 00:00:15.088 Real Time: 00 00:00:15.884

At startup image.

Design Choices:

1. Path Smoothing:

- Implemented parametric cubic spline
- Arc-length parameterization

Why?

- C2 continuity
- Smooth curvature
- Suitable for differential drive robots

2. Trajectory Generation

- Sampled at fixed arc-length intervals

Each trajectory point:

Here theta is the heading at particular point on the curve.

```
(x,y,theta,t)
```

3. Trajectory Tracking

Controller: Implemented Pure Pursuit

Curvature = 1/R

R = L^2/Y_r

here L is length of the arc and Y_R is the lateral error.

Angular velocity:

ω=v·κ\omega = v \cdot \kappa

ω=v·κ

Enhancements:

- Goal alignment correction
- Velocity scaling
- Lookahead tuning
- Curvature gain for reducing velocities at sharp curves depending on the sharpness of the curve.

4. Controller flow:

Waypoints
↓
Spline2D (Path Smoothing)
↓
Trajectory Generator
↓
Pure Pursuit Controller
↓
cmd_vel → TurtleBot3
↓
Odometry Feedback

The architecutre is as above and below is the code base structure:

```
├── include
│   └── trajectory_control
│       ├── ppt_controller.hpp
│       ├── spline.hpp
│       └── trajectory.hpp
├── launch
│   └── trajectory.launch.py
├── package.xml
├── README.md
├── rviz
│   └── traj_config.rviz
├── src
│   ├── controller.cpp
│   ├── controller_node.cpp
│   ├── spline.cpp
│   └── trajectory.cpp
```

Extension To A Real Robot:

- This could be implemented on the real robot understanding the motor tolerance so that the trajectory tracking doesnt stress the motor a lot. In such cases more feedback mechanisms as in one like the curvature gain that help reduce the velocity on curves similarly acceleration, deccelration can be implemented depending on the distance of robot from the desired goal.
- Aparty from this PID controllers can be implemented to achieve stability and less steady state errors and do the reauired balancing.
- Replace simulated odometry with wheel encoder data
- Use real LiDAR for obstacle detection
- Handle sensor noise and latency

AI tools used:

- I used chatgpt to understad spline path generation and parametric function generation for implementing 2D spline generation c2 continuity also took help in coding the spine generation.

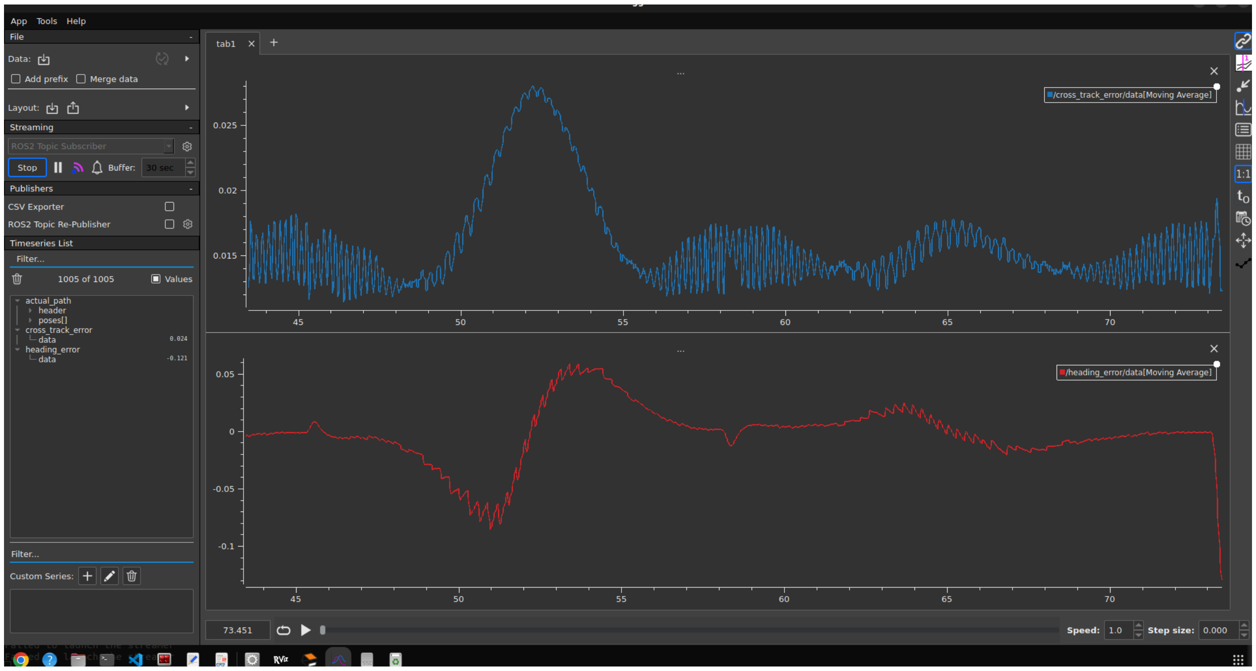
Extension to avoid obstacles:

- The controller can be extended to avoid obstacles by implmenting first costmap around obstacles, considering the robot’s footprint.

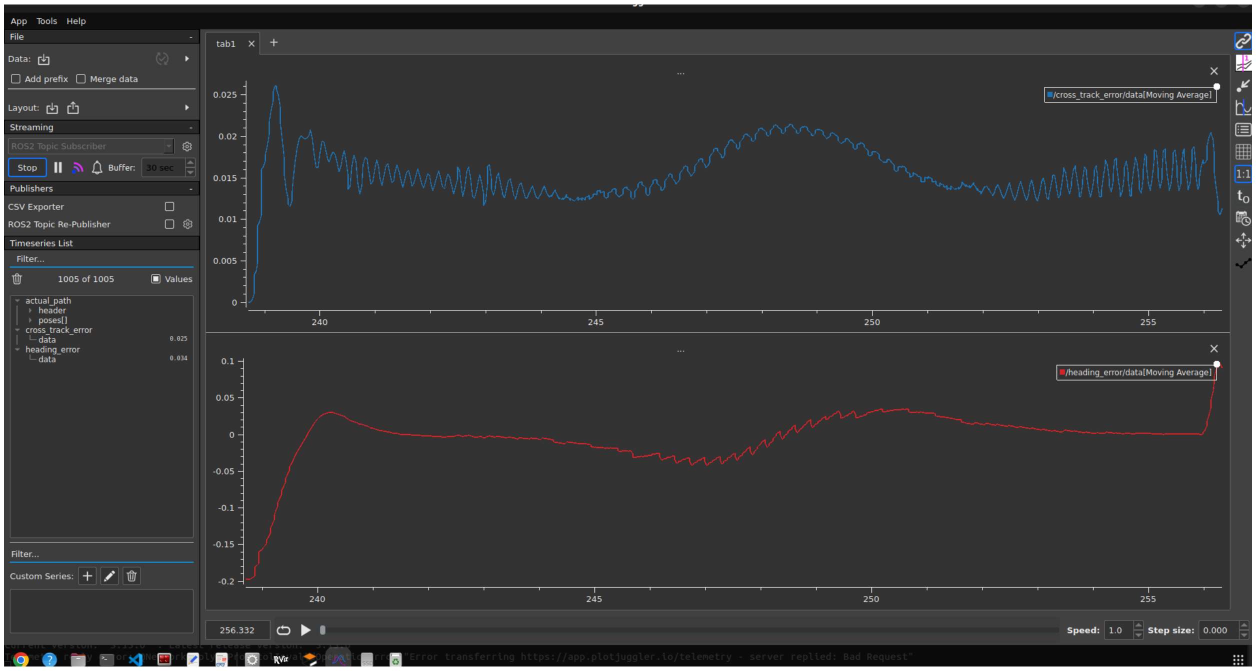
- Next whenever robot is 1-2 m from the object stop the robot, generate an alternative path to the nearest waypoint after the obstacle by implementing algorithms like A* or Djikstra.
- After reaching the goal just after the obstacle continue on the path.

Plots and Results:

Cross track and heading error for S- shaped smoothened curve.

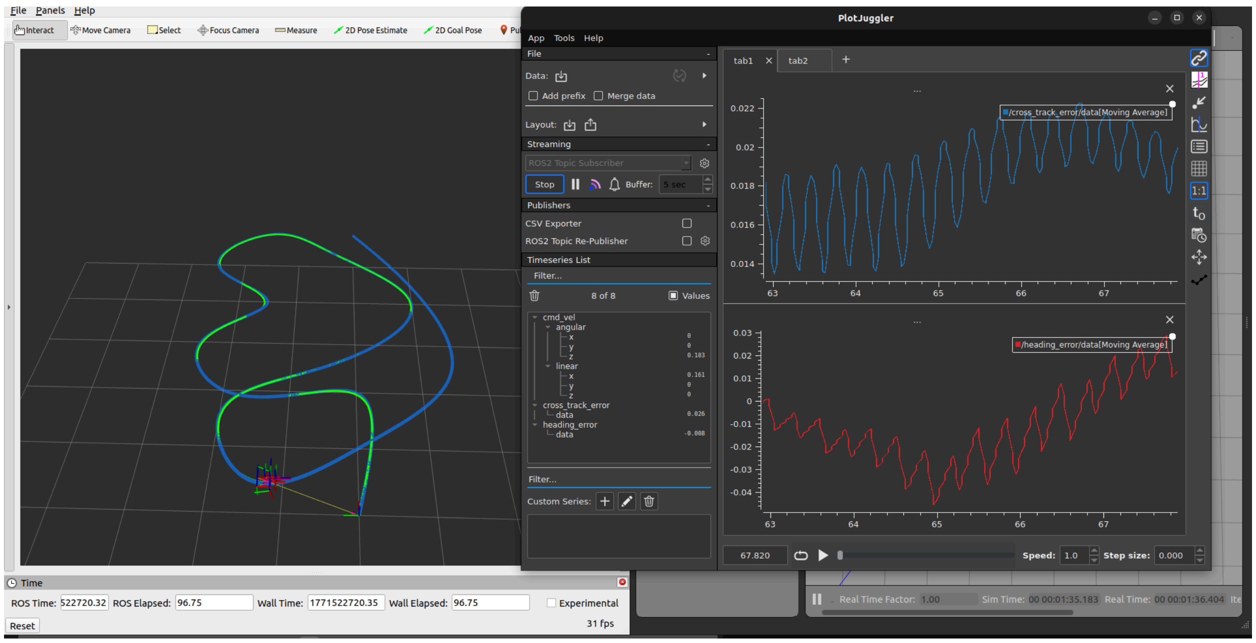


Cross track and heading error for Sharp Turned curve:



```
Error transferring https://app.plotjuggler.io/telemetry - server replied: Bad Request
```

Linear and angular velocity profiles:



ROS Time: 522720.32 ROS Elapsed: 96.75 Wall Time: 1771522720.35 Wall Elapsed: 96.75 Experimental

Reset 31 fps

Real Time Factor: 1.00 Sim Time: 00 00:01:35.183 Real Time: 00 00:01:36.404