ROS106 Assignment

MCP100: Filtering Noisy Robot Pose and Transforming Points

IIT Delhi

May 26, 2025

Disclaimer

This assignment is for fun and learning only! We will check mainly for approach part and not what is final result. We will try to provide an evaluator, but the main goal is for you to learn and experiment. Please try the implementation yourself—it will help you a lot in the future!(regarding ROS2 only ;)

Background Story

You have just finished your first year at IITD (well, you have done MCP100, but imagine you are making the project again). As part of your MCP100 project, you built a robot called **Robot A**. Robot A can move around and tell you its position in the **world frame** (where it is and which way it is facing). But there is a problem: Robot A's position data comes in very fast and is **noisy** (jumpy and not smooth). Because of this, Robot A sometimes moves in a shaky way.

Now, there is a **Professor** standing at a fixed point in the world. Sometimes, you want to know: *Where is Robot A in the Professor's frame?*

Your job: Make the position data smooth, and use this smooth data to answer the question: "Where is Robot A in the **Professor's frame**?"

Assignment Overview

You will create **two ROS 2 nodes**:

- **Pose Filter Node:** Reads noisy pose data, smooths it, and publishes the filtered data.
- **Transform Service Node:** Uses the filtered data to transform Robot A's position from the world frame to the **Professor's frame**.

Task 1: Pose Filter Node

Node Details

- **Name:** `pose_filter_node`
- **Subscribes to:** `/robot_a/raw_pose` (custom message, pose in world frame)
- **Publishes to:** `/robot_a/filtered_pose` (custom message, pose in world frame)
- **Buffer:** Always keeps the last 10 messages
- **Publish Rate:** 10 Hz (every 0.1 seconds)
- **Publish Condition:** Only publishes when buffer is full (10 messages)

Custom Message

File: `msg/RobotPose2D.msg`

```
float64 x
float64 y
float64 theta
```

Averaging Algorithm

- Keep a buffer (list) of last 10 poses.
- For each new message:
  - Add to buffer.
  - If buffer > 10, remove oldest.
- To smooth: Average each of x, y, and θ over the buffer.
- Publish the averaged pose.

Task 2: Transform Service Node

Node Details

- **Name:** `transform_service_node`
- **Subscribes to:** `/robot_a/filtered_pose` (custom message, pose in world frame)
- **Provides service:** `/get_robot_position_in_professor_frame` (custom service)
- **Always keeps the latest filtered pose.**
- **Professor's Pose:** Assume the Professor is standing at (x_p, y_p, θ_p) in the world frame (these values are fixed and known).

Custom Service

File: `srv/GetRobotPosition.srv`

```
# Request (empty - no parameters needed)
---
# Response
float64 x
float64 y
float64 theta
```

How the Service Works

- **Service Call:** Client calls the service with no parameters (empty request)
- **Processing:** Service node uses the latest filtered pose from `/robot_a/filtered_pose` topic
- **Transformation:** Converts Robot A's world coordinates to Professor's frame coordinates
- **Response:** Returns Robot A's complete pose (x, y, theta) relative to Professor's frame

Transformation Math

Given:

- Latest pose of Robot A in world: (x_r, y_r, θ_r)
- Professor's pose in world: (x_p, y_p, θ_p)

Steps:

1. **Step 1:** Calculate the difference between Robot A's position and Professor's position in world frame:

$$dx = x_r - x_p \tag{1}$$
$$dy = y_r - y_p \tag{2}$$

2. **Step 2:** Transform world coordinates to Professor's frame:

$$x_{prof} = \cos(-\theta_p) \cdot dx - \sin(-\theta_p) \cdot dy \tag{3}$$
$$y_{prof} = \sin(-\theta_p) \cdot dx + \cos(-\theta_p) \cdot dy \tag{4}$$

3. **Step 3:** Calculate Robot A's orientation relative to Professor's frame:

$$\theta_{prof} = \theta_r - \theta_p \tag{5}$$

   Note: Normalize θ_{prof} to $[-\pi, \pi]$ range if needed.

The service should return $(x_{prof}, y_{prof}, \theta_{prof})$ as Robot A's complete pose in the Professor's frame.

Implementation Details

Package Structure

Create your ROS 2 package with the following structure:

```
ros2_assignment_pkg/
 package.xml
 setup.py
 CMakeLists.txt (if using C++)
 msg/
    RobotPose2D.msg
 srv/
    GetRobotPosition.srv
 ros2_assignment_pkg/
     __init__.py
     pose_filter_node.py
     transform_service_node.py
```

Key Implementation Points

- **Pose Filter Node:**
  - Use a circular buffer or list to store last 10 poses

  - Calculate moving average for x, y, and theta separately
  - Handle angle averaging carefully (consider wrapping around at $\pm\pi$)
  - Only publish when buffer has exactly 10 messages
- **Transform Service Node:**
  - Store Professor's fixed pose as class variables
  - Keep track of the latest filtered pose from subscription
  - Implement coordinate transformation using rotation matrices
  - Handle empty service requests properly
  - Return complete pose (x, y, theta) in response

Testing Your Implementation

- Create test scenarios to verify your pose filtering works correctly
- Test the coordinate transformation math with known inputs and expected outputs
- Verify that both nodes communicate properly through topics and services
- Check that your implementation handles edge cases gracefully

Checklist

Created `msg/RobotPose2D.msg` with `float64 x`, `float64 y`, `float64 theta`

Created `srv/GetRobotPosition.srv` with empty request and `float64 x`, `float64 y`, `float64 theta` response

Pose filter node subscribes to `/robot_a/raw_pose` (custom message)

Pose filter node publishes to `/robot_a/filtered_pose` (custom message)

Pose filter node keeps buffer of last 10 messages, averages, and publishes at 10 Hz

Transform service node subscribes to `/robot_a/filtered_pose` (custom message)

Transform service node provides `/get_robot_position_in_professor_frame` service (custom service)

Service uses latest filtered pose to transform Robot A's complete pose (x, y, theta) to Professor's frame and returns answer

All topics and services use correct names and custom messages/services

All code is easy to read and matches assignment requirements

Submission Instructions

- Create a complete ROS 2 workspace with your package following the specified structure
- Ensure your package builds successfully using `colcon build`
- Include all required files: custom message, custom service, and both Python nodes
- Test your implementation to verify it works as expected(will provide an evaluator but will take some time for us to build.)
- Document(just write in code comments) any assumptions you made (e.g., Professor's fixed pose values)
- Submit your complete package - anything that works will fetch non-zero results!
- Include a brief README explaining how to run your nodes and test the functionality

Good Luck!(I guess no need for its cause not an insitute level course but ok :)

Have fun!