Homework Report — ROS2 Teleoperation and Stanley Controller

Umut ŞENER

1. Introduction

The goal of this homework was to design and implement a simple ROS2-based system that allows manual and autonomous control of a `turtlesim` robot. The system consists of three main nodes:

- **Teleoperation Node** (`virdult_teleop_key.cpp`) for manual keyboard control and toggling autonomous mode.
- **Path Publisher Node** (`path_publisher.cpp`) for generating and publishing a circular or square path.
- **Stanley Controller Node** (`stanley_controller.cpp`) for following the path using the Stanley control method.

2. Explanations and Observations

Teleoperation Node

This node reads keyboard inputs directly from a Linux input device:
(e.g., `/dev/input/by-path/...-event-kbd`) and publishes velocity commands to the topic `/turtle1/cmd_vel`. Pressing **X** toggles autonomous mode, publishing a Boolean value to the topic `autonomous_mode`. While in autonomous mode, manual control is disabled.

Path Publisher Node

When autonomous mode is activated, this node locks the turtle's current position as the center and publishes a predefined path on the topic. Also, I made it so that code actually always draws a circle around the turtle, yet however many destinations you assign, the path changes accordingly. If you change the code as 4, it will be a square; if 3, it will be a triangle; if around 100, it will be a circular path since the destinations have 3.6 degrees in between. The node continuously republishes the path at regular intervals to ensure it stays available for the controller.

Stanley Controller Node

This node subscribes to the current pose (`/turtle1/pose`) and the path (`/path`), and computes steering commands based on the Stanley controller algorithm. It calculates cross-track and heading errors and publishes the resulting linear and angular velocities to `/turtle1/cmd_vel`. The controller maintains a smooth path-following behavior and updates in real time.

3. Screenshots

The following screenshots demonstrate the working system.

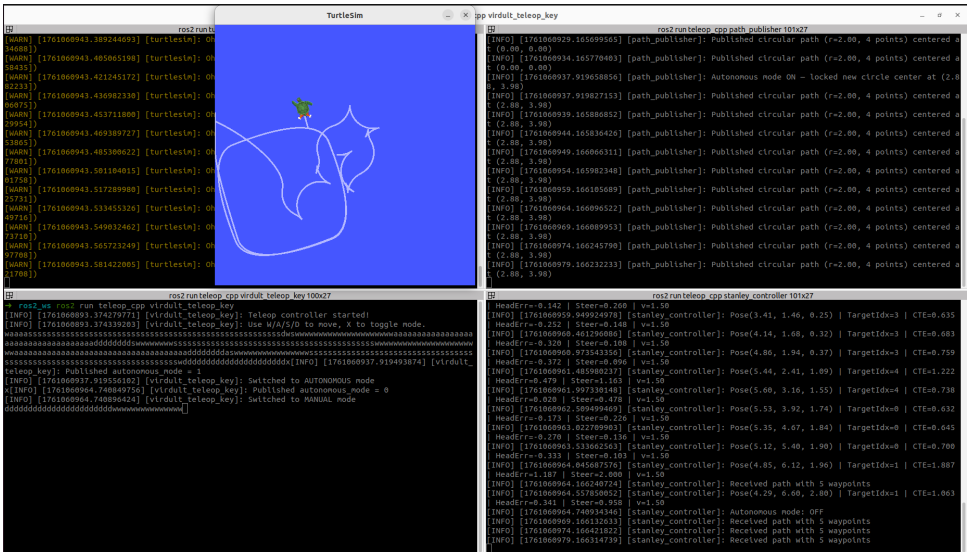

Figure 1: Teleoperation node successfully started and awaiting keyboard input.

4. Challenges and Solutions

- **Pressing Multiple Keys At Once:** Pressing multiple keys at once and making the turtle turn at the time it is also moving was not possible. Therefore I found a solution to it by using the current computers keyboard device in the code and directly take the keyboards input.
- **Keyboard Device Access:** Initially, the teleop node could not open the keyboard device due to permission issues. This was solved by running the node with `sudo` privileges or changing device permissions using `chmod a+rw /dev/input/...-event-kbd`.
- **Center Locking for Path:** Ensuring the path only updated when autonomous mode toggled ON required adding a mode callback and proper flag management.

5. Conclusion

This assignment helped me understand how to integrate multiple ROS2 nodes for interactive control and path-following tasks. It demonstrated how manual teleoperation can coexist with autonomous control and how to apply the Stanley controller for trajectory tracking. The challenges faced with device access and ROS2 environment setup strengthened my understanding of ROS2's structure and debugging process. Also I learned how to use the terminal more efficiently since I was new to linux and I have never used it before. (e.g., ctrl+shift+e and ctrl+shift+o . Also, these new terminals opened with the shortcuts also opens in the same file, but does not carry sourcing.)