ITU ZES Solar Car Team Autonomous Group Homework 1

1 Introduction

Welcome to your first homework in the ITU ZES Solar Car Team Autonomous Systems Group! This assignment consists of three main parts that will help you understand the fundamentals of ROS 2 basics and trajectory tracking control. Each step builds upon the previous one, leading you from basic ROS 2 operations to implementing a fully functional path-following controller using the Stanley method, which is described in this document (starting from page 14).

You are expected to document your progress with screenshots, explanations, and a short summary for each step.

2 1. ROS 2 Humble Installation

Objective: Install and verify ROS 2 Humble Hawksbill on your computer.

Tasks:

- Follow the official installation guide for ROS 2 Humble on Ubuntu 22.04: `https://docs.ros.org/en/humble/Installation.html`
- After installation, source the setup file and verify your ROS environment:

```
source /opt/ros/humble/setup.bash
ros2 run demo_nodes_cpp talker
```

Open another terminal and run:

```
source /opt/ros/humble/setup.bash
ros2 run demo_nodes_py listener
```

- Take screenshots of your terminal showing successful ROS 2 installation and the output of the above commands.
- **Tip:** To avoid sourcing ROS 2 every time you open a new terminal, add the following line to your **bashrc** file:

  ```
  echo "source /opt/ros/humble/setup.bash" >> ~/.bashrc
  source ~/.bashrc
  ```

  This ensures that your ROS 2 environment is automatically set up each time you start a terminal session, making your workflow more efficient.

3 Turtlesim and Keyboard Controller

Objective: Learn ROS 2 publisher–subscriber communication by creating your own keyboard controller node for Turtlesim.

Tasks:

- Launch the Turtlesim node:

  ```
  ros2 run turtlesim turtlesim_node
  ```

- Create your own C++(do not use `turtle_teleop_key`) that:
  - Subscribes to keyboard input from the terminal,
  - Publishes `geometry_msgs/msg/Twist` messages to the topic `/turtle1/cmd_vel`,
  - Allows manual control using the following keys:

    ```
    w - move forward
    s - move backward
    a - turn left
    d - turn right
    ```

  - Includes a special key (e.g., `x`) to toggle between **manual** and **autonomous** modes:
    * Pressing `x` once → switch to autonomous mode
    * Pressing `x` again → return to manual control

- Prints the current mode (manual/autonomous) to the terminal each time the mode changes.

Expected Deliverables:

- Screenshot of your running node controlling the turtle manually.
- Screenshot showing autonomous mode activated (with printed mode change).
- Short explanation describing how you implemented mode switching.

4 Path Following with Stanley Controller

Objective: Implement a Stanley Controller that follows a predefined path and evaluates total tracking error.

Tasks:

- Create your own `path_publisher` node instead of using a provided one. The node should publish a simple path (as a list of positions) that the turtle will follow.
- Define the path as a sequence of waypoints in 2D space. You may create the path in any shape (e.g., straight line, curve, square) but it should be continuous. The turtle starts at approximately the center of the Turtlesim window, around coordinates (5.5, 5.5).
- The path can be published using either:
  - `nav_msgs/msg/Path` message (recommended), or
  - a custom message containing an array of (x, y, θ) coordinates, where θ represents the orientation (yaw angle) at each waypoint.
- Build and run your path publisher:

```
colcon build
source install/setup.bash
ros2 run <your_package_name> path_publisher
```

- Implement your own Stanley Controller node in C++. It should:
  1. Subscribe to the published path topic.

2. Subscribe to the turtle's current pose.
3. Calculate the steering command based on the Stanley control law.
4. Publish corresponding velocity/heading commands to `/turtle1/cmd_vel`.
5. Continuously compute the absolute lateral error during motion and accumulate it.

Expected Deliverables:

- Screenshot of your visualization (path + trajectory) from rviz.
- Terminal output showing total accumulated lateral error.
- Explanation of how you computed the error term.

5 Reporting and Submission

Deliverables:

- A short report (in PDF format) that includes:
  - Screenshots from each part of the homework.
  - Explanations and observations for each section.
  - Discussion of challenges faced and how you solved them.
- Upload all your source code to a public GitHub repository.
- Include a clear **README.md** file describing:
  - How to build and run each node,
  - ROS topics used in the project,
  - Explanation of your Stanley Controller implementation.

6 Conclusion

You are not required to have prior experience with ROS 2 or control theory to complete this assignment. The main goal is to help you learn through implementation and experimentation. We are excited to see your creative solutions — good luck and have fun coding!