Instruction for Preliminary Project 2

RC Car Keyboard Control

1. Introduction

In this preliminary project, we will practice implementing ROS2 communication and get used to our gymnasium environment by controlling virtual RC car with keyboard input. Skeleton codes are available at https://github.com/rllab-snu/Intelligent-Systems-2024-Pre.git.

First, clone the repository and install properly following the instructions in the next section. If you run the ROS2 packages properly, RC car will be shown on the track, but it won't move even if you press any keys. You should **fill in five TODO parts** in **run_env.py** and **keyboard_control.py** to make the whole system work.

2. Gym environment & ROS2 setting

2.1. Gym environment setting

To begin with, you need to install gym environment for the RC car and its dependencies. We recommend you install packages inside a virtual environment such as Anaconda (or virtualenv).

```
conda create -n rccar python=3.8
conda activate rccar

git clone https://github.com/rllab-snu/Intelligent-Systems-2024-Pre.git
cd Intelligent-Systems-2024-Pre/rccar_gym
pip install -e .
```

2.2. ROS2 setting

We use 'ROS2 Foxy' to run the gym environment and project codes. We assume all of you have already installed ROS2 Foxy in preproject 1. (If you have not installed it yet, please refer to https://docs.ros.org/en/foxy/Installation/Ubuntu-Install-Debians.html)

Since ROS2 can share nodes and topics within the same networks, it would be safe to run ROS2 executables only in localhost. Thus, we recommend adding the following command at '~/.bashrc'.

```
export ROS_LOCALHOST_ONLY=1
```

To build the ROS2 packages to use your specific python virtual environment, you should install colcon building tools after activating your virtual environment.

```
conda activate rccar
pip install colcon-common-extensions
```

This enables installed files resulting from colcon build to use desired packages in your environment.

Next, install dependencies and build the packages.

```
cd Intelligent-Systems-2024-Pre
rosdep update --rosdistro foxy
rosdep install -i --from-path src --rosdistro foxy -y
colcon build --symlink-install
```

Note that '--rosdistro foxy' is required for 'rosdep update' since foxy is an end-of-life version, and '--symlink-install' is required to use modified python files directly without building again.

After building the package, you must type the following commands in every terminal you want to use your packages.

```
source install/setup.bash
```

To run the system, you need to run two ROS2 executables in two separate terminals.

You can run the 'rccar_bringup' node by following command in the first terminal.

```
ros2 run rccar_bringup rccar_bringup
```

You can run the 'keyboard_control' node by following command in the second terminal.

```
ros2 run rccar_bringup keyboard_control
```

* When the rendering window does not appear after running two commands, there might be a communication error, so please shut down both commands and run again.

3. Skeleton Code

- You need to implement five TODO parts in run_env.py, keyboard_control.py at Intelligent-Systems-2024-Pre/src/rccar_bringup/rccar_bringup directory.

For an understanding of publisher and subscriber creation, please refer to the following link. https://docs.ros.org/en/foxy/Tutorials/Beginner-Client-Libraries/Writing-A-Simple-Py-Publisher-And-Subscriber.html

3.1 Overview

As we have studied in the ROS2 tutorial session, publisher and subscriber exchange data via messages. Among the message types, topic is most commonly used. Here, our system consists of **two nodes**:

- Node **'rccar_bridge'**
  (Intelligent-Systems-2024-Pre/src/rccar_bringup/rccar_bringup/run_env.py)
- Node **'keyboard_control'**
  (Intelligent-Systems-2024-Pre/src/rccar_bringup/rccar_bringup/keyboard_control.py)

The 'rccar_bridge' node is a bridge for the 'rccar_gym' environment to be used in ROS2 simulation. The 'keyboard_control' node gets control input from the keyboard and hands it over to the simulation.

We use **four topics** for communications: **'/running'**, **'/reset'**, **'/action'**, and **'/terminate'**. The brief explanations of the topics are as follows:

- The node 'rccar_bridge' publishes the topic '/running' every 0.005 seconds using a timer to notify the controller node that the environment is ready to be reset.
- The node 'keyboard_control' subscribes to the topic '/running' and publishes the topic '/reset'. Once the node 'rccar_bridge' receives the topic '/reset', rccar simulation based on the pygame library resets and the rendering begins.
- The node 'keyboard_control' receives keyboard input from the user and publishes the topic '/action' which contains corresponding speed and steering angle information. The node 'rccar_bridge' subscribes to this topic and moves the RC car accordingly.
- When the car collides with the wall or completes 1 lap without crashing, the node 'rccar_bridge' publishes the topic '/terminate'. The node 'keyboard_control' subscribes to the topic '/terminate' to know whether the environment is terminated.

The following figure is the relationship between the nodes and topics illustrated by 'rqt_graph'.

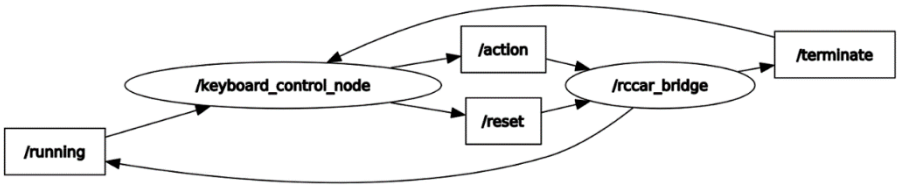

Figure 1. Nodes and topics illustrated by 'rqt_graph'.

3.2 run_env.py

Receiving '/reset' from the node 'keyboard_control', 'reset_callback' is called and the RC car is spawned on a track (env.reset). When '/action' arrives, 'action_callback' moves the RC car accordingly (env.step) and gets current state information of the environment. When the car collides with the wall or completes 1 lap without crashing, 'action_callback' publishes '/terminate' containing 'True'.

3.3. keyboard_control.py

w	Move straight forward	**a**	Turn left
s	Stop	**d**	Turn right

Function 'run_control' gets keyboard input, decides appropriate speed and steer (please check the limitations), and passes them through the '/action' topic. Whole processes repeat within a while loop to enable continuous RC car control. When '/terminate' is received, 'terminate_callback' is called. You can verify whether '/terminate' is properly published and received by checking ">>> Terminated" printed in both terminals.

In all projects, we are going to use the 'AckermannDriveStamped' type message as the '/action' topic. 'AckermannDriveStamped' is a message type used to represent the driving commands for vehicles that utilize the Ackermann steering mechanism. This type of steering is common in traditional automobiles, where the front wheels steer while the rear wheels follow a curved path. To learn how to use the message type 'AckermannDriveStamped' imported from 'ackermann_msgs.msg' module, please refer to the following documentation.
https://docs.ros.org/en/jade/api/ackermann_msgs/html/msg/AckermannDriveStamped.html

As mentioned in ROS2 tutorial session, you can check current states of nodes and topics by following commands in individual terminals.

```
ros2 node list
ros2 topic list
```

If you want to visualize ROS graph, open a new terminal and execute 'rqt_graph'.

```
ros2 node list
ros2 topic list
rqt_graph
```

4. Submission Format

You should submit three contents:

1) Your codes: Compress your 'Intelligent-Systems-2024-Pre' folder which includes all your files
2) A short video or GIF file showing your RC car moving on the map (It must show that your car can make clear turns at the corners.)
3) Screenshots of ">>> Terminated" in both terminals when the episode is terminated

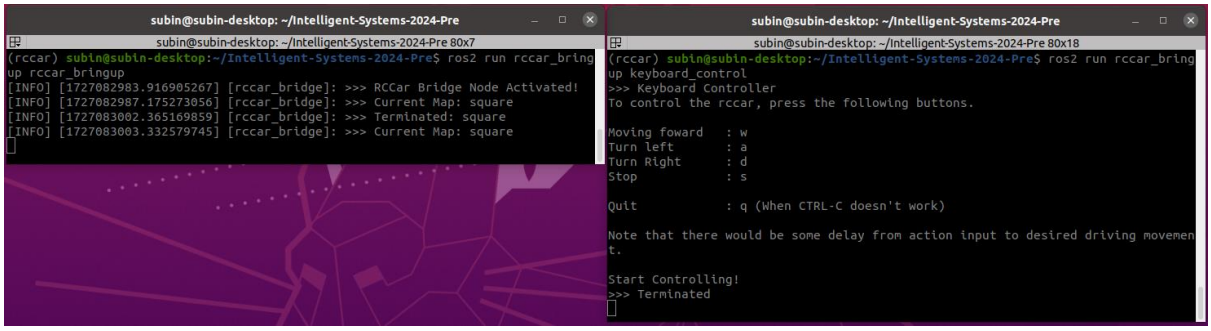

Figure 2. Example of the screenshots

Upload these on eTL in single compressed file named '**IS_{StudentNumber}_Pre_Project2.zip**'.

Due to: 2024.10.07 23:59 KST