

Project 1

Mu-Ruei Tseng

September 13, 2024

1 Task 3: Explore

While *turtlesim* is still running, use the `ros2` command line tool to explore your ROS installation.

1. Find a list of all of the topics in the system.
2. Choose one of these topics to use for the following steps.
3. Find the name of your topic's data type. Find the number of publishers and the number of subscribers.
4. Find the details of your topic's data type. That is, find a list of fields included in that message type.
5. Find the total number of interfaces in your ROS installation, including message types, service types, and action types.
6. Find the number of packages in your ROS installation that start with the letter 's'.

Paste the commands you used and their outputs into your report.

1.1 List of all the topics in the system

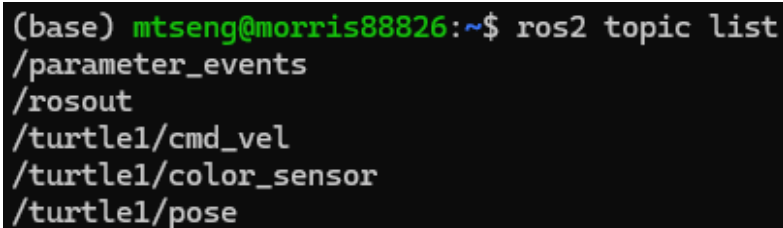
Run the *turtlesim* with

```
$ ros2 run turtlesim turtlesim_node
```

While *turtlesim* is still running, use the following command to list all the topics in the system:

```
$ ros2 topic list
```

Terminal output:



```
(base) mtseng@morris88826:~$ ros2 topic list
/parameter_events
/rosout
/turtle1/cmd_vel
/turtle1/color_sensor
/turtle1/pose
```

I choose */turtle1/pose* to use for the following steps.

1.2 Find the name of your topic's data type. Find the number of publishers and the number of subscribers

To find the name of your topic's data type and the number of publishers and subscribers, use the following command:

```
$ ros2 topic info /turtle1/pose
```

Terminal output:

```
(base) mtseng@morris88826:~$ ros2 topic info /turtle1/pose
Type: turtlesim/msg/Pose
Publisher count: 1
Subscription count: 0
```

The information is as follows:

- Data type: turtlesim/msg/Pose
- Number of publisher: 1
- Number of subscribers: 0

1.3 Find the details of your topic's data type. That is, find a list of fields included in that message type

To find the details of the selected topic's data type (*turtle1/pose*), use the following command:

```
$ ros2 interface show turtlesim/msg/Pose
```

Terminal output:

```
(base) mtseng@morris88826:~$ ros2 interface show turtlesim/msg/Pose
float32 x
float32 y
float32 theta

float32 linear_velocity
float32 angular_velocity
```

The list of fields and the corresponding message type:

- x: float 32
- y: float 32
- theta: float 32
- linear_velocity: float 32
- angular_velocity: float 32

1.4 Find the total number of interfaces in your ROS installation

We can use the following command to list all the interfaces in ROS, including message types, service types, and action types.

```
$ ros2 interface list
```

Terminal output:

```
(base) mtseng@morris88826:~$ ros2 interface list
Messages:
  action_msgs/msg/GoalInfo
  action_msgs/msg/GoalStatus
  action_msgs/msg/GoalStatusArray
  actionlib_msgs/msg/GoalID
  actionlib_msgs/msg/GoalStatus
  actionlib_msgs/msg/GoalStatusArray
  builtin_interfaces/msg/Duration
  builtin_interfaces/msg/Time
  diagnostic_msgs/msg/DiagnosticArray
  diagnostic_msgs/msg/DiagnosticStatus
```

To get the total number within each category, we can use piping to filter and count interfaces. For example:

```
$ ros2 interface list | grep "{type}/" | wc -l
```

Terminal output:

```
(base) mtseng@morris88826:~$ ros2 interface list | grep "msg/" | wc -l
193
(base) mtseng@morris88826:~$ ros2 interface list | grep "srv/" | wc -l
52
(base) mtseng@morris88826:~$ ros2 interface list | grep "action/" | wc -l
4
```

From the output, we can observe the total number of interfaces in each category:

- Number of message types: 193
- Number of service types: 52
- Number of action types: 4

The total number of interfaces on my ROS installation is 249.

1.5 Find the number of packages in your ROS installation that start with the letter 's'

To find the list of packages in the ROS installation that start with the letter 's', we can use the following command:

```
$ ros2 pkg list | grep ^s
```

See the result below: Similar to the previous task, we can use piping to get the number of packages in

```
(base) mtseng@morris88826:~$ ros2 pkg list | grep ^s
sdl2_vendor
sensor_msgs
sensor_msgs_py
shape_msgs
shared_queues_vendor
spdlog_vendor
sqlite3_vendor
sros2
sros2_cmake
statistics_msgs
std_msgs
std_srvs
stereo_msgs
```

ROS that start with the letter 's'.

```
$ ros2 pkg list | grep ^s | wc -l
```

Terminal output:

```
(base) mtseng@morris88826:~$ ros2 pkg list | grep ^s | wc -l
13
```

The number of packages in my ROS installation that start with the letter 's' is 13.

2 Task 4: Implement

Write one program, using Python or another ROS-supported language of your choice, that does these two things at the same time:

- Publish messages on the topic `/turtle1/cmd_vel` that tell the turtle to move slowly forward with a slight turn to the left. Your program should publish the same message approximately once per second. If you are doing this correctly, the turtle should move smoothly in a circular pattern.
- Subscribe to message on the topic `/turtle1/color_sensor`. When a message published on that topic arrives, your program should print the message to the console.

2.1 Create a ROS workspace and a ROS package within that workspace

Some installation:

```
$ echo source /opt/ros/humble/setup.bash >> ~/.bashrc
$ sudo apt install python3-colcon-common-extensions
$ echo source /usr/share/colcon_argcomplete/hook/colcon-argcomplete.bash >> ~/.bashrc
$ source ~/.bashrc
```

Within a directory (ex: *ros_workspace*), run:

```
$ mkdir src # where to write the ROS nodes
$ colcon build
$ source ./install/setup.bash
```

To create our package, use:

```
$ cd src
$ ros2 pkg create turtle_circle_controller --build-type ament_python --dependencies rclpy
```

Here, I created a custom package called **turtle_circle_controller**. Since I am using Python to write the programming to control ROS, I chose **ament_python** as the build type. I also added the **rclpy** dependencies for the Python library.

After creating the package, we can go back to the parent directory and build the package:

```
$ cd ..
$ colcon build
```

2.2 Create a ROS2 node with Python

Now we can start writing the code for the package inside `src/turtle_circle_controller/turtle_circle_controller.py`.

Problems when executing the python code

1. The system raised the following error:

```
ModuleNotFoundError: No module named 'rclpy._rclpy_pybind11'
The C extension
→ '/opt/ros/humble/lib/python3.10/site-packages/_rclpy_pybind11.cpython-312-x86_64-linux-gnu.so'
→ isn't present on the system.
```

This is because the rclpy library is only compatible with the python version 3.10. Therefore, I create a new conda environment with:

```
$ conda create -n ros python=3.10
$ conda activate ros
```

2. After that it produces the other error of:

```
ImportError: {miniconda_path}/envs/ros/bin/./lib/libstdc++.so.6: version
→ GLIBCXX_3.4.30 not found (required by
→ /opt/ros/humble/local/lib/python3.10/dist-packages/rclpy/
_rclpy_pybind11.cpython-310-x86_64-linux-gnu.so)
```

This is because the version of libstdc++.so.6 in my Miniconda environment does not contain the required GLIBCXX_3.4.30. I solved this by adding the following line into the `.bashrc`.

```
export LD_PRELOAD=/usr/lib/x86_64-linux-gnu/libstdc++.so.6
```

After finishing writing the code, add the following line in the `entry_points` part of the `setup.py` file.

```
1 entry_points={
2     'console_scripts': [
3         'circular_node = turtle_circle_controller.circular_node:main',
4     ],
5 }
```

Next, go back to the `ros_workspace` directory and build the package:

```
$ colcon build
$ source ./install/setup.bash
```

To prevent rebuilding it every time, use:

```
$ colcon build --symlink-install
```

Make sure to source the setup every time you open a new terminal.

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

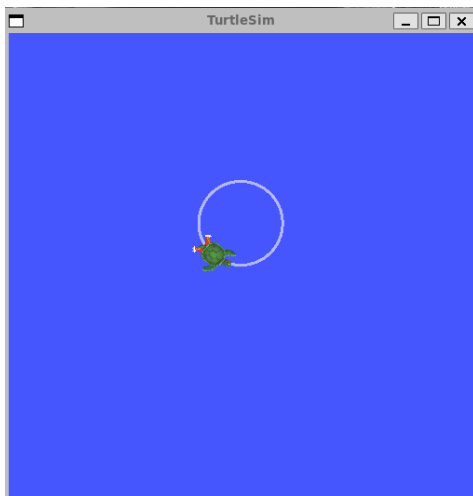
After all the edits and builds, we can use this to run our program:

```
$ ros2 run turtle_circle_controller circular_node
```

2.3 Result

The data type for the topic `/turtle1/cmd_vel` is `geometry_msgs/msg/Twist`, which contains 2 fields: **linear** and **angular**. I set `linear.x = 1.0` to make the turtle move forward with a linear velocity of 1.0 units per second, and I set `angular.z = 1.0` to make the turtle rotate counterclockwise with an angular velocity of 1.0 radians per second. The figure below shows the circle drawing from the program.

The data type for the topic `/turtle1/color_sensor` is `turtlesim/msg/Color`, which is the message published from `turtlesim`.



(a) TurtleSim Output

```
(ros) ntseong@hor-i58826:/mnt/c/Users/ntseong/Desktop/TAMU/CSCI-752-ROBOTICS-AND-SPATIAL-INTRO
[1726236784.634933684] [circular_node]: Circular node has been started
[INFO] [1726236784.719654495] [circular_node]: Color sensor callback: R=179, G=184, B=255
[INFO] [1726236784.77898769] [circular_node]: Color sensor callback: R=179, G=184, B=255
```

(b) Terminal Output

Figure 1: When executing `circular_node` from my `turtle_circle_controller` package

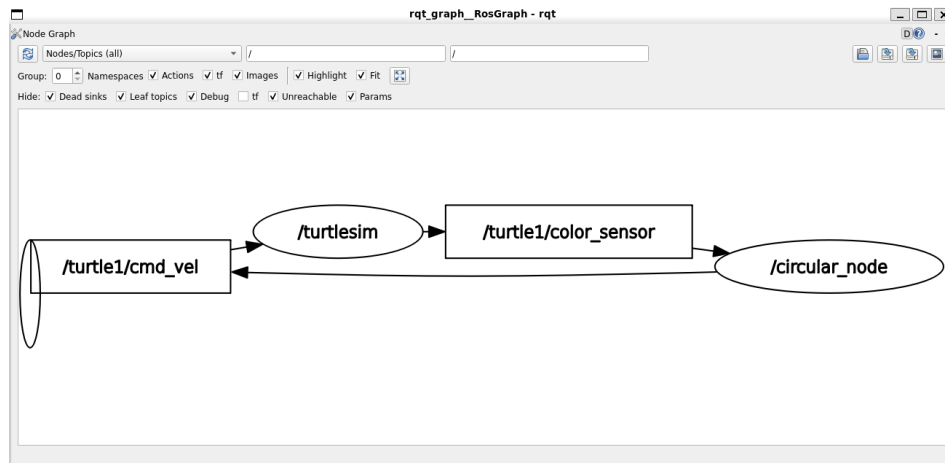


Figure 2: Node graph of the ROS system showing the `/turtle1/cmd_vel` and `/turtle1/color_sensor` topics with connections between the `turtlesim` node and `circular_node`.

3 Task 5: Install

Download the *sim* package from this link and extract it into the *src* subdirectory in your ROS workspace.

Build your workspace with this new package.

```
$ colcon build
$ source ./install/setup.bash
```

```

(base) mtseng@morris88826:/mnt/c/Users/mtseng/Desktop/TAMU/CSCE-752-ROBOTICS-AND-SPATIAL-INTELLIGENCE/project1/ros_workspace$ colcon build
Starting >>> sim
Starting >>> turtle_circle_controller
Finished <<< turtle_circle_controller [6.19s]
Finished <<< sim [6.94s]
Summary: 2 packages finished [7.81s]
```

To verify the installation, run:

```
$ ros2 pkg prefix sim
```

Terminal output:

```

(ros) mtseng@morris88826:/mnt/c/Users/mtseng/Desktop/TAMU/CSCE-752-ROBOTICS-AND-SPATIAL-INTELLIGENCE/project1/ros_workspace$ ros2 pkg prefix sim
/mnt/c/Users/mtseng/Desktop/TAMU/CSCE-752-ROBOTICS-AND-SPATIAL-INTELLIGENCE/project1/ros_workspace/install/sim
```

4 Task 6: Visualize

To move the robot in the simulation, I created a new node called 'control_node' similar to Task 4 and subscribed to the /cmd_vel topic. This allows me to control the robot in a circular motion. Here are the results:

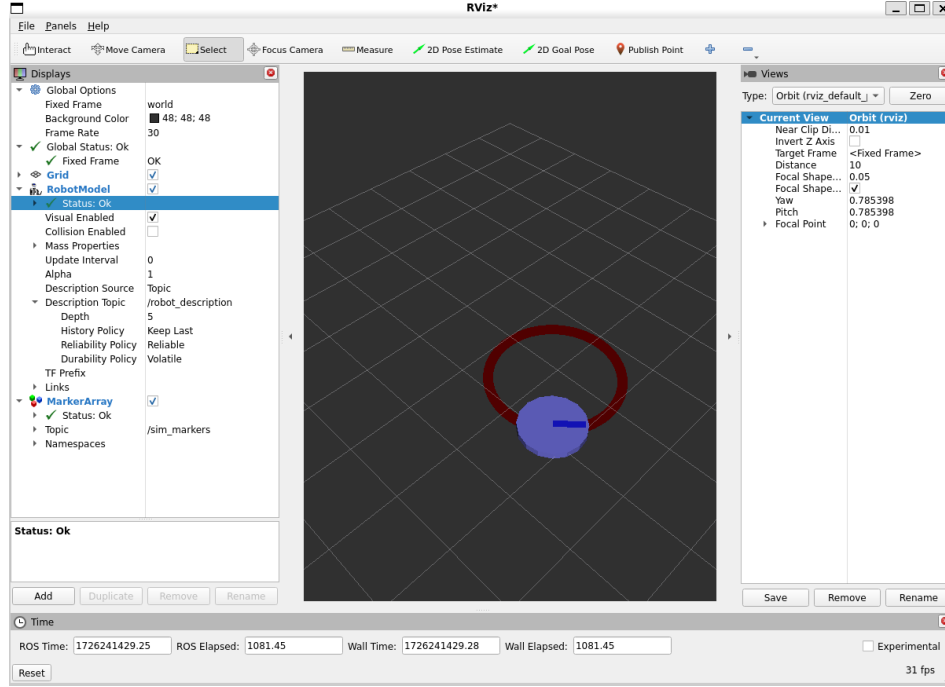


Figure 3: Screenshot of the sim1 simulation in rviz2, controlled by the control_node publisher.

Here I also include the graph while running the simulation.

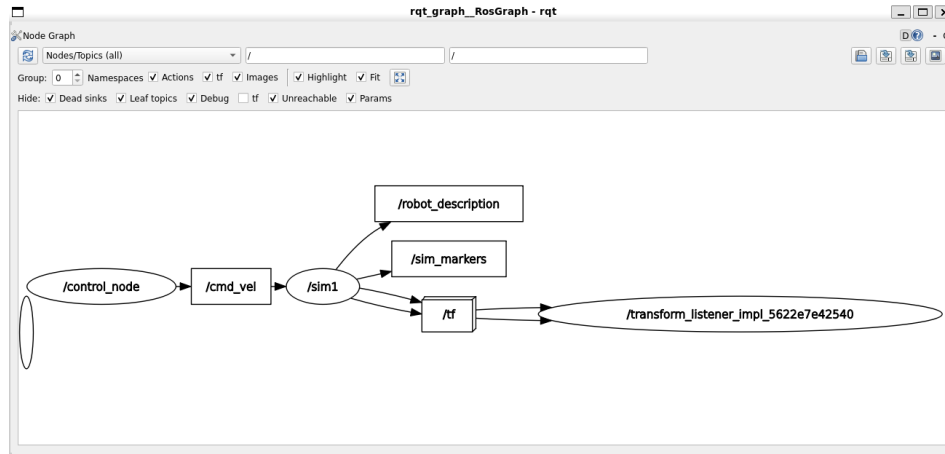


Figure 4: Node graph for the running sim1 simulation.