Tinkerers' Lab
IIT Hyderabad

Mechatronics Summer School 2025

Prepared by:

The Senior Mechatronics Team

Jaideep Nirmal AJ, *Head*
Abhijith Raj, *Senior Core*
Satyavada Sri Harini, *Senior Core*
Taha Mohiuddin, *Senior Core*

Tinkerers' Lab, IIT Hyderabad
Between A, B, G, H Hostel Block, IIT Hyderabad
IITH Main Road, Near NH-65, Sangareddy, Telangana 502285

Tinkerers' Lab
IIT Hyderabad

.

Module 2
ROS 102: Turtlesim

Contents

This is a relatively smaller module to give you some more time to catch up with the pace.

1 What is Turtlesim?

Turtlesim is a lightweight 2D simulator. It is often used as a "toy-problem" to demonstrate the capabilities and working of ROS. It provides a simple 2D world where numerous turtles can be spawned and be played around with.
Historically, the LOGO programming language in the 1960s introduced "turtle graphics," allowing students to program a virtual or physical turtle to draw shapes and patterns. This made abstract programming concepts more tangible and interactive, and LOGO's turtle became a widely recognized tool for teaching both programming and robotics fundamentals. Turtlesim also follows the same philosophy.

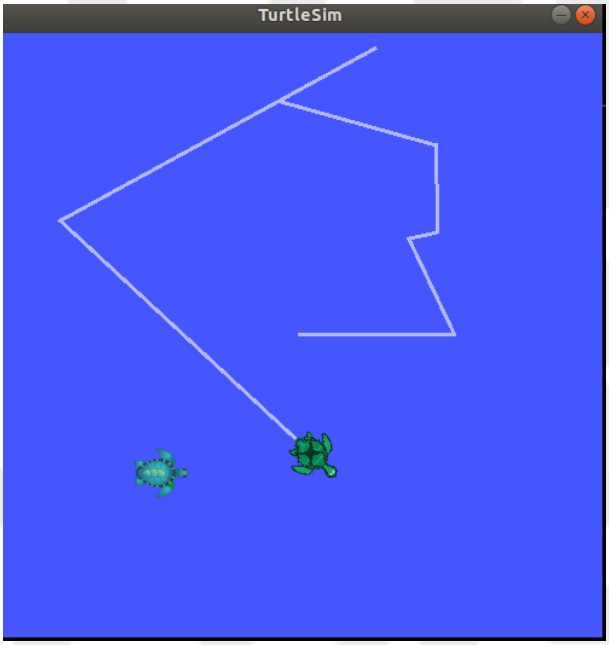

Figure 1: Turtlesim

2 Launching Turtlesim

Turtlesim should come pre-installed with ROS. Start turtlesime using this command:

```
Terminal                                                    - □ X
$ ros2 run turtlesim turtlesim_node
INFO [turtlesim]: Starting turtlesim with node name /turtlesim
INFO [turtlesim]: Spawning turtle [turtle1] at x=[5.544445], y=[5.544445],
theta=[0.000000]
```

You should get the following output along with this screen. If you get any error

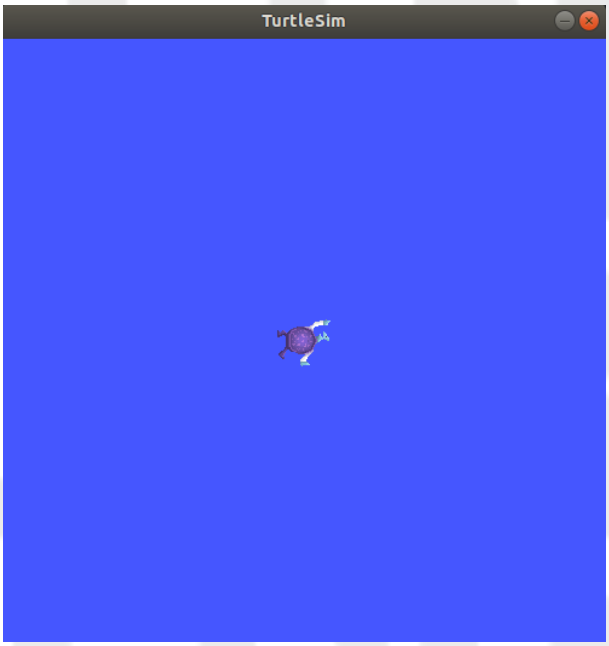

Figure 2: Default screen of Turtlesim

indicating turtlesim might not be installed, use the first two commands to install and the last command to check if Turtlesim is installed properly (The last 4 lines are the output you should be getting).

```
Terminal                                                    - □ X
$ sudo apt update
$ sudo apt install ros-humble-turtlesim
$ ros2 pkg executables turtlesim
turtlesim draw_square
turtlesim mimic
turtlesim turtle_teleop_key
turtlesim turtlesim_node
```

3 Exploring Turtlesim

Let's look at what all topics turtlesim has, by default:

```
Terminal                                                    - □ X
$ ros2 topic list
/parameter_events
/rosout
/turtle1/cmd_vel
/turtle1/color_sensor
$ ros2 node list
/turtlesim
```

We infer here that there is only one node and there are 4 topics that are available. The `cmd_vel` topic is the most important one.

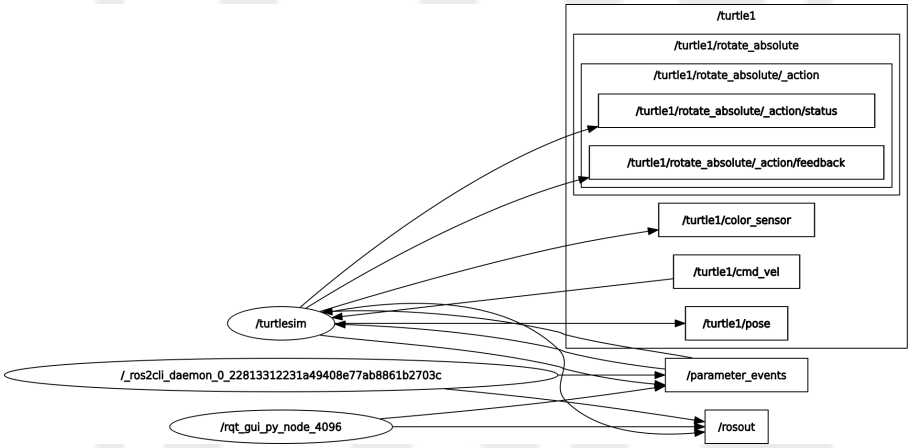

Figure 3: `rqt_graph`

3.1 `/pose` Topic

`/pose` is short for position and as the name says, it publishes information about the position as well as the velocity of the turtle. Lets see what is being published.

```
Terminal                                                    - □ X
$ ros2 topic echo /turtle1/pose -
x: 5.544444561004639 y: 5.544444561004639
theta: 0.0
linear_velocity: 0.0
angular_velocity: 0.0
-
```

When we want to describe the position of something, the most natural and intuitive way is to express it in the cartesian coordinates. Note that the origin is at the bottom left corner of the screen.

Now, we want to describe the velocity of something. What's the most intuitive way possible? Do you think cartesian approach is *practical*? Here, think in terms of a robot that is moving around in a space. Say it can easily keep track of the origin and tell you where it is in a given space. Now you want it to move from A to B. Wouldn't it be easy to ask it go *towards* B instead of telling it to travel for X to along $\hat{i}$ and Y along $\hat{j}$.
Also, think- if you specify the robot with a certain x and y velocity, will every robot be able to achive that? Most robots have fixed limbs or wheels. They can either move ahead or turn around at the same point- never both.

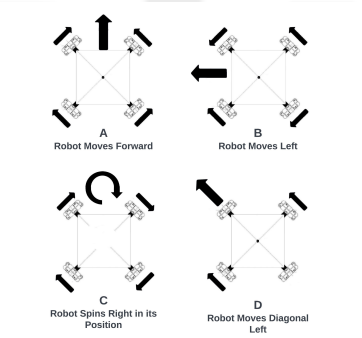

Figure 4: Most mobile robots move only one direction at a time

This is where the radial coordinates come into picture. We specify the velocity of the robot with two variables- the speed, the compass heading. The compass headings (0 to 360) is universally defined and say the robot can easily track the direction it is facing. The variables we just defined are very simple to measure. As mentioned, most robot can move in only one direction at once- measure it! It's easy to read out of a compass (usually out of an IMU sensor)- measure it! As simple as that. This also happens to be the standard working rules for speed reporting in aircraft and ships!
These two variables are necessarily the radial velocity and the and the angle the position vector makes w.r.t the origin. Once we have these two variables, with some basic vector

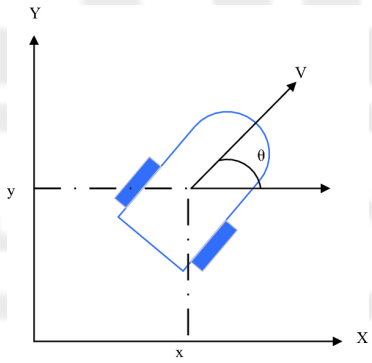

Figure 5: Velocity Description

algebra, we can translate this to velocity along x and y directions. If it is intuitive to think in cartesian coordinates, it is easier for the robot to read and execute in radial coordinates. Therefore, most computation for path algorithms may happen in cartesian coordinates after a simple conversion.

3.2 `/cmd_vel` Topic

/cmd_vel topic stands for command velocity. As the name suggests this topic is responsible for sending velocity commands to your object (here the turtle). Now for an example if you are using your keyboard to move around the turtle a node called /teleop_turtle publishes this data to /turtle1/cmd_vel to which the node /turtlesim is subscribed. The rqt graph for such an example is as follows: To visualize the data being send from the

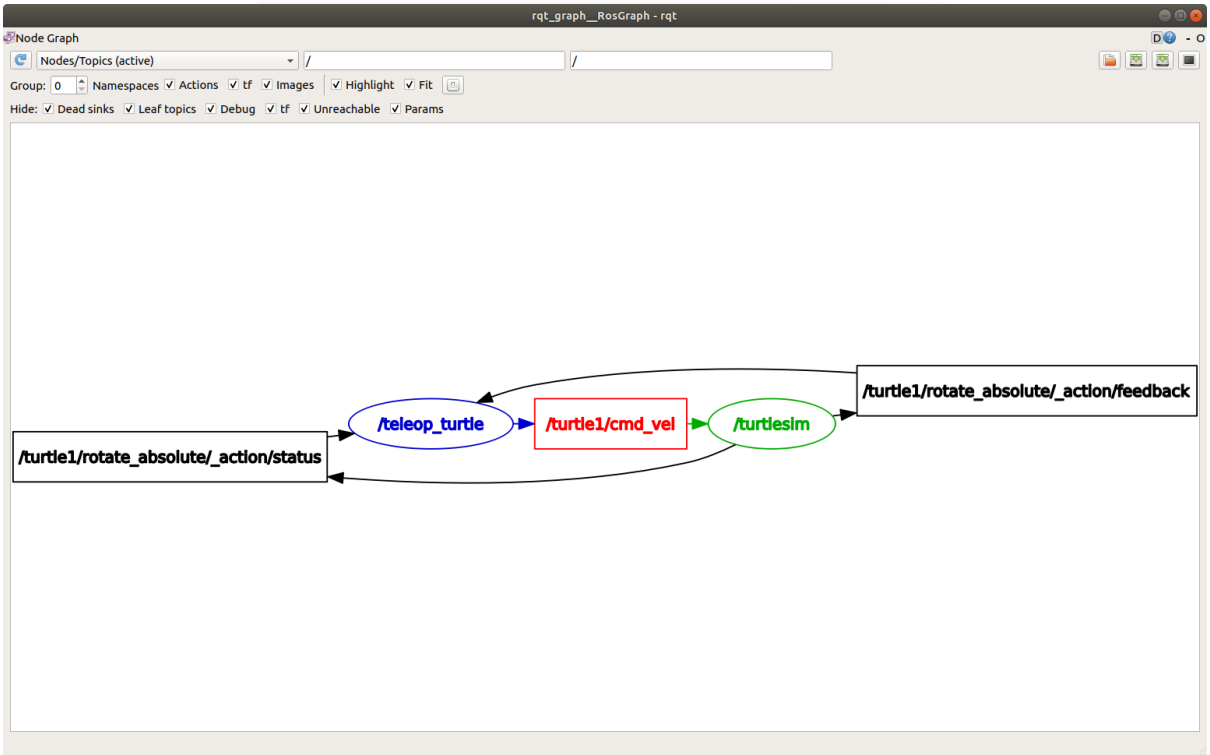

Figure 6: rqt_graph

teleop node to cmd_vel topic use:

```
Terminal
ros2 topic echo /turtle1/cmd_vel
```

in a new terminal. At first, this command won't return any data. That's because it's waiting for /teleop_turtle to publish something. Return to the terminal where turtle_teleop_key is running and use the arrows to move the turtle around. Watch the terminal where your echo is running at the same time, and you'll see velocity data being published for every movement you make:

```
Terminal
linear:
x: 2.0
y: 0.0
z: 0.0
angular:
x: 0.0
y: 0.0
z: 0.0
- - -
```

The velocity in cmd_vel is divided into two types 'linear' and 'angular'; the linear velocity notation is the same as in /pose topic but unlike in the /pose topic the angular velocity is described in components about each axis. Having understood the role of the /cmd_vel topic and its syntax lets use it to send the velocity commands directly and draw a circle:

```
Terminal
ros2 topic pub /turtle1/cmd_vel geometry_msgs/msg/Twist "{linear: {x: 2.0, y: 0.0,
z: 0.0}, angular: {x: 0.0, y: 0.0, z: 1.8}}"
```

The result should look something like this:

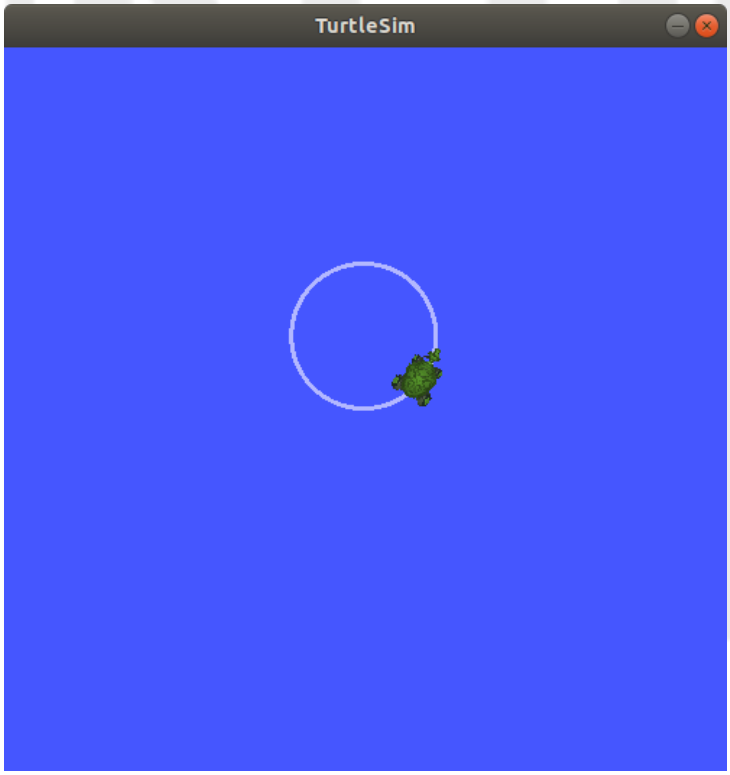

Figure 7: circle

3.3 `teleop_keyboard`

turtle_teleop_key is a keyboard teleoperation node for controlling the turtle in `turtlesim`. It reads keypresses and publishes velocity commands on the `/turtle1/cmd_vel` topic (or a remapped topic).

Run Command

To start teleoperation with default topic (`/turtle1/cmd_vel`):

```
Terminal                                                    - □ X
ros2 run turtlesim turtle_teleop_key
```

Publishing to a Different Topic (Remapping)

To control a turtle in a different namespace or topic:

```
Terminal                                                    - □ X
ros2 run turtlesim turtle_teleop_key --ros-args -r /cmd_vel:=/robot1/cmd_vel
```

This is useful when launching multiple turtlesim instances, each in its own namespace (The topic of Namespace is covered at the end of the document).

Usage

This node maps specific keys to movement commands. Pressing a key sends a `geometry_msgs/Twist` message to the velocity topic.

Key Mappings

```
Arrow Up    : Move forward
Arrow Down  : Move backward
Arrow Left  : Turn left
Arrow Right : Turn right
```

Other Controls

```
CTRL-C : Quit the node
```

NOTES

- By default, the node publishes to `/turtle1/cmd_vel`.
- Use remapping if controlling a turtle in a different namespace (e.g., `/turtle2/cmd_vel`).
- Only one turtle can be controlled per instance of `turtle_teleop_key`.

4 Turtlesim Services

The `turtlesim` package in ROS 2 provides a simple and visual way to understand the core concepts of the ROS 2 communication system, including services. A service in ROS 2 is a synchronous request-response mechanism, and `turtlesim` exposes several useful services for interacting with the turtles.

1. `/spawn`

This service is used to create a new turtle at a specified location and orientation on the screen. It accepts the `turtlesim/srv/Spawn` service type.

- **Request fields:** `x`, `y`, `theta`, `name`
- **Response:** Name of the newly spawned turtle

```
Spawn a New Turtle                                                    - □ X
ros2 service call /spawn turtlesim/srv/Spawn "x: 2.0, y: 3.0, theta: 0.0, name:
'turtle2'"
```

2. `/kill`

This service is used to remove a turtle from the simulation. It accepts the `turtlesim/srv/Kill` service type.

- **Request fields:** `name` of the turtle to kill
- **Response:** None (empty response)

```
Kill a Turtle                                                         - □ X
ros2 service call /kill turtlesim/srv/Kill "name: 'turtle2'"
```

3. `/reset`

This service resets the turtlesim window to its initial state with only `turtle1` in the center. It uses the `std_srvs/srv/Empty` service type.

```
Reset Turtlesim                                                       - □ X
ros2 service call /reset std_srvs/srv/Empty ""
```

4. `/clear`

Clears the drawing path made by the turtles without resetting their positions. Also uses the `std_srvs/srv/Empty` service type.

```
Clear Drawing                                                         - □ X
ros2 service call /clear std_srvs/srv/Empty ""
```

5. `/teleport_absolute` and `/teleport_relative`

These services are used to instantly move a turtle to a new position:

- `/teleport_absolute` moves the turtle to a specific coordinate and orientation.
- `/teleport_relative` moves it a relative distance forward and angle.

Teleport a Turtle to an Absolute Position

```
ros2 service call /turtle1/teleport_absolute turtlesim/srv/TeleportAbsolute "x: 5.0, y: 5.0, theta: 1.57"
```

Teleport a Turtle Relatively

```
ros2 service call /turtle1/teleport_relative turtlesim/srv/TeleportRelative "linear: 2.0, angular: 1.57"
```

Usefulness in Learning

These services are especially helpful in learning ROS 2 concepts because they allow for interactive experimentation. By using these commands, users can understand service requests, visualize synchronous interactions, and observe real-time effects in the simulation.

5 Namespaces

In ROS 2, a **namespace** is a way to group related resources—such as nodes, topics, services, actions, and parameters—under a hierarchical name. Namespaces help manage complexity, prevent naming conflicts, and enable modular system design.

Why Namespaces Matter

- **Isolation**: You can run multiple instances of the same node without topic clashes.
- **Scalability**: Useful in multi-robot systems (e.g., `/robot1/`, `/robot2/`).
- **Reusability**: Launch the same code under different namespaces for different parts of a system.

Example Imagine two robots publishing camera images:

```
/robot1/camera/image_raw
/robot2/camera/image_raw
```

Here, `robot1` and `robot2` are namespaces, keeping the data streams separate.
Setting Namespaces in Launch Files:

```
from launch_ros.actions import Node

Node(
    package='my_package',
    executable='my_node',
    namespace='robot1'
)
```

Setting Namespaces from the Command Line:

```
Terminal                                                    - □ X
ros2 run my_package my_node --ros-args -r __ns:=/robot1
```

Example:

```
Terminal                                                    - □ X
ros2 run turtlesim turtlesim_node --ros-args -r __ns:=/turtle1
```

You can check the topics by typing in 'ros2 topic list' in the terminal:

```
Terminal                                                    - □ X
username@LAPTOP-RJMPLMAJ:~$ ros2 topic list
/parameter_events
/rosout
/turtle1/turtle1/cmd_vel
/turtle1/turtle1/color_sensor
/turtle1/turtle1/pose
```

Remapping

Remapping allows you to change the names of topics, services, or entire namespaces at runtime without modifying the node's source code.

Command Line for Remapping:

```
Terminal
ros2 run my_package my_node --ros-args -r original_name:=new_name
```

Remap a namespace:

```
Terminal
ros2 run my_package my_node --ros-args -r __ns:=/robot1
```

Remap a topic:

```
Terminal
ros2 run my_package my_node --ros-args -r /cmd_vel:=/robot1/cmd_vel
```

Remapping in Launch Files:

```
Node(
    package='my_package',
    executable='my_node',
    remappings=[
        ('/cmd_vel', '/robot1/cmd_vel')
    ]
)
```

Any reference to the absolute topic /cmd_vel in the node will now use /robot1/cmd_vel.

The remapping ignores any namespace because /cmd_vel is absolute.

6 DIY Tasks

For both the tasks, a launch file is a must.

1. **Turtle Paints a Rainbow!**
   Our turtle wants to paint a rainbow. Use the teleop to control the turtle, while the turtle changes its trace color with its position, as per the image given below (The exact numbers are not being given here, you can choose the radii yourself but be sure to include all 7 colors + white)

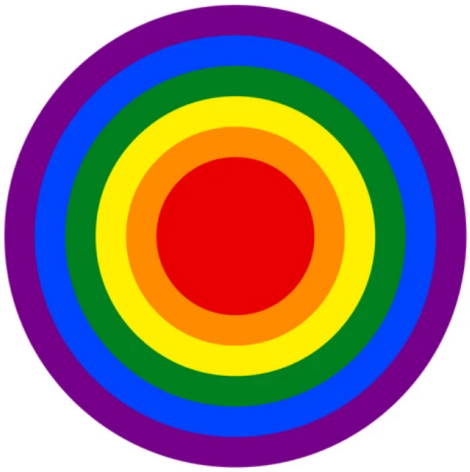

Figure 8: Trace color required at different positions

2. **Catch Me If You Can!**
   Spawn 4 turtles at random positions into the world. One of them is a thief and the rest are cops trying to chase the former. Give each turtle an interesting and unique name, passed in from command line. The thief turtle moves randomly. All the police turtles need to move towards the thief turtle. As soon as one of the cop turtles catch the thief turtle, the thief turtle teleports itself to a random location in the world. The cop that caught the thief gets a point. Continue the game till a cop turtle wins 5 points.