Lab 2 Report

Robotics Integration Group Project I

Yuwei ZHAO (23020036096)
Group #31 2025-11-12

Abstract

This report presents the setup and experimentation process of Lab 2. The primary objective of this lab is to install and configure the Robot Operating System (*ROS*) environment, establish a working ROS workspace and gain hands-on experience with ROS nodes, topics, and message communication. Through a series of practical exercises, the lab introduces core *ROS* concepts such as node creation, publisher–subscriber mechanisms, and visualization using *rqt_graph* and *roscore*. The experiment provides a foundational understanding of how *ROS* enables modular and distributed robotics software development.

See Resources on git@github.com:RamessesN/Robotics_MIT.

1 Introduction

This laboratory session focuses on the installation, configuration and initial exploration of the Robot Operating System (*ROS*), which serves as the middleware framework for subsequent robotics development. Before implementing perception, planning or control modules, it is essential to understand the *ROS* architecture and its communication mechanisms. The experiment involves setting up the *ROS* environment, creating and managing a catkin workspace, and developing simple publisher and subscriber nodes to exchange data through topics.

2 Procedure

2.1 Part I

2.1.1 Objective

The objective of this section is to set up the *ROS* environment on the Ubuntu 24.04 operating system, including the installation of necessary packages and configuration of a functional workspace.

2.1.2 Methodology

The original method for installing *ROS Noetic* was as follows:

1. Add the *ROS* package sources and install *ROS*:

```
sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu $(lsb_release -sc)
↪  main" > /etc/apt/sources.list.d/ros-latest.list'
curl -s https://raw.githubusercontent.com/ros/rosdistro/master/ros.asc | sudo
↪  apt-key add -
sudo apt update
sudo apt install ros-noetic-desktop-full
```

2. Configure the *ROS* environment:

```
echo "source /opt/ros/noetic/setup.bash" >> ~/.bashrc
source ~/.bashrc
sudo apt install python3-rosdep python3-rosinstall python3-rosinstall-generator
↪  python3-vcstool build-essential python3-catkin-tools python-is-python3
```

3. Initialize *rosdep*:

```
sudo rosdep init
rosdep update
```

However, the above method is not fully compatible with *Ubuntu 24.04*. Therefore, an alternative approach using the *shrike* repository was adopted, which proved successful:

1. Clone the *shrike* repository from GitHub:

```
git clone git@github.com:Minoic-Intelligence/shrike.git
```

2. Follow the instructions provided in the *shrike* repository to complete the installation:

```
./scripts/install_ubuntu24.sh
./src/catkin/bin/catkin_make_isolated --install -DCMAKE_BUILD_TYPE=Release
source ./install_isolated/setup.bash
```

2.1.3 Observations

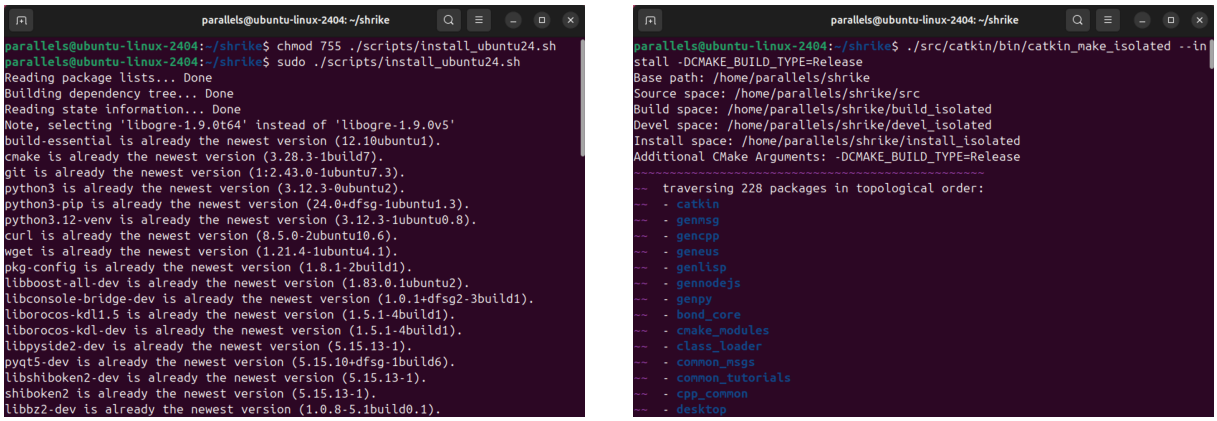


Figure 1: Installation and configuration of *ROS Noetic* on *Ubuntu 24.04*

2.1.4 Discussion

Installing *ROS Noetic* on *Ubuntu 24.04* presents challenges due to compatibility issues with the standard installation procedure. By leveraging the *shrike* repository, which provides customized scripts tailored for this specific operating system version, the installation was completed successfully. This process underscores the importance of adapting installation methods to the target environment and utilizing community-maintained resources when standard approaches fail.

2.2 Part II

2.2.1 Objective

The goal of this section is to deepen understanding of the *ROS* communication infrastructure by implementing publisher and subscriber nodes, exploring topic-based message exchange, and visualizing the inter-node topology. This lays the foundation for modular, event-driven robotic behavior by:

1. Implementing *ROS* nodes that publish and subscribe to custom messages.
2. Demonstrating data flow between nodes via topics and verifying correct message delivery.
3. Utilizing diagnostic and visualization tools (such as `rqt_graph`) to inspect the connectivity of *ROS* nodes and topics.
4. Ensuring that the workspace is properly configured to build, launch, and manage multiple nodes within a *ROS* ecosystem.

2.2.2 Methodology

- ***ROS* Master**
  Start the *ROS* master with:

  ```
  roscore
  ```

- ***ROS* Nodes**

  1. Launch the *turtlesim* node in a new terminal:

     ```
     rosrun turtlesim turtlesim_node
     ```

  2. Query the running nodes via:

     ```
     rosnode list
     ```

  3. Launch the *turtle_teleop_key* node to control the turtle using the keyboard:

     ```
     rosrun turtlesim turtle_teleop_key
     ```

- ***ROS* Topics**

  1. Visualize running nodes and topics with:

```
rosrun rqt_graph rqt_graph
```

2. Monitor the velocity commands sent to the turtle:

```
rostopic echo /turtle1/cmd_vel
```

3. Run the *C++* sample

```cpp
#include <ros/ros.h>

int main(int argc, char** argv) {
    ros::init(argc, argv, "example_node");
    ros::NodeHandle n;

    ros::Rate loop_rate(50);

    while (ros::ok()) {
        ros::spinOnce();
        loop_rate.sleep();
    }
    return 0;
}
```

To compile and run this code using *Shrike*, follow these steps:

```
cd ~/shrike/ros_ws/src
catkin_create_pkg ros_sample roscpp std_msgs
```

The resulting workspace structure is:

```
- ros_ws/
  - src/
    - ros_sample/
      - CMakeLists.txt
      - include/
      - package.xml
      - src/
        - ros_sample.cpp
```

Replace *example_node.cpp* with *ros_sample.cpp* and update *CMakeLists.txt* as follows:

```
cmake_minimum_required(VERSION 3.0.2)
project(ros_sample)

find_package(catkin REQUIRED COMPONENTS
  roscpp
  std_msgs
)

catkin_package()

include_directories(
  ${catkin_INCLUDE_DIRS}
)

add_executable(ros_sample_node src/ros_sample.cpp)
target_link_libraries(ros_sample_node ${catkin_LIBRARIES})
```

Build and source the workspace:

```
cd ~/shrike/ros_ws
catkin_make_isolated --install -DCMAKE_BUILD_TYPE=Release
source ~/shrike/ros_ws/devel_isolated/setup.bash
rosrun ros_sample ros_sample_node
```

Verify the node is running with *rosnode list*.

4. *TF* Tools
   - Launch the Turtle TF demo:
     `roslaunch turtle_tf turtle_tf_demo.launch`
     Note: On Ubuntu 24.04 with *Shrike*, older Noetic packages rely on Python 2, but only Python 3 is installed. To fix:
     `sudo ln -s /usr/bin/python3 /usr/bin/python`
   - Visualize TF tree using:
     `rosrun rqt_tf_tree rqt_tf_tree`
     This shows three frames: *world* (parent), *turtle1*, and *turtle2*.
   - Inspect transforms with:
     `rosrun tf tf_echo /turtle1 /turtle2`
   - Run RViz:
     `LIBGL_ALWAYS_SOFTWARE=1 rviz`
     Software rendering is required because RViz depends on OpenGL, which may not be fully supported in virtual environments.
5. Modify *view_frames* script for *Python 3.x* compatibility:

```
try:
    vstr = subprocess.Popen(args, stdout=subprocess.PIPE,
    ↪  stderr=subprocess.STDOUT).communicate()[0]
    vstr = vstr.decode('utf-8')
except OSError as ex:
    print("Warning: Could not execute `dot -V`.  Is graphviz installed?")
    sys.exit(-1)

v = distutils.version.StrictVersion('2.16')
r = re.compile(r".*version (\d+\.?\d*)")
print(vstr)
m = r.search(vstr)
```

2.2.3 Observations

- *ROS* Master

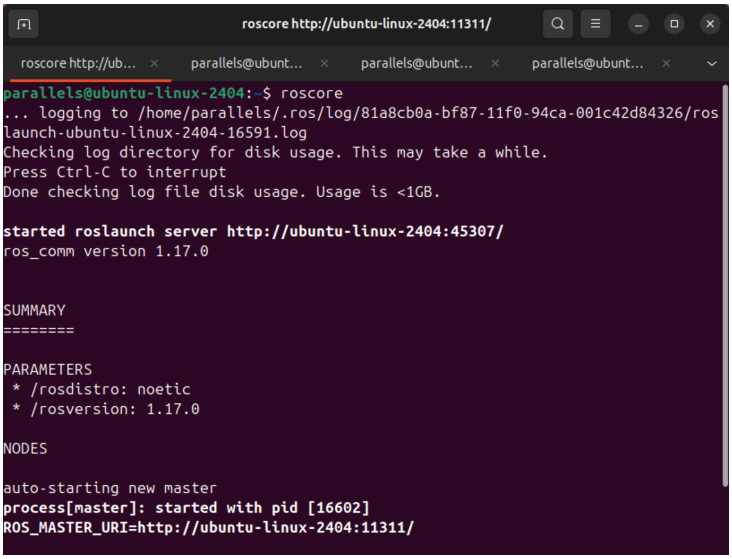
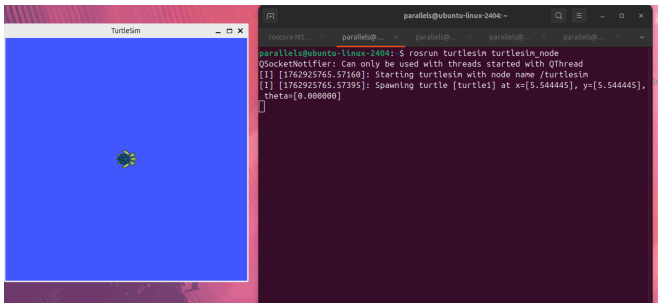
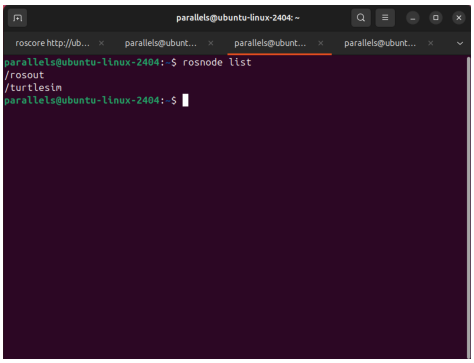


Figure 2: ROS Master status

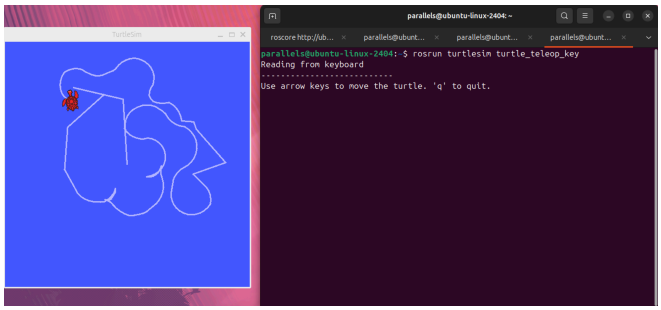
- *ROS* Nodes



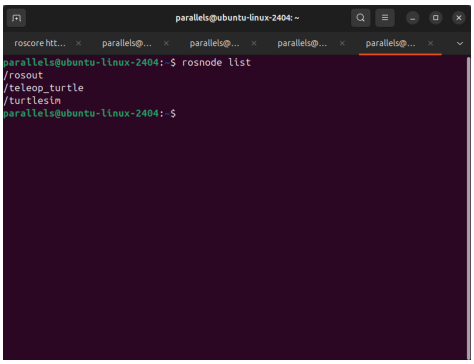
Turtlesim node running



rosnode list output



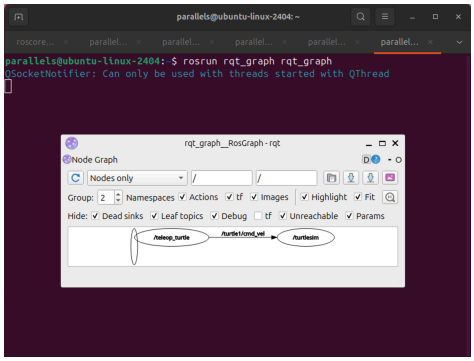
Turtle teleop node running



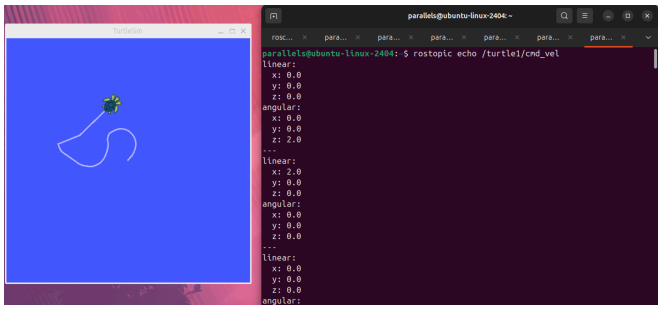
rosnode list output

Figure 3: ROS nodes status

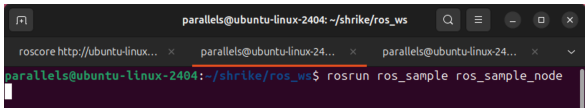
- *ROS* Topics



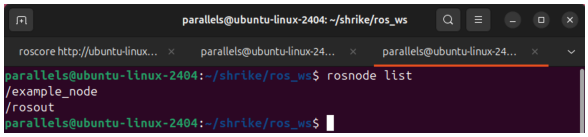
rqt_graph output



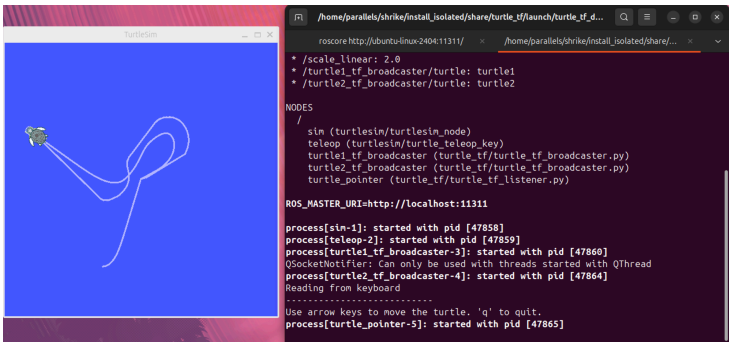
Velocity topic output



ROS node running

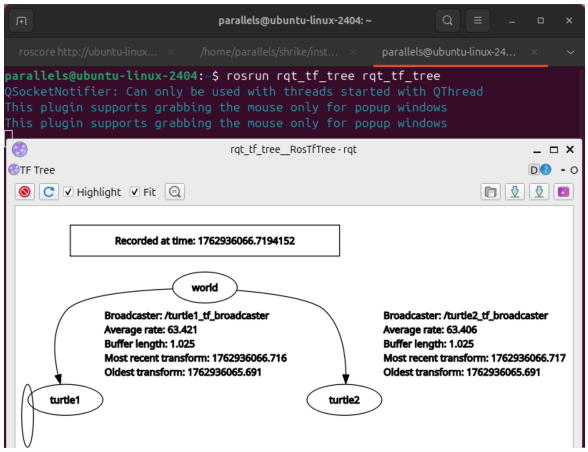


rosnode list output

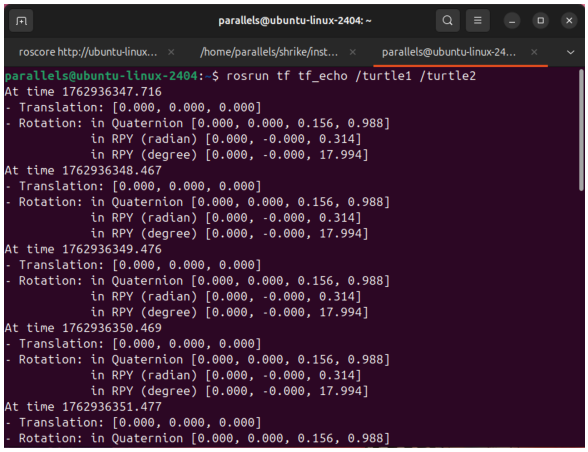


Roslaunch output

Figure 4: ROS topics observation



(a) rqt_tf_tree output



(b) tf_echo output

(b) TF tools observation

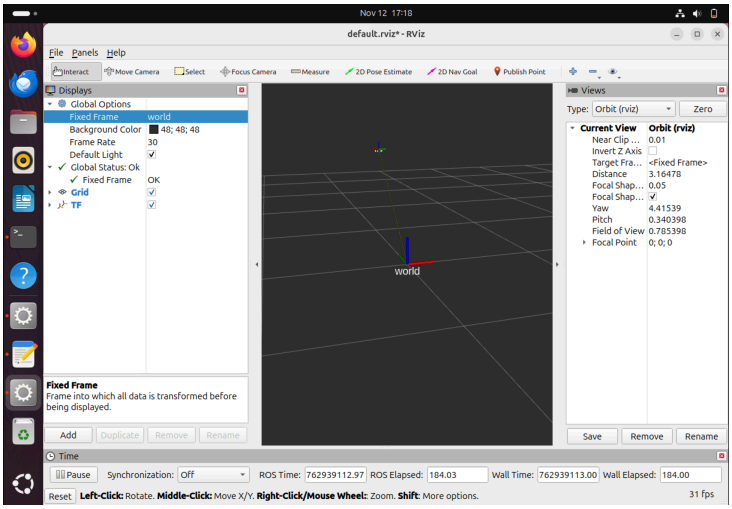


Figure 5: RViz output

2.2.4 Discussion

This section demonstrates the fundamental principles of communication and visualization within the *ROS* ecosystem. By implementing publisher and subscriber nodes and observing the resulting message flows through *rqt_graph* and *rostopic echo*, it becomes evident how *ROS* topics facilitate modular and decoupled inter-node communication. The ability to monitor live message data allows developers to verify correct behavior and quickly identify misconfigurations or runtime errors.

The use of *TF* frames further highlights the importance of coordinate transformations in robotic systems. The hierarchical relationship between *world*, *turtle1*, and *turtle2* frames emphasizes how spatial relationships are managed and broadcast across the *ROS* network. Tools such as *rqt_tf_tree* and *tf_echo* provide intuitive visualization and debugging capabilities, reinforcing the understanding of frame relationships and motion propagation.

Running RViz in a virtualized environment illustrated practical considerations in deploying *ROS* visualization tools. The necessity to force software rendering with `LIBGL_ALWAYS_SOFTWARE=1` underscores how hardware limitations or OpenGL compatibility issues can affect 3D visualization. This reinforces the broader lesson that deployment environments can significantly influence system behavior and must be accounted for during development and testing.

2.3 Part III

2.3.1 Objective

2.3.2 Methodology

2.3.3 Observations

2.3.4 Discussion

2.4 Part IV

2.4.1 Objective

2.4.2 Methodology

2.4.3 Observations

2.4.4 Discussion

3 Reflection and Analysis

4 Conclusion

5 Source Code