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

To set up the *ROS* environment on the *Ubuntu 24.04* operating system, including the installation of essential packages and the configuration of a functional Catkin workspace.

2.1.2 Methodology

The initial approach for installing *ROS Noetic* was as follows:

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

```
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 and install additional dependencies:

```
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* to enable dependency management:

```
sudo rosdep init
rosdep update
```

However, the above method is not fully compatible with *Ubuntu 24.04*. To address this, an alternative installation procedure using the *Shrike* repository was adopted, which provided a successful setup.

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. The resulting terminal output is shown in Figure 1.

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

2.1.3 Observations

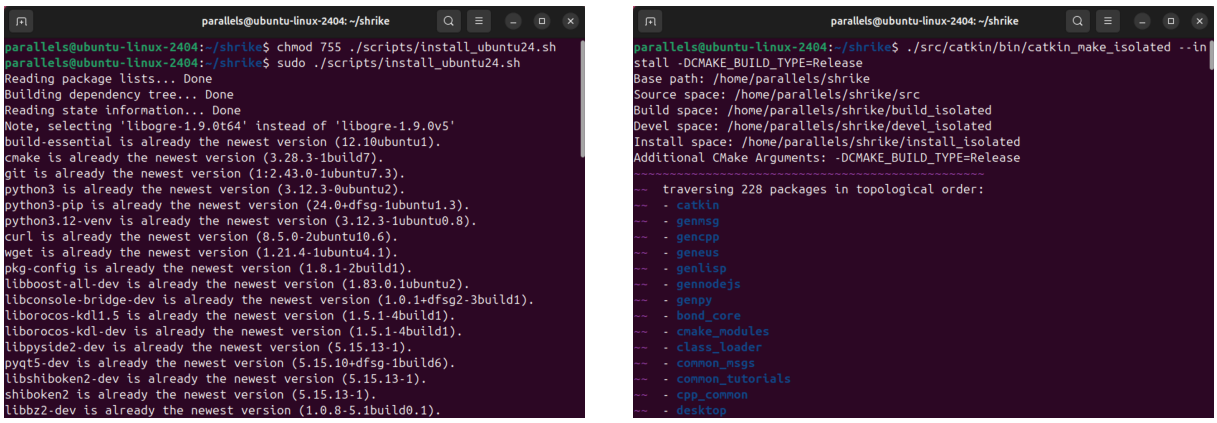


Figure 1: Installation of *ROS Noetic* on *Ubuntu 24.04*

2.1.4 Discussion

Installing *ROS Noetic* on *Ubuntu 24.04* presents several challenges due to incompatibilities between the standard Noetic packages and the latest system libraries. By utilizing the *Shrike* repository—which provides customized build scripts tailored for Ubuntu 24.04—the installation process was completed successfully. This outcome highlights the necessity of adapting traditional installation procedures to evolving system environments and demonstrates the value of community-maintained solutions when official support is unavailable.

2.2 Part II

2.2.1 Objective

To deepen the understanding of the *ROS* communication framework by implementing publisher and subscriber nodes, exploring topic-based message exchange, and visualizing the inter-node topology. This experiment establishes the foundation for modular, event-driven robotic behavior through:

1. Implementing *ROS* nodes that publish and subscribe to custom message types.
2. Demonstrating data flow between nodes via topics and verifying correct message delivery.
3. Utilizing diagnostic and visualization tools (such as `rqt_graph`) to examine node–topic connectivity.
4. Ensuring 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 using the following command. The resulting output is shown in Figure 2.

  ```
  roscore
  ```

- ***ROS* Nodes**

  1. Launch the *turtlesim* node in a new terminal:
     ```
     rosrun turtlesim turtlesim_node
     ```
  2. List active nodes using:
     ```
     rosnode list
     ```
  3. Launch the *turtle_teleop_key* node to control the turtle with keyboard input:
     ```
     rosrun turtlesim turtle_teleop_key
     ```
  4. The corresponding runtime output is illustrated in Figure 3.

- ***ROS* Topics**

1. Visualize the communication graph of running nodes and topics:

   ```
   rosrun rqt_graph rqt_graph
   ```

2. Monitor velocity commands sent to the turtle in real time:

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

3. Create a simple *C++* example node:

```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 execute this example 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 the *CMakeLists.txt* file 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 that the node is active using `rosnode list`. The corresponding results are shown in Figure 4.

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 may depend on Python 2, while only Python 3 is installed. To resolve this compatibility issue, create a symbolic link:
     `sudo ln -s /usr/bin/python3 /usr/bin/python`
   - Visualize the TF tree using:
     `rosrun rqt_tf_tree rqt_tf_tree`
     The visualization shows three coordinate frames: *world* (parent), *turtle1*, and *turtle2*.
   - Inspect transformations in real time:
     `rosrun tf tf_echo /turtle1 /turtle2`
   - The resulting output is illustrated in Figure 5.
5. **Using *RViz***
   Run the following command to launch *RViz*:
   `LIBGL_ALWAYS_SOFTWARE=1 rviz`
   Software rendering is required because *RViz* relies on OpenGL, which may not be fully supported in virtualized environments. The visualization result is shown in Figure 6.

Notice: Modify the *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

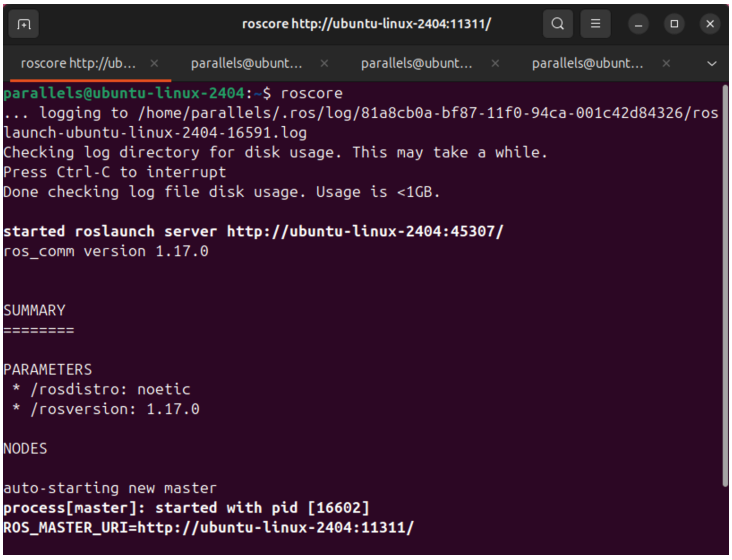
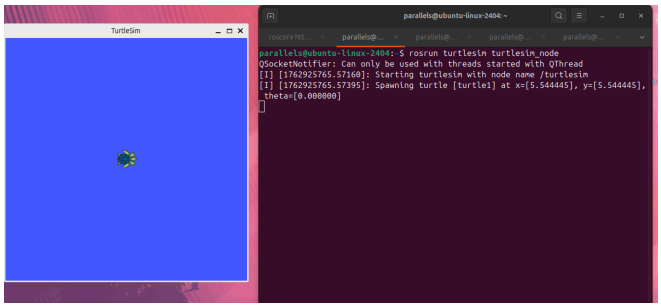
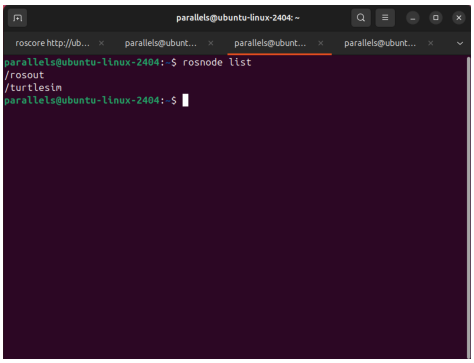


Figure 2: ROS Master

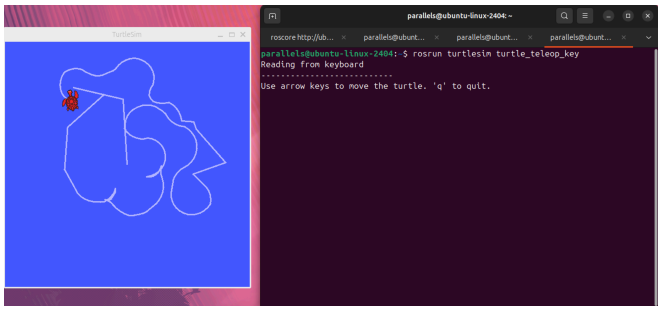
- *ROS* Nodes



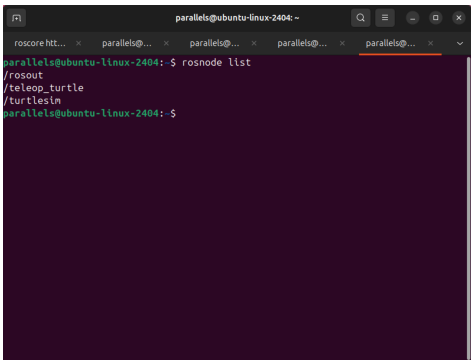
Turtlesim node running



rosnode list output



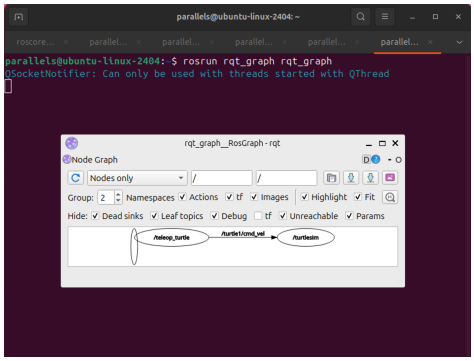
Turtle teleop node running



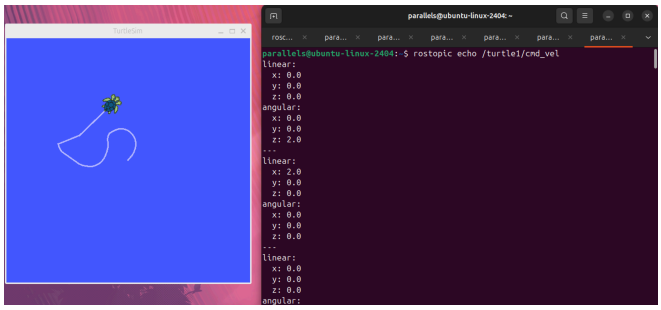
rosnode list output

Figure 3: ROS nodes

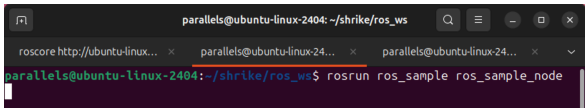
- *ROS* Topics



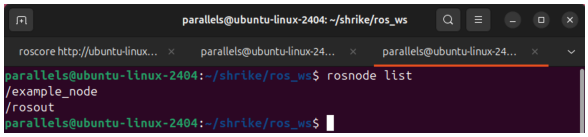
rqt_graph output



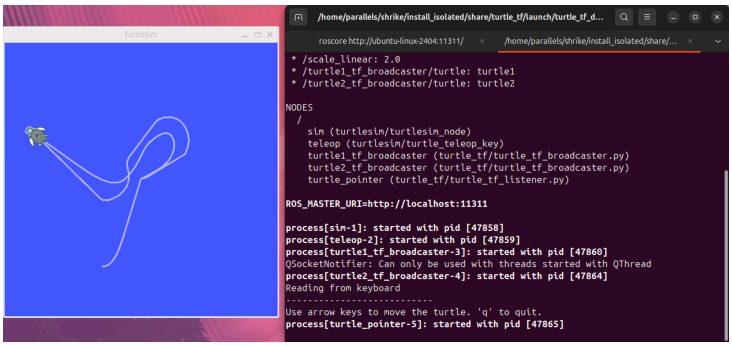
Velocity topic output



ROS node running

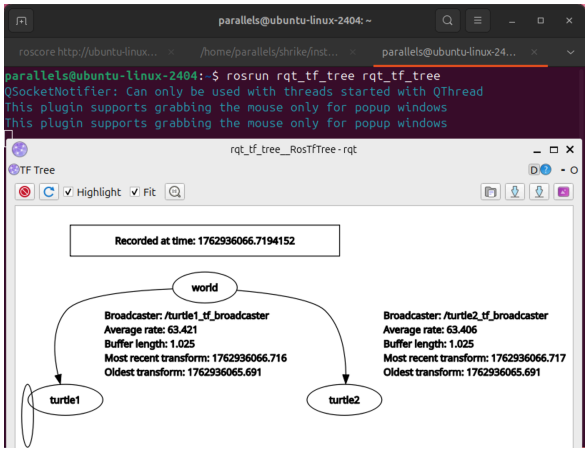


rosnode list output

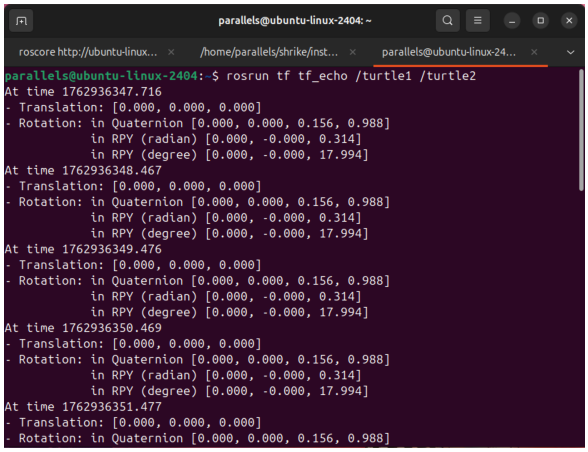


Roslaunch output

Figure 4: ROS topics



(a) rqt_tf_tree output



(b) tf_echo output

Figure 5: TF tools

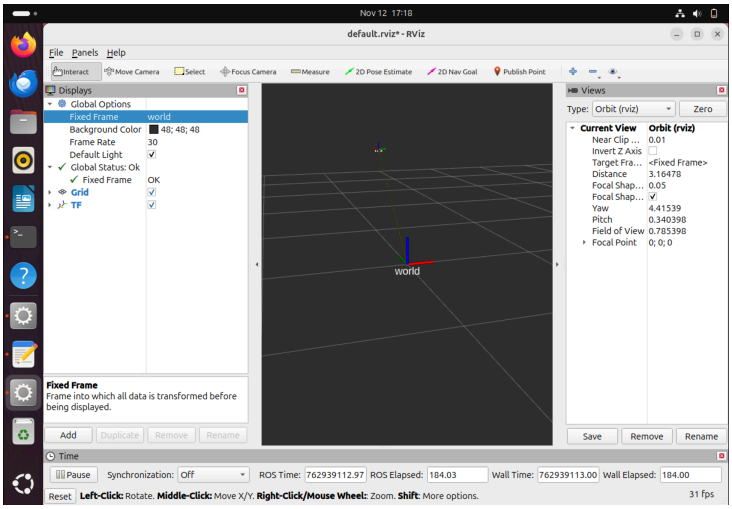


Figure 6: RViz

2.2.4 Discussion

This section demonstrates the core principles of inter-node communication and system visualization within the *ROS* framework. By implementing publisher and subscriber nodes and observing message flow using *rqt_graph* and *rostopic echo*, it becomes clear how topic-based communication enables modular and decoupled node interaction. The ability to monitor live message streams allows developers to validate correct behavior and quickly identify configuration or runtime issues.

The use of *TF* frames further illustrates the importance of coordinate transformation management in robotic systems. The hierarchical relationship among the *world*, *turtle1*, and *turtle2* frames highlights how spatial relationships are maintained and broadcast across the *ROS* network. Tools such as *rqt_tf_tree* and *tf_echo* provide intuitive means of visualizing and debugging frame hierarchies, reinforcing understanding of kinematic relationships and transformation propagation.

Running *RViz* in a virtualized environment emphasizes practical considerations for deploying visualization tools. The need to enforce software rendering with `LIBGL_ALWAYS_SOFTWARE=1` underscores how hardware acceleration and OpenGL support directly affect performance and usability.

2.3 Part III

2.3.1 Objective

To utilize *TF* to control the movement of drones.

2.3.2 Methodology

1. **Setting up the workspace**
   First, create a Catkin workspace named *vnav_ws* under the home directory:

   ```
   mkdir -p ~/vnav_ws/src
   cd ~/vnav_ws/
   catkin init
   ```

   The output should appear as shown in Figure 7. Next, copy the *two_drones_pkg* package into the *src* folder:

   ```
   cp -a /source_path/two_drones_pkg ~/vnav_ws/src
   ```

   Then, build the workspace using:

   ```
   catkin build
   ```

   The build output is shown in Figure 7. After the build completes, source the setup file to refresh the environment:

   ```
   source devel/setup.bash
   ```

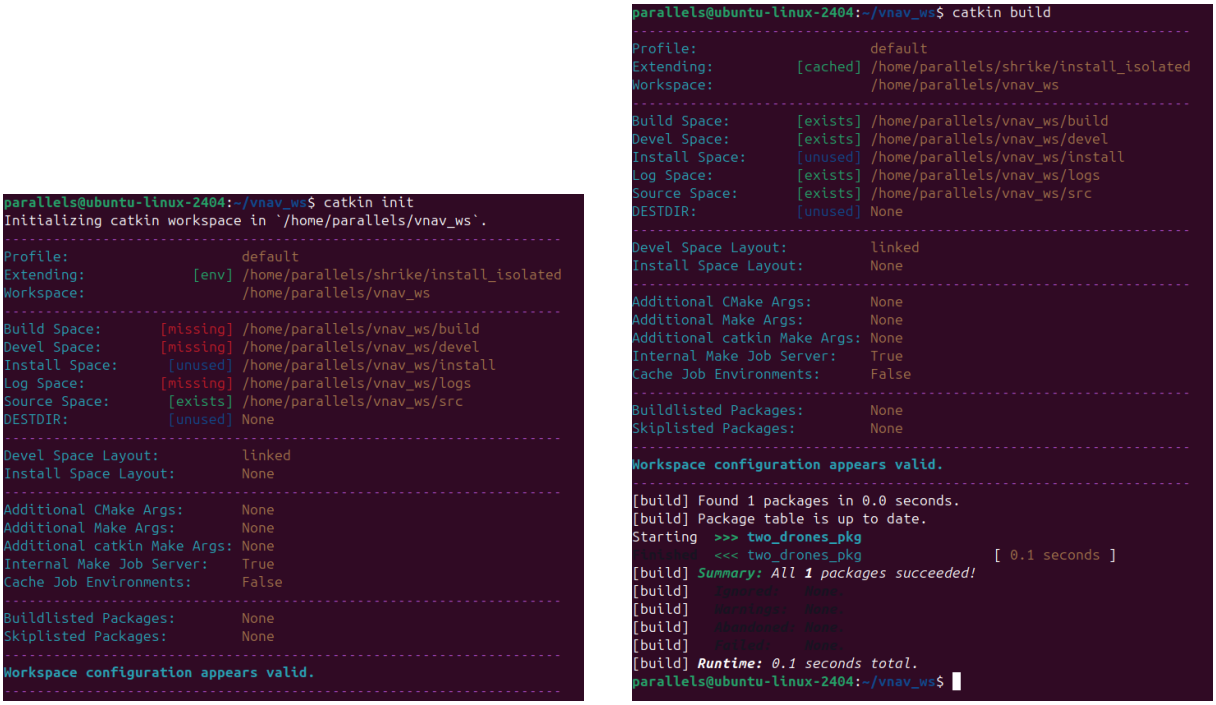
   Finally, launch *ROS* to visualize the static drone scene in *RViz*:

   ```
   roslaunch two_drones_pkg two_drones.launch static:=True
   ```

   After adding the *TF* display, three colored axes will appear for each coordinate frame, as illustrated in Figure 8. Remember to save the RViz configuration once the setup is complete.

2.

2.3.3 Observations



(a) catkin init (b) catkin build

Figure 7: catkin init & build

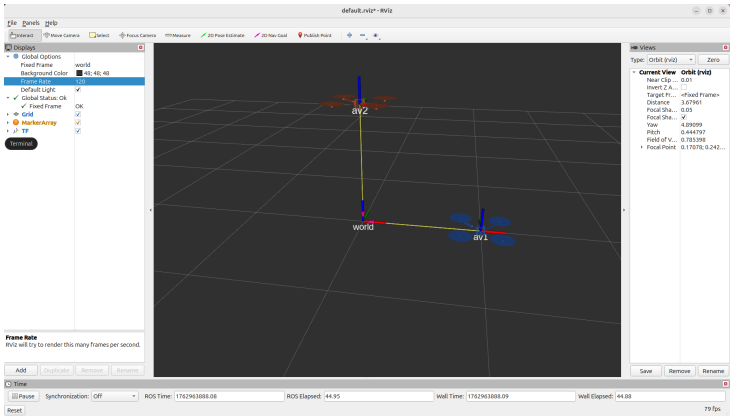


Figure 8: Two drones

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