ROBT 611 - ROS2 BASIC PRACTICE

Kanagat Kantoreyeva, Assanali Tokenov

***Abstract*—This assignment involves completing foundational training in the Robot Operating System (ROS) 2 environment. Initially, the task requires following the ROS 2 tutorials on fundamental concepts and basic applications. Further, the assignment progresses to modifying the tutorial robot setup by integrating additional components: a new ROS node, an additional topic, a service, and an action, each featuring distinct messages and functionalities. The final step involves updating the launch file to incorporate and manage these newly added elements, thereby demonstrating practical skills in configuring and deploying ROS 2 systems.**

I. Task 1. Follow the instructions and complete ROS 2 tutorial Session 1 - ROS Concepts and Fundamentals (ROS2) and Session 2 - Basic ROS Applications (ROS2)

A. Session 1

1) ROS Setup: This task required cleaning up the Virtual Machine space because there was not enough space on the disk for installation. Several errors appeared during installation, one of them is a package failed to process. This is shown on Fig 1. Other packages took more than 2 hours to complete installation.

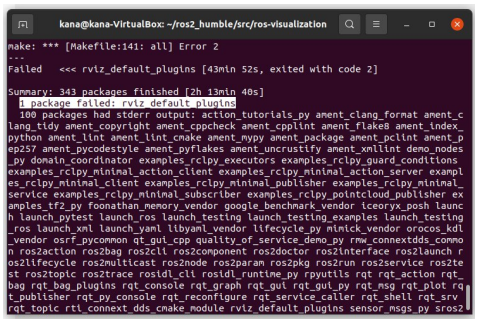

Figure 1. Package failed

After using $COLCON\_IGNORE$, the installation was successful. To test, talker and listener was run. Fig 2 shows the listener running successfully. Later edited: Creating $COLCON\_IGNORE$ created a lot of errors afterwards, so it was removed, and all the steps were done again.

Sourcing the setup also caused some troubles at first. Fig 3 shows the warning which appeared. To deal with this problem $ros\_switch.sh$ file was created which could switch between two ROS distributions. It can be seen from Fig 4 how it is working. The insides of the $ros\_switch.sh$ is below:

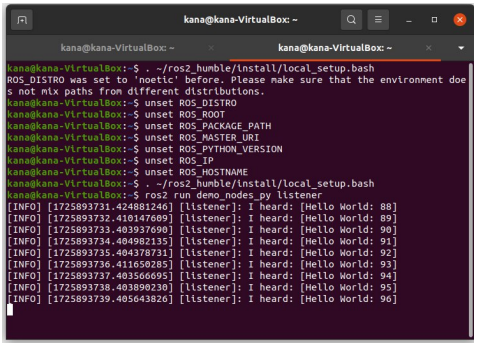

Figure 2. Listener successful

```
#!/bin/bash
if [ "$1" == "ros1" ]; then
    unset ROS_DISTRO
    unset ROS_ROOT
    unset ROS_PACKAGE_PATH
    unset ROS_MASTER_URI
    unset ROS_PYTHON_VERSION
    unset ROS_IP
    unset ROS_HOSTNAME
    source /opt/ros/noetic/setup.bash
    echo "Switched to ROS 1 (Noetic)"
elif [ "$1" == "ros2" ]; then
    unset ROS_DISTRO
    unset ROS_ROOT
    unset ROS_PACKAGE_PATH
    unset ROS_MASTER_URI
    unset ROS_PYTHON_VERSION
    unset ROS_IP
    unset ROS_HOSTNAME
    source ~/ros2_humble/install/local_setup.bash
    echo "Switched to ROS 2 (Humble)"
else
    echo "Usage: ./ros_switch.sh [ros1|ros2]"
fi
```

After this, We were able to source the setup. It can be seen from Fig 5.

2) Installing Packages : All the steps from this part was accomplished and the package was successfully located. It can be seen from Fig 6. Installation was done from source, not from apt Repository. After this task, later a lot of errors appeared, so we had to switch Ubuntu system from 20.04 to 22.04 in my VirtualMachine. All of the above steps were completed again. It can be seen from Fig 7.

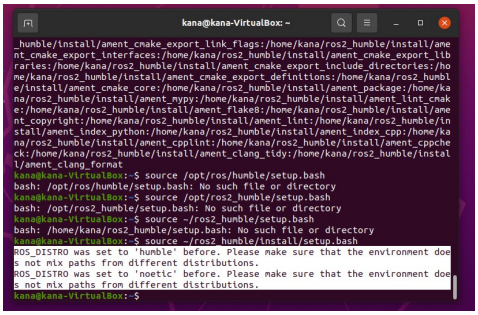

Figure 3. Warning

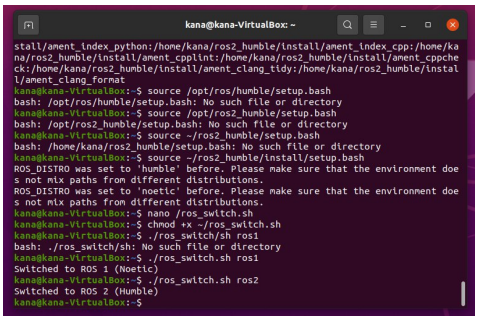

Figure 4. ROS switch at work

3) Packages and Nodes: All the steps were successfully completed using the provided instructions. In Fig 8, the 'package.xml' file for the project can be seen, showing the configuration of dependencies and build settings. In Fig 9, the ROS node is running successfully after being launched. Finally, in Fig 10, it is confirmed that the '*vision_node*' is running, and the output message "Hello, World!" has been printed.

4) Topics and Messages: In Fig 11, the process of completing the task using the 'rqt plot' tool is captured. The plot displays the values for the x and y positions of the AR pose marker. This figure provides insight into the continuous data being published during the task execution. In Fig 12, it can be seen that the task was completed successfully. The '*vision_node*' consistently received pose data with x, y, and z coordinates of -0.6, 0.2, and 0.5, respectively. This indicates that the system was able to continuously track the object's position without any interruptions, verifying the successful operation of the node.

B. Session 2

1) Services: The services part was successfully completed by following the given instructions. As shown in Fig 13, the node executed properly, and the part localization was achieved with coordinates x: -0.6, y: 0.2, and z: 0.5.s

2) Launch files: The instructions were followed and the expected results were achieved. As seen in

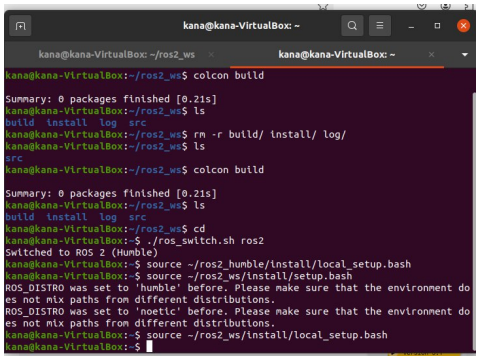

Figure 5. Source setup.bash

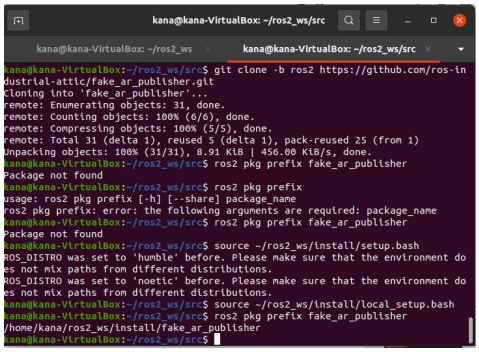

Figure 6. *Fake_ar_publisher* package

Fig 14, the process executed correctly, though several warnings and errors related to publisher and subscription handles were logged. Despite these errors, the node completed its execution and finished cleanly, as indicated by the final message.

3) Parameters: The task and the tutorial were completed successfully. As seen in Fig 15, the '*vision_node*' was executed alongside the '*fake_ar_publisher_node*'. Both nodes started without errors, indicating that the system is correctly running and the simulated AR marker publisher is functioning as expected.

II. Task 2. Modify the tutorial robot setup by adding an additional ROS node, a topic, a service and an action with other messages/functionalities.

For this task, we have created new package inside which we created cpp file. The resulting *robot_node.cpp* will be attached to this report. The *robot_node.cpp* file defines a ROS 2 node named *robot_node* that performs three main tasks: 1) Publishes to a Topic: Every second, the node publishes a message "Robot is operational" to the *robot_status* topic. 2) Provides a Service: The node offers a service *add_two_ints*, which takes two integers as input and returns their sum. This service can be called by other nodes. 3) Implements an Action: The node hosts

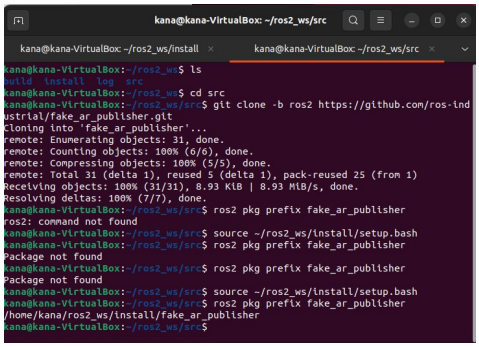

Figure 7. *Fake_ar_publisher* package after switching UBUNTU

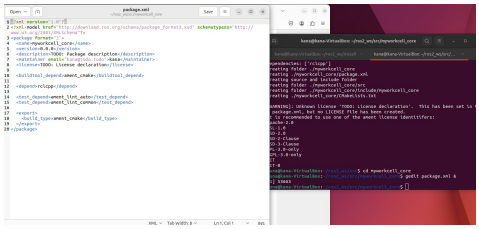

Figure 8. package.xml

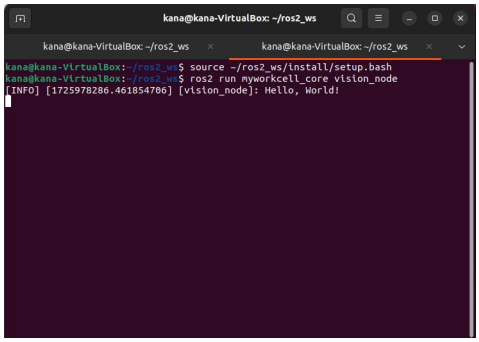

Figure 9. *vision_node*

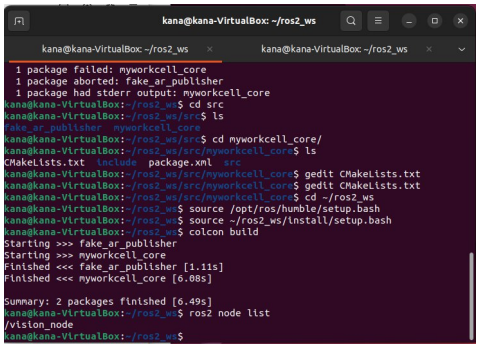

Figure 10. ros2 node list

a *count_up* action that counts from 0 to a given number (specified in the goal) and sends feedback at each count. The action can be canceled by the client mid-execution, and it returns a result when finished. This node showcases basic ROS 2 functionalities: publishing, servicing, and actions.

All the Figures included and labeled.

III. Task 3. Modify the launch file to run your new elements in addition.

The '*robot_launch.py*' file is a ROS 2 launch file that starts the '*robot_node*'. It launches the node with the name '*robot_node*', ensuring that it runs and all its functionalities (topic publishing, service, and action) are available. It outputs logs to the terminal so you can monitor the node's activity.

This launch file makes it easy to start the '*robot_node*' without manually running the executable.

IV. Conclusion

This assignment successfully introduced key concepts of ROS 2, covering installation challenges, package setup, and execution of nodes, topics, and services. We resolved installation issues with solutions like the '*ros_switch.sh*' script, and tested fundamental ROS functionalities. Extending the tutorial setup by adding a new ROS node, topic, service, and action demonstrated practical skills in configuring and deploying ROS 2 systems. Despite some troubleshooting, the tasks were completed successfully, providing a solid foundation for more advanced ROS 2 applications.

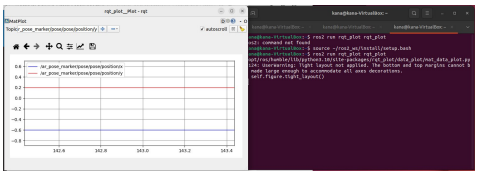

Figure 11. rqt plot

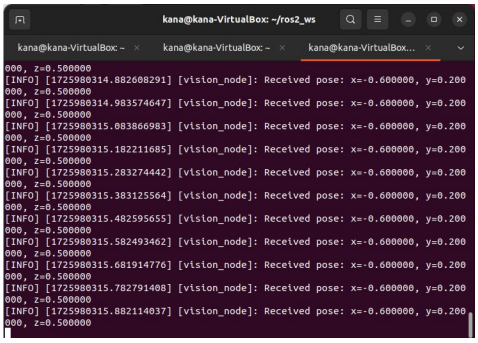

Figure 12. Node

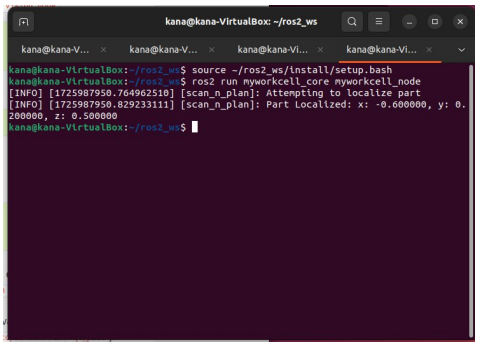

Figure 13. Services

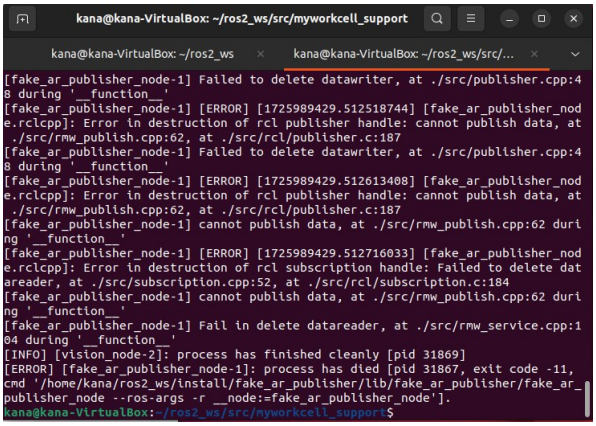

Figure 14. Launch files

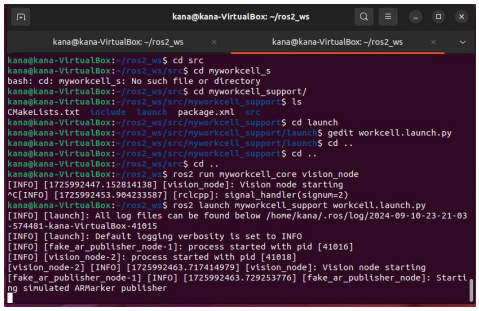

Figure 15. Launch files

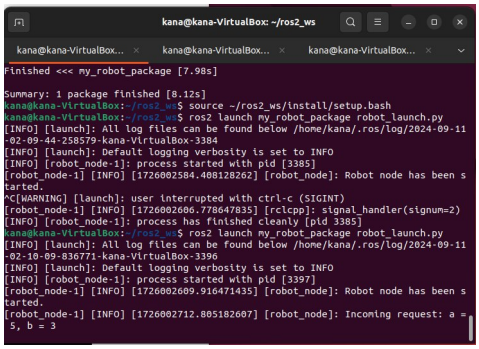

Figure 16. my robot package 1

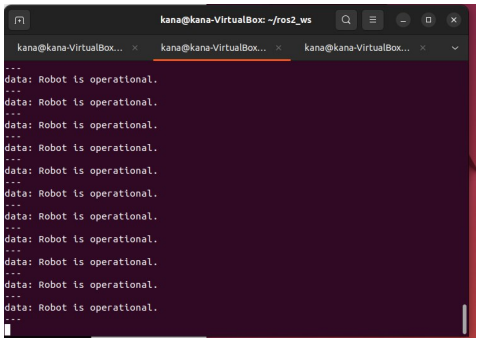

Figure 17. my robot package 2

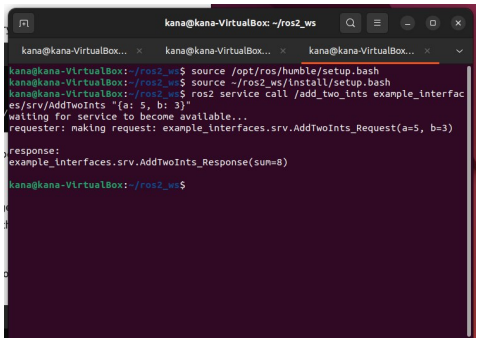

Figure 18. my robot package 3

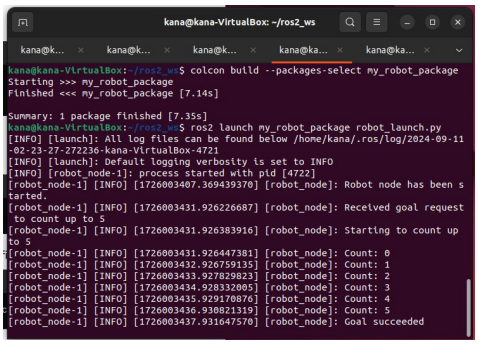

Figure 19. my robot package 4

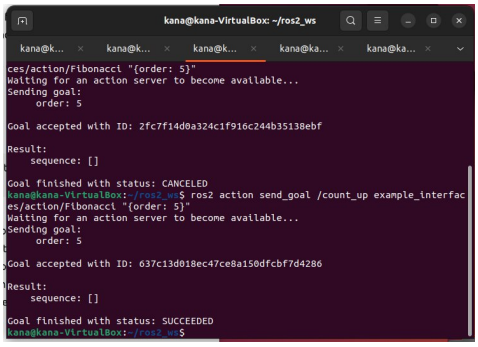

Figure 20. my robot package 5