# **AIG 240 GROUP-3 PROJECT 1**

**Apoorva’s Answers:**

1. What command did you use to create a ROS 2 package?

Answer: ros2 pkg create my\_robot\_controller --build-type

ament\_python --dependencies rclpy geometry\_msgs

1. Explain why and how you used ROS messages in your program.

Answer:

* ROS messages enable nodes to interact with one another through a publisher-subscriber framework.
* In this project, messages are utilized to transmit velocity commands for controlling the robot's movement.
* The node publishes geometry\_msgs/msg/Twist messages on the /cmd\_vel topic to indicate both linear and angular velocities.

1. How do you verify that your ROS 2 node is publishing messages correctly?

* Answer: Utilize ros2 topic list to view the topics that are available.
* Utilize ros2 topic echo /cmd\_vel to observe the messages that are being sent.
* Utilize ros2 node list to verify that your node is operational.
* Utilize ros2 topic info /cmd\_vel to verify the publishers and subscribers of the topic.

1. Describe the steps to launch Gazebo and your ROS 2 node simultaneously.

Answer:

To launch Gazebo and run the ROS 2 node, these are the following steps:

1. **Build the package in the ROS 2 workspace root:**

* Colcon build
* source install/setup.sh

1. **Start Gazebo in one terminal:**

* ros2 launch gazebo\_ros gazebo.launch.py

1. **Run the Python node in another terminal:**

* ros2 run my\_robot\_controller robot\_controller

**YAN’S ANSWERS**

1. What command did you use to create a ROS 2 package?

ros2 pkg create my\_robot\_controller --build-type ament\_python --dependencies rclpy geometry\_msgs

2. Explain why and how you used ROS messages in your program.

**Why:**

1. In ROS 2, data is transferred via topics using a publisher/subscriber mechanism. To control a robot’s movement, we publish messages to a known topic.
2. By using built-in ROS message types, we keep data representation consistent and easily shareable between different nodes and packages.
3. Different nodes can subscribe and publish without knowing each other’s implementation details, as they only need to agree on message types and topics.

**How:**

A Python node that publishes a geometry\_msgs/Twist message:

import rclpy

from rclpy.node import Node

from geometry\_msgs.msg import Twist

class RobotController(Node):

def \_\_init\_\_(self):

super().\_\_init\_\_('robot\_controller')

# Create a publisher for Twist messages on the '/cmd\_vel' topic

self.publisher\_ = self.create\_publisher(Twist, '/cmd\_vel', 10)

# Option 1: Use a timer callback to publish periodically

timer\_period = 0.1 # Publish at 10 Hz

self.timer = self.create\_timer(timer\_period, self.publish\_cmd\_vel)

def publish\_cmd\_vel(self):

# Construct a Twist message

cmd\_msg = Twist()

cmd\_msg.linear.x = 0.5 # Move the robot forward

cmd\_msg.angular.z = 0.5 # Rotate the robot

self.publisher\_.publish(cmd\_msg)

self.get\_logger().info(f'Publishing: linear.x={cmd\_msg.linear.x}, angular.z={cmd\_msg.angular.z}')

def main(args=None):

rclpy.init(args=args)

node = RobotController()

try:

rclpy.spin(node)

except KeyboardInterrupt:

pass

node.destroy\_node()

rclpy.shutdown()

if \_\_name\_\_ == '\_\_main\_\_':

main()

**Publisher**: create\_publisher(Twist, '/cmd\_vel', 10).

**Timer**: Calls publish\_cmd\_vel() at a fixed rate.

**Message Content**: Set linear and angular fields based on desired robot movement.

3. How do you verify that your ROS 2 node is publishing messages correctly?

1. **List Topics**:  
   ros2 topic list

Confirms the /cmd\_vel topic is visible.

1. **Echo Messages**:  
   ros2 topic echo /cmd\_vel

Displays the published messages (linear, angular components) in real time.

1. **Check Node Info**:  
   ros2 node list

ros2 node info robot\_controller

Ensures thenode exists and is publishing to /cmd\_vel.

1. **Check Publish Rate**:  
   ros2 topic hz /cmd\_vel

Shows the publishing frequency

4. Describe the steps to launch Gazebo and your ROS 2 node simultaneously.

In the ROS 2 workspace root, build the package:

colcon build

source install/setup.sh

**Terminal A**: Start Gazebo (gazebo.launch.py):  
ros2 launch gazebo\_ros gazebo.launch.py

**Terminal B**: Run Python node:  
ros2 run my\_robot\_controller robot\_controller

ASGARI’S ASNWERS

**1. What command did you use to create a ROS 2 package?**

To create a ROS 2 package, we used the following command:

ros2 pkg create my\_robot\_controller --build-type ament\_python --dependencies rclpy geometry\_msgs

**2. Explain why and how you used ROS messages in your program**

We used ROS messages because they enable communication between nodes via a publisher-subscriber framework. In this project, they helped sending commands to control the robot’s movement. Also, built-in ROS message types like geometry\_msgs/Twist, keeps data representation standardized, ensuring compatibility across nodes which interact independently, requiring only agreement on topics and message structures.

The program implements a Python node that publishes messages of type geometry\_msgs/Twist on the /cmd\_vel topic:

* **Publisher Creation:** A publisher is created to send Twist messages.
* **Timer Callback:** The node uses a timer to periodically call the function publish\_cmd\_vel() at a fixed interval (e.g., 10 Hz).
* **Message Content:** The Twist message specifies linear and angular velocities to dictate robot movement. For example:

cmd\_msg = Twist()

cmd\_msg.linear.x = 0.5 # Move forward

cmd\_msg.angular.z = 0.5 # Rotate

self.publisher\_.publish(cmd\_msg)

**3. How do you verify that your ROS 2 node is publishing messages correctly?**

To confirm that ROS 2 node is functioning as intended, the following checks should be done:

**List Topics:**

ros2 topic list

Ensures the /cmd\_vel topic is visible.

**Echo Messages:**

ros2 topic echo /cmd\_vel

Displays the real-time message content, showing linear and angular velocity values.

**Check Node Info:**

ros2 node list

ros2 node info robot\_controller

Verifies the node exists and confirms it’s publishing to the /cmd\_vel topic.

**Check Publish Rate:**

ros2 topic hz /cmd\_vel

Displays the rate at which messages are being published.

**4. Describe the steps to launch Gazebo and your ROS 2 node simultaneously.**

To run Gazebo and our ROS 2 node concurrently:

1.Build the ROS 2 Package:

colcon build

source install/setup.sh

2. Start Gazebo (Terminal A):

ros2 launch gazebo\_ros gazebo.launch.py

This launches the Gazebo simulation environment.

3. Run the Python Node (Terminal B):

ros2 run my\_robot\_controller robot\_controller

This starts your custom node to control the robot.

**VIPIN’S ANSWERS**

1. What command did you use to create a ROS 2 package?

ros2 pkg create my\_robot\_controller --build-type ament\_python --dependencies rclpy geometry\_msgs

my\_robot\_controller → The name of the package.

--build-type ament\_python → Specifies that it is a Python-based ROS 2 package.

--dependencies rclpy geometry\_msgs → Ensures that the package includes rclpy (for ROS 2 nodes) and geometry\_msgs (for sending movement commands).

2. Explain why and how you used ROS messages in your program

**Why use ROS messages?**

* ROS messages enable communication between different nodes using a **publisher-subscriber model**.
* In this project, **geometry\_msgs/Twist** messages are used to send movement commands to the robot.
* The Twist message defines both:
  + **Linear velocity (linear.x)** → Moves the robot forward.
  + **Angular velocity (angular.z)** → Rotates the robot.

**How were ROS messages used?**

* A **publisher node** was created to send movement commands.
* The node sends a **Twist message** to the /cmd\_vel topic every **0.1 seconds**.
* The message specifies the **robot's speed and rotation**

**3. How do you verify that your ROS 2 node is publishing messages correctly?**

To confirm that ROS 2 node is functioning as intended, the following checks should be done:

**List Topics:**

ros2 topic list

Ensures the /cmd\_vel topic is visible.

**Echo Messages:**

ros2 topic echo /cmd\_vel

Displays the real-time message content, showing linear and angular velocity values.

**Check Node Info:**

ros2 node list

ros2 node info robot\_controller

Verifies the node exists and confirms it’s publishing to the /cmd\_vel topic.

**Check Publish Rate:**

ros2 topic hz /cmd\_vel

Displays the rate at which messages are being published.

**4. Describe the steps to launch Gazebo and your ROS 2 node simultaneously.**

Build the ROS 2 Package

cd ~/workspace

colcon build --merge-install

source install/setup.bash

**Start Gazebo (Terminal A)**

ros2 launch gz\_sim gazebo.launch.py

**Run the Python Node (Terminal B)**

ros2 run my\_robot\_controller move\_robot

* Starts the **ROS 2 node to control the robot**.

**Verify Movement**

ros2 topic echo /cmd\_vel

* Checks if movement commands are being sent to the robot.