National University of Sciences and Technology (NUST)
School of Electrical Engineering and Computer Science

Department of Electrical Engineering and Computer Science

Faculty Member: Dr. Hafsa Iqbal **Date:** 10/03/2024

Semester: 8th **Group:** GP-1

Lab Engineer: Munadi Ahmad Sial

EE-381 Robotics

Lab 5: Gazebo Simulator & Twist Messages

Group Members

		PLO5 – CLO4		PLO9 – CLO4	
Name	**Reg. No**	**Analysis of Data in Lab Report**	**Modern Tool Usage**	**Individual and Team Work**	**Total Marks**
		10 Marks	**5 Marks**	**5 Marks**	**20 Marks**
Muhammad Abdullah Sohail	343642				
Muhammad Ahmed Mohsin	333060				
Muhammad Umer	345834				
Hassan Rizwan	335753				
Muhammad Saad	333414				

Table of Contents

Introduction

This laboratory exercise is intended to introduce Gazebo, a popular physics simulator used widely with ROS. The major advantage of a simulator is that it allows for robot programming without having to buy an actual robot. The physics of the simulator closely approximates the real-life hardware implementation so it is useful for tests that risk hardware damage. This lab will utilize the robot simulation in Gazebo and will focus on "Twist" messages which are used to command the robot to move.

Objectives

The following are the main objectives of this lab:

- Execution of Gazebo Simulator with ROS
- Cloning of an existing package from a GitHub repository
- Performing teleoperation of a robot in Gazebo
- Publishing twist messages for commanding actuator velocities

Lab Task 1 – Simulation Startup

In this task, we will start Gazebo, clone the robot simulator packages from the repository and spawn the robot in the simulator.

Go to the link of the repository for the robot simulator. You will find the terminal commands for the installation which you must execute. Once complete, a workspace for the robot simulator will be created containing the required packages.

Start up the Gazebo simulator with the command given below. You need to be in the workspace root for this. (Also, ensure you have built the packages and sourced the workspace).

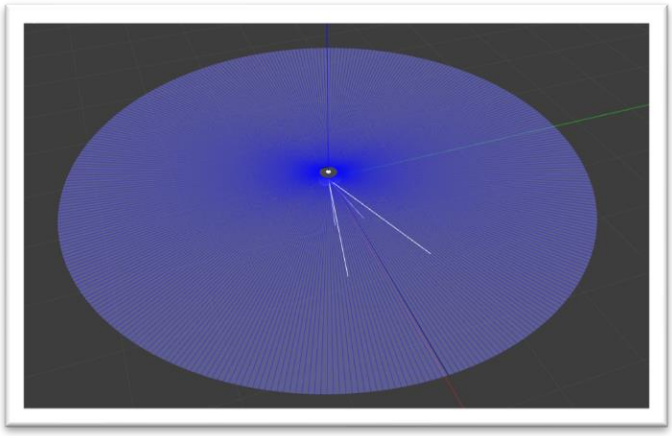
gazebo

You can install Gazebo from the following link if it is not installed:

https://classic.gazebosim.org/tutorials?tut=install_ubuntu&cat=install

Follow the commands for installing the simulator packages (you will be given instructions for the TurtleBot3 package). After installing, launch the simulation and place the *Waffle* model. The robot will start with a laser span of 360 degrees. You need to take the screenshot of this.

Output:

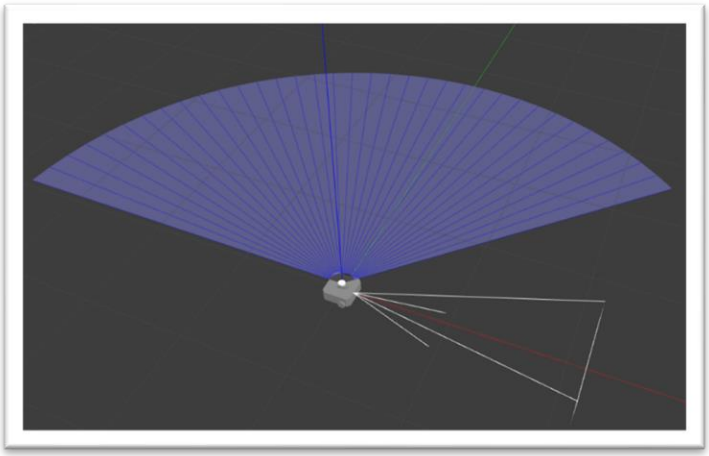


The 360 degree laser is highly useful for practical tasks. Due to the introductory nature of the lab, we will modify the laser scanner to make the visuals easier to see. (The 360 degree span will be revisited in a later lab). You need to modify the laser before proceeding to the next tasks. Go to the models file in the package:

turtlebot3_simulations/turtlebot3_gazebo/models/…/model.sdf

Open the model.sdf file and find the <sensor name = "…" type = "ray"> tag of the laser scanner. Inside the tag, you will find the <samples>, <min_angle> and <max_angle> sub-tags. Lower the samples to 36 and change the span of laser from 45-180 degrees. Save the file, build the package and rerun the simulation. Now the robot will have the modified laser. You need to take the screenshot and add it to your report.

Modified Laser Output:



Lab Task 2 – Teleoperation

In the simulation (with the modified laser), open a new terminal and launch the following command to start the teleoperation node:

```
ros2 run teleop_twist_keyboard teleop_twist_keyboard
```

With the terminal still active, press the mentioned keys to move the robot around. You need to make a video showing the teleop command being given then moving the robot around. Ensure that your group names are also shown on the terminal. The video will be named lab5_teleop for the submission.

Lab Task 3 – Linear and Angular Velocities

Before publishing velocity commands to the robot via topics, it is helpful to gain insights into the concepts of linear and angular velocities. For this, the TurtleSim program will be used. Open the TurtleSim node and start up the teleoperation node (which you will need to alter the angular orientation).

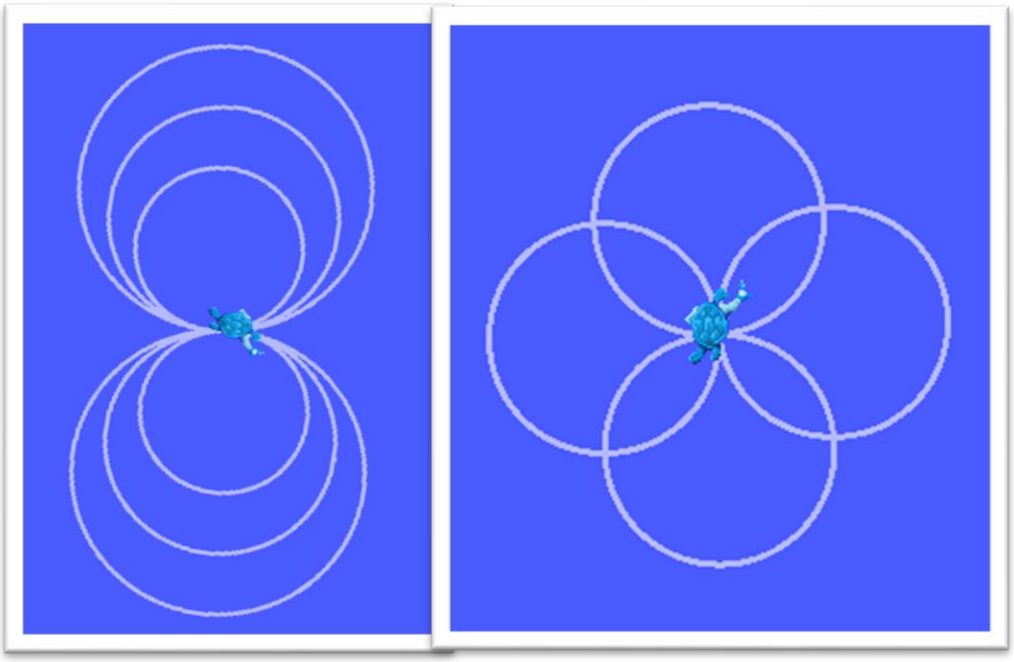
```
ros2 run turtlesim turtlesim_node

ros2 run turtlesim turtle_teleop_key
```

The velocity commands are given in the *cmd_vel* topic. The data type of such commands is the "Twist" message which contains the linear x, y, z and the angular x, y, z velocities. The turtle can only move for the linear x and angular z velocities only. It cannot move/strafe sideways in the linear y direction (the other 3 velocities do not exist as the TurtleSim is a 2-D environment). In a new terminal, use the following command to publish a Twist message to move the turtle:

```
ros2 topic pub --once turtle1/cmd_vel geometry_msgs/msg/Twist
"linear:
  x: 0.0
  y: 0.0
  z: 0.0
angular:
  x: 0.0
  y: 0.0
  z: 0.0"
```

Experiment with the linear and angular velocities. Then create the following two patterns and take their screenshots showing the turtle. Hint: try experimenting with only the angular velocities first.

Pattern Screenshots:



Lab Task 4 – Turning the Robot

In this task, you will use your understanding of linear and angular velocities in order to publish Twist messages to the robot. Start up the simulation and spawn the robot. Next, create a new package (with rclpy, std_msgs and geometry_msgs dependencies) in the workspace called velocity_package. In the package, create a node called turning.py and place the minimal publisher code.

Next, you need to make the following changes to the node:

Change all of the class and object names corresponding to the following:

```
vel_pub = VelocityPublisher()
```

Add the import for the Twist messages:

```
from geometry_msgs.msg import Twist
```

To initialize the twist message datatype, use the following command:

```
msg = Twist()
```

To enter a value for the twist message, use the following commands:

msg.linear.x = 1.0

msg.angular.x = 1.0

You need to use the above commands to change the node so that it will publish a turning command to the robot (no linear velocity). Provide the class code and also make a video (lab5_turning) showing execution of the node in the terminal. The video must show both the terminal and gazebo.

Code:

```
import rclpy
from geometry_msgs.msg import Twist
from rclpy.node import Node
from std_msgs.msg import Int32


class VelocityPublisher(Node):
    def __init__(self):
        super().__init__("turning")
        self.publisher_ = self.create_publisher(Twist, "cmd_vel", 10)

        timer_period = 1
        self.timer = self.create_timer(timer_period, self.timer_callback)

    def timer_callback(self):
        msg = Twist()
        msg.linear.x = 0.0
        msg.angular.z = 0.785  # 45 degrees per second
        self.publisher_.publish(msg)
        self.get_logger().info(f"Turn by {msg.angular.z} rad/s")


def main(args=None):
    rclpy.init(args=args)
    vel_pub = VelocityPublisher()
    rclpy.spin(vel_pub)
    vel_pub.destroy_node()
    rclpy.shutdown()


if __name__ == "__main__":
    main()
```

Lab Task 5 – Start-Stop

Create a second node called redgreen.py which contains the code for publishing Twist messages that will make the robot move forward, then stop, then move forward, then stop and so on. The duration of both the moving and stopping can be 1 second each. Provide the class code and make a video (lab5_redgreen) showing the execution.

Code:

```
import rclpy
from geometry_msgs.msg import Twist
from rclpy.node import Node
from std_msgs.msg import Int32


class VelocityPublisher(Node):
    def __init__(self):
        super().__init__("redgreen")
        self.publisher_ = self.create_publisher(Twist, "cmd_vel", 10)

        timer_period = 0.5  # move every 1 second
        self.move_timer = self.create_timer(timer_period, self.move_callback)

        stop_timer_period = 1  # stop after 2 seconds
        self.stop_timer = self.create_timer(stop_timer_period,
self.stop_callback)

    def move_callback(self):
        msg = Twist()
        msg.linear.x = 0.5  # move forward
        self.publisher_.publish(msg)
        self.get_logger().info(f"Move forward by {msg.linear.x} m/s")

    def stop_callback(self):
        msg = Twist()
        msg.linear.x = 0.0  # stop
        self.publisher_.publish(msg)
        self.get_logger().info(f"Stop")


def main(args=None):
    rclpy.init(args=args)
    vel_pub = VelocityPublisher()
    rclpy.spin(vel_pub)
    vel_pub.destroy_node()
    rclpy.shutdown()
```

```
if __name__ == "__main__":
    main()
```

Lab Task 6 – Moving in a Circle

Create a third node called circles.py. Write the code for publishing Twist messages that will make the robot move in a circular path. Provide the class code and make a video (lab5_circles) showing the execution.

Code:

```
import rclpy
from geometry_msgs.msg import Twist
from rclpy.node import Node
from std_msgs.msg import Int32


class VelocityPublisher(Node):
    def __init__(self):
        super().__init__("circles")
        self.publisher_ = self.create_publisher(Twist, "cmd_vel", 10)

        timer_period = 1
        self.timer = self.create_timer(timer_period, self.timer_callback)

    def timer_callback(self):
        msg = Twist()
        msg.linear.x = 1
        msg.angular.z = 0.785
        self.publisher_.publish(msg)
        self.get_logger().info(f"Move in a circle")


def main(args=None):
    rclpy.init(args=args)
    vel_pub = VelocityPublisher()
    rclpy.spin(vel_pub)
    vel_pub.destroy_node()
    rclpy.shutdown()
if __name__ == "__main__":
    main()
```

Conclusion:

In this lab, we explored how ROS nodes make the use of Twist messages to make turtlesim node move in various patterns through modifying linear and angular velocities. Furthermore, we explored the Gazebo simulating environment on turtlebot3 and performed tasks including observing laser scan, turning, and moving in a circle.