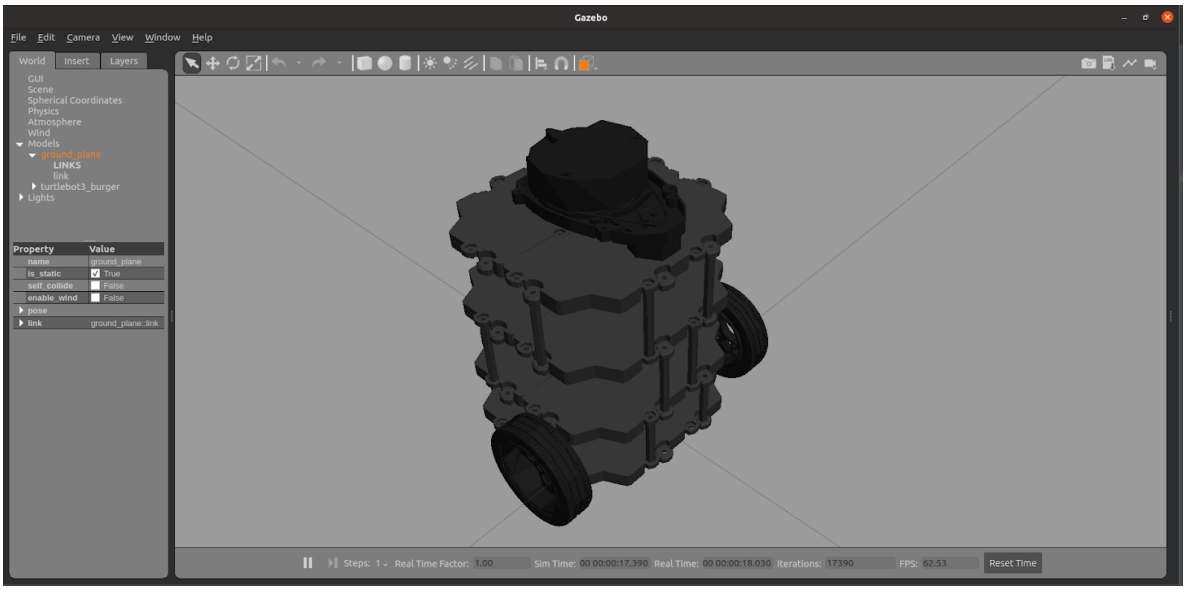
Sadaan Tahir - 25100224

EE 3002 L1 (Junior Design Studio - Robotics)

Spring 2025 - LAB 3

Turtlebot is a ROS standard platform robot. There are 3 versions of the TurtleBot model. TurtleBot1, TurtleBot2 and TurtleBot3. TurtleBot3 is a small, affordable, programmable, ROS-based mobile robot for education, research, hobby, and product prototyping. The TurtleBot3 can be customized in various ways depending on how you reconstruct the mechanical parts and use optional parts such as the computer and sensor. In addition, TurtleBot3 has evolved with cost-effective and small-sized SBCs suitable for robust embedded systems, 360-degree distance sensors, and 3D printing technology.
Learn more about it over here: https://www.turtlebot.com/turtlebot3/

For this lab, we will be running our simulations on the **TurtleBot3 Burger** model shown below:



Task 1: TurtleBot3 Setup [20 MARKS]

In this task, we will configure the **TurtleBot3 Burger** robot for simulation in **Gazebo v11** within our **ROS Noetic** environment. As ROS Noetic includes Gazebo 11 by default, the **primary objective** of Task 1 will be to **install the required TurtleBot3 packages** to begin with the simulations.

1.1 Initial Setup:

Steps:

1. **Install the TurtleBot3 package** by typing this command in a new terminal: sudo apt-get install ros-noetic-turtlebot3.
2. **Make a new workspace** namely **lab3** using mkdir ~/lab3/src followed by catkin_make.
3. Next, **clone the following packages** in the directory **jdsrobo@ubuntu:~/lab3/src$** using these terminal commands:

   3.1. **ROS msgs package for TurtleBot3**: git clone --branch noetic https://github.com/ROBOTIS-GIT/turtlebot3_msgs.git
   3.2. **TurtleBot3 package**: git clone --branch noetic https://github.com/ROBOTIS-GIT/turtlebot3.git
4. **Build** all the ROS packages within your workspace using catkin_make.
5. Next, to **optimize performance on your virtual machines**, add the following lines towards the very end of the **~/.bashrc file** (open it using gedit ~/.bashrc):
   5.1. export SVGA_VGPU10=0
   5.2. export TURTLEBOT3_MODEL=burger
6. **Save** the **~/.bashrc file** and apply the changes using this terminal command: source ~/.bashrc.
7. Next, clone the **simulation package** using these terminal commands:
   7.1. cd ~/lab3_ws/src
   7.2. git clone --branch noetic https://github.com/ROBOTIS-GIT/turtlebot3_simulations.git
8. **Build the workspace again** and make sure you source the setup.bash file by running source devel/setup.bash.

[5 MARKS]

1.2 Launching TurtleBot3 in a Gazebo World:

Now, in this task you will launch a TurtleBot3 simulation in Gazebo, control the robot using teleoperation, and write Python scripts to move the robot in a circle and navigate to a specified goal location.

Steps:

1. In a new terminal, launch the **TurtleBot3 simulation** in an **empty Gazebo world** by running: roslaunch turtlebot3_gazebo turtlebot3_empty_world.launch.
2. Open a second terminal to control the bot using the **keyboard teleoperation** command: roslaunch turtlebot3_teleop turtlebot3_teleop_key.launch.
   2.1. In case you get an error regarding the "**keyboard**" library being missing, install it using pip install keyboard in another terminal and rerun the launch file.
3. Create a Python file named **turtlebot3_circle.py** in your workspace's **scripts** directory. This Python script needs to be such that when it runs, **TurtleBot3 moves in a circle of radius 5 meters** from its initial position in the Gazebo world.
4. Create another Python file named **turtlebot3_go2goal.py** in the same scripts directory. This one should **move TurtleBot3 to any preset goal position** from its initial position in the Gazebo world.

[15 MARKS]

HINTS:

- For position and heading (orientation), you need to extract the relevant information from Odometry: http://docs.ros.org/en/noetic/api/nav_msgs/html/msg/Odometry.html. The orientation data provided by the **/odom** topic is in **quaternion format**. To work with this data in a more conventional format, it must be converted into the standard **(roll, pitch, yaw)** representation. To achieve this, you need to import the **euler_from_quaternion** function from the **tf.transformations** module.

You can use this callback function for the **/odom** subscriber:

```
def odom_callback(msg):
    global x
    global y
    global theta

    x = msg.pose.pose.position.x
    y = msg.pose.pose.position.y

    rot_q = msg.pose.pose.orientation

    # Converting the pose info in quaternions into euler form
    (roll, pitch, theta) = euler_from_quaternion([rot_q.x, rot_q.y,
rot_q.z, rot_q.w])
```

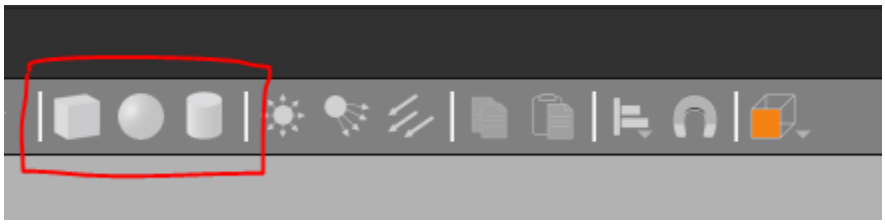
Task 2: Using LaserScan Data [10 MARKS]

In this task you need to **write a Python script that subscribes to the laser scan data and prints** the following:

- Total number of valid readings.
- Range of closest point and its angle from TurtleBot3's location.
- Range of farthest point and its angle from TurtleBot3's location.

Steps:

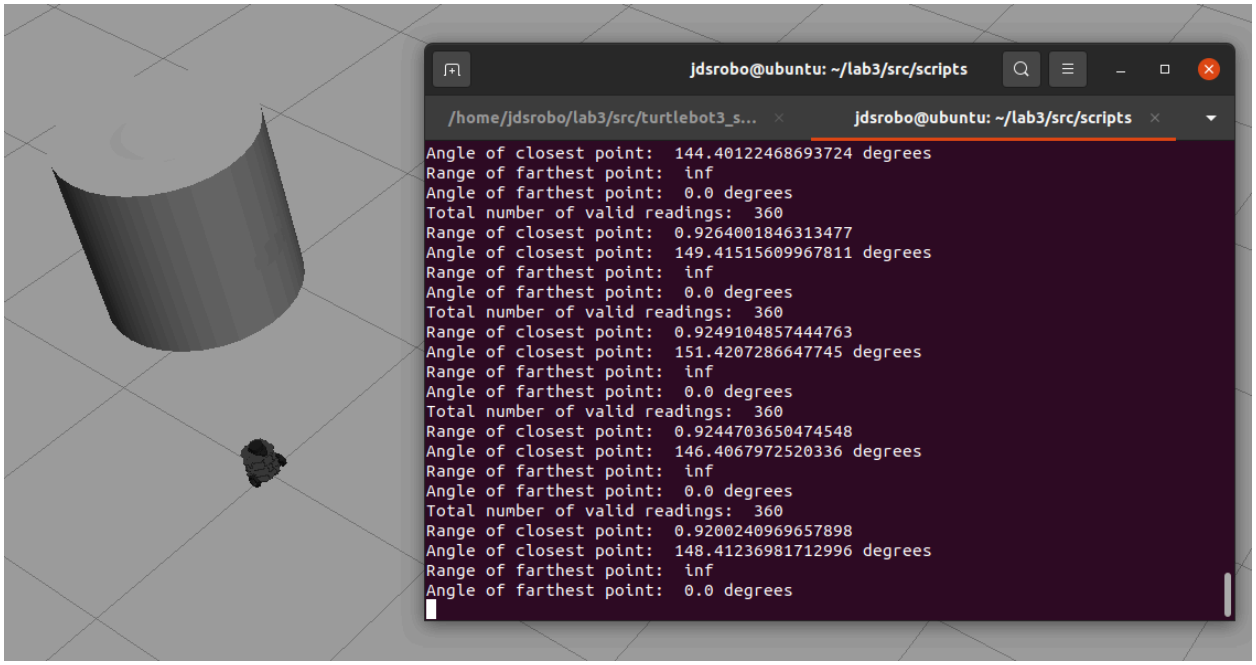
1. Make a new Python file under **jdsrobo@ubuntu:~/lab3/src/scripts$** called **turtlebot3_laserscan.py** that does the aforementioned.
2. Close any previous Gazebo worlds using pkill gzserver followed by pkill gzclient in terminal. Then, **launch an empty Gazebo world with a TurtleBot3** by running: roslaunch turtlebot3_gazebo turtlebot3_empty_world.launch in the same terminal.
3. Add any solid shape into the 3D world. Click on these preset shapes to select them:



4. Run your Python script, **turtlebot3_laserscan.py** and take screenshots of the terminal output.

[10 MARKS]

Your final output for this task should look similar to this:



HINTS:

- Import the scan message type like this: from sensor_msgs.msg import LaserScan. More information about LaserScan can be found here: http://docs.ros.org/en/noetic/api/sensor_msgs/html/msg/LaserScan.html
- Subscribe to the **/scan** topic using this: rospy.Subscriber("/scan", LaserScan, scan_callback).

Task 3: Transformations [5 MARKS]

In this task you need to **write a Python script that publishes the following transformations using tf2_ros library** (you can import it like import tf2_ros):

- Static transformation between robot frame and lidar frame.
    - Implement Static Transform publisher in launch file (transform_pub.luanch) , use the following link to understand how to publish static transform : http://wiki.ros.org/tf2_ros#:~:text=Tools-,static_transform_publisher,-static_transform_publisher%C2%A0x%C2%A0y

- Dynamic transformation between robot frame and world frame.
    - Use the following link to understand how to make dynamic transform, publisher in python : http://wiki.ros.org/tf2/Tutorials/Writing%20a%20tf2%20broadcaster%20%28Python%29#:~:text=%23!/usr/bin/env,.spin()
- Run both static and dynamic transform publisher nodes from transform_pub.launch.

Verification :

- To verify that transform is being published, do following :
- Using “rosrun rqt_tf_tree rqt_tf_tree”

More tutorials about doing transformations using tf2_ros library can be found here: https://wiki.ros.org/tf2_ros/Tutorials

Task 4: Mapping [15 MARKS]

In this task you need to **write a Python script** that first moves the Turtlebot3 **towards an object** in Gazebo world and then **circles around it once using laser scan data** in order to **create a map of the object:**

- Launch empty world in Gazebo with turtlebot3 model.
- Import an object i.e. sphere or cube in Gazebo.
- It should go to the object if its not already within some distance of it (maintain a minimum distance threshold it should keep from the object)
- Once it approaches the object, it should circle it while maintaining this threshold distance from the object.
- As it circles, it should create a plot of the object which should ideally represent the shape of the object. You **can not hardcode** this part to move it in a circle, you have to use the laser scan data to do this.

HINTS:

- The robot starting position acts as a world frame, so you have translation (tx,ty) and rotation (theta) of the robot from initial position in the form of position and orientation.
- Create a transform listener from robot frame (“base_link”) to world (“world”) frame, to listen to dynamic transform from robot to world frame.
    - Use the following link to understand how to write transform listener in python : http://wiki.ros.org/tf2/Tutorials/Writing%20a%20tf2%20listener%20%28Python%29#:~:text=%23!/usr/bin/env%20python%20%20%0A%20%20%202,37%20%20%20%20%20%20%20%20%20%20rate.sleep()

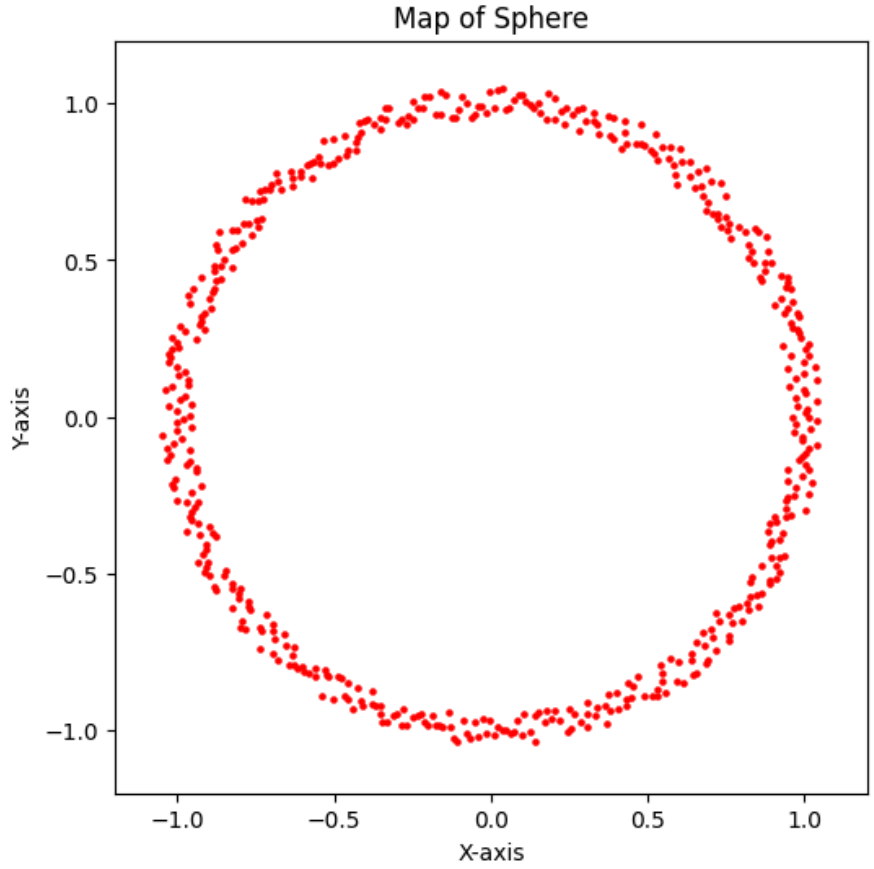
Get x_robot, y_robot and theta from transform listener to create below transformation matrix.

- When the robot sees an obstacle in its frame. At 90 or 270 degrees, you have distance at 90 or 270 degrees, and you can find its x and y coordinates in the robot frame.
- Once you get x,y coordinates of obstacles at each point (in robot frame), you need to convert all the obstacle points to world frame using a transformation matrix which requires translation (tx,ty) and rotation (theta) from the world origin.

$$\begin{bmatrix} X_{\text{world}} \\ Y_{\text{world}} \\ 1 \end{bmatrix} = \begin{bmatrix} \cos(\theta) & -\sin(\theta) & t_x \\ \sin(\theta) & \cos(\theta) & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_{\text{robot}} \\ y_{\text{robot}} \\ 1 \end{bmatrix}$$

- You need to make sure that always 90 degrees of robot face the obstacle aside, as you need to handle the corners of the object.
- **ranges[269]** will be useful to correct the robot's path by rotating if the distance changes, while **ranges[0]** will be useful so that the robot doesn't collide with obstacles in front of it.
- Angle of the closest obstacle will be useful for aligning the robot.

Your final output for this task should look similar to this (ideally):



Sadaan Tahir - 25100224

Submission Requirements:

1. Include a **zip file** containing the following items:
   1.1. The entire workspace **~/lab3** with all the Python script files.
   1.2. **Screenshots for all the tasks** except Task 1, sub-task 1.1. Screenshots may include either terminal outputs or Gazebo world snapshots or even both depending on the task.

2. Submit a **LAB report** in PDF format (please do not submit word files). This report should provide explanations for all tasks performed along with screenshots.