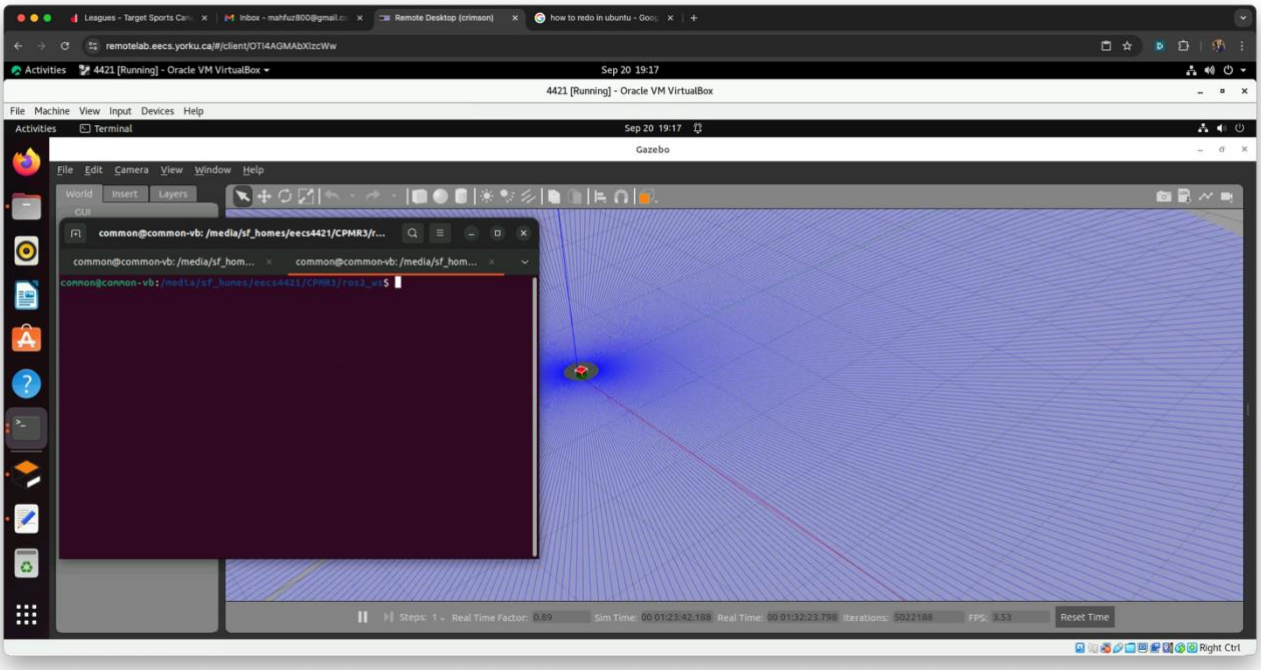
EECS 4421 LAB1

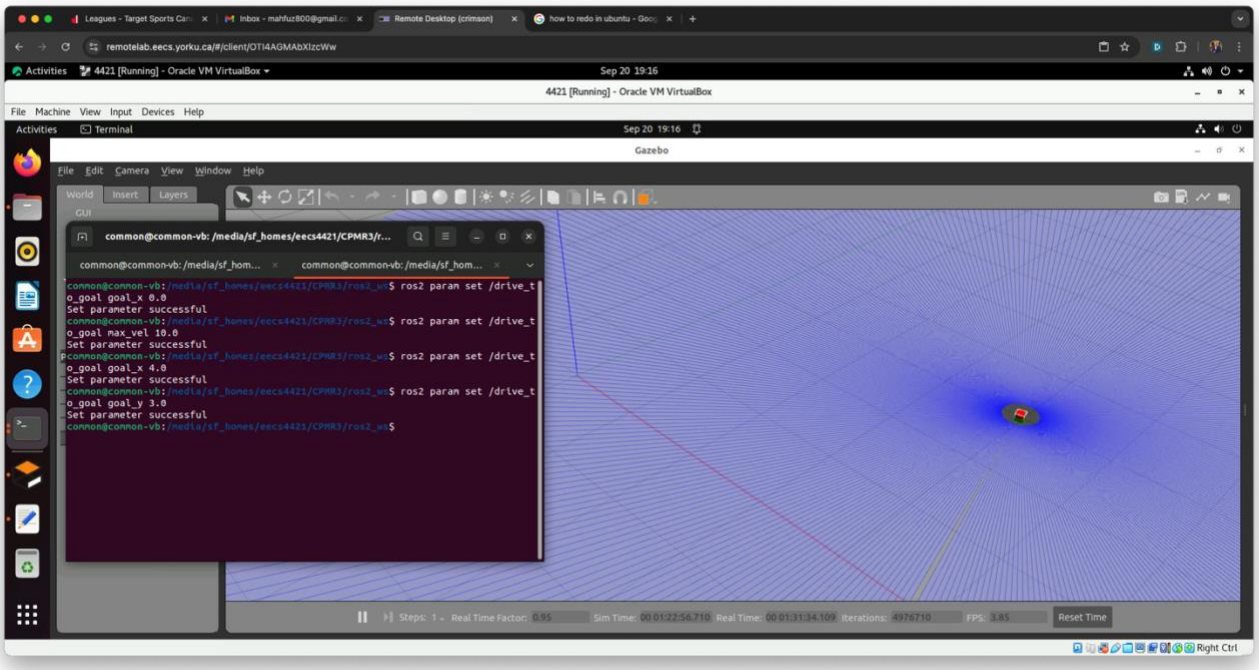
Name: Mahfuz Rahman

Student No.: 217847518

Initial State of robot:



State of robot after executing code:



drive_robot.launch.py

```
import os
from ament_index_python.packages import get_package_share_directory
from launch import LaunchDescription
from launch.actions import DeclareLaunchArgument, IncludeLaunchDescription
from launch.launch_description_sources import PythonLaunchDescriptionSource
from launch.substitutions import LaunchConfiguration
from launch_ros.actions import Node

def generate_launch_description():
    return LaunchDescription([
        DeclareLaunchArgument('goal_x', default_value = '0.0', description = 'goal (x)'),
        DeclareLaunchArgument('goal_y', default_value = '0.0', description = 'goal (y)'),
        DeclareLaunchArgument('goal_t', default_value = '0.0', description = 'goal (t)'),

        #Code added by Mahfuz Rahman
        DeclareLaunchArgument('max_vel', default_value = '0.2', description = 'max_vel
(vel)'),
        DeclareLaunchArgument('vel_gain', default_value = '5.0', description = 'vel_gain
(gain)'),

        Node(
            package = 'cpmr_ch2',
            executable = 'drive_to_goal',
            name = 'drive_to_goal',
            parameters = [
                {'goal_x' : LaunchConfiguration('goal_x')},
                {'goal_y' : LaunchConfiguration('goal_y')},
                {'goal_t' : LaunchConfiguration('goal_t')},
            ],
```

```
        ),
    ])
```

drive_to_goal.py

```
import math
import numpy as np
import rclpy
from rclpy.node import Node
from rclpy.parameter import Parameter
from rcl_interfaces.msg import SetParametersResult
from nav_msgs.msg import Odometry
from geometry_msgs.msg import Twist, Pose, Point, Quaternion
from nav_msgs.msg import Odometry

def euler_from_quaternion(quaternion):
    """
    Converts quaternion (w in last place) to euler roll, pitch, yaw
    quaternion = [x, y, z, w]
    """
    x = quaternion.x
    y = quaternion.y
    z = quaternion.z
    w = quaternion.w

    sinr_cosp = 2 * (w * x + y * z)
```

```
    cosr_cosp = 1 - 2 * (x * x + y * y)
    roll = np.arctan2(sinr_cosp, cosr_cosp)

    sinp = 2 * (w * y - z * x)
    pitch = np.arcsin(sinp)

    siny_cosp = 2 * (w * z + x * y)
    cosy_cosp = 1 - 2 * (y * y + z * z)
    yaw = np.arctan2(siny_cosp, cosy_cosp)

    return roll, pitch, yaw


class MoveToGoal(Node):
    def __init__(self):
        super().__init__('move_robot_to_goal')
        self.get_logger().info(f'{self.get_name()} created')

        self.declare_parameter('goal_x', 0.0)
        self._goal_x = self.get_parameter('goal_x').get_parameter_value().double_value
        self.declare_parameter('goal_y', 0.0)
        self._goal_y = self.get_parameter('goal_y').get_parameter_value().double_value
        self.declare_parameter('goal_t', 0.0)
        self._goal_t = self.get_parameter('goal_t').get_parameter_value().double_value
        self.add_on_set_parameters_callback(self.parameter_callback)
        self.get_logger().info(f"initial goal {self._goal_x} {self._goal_y} {self._goal_t}")

        #Code added by Mahfuz Rahman
        self.declare_parameter('max_vel', 0.2)
        self.declare_parameter('vel_gain', 5.0)

        self._max_vel = self.get_parameter('max_vel').get_parameter_value().double_value
```

```
        self._vel_gain = self.get_parameter('vel_gain').get_parameter_value().double_value


        ##########################

        self._subscriber = self.create_subscription(Odometry, "/odom",
self._listener_callback, 1)
        self._publisher = self.create_publisher(Twist, "/cmd_vel", 1)


    def _listener_callback(self, msg, max_pos_err=0.05):
        pose = msg.pose.pose

        cur_x = pose.position.x
        cur_y = pose.position.y
        o = pose.orientation
        roll, pitchc, yaw = euler_from_quaternion(o)
        cur_t = yaw

        x_diff = self._goal_x - cur_x
        y_diff = self._goal_y - cur_y
        dist = math.sqrt(x_diff * x_diff + y_diff * y_diff)

        twist = Twist()
        if dist > max_pos_err:
            x = max(min(x_diff * self._vel_gain, self._max_vel), -self._max_vel)
            y = max(min(y_diff * self._vel_gain, self._max_vel), -self._max_vel)
            twist.linear.x = x * math.cos(cur_t) + y * math.sin(cur_t)
            twist.linear.y = -x * math.sin(cur_t) + y * math.cos(cur_t)
            self.get_logger().info(f"at ({cur_x},{cur_y},{cur_t}) goal
({self._goal_x},{self._goal_y},{self._goal_t})")
        self._publisher.publish(twist)
```

```
    def parameter_callback(self, params):
        self.get_logger().info(f'move_robot_to_goal parameter callback {params}')
        for param in params:
            self.get_logger().info(f'move_robot_to_goal processing {param.name}')
            if param.name == 'goal_x' and param.type_ == Parameter.Type.DOUBLE:
                self._goal_x = param.value
            elif param.name == 'goal_y' and param.type_ == Parameter.Type.DOUBLE:
                self._goal_y = param.value
            elif param.name == 'goal_t' and param.type_ == Parameter.Type.DOUBLE:
                self._goal_t = param.value

            ########
            elif param.name == 'max_vel' and param.type_ == Parameter.Type.DOUBLE:
                self._max_vel = param.value
            elif param.name == 'vel_gain' and param.type_ == Parameter.Type.DOUBLE:
                self._vel_gain = param.value
            #######
            else:
                self.get_logger().warn(f'{self.get_name()} Invalid parameter {param.name}')
                return SetParametersResult(successful=False)
            self.get_logger().warn(f"Changing goal {self._goal_x} {self._goal_y}
{self._goal_t}")
        return SetParametersResult(successful=True)



def main(args=None):
    rclpy.init(args=args)
    node = MoveToGoal()
    try:
        rclpy.spin(node)
```

```
    except KeyboardInterrupt:
        pass
    rclpy.shutdown()

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