M. Huraira Anwer - 25100314 Sadaan Tahir - 25100224

EE 3002 L1 (Junior Design Studio - Robotics)

Spring 2025 - LAB 3

SOLUTION

Task 1: TurtleBot3 Setup [20 MARKS]

1.1 Initial Setup:

Just follow the steps in the manual.

1.2 Launching the TurtleBot3 Gazebo World:

The python script **turtlebot3_circle.py**:

```
#!/usr/bin/env python3
import rospy
from geometry_msgs.msg import Twist
import keyboard

PI = 3.1415926535897

def circle():
    try:
        # initializing a new node
        rospy.init_node('robot_cleaner', anonymous=True)
        velocity_publisher = rospy.Publisher('/cmd_vel', Twist, queue_size=10)
        vel_msg = Twist()

        # setting speed and radius
        speed = 1.0
        radius = 1.0

        # calculating the linear and angular velocities
        vel_msg.linear.x = speed
        vel_msg.linear.y = 0
        vel_msg.linear.z = 0

        angular_speed = speed / radius
        vel_msg.angular.x = 0
```

```
        vel_msg.angular.y = 0
        vel_msg.angular.z = angular_speed

        # publishing twist commands at a rate of 10 Hz
        rate = rospy.Rate(10)

        while not rospy.is_shutdown():
            velocity_publisher.publish(vel_msg)
            rate.sleep()

        vel_msg.linear.x = 0
        vel_msg.angular.z = 0
        velocity_publisher.publish(vel_msg)

    except rospy.ROSInterruptException:
        pass

if __name__ == '__main__':
    try:
        # testing the function
        circle()
    except rospy.ROSInterruptException:
        pass
```

The python script **turtlebot3_go2goal.py**:

```
#!/usr/bin/env python3
import rospy # for creating ros nodes
from geometry_msgs.msg import Twist # for sending linear and angular velocities
to the robot
from nav_msgs.msg import Odometry # contains info about robot's pose (position
and orientation)
from tf.transformations import euler_from_quaternion # from quaternion to euler
angle conversion
from math import atan2, pi

# Global variables
x = 0.0
y = 0.0
theta = 0.0
```

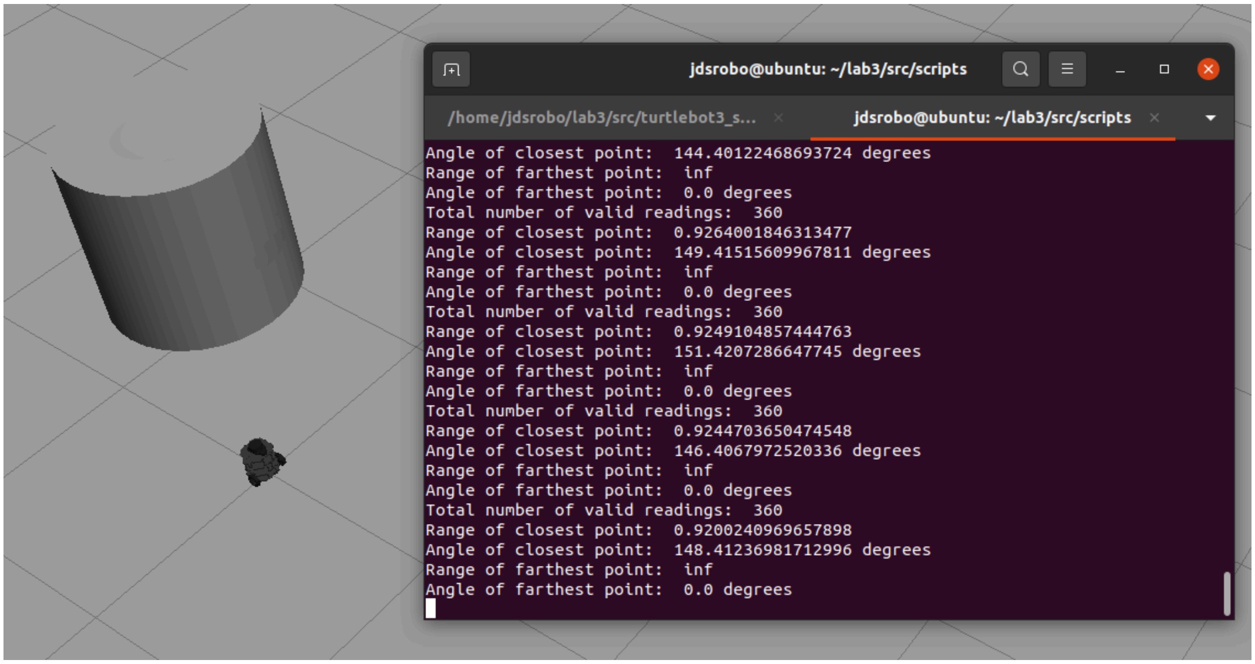
```
# callback function for the /odom subscriber. Extracts the robot's pose from the
received odometry message
def odom_callback(msg):
    global x
    global y
    global theta

    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])

# the main function that moves the robot to the set goal
def move_to_goal(goal_x, goal_y):
    # initializing the node and setting up the publisher and subscriber
    rospy.init_node('node_for_go2goal_operation', anonymous = True)
    velocity_publisher = rospy.Publisher('/cmd_vel', Twist, queue_size = 10)
    rospy.Subscriber("/odom", Odometry, odom_callback)


    while not rospy.is_shutdown():

        vel_msg = Twist()

        # computing the error in the current pose with the goal pose
        error_x = goal_x - x
        error_y = goal_y - y
        angle_to_goal = atan2(error_y, error_x)

        if abs(angle_to_goal - theta) > 0.3:
            # rotating the robot to face the goal
            vel_msg.linear.x = 0
            vel_msg.angular.z = 0.3
        else:
            # moving it towards the goal
            vel_msg.linear.x = 0.5
            vel_msg.angular.z = 0
```

```
        # publishing the twist message
        velocity_publisher.publish(vel_msg)

        # avoiding spamming the cmd_vel topic
        rospy.sleep(0.1)

if __name__ == '__main__':
    try:
        # setting the goal position
        goal_x = -1
        goal_y = 4

        # running the code
        move_to_goal(goal_x, goal_y)
    except rospy.ROSInterruptException:
        pass
```

Task 2: Using Laserscan Data [10 MARKS]

The python script **turtlebot3_laserscan.py**:

```
#!/usr/bin/env python3
import rospy
from sensor_msgs.msg import LaserScan
from math import pi

def scan_callback(msg):
    # printing the total number of valid readings
    total_readings = len(msg.ranges)
    print("Total number of valid readings: ", total_readings)

    # printing the range of closest point and its angle
    min_range = min(msg.ranges)
    min_angle = msg.angle_min + msg.ranges.index(min_range) * msg.angle_increment
    print("Range of closest point: ", min_range)
    print("Angle of closest point: ", min_angle * 180 / pi, "degrees")

    # printing the range of farthest point and its angle
    max_range = max(msg.ranges)
    max_angle = msg.angle_min + msg.ranges.index(max_range) * msg.angle_increment
    print("Range of farthest point: ", max_range)
    print("Angle of farthest point: ", max_angle * 180 / pi, "degrees")

def laser_listener():
    rospy.init_node('laser_listener', anonymous=True)
    rospy.Subscriber("/scan", LaserScan, scan_callback)
    rospy.spin()

if __name__ == '__main__':
    try:
        # testing the function
        laser_listener()
    except rospy.ROSInterruptException:
        pass
```



The **terminal output** when I used a **cylinder** for the solid object.

Task 3: Publishing Transformations [5 MARKS]

The python script **static_dynamic_tfms.py**:

```
import rospy
import tf2_ros
import geometry_msgs.msg
import numpy as np
import math

class RobotTransformPublisher:
    def __init__(self):
        rospy.init_node("robot_tf_broadcaster")

        # Broadcasters
        self.tf_broadcaster = tf2_ros.TransformBroadcaster()
        self.static_broadcaster = tf2_ros.StaticTransformBroadcaster()

        # Robot starts at (0,0,0) in world frame
        self.x = 0
        self.y = 0
        self.theta = 0  # Initial orientation
```

```python
        # Publish static transform between LiDAR and robot
        self.publish_static_lidar_transform()

    def publish_static_lidar_transform(self):
        """Publishes a static transform from robot frame to LiDAR frame."""
        static_transform = geometry_msgs.msg.TransformStamped()
        static_transform.header.stamp = rospy.Time.now()
        static_transform.header.frame_id = "robot_base"  # Parent frame (robot)
        static_transform.child_frame_id = "lidar"  # Child frame (LiDAR)

        # Set LiDAR position relative to the robot
        static_transform.transform.translation.x = 0.2  # Adjust based on LiDAR mounting
        static_transform.transform.translation.y = 0.0
        static_transform.transform.translation.z = 0.1

        # No rotation needed for LiDAR in robot frame
        static_transform.transform.rotation.x = 0
        static_transform.transform.rotation.y = 0
        static_transform.transform.rotation.z = 0
        static_transform.transform.rotation.w = 1  # Identity quaternion

        self.static_broadcaster.sendTransform(static_transform)
        rospy.loginfo("Published static transform: Robot -> LiDAR")

    def publish_dynamic_robot_transform(self):
        """Publishes dynamic transform from world frame to robot frame."""
        transform = geometry_msgs.msg.TransformStamped()
        transform.header.stamp = rospy.Time.now()
        transform.header.frame_id = "world"  # Parent frame (fixed world)
        transform.child_frame_id = "robot_base"  # Child frame (robot)

        # Update robot's position
        transform.transform.translation.x = self.x
        transform.transform.translation.y = self.y
        transform.transform.translation.z = 0.0  # Assuming 2D plane

        # Convert theta (yaw) to quaternion
        q = self.yaw_to_quaternion(self.theta)
```

```
        transform.transform.rotation.x = q[0]
        transform.transform.rotation.y = q[1]
        transform.transform.rotation.z = q[2]
        transform.transform.rotation.w = q[3]

        self.tf_broadcaster.sendTransform(transform)
        rospy.loginfo("Published dynamic transform: World -> Robot")

    def yaw_to_quaternion(self, yaw):
        """Convert a yaw angle (in radians) to a quaternion (x, y, z, w)."""
        return [0, 0, math.sin(yaw / 2), math.cos(yaw / 2)]

    def obj_coor(self, min_dis, ref_angle):
        """Compute object coordinates in world frame given LiDAR distance &
angle."""
        ref_angle = math.radians(ref_angle)

        # Transform object from robot frame to world frame
        obs_robot_coordinates = np.array([min_dis * math.cos(ref_angle),
                                          min_dis * math.sin(ref_angle),
                                          1]).reshape(3, 1)

        # Transformation matrix (robot to world)
        transformation_matrix = np.array([
            [math.cos(self.theta), -math.sin(self.theta), self.x],
            [math.sin(self.theta), math.cos(self.theta), self.y],
            [0, 0, 1]
        ])

        # Transform object to world coordinates
        obs_world = np.matmul(transformation_matrix, obs_robot_coordinates)
        return obs_world[0], obs_world[1]

    def update_robot_position(self, new_x, new_y, new_theta):
        """Updates the robot's position and publishes the new transform."""
        self.x = new_x
        self.y = new_y
        self.theta = new_theta
        self.publish_dynamic_robot_transform()
```

```
if __name__ == "__main__":
    robot_tf = RobotTransformPublisher()
    rate = rospy.Rate(10)  # 10 Hz

    while not rospy.is_shutdown():
        # Simulate robot movement (example)
        robot_tf.update_robot_position(robot_tf.x + 0.01, robot_tf.y,
robot_tf.theta + 0.01)
        rate.sleep()
```

Task 4: Mapping an Object [15 MARKS]

The python script **mapping.py**:

```
#!/usr/bin/env python
import rospy
from nav_msgs.msg import Odometry
import math
from tf.transformations import euler_from_quaternion, quaternion_from_euler
import time
from math import pow, atan2, sqrt,sin,cos
from geometry_msgs.msg import Twist
import numpy as np
from sensor_msgs.msg import LaserScan
import matplotlib.pyplot as plt
import keyboard
class my_turtlebot3_map:
    def __init__(self):
        self.x=0
        self.y=0
        self.theta=0
        rospy.init_node('turtlebot3_controller')
        self.minimun_distance=float(input("Enter Minimum distance to maintain:
"))
        self.velocity_publisher = rospy.Publisher('/cmd_vel',
                                                  Twist, queue_size=10)
        rospy.Subscriber("/odom", Odometry, self.call_back)
```

```
        rospy.Subscriber('/scan', LaserScan, self.scan_call_back)
        self.rate = rospy.Rate(25)

        self.scan_ranges=None




    def scan_call_back(self,msg):
        self.scan_ranges=msg.ranges



    def call_back(self,data):
        self.x=round(data.pose.pose.position.x,4)
        self.y=round(data.pose.pose.position.y,4)

        orientation_q = data.pose.pose.orientation
        orientation_list = [orientation_q.x, orientation_q.y, orientation_q.z,
orientation_q.w]
        roll, pitch, yaw_rad = euler_from_quaternion (orientation_list)

        yaw=math.degrees(yaw_rad)

        if yaw<0:
            yaw=yaw+360

        if yaw>360:
            yaw=yaw-360

        self.theta=math.radians(yaw)

    def move_minimum_distance(self):
        vel_msg = Twist()
        while(self.scan_ranges[0]>self.minimun_distance):
            print("here ",self.scan_ranges[0])

            vel_msg.linear.x = 0.1
            vel_msg.linear.y = 0
            vel_msg.linear.z = 0
```

```
            # Angular velocity in the z-axis.
            vel_msg.angular.x = 0
            vel_msg.angular.y = 0
            vel_msg.angular.z = 0


            self.velocity_publisher.publish(vel_msg)
            self.rate.sleep()

        vel_msg.linear.x = 0
        self.velocity_publisher.publish(vel_msg)
        self.rate.sleep()

        while(self.theta<math.radians(90)):
            vel_msg.angular.z = math.radians(20)
            self.velocity_publisher.publish(vel_msg)
            self.rate.sleep()
        vel_msg.angular.z = 0
        self.velocity_publisher.publish(vel_msg)
        self.rate.sleep()



    def rotate_minimum(self):
        min_dis=np.min(self.scan_ranges)
        vel_msg = Twist()
        print('min_dis ',min_dis, np.argmin(self.scan_ranges))
        time.sleep(1)
        while(abs(self.scan_ranges[269]-min_dis)>0.1):
            vel_msg.linear.x = 0
            vel_msg.linear.y = 0
            vel_msg.linear.z = 0

            # Angular velocity in the z-axis.
            vel_msg.angular.x = 0
            vel_msg.angular.y = 0
            vel_msg.angular.z = math.radians(-25)
```

```
            self.velocity_publisher.publish(vel_msg)
            self.rate.sleep()
            print("at 90 here",self.scan_ranges[270] )

        vel_msg.angular.z = 0
        self.velocity_publisher.publish(vel_msg)
        self.rate.sleep()
        print("exisiting")


    def obj_coor(self,min_dis, ref_angle):
        ref_angle=math.radians(ref_angle)

transformation_matrix=np.array([[cos(self.theta),-1*sin(self.theta),self.x],
        [sin(self.theta),cos(self.theta),self.y],
        [0,0,1]])


obs_robot_coordinates=np.array([min_dis*cos(ref_angle),min_dis*sin(ref_angle),1])
.reshape(3,1)

        obs_world=np.matmul(transformation_matrix,obs_robot_coordinates)
        return obs_world[0],obs_world[1]

    def rotate_goal(self,goal):
        des_angle=np.argmin(self.scan_ranges)


        error=des_angle-goal
        vel_msg = Twist()
        print("error : ",error)

        while (abs(error)>3):
            print("rotating")
            vel_msg.linear.x = 0
            vel_msg.linear.y = 0
            vel_msg.linear.z = 0

            # Angular velocity in the z-axis.
            vel_msg.angular.x = 0
```

```
            vel_msg.angular.y = 0
            vel_msg.angular.z = math.radians(error)


            self.velocity_publisher.publish(vel_msg)
            self.rate.sleep()
            des_angle=np.argmin(self.scan_ranges)
            error=des_angle-goal



        vel_msg.angular.z = 0
        self.velocity_publisher.publish(vel_msg)
        self.rate.sleep()



    def map_object(self):
        self.rotate_goal(0)
        self.move_minimum_distance()
        vel_msg = Twist()

        while not rospy.is_shutdown():

            vel_msg.linear.x = 0.1
            vel_msg.linear.y = 0
            vel_msg.linear.z = 0

            # Angular velocity in the z-axis.
            vel_msg.angular.x = 0
            vel_msg.angular.y = 0
            vel_msg.angular.z = 0


            self.velocity_publisher.publish(vel_msg)
            self.rate.sleep()

            if self.scan_ranges[270]<10:
                print("mapping ")
                obs_x_w,obs_y_w=self.obj_coor(self.scan_ranges[269],270)
```

```
                print(obs_x_w,obs_y_w)
                plt.plot(obs_x_w,obs_y_w,'.r')
                plt.draw()
                plt.xlim(0, 3)
                plt.ylim(0,3)
                plt.pause(0.00001)
            print("at 90 ",self.scan_ranges[269] )

            min_range=min(self.scan_ranges)
            if abs(self.scan_ranges[269]-self.minimun_distance)>0.1:
                self.rotate_goal(270)

        vel_msg.linear.x=0
        self.velocity_publisher.publish(vel_msg)
        self.rate.sleep()

        plt.savefig("map.png")
        print("figure is saved ")
        plt.show()

if __name__ == '__main__':
    robo=my_turtlebot3_map()
    time.sleep(1)
    robo.map_object()
```