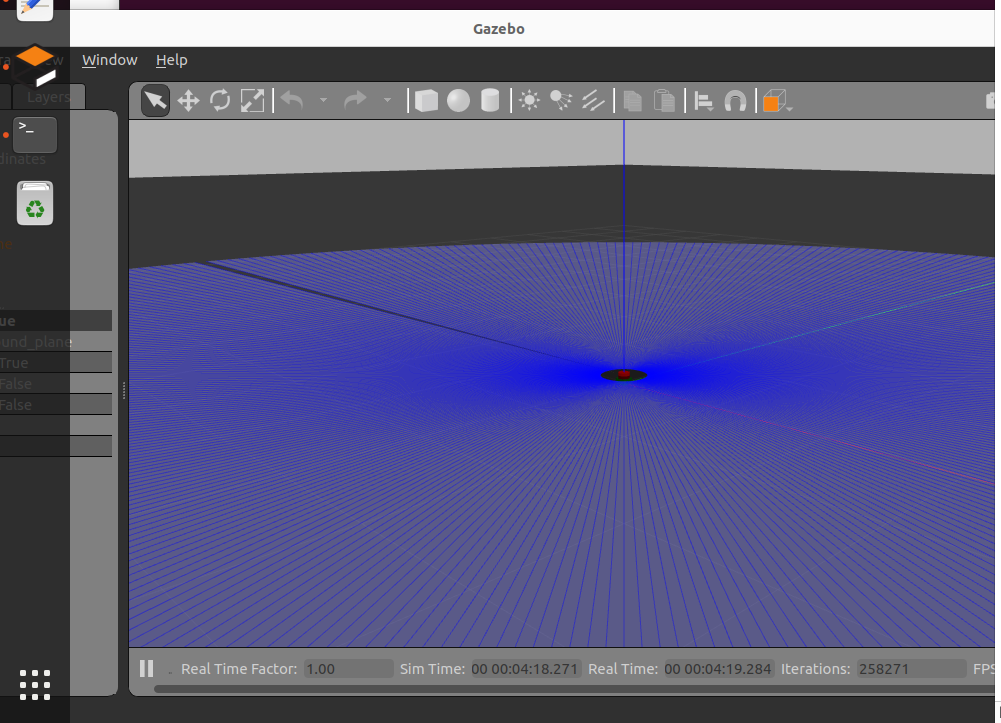
Name: Christopher Budd

Student Number: 218919068

Before Movement:



After Movement:

A screenshot of a computer

Description automatically generated

Python File: drive\_to\_goal

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.declare\_parameter('max\_vel', 0.2)

        self.\_max\_vel = self.get\_parameter('max\_vel').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}")

        self.get\_logger().info(f"maximum velocity {self.\_max\_vel}")

        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, vel\_gain=5.0, max\_vel=0.2, max\_pos\_err=0.05):

        pose = msg.pose.pose

        max\_vel = self.\_max\_vel

        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 \* vel\_gain, max\_vel), -max\_vel)

            y = max(min(y\_diff \* vel\_gain, max\_vel), -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)

        angle\_diff = math.atan2(math.sin(self.\_goal\_t - cur\_t), math.cos(self.\_goal\_t - cur\_t))

        if abs(angle\_diff) > max\_pos\_err:

            self.get\_logger().info(f"Twist {angle\_diff}")

            twist.angular.z = max(min(angle\_diff \* vel\_gain\*4, max\_vel\*5), -max\_vel\*5)

            self.get\_logger().info(f"Twist ang {twist.angular.z}")

        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

            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()