Assignment 1c Presentation

https://github.com/jleal7/AuE8230_Group3/tree/main/catkin_ws/src/assignment1c_turtlebot3/videos

AuE8230 Spring 2023

Group 3:
Jairo Leal
Rohit Ravikumar
Saachi Walia
Vasanth Seethapathi

Launch File: Step 1. Launch Gazebo with empty world and turtlebot3

```
<launch>

  <!--copy paste of turtlebot3_empty_world.launch-->
  <arg name="model" default="$(env TURTLEBOT3_MODEL)" doc="model type [burger, waffle, waffle_pi]"/>
  <arg name="x_pos" default="0.0"/>
  <arg name="y_pos" default="0.0"/>
  <arg name="z_pos" default="0.0"/>

  <include file="$(find gazebo_ros)/launch/empty_world.launch">
    <arg name="world_name" value="$(find turtlebot3_gazebo)/worlds/empty.world"/>
    <arg name="paused" value="false"/>
    <arg name="use_sim_time" value="true"/>
    <arg name="gui" value="true"/>
    <arg name="headless" value="false"/>
    <arg name="debug" value="false"/>
  </include>

  <param name="robot_description" command="$(find xacro)/xacro --inorder $(find turtlebot3_description)/urdf/turtlebot3_$(arg model).urdf.xacro" />

  <node pkg="gazebo_ros" type="spawn_model" name="spawn_urdf" args="-urdf -model turtlebot3_$(arg model) -x $(arg x_pos) -y $(arg y_pos) -z $(arg
z_pos) -param robot_description" />
```

Copy paste from turtlebot3_empty_world.launch

Launch File: Step 2. Launch circle.py or square.py based on roslaunch argument

```
23  <!--whole "group" of code only runs if condition is true-->
24  <group if="$(eval arg('code') == 'circle')">
25    <!--Set default values for v_x and w_z. Slow = 0.1 for both. Med = 0.3 for both. Fast = 1 for both-->
26    <arg name="v_x" default = "0.3"/>
27    <arg name="w_z" default = "0.3"/>
28
29    <!--node I added to control turtlebot inside of Gazebo by using Python script-->
30    <node pkg="assignment1c_turtlebot3" type="circle.py" name="PythonScript">
31      <param name="v_x" value="$(arg v_x)"/>
32      <param name="w_z" value="$(arg w_z)"/>
33    </node>
34  </group>
35
36  <group if="$(eval arg('code') == 'square')">
37    <!--ROHIT OR VASATH SET YOUR PARAMETERS HERE FOR SQUARE.PY!!! These are just copy pastes from my circle.py, feel free to change them.-->
38    <arg name="v_x" default = "1"/>
39    <arg name="w_z" default = "1"/>
40
41    <!--node I added to control turtlebot inside of Gazebo by using Python script-->
42    <node pkg="assignment1c_turtlebot3" type="square.py" name="PythonScript">
43      <!--ALSO NEED TO CALL THE PARAMETERS YOU CREATED ABOVE HERE!!!-->
44      <param name="v_x" value="$(arg v_x)"/>
45      <param name="w_z" value="$(arg w_z)"/>
46    </node>
47  </group>
48
49
50 </launch>
```

roslaunch assignment1c_turtlebot3 move.launch code:= ‘circle’ or ‘square’

Circle.py

```python
#!/usr/bin/env python3

import rospy
#from echoing gazebo, it uses the same Twist geometry_msgs as turtlesim
from geometry_msgs.msg import Twist


if __name__ == '__main__':
        try:

                #launch file created node that opened python file. Still need node to control turtlebot
                rospy.init_node('turtlesim_controller',anonymous=True)
                #to move the turtle in a circle we need to change it's linear and angular velocity. We
so need to create a publisher:
                publisher = rospy.Publisher('/cmd_vel',Twist,queue_size=10)

                vel = Twist() #move forward at a velocity of 1 and rotate at 0.5 rad/s
                rate = rospy.Rate(10)

                while not rospy.is_shutdown():
                        #slow settings:
                        vel.linear.x = rospy.get_param('~v_x')
                        vel.linear.y = 0
                        vel.linear.z = 0
                        vel.angular.x = 0
                        vel.angular.y = 0
                        vel.angular.z = rospy.get_param('~w_z')

                        publisher.publish(vel)

                        rate.sleep()

        except rospy.ROSInterruptException:
                pass
```

- Unlike turtlesim now we need to publish to /cmd_vel instead of /turtle1/cmd_vel
- Gazebo uses same Twist message as turtlesim
- Use rospy.get_param('~') to retrieve args from launch file

Slow and Medium Speed

Wheels have not saturated. Circle ends where it starts.

Slow Speed
($v_x = w_z = 0.1$)

Medium Speed
($v_x = w_z = 0.3$)

Fast Speed

Wheels have reached limit of grip. Inertia of car takes over and loose circle.

Fast Speed
(v_x = w_z = 1)

Square.py - CODE

```
    t0 = rospy.Time.now().to_sec() # current time before startinh the manouver

    #first straight
    while(current_distance <= 2):


        vel.linear.x = rospy.get_param('~v_x')
        vel.linear.y = 0
        vel.linear.z = 0
        vel.angular.x = 0
        vel.angular.y = 0
        vel.angular.z = 0
        pub.publish(vel)  # publishes the velocity
        t1=rospy.Time.now().to_sec()  # current time in the loop
        current_distance = rospy.get_param('~v_x')*(t1-t0)

    vel.linear.x = 0 # sets the vel back to 0
    pub.publish(vel)
    rospy.sleep(2)

     # first rotation first vertex
    current_angle= 0
    t2 = rospy.Time.now().to_sec()


    while(current_angle<=1.5707963267948): # angle is in radians

        vel.linear.x = 0
        vel.linear.y = 0
        vel.linear.z = 0
        vel.angular.x = 0
        vel.angular.y = 0
        vel.angular.z = rospy.get_param('~w_z')

        pub.publish(vel)
        t3 = rospy.Time.now().to_sec() # current time in the loop
        current_angle = rospy.get_param('~w_z')*(t3-t2) # current angle

```

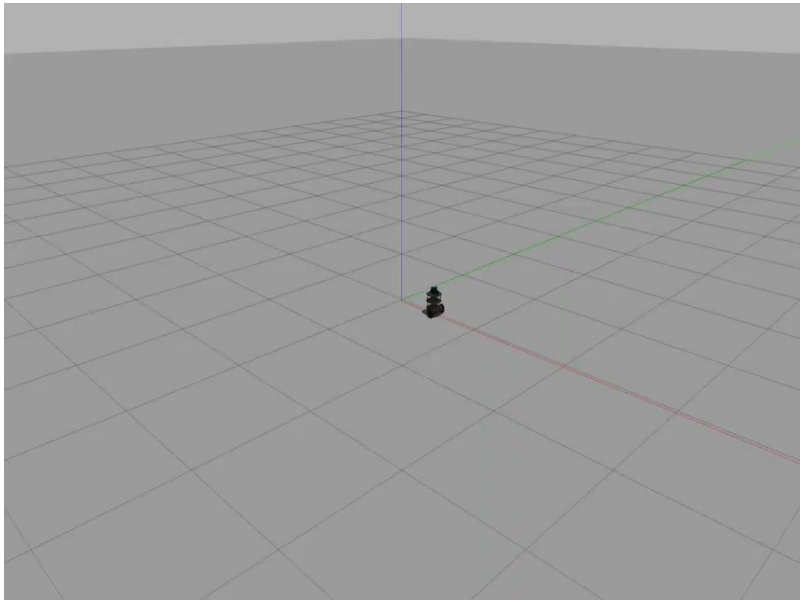
Initially rospy.sleep() was not used which led to the reaction time between the turn and the straight to be too less and it caused the bot to swerve to one side.

Each segment of the square represents one while loop and it is run sequentially.

Slow and Medium Speed

The speed with 0.1 gives the best results as at this speed there is minimum slipping of the wheels. At speeds of 0.3, we can notice that there is additional slip when the bot starts. The additional

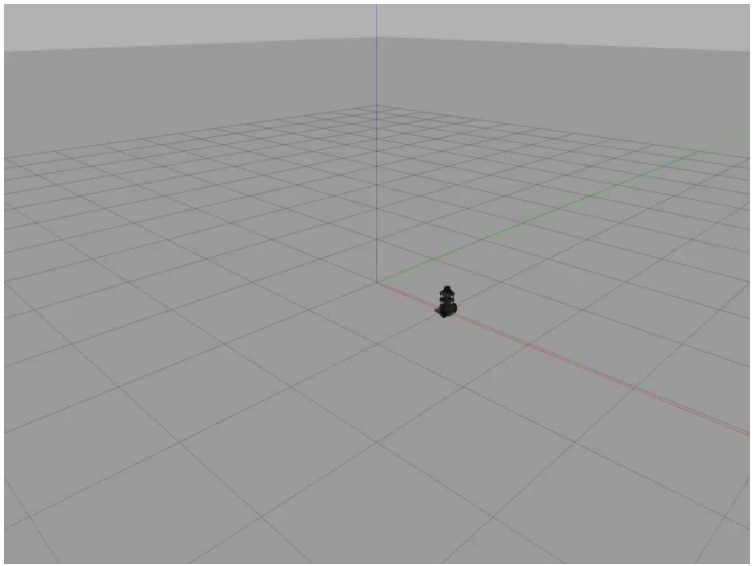
SLOW SPEED (BEST RESULT)
($v_x = w_z = 0.1$)



MEDIUM SPEED
($v_x = w_z = 0.3$)

Fast Speed

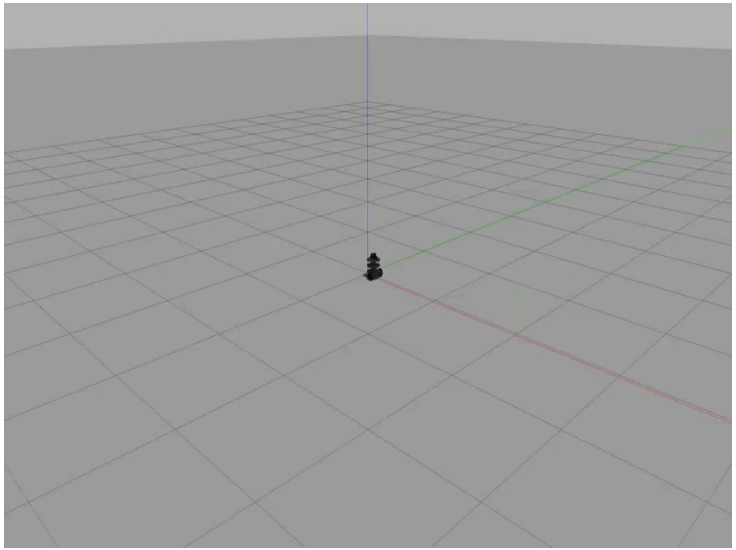
At higher speeds, the vehicle completely lost traction and often steered to one side. This is caused the initial burst of acceleration as the car reached high vels immediately, which caused the wheels to slip longitudinally



Fast Speed
(v_x = 0.5 w_z = 0.1)

Very slow speeds

Another interesting phenomena that was noticed was that at very low velocities, the car instead of performing ideally experienced a yaw and started drifting. This was because the cross wind was on and the car faced a yaw motion or lateral slip, as seen in the video below.



Very Slow speeds
(v_x = 0.05 w_z = 0.1)

Observations

The same code which worked fine in Turtlebot simulator, didn’t follow the desired trajectory in Gazebo environment.The reasons could be -

- Turtlebot simulator runs in a 2D physics engine but Gazebo uses a 3D engine.
- Turtlebot is a differential drive agent - where two wheels are mounted on an axle - where each wheel has its own velocity which is controlled independently.
- The simulation is performed based on the default simulator parameters. These settings might be need to be tuned for an optimal performance.

Conclusions

All the simulations were run on the default physics setting on gazebo, and hence the tire to ground contact friction coefficient was not touched. Hence this led to the bot following some absurd trajectories even at low speeds, due to low coefficients of friction between the two surfaces, which led the turtle to slip even while turning and align itself some degrees off from the intended path. This led to a skewed square.

As the speed increased, the turtlebot kept breaking the traction limit instantly after starting off and veered of into a new path.