CSCI 4972/6972: Algorithmic Robotics
Programming Assignment Two (PA2)
Deadline: 2pm GMT-05 February 21, 2019

This assignment will deepen your knowledge of ROS and trajectory control, and introduce you to Gazebo and Rviz. Gazebo is a robot simulator that was designed to allow roboticists to simulate robot systems before implementing them in hardware. Gazebo uses rigid body dynamics to simulation robot motions including when contact occurs with the environment and other objects, and it uses physics-based sensor models to simulate sensor readings that your robots would get in from real sensors. A goal of Gazebo is that the ROS code that drives a simulation can drive the real system without changes. Rviz is a 3D debugging tool that allows the programmer to see what the robot is seeing, thinking, and doing. It displays sensor data and state information coming from ROS in a variety of user-settable modes.

You will submit a written report answering all the questions of all parts of this assignment and you will submit a ROS package that can perform the required demos. Click here to pa2_student.zip for all parts of this assignment. Read each part of the assignment carefully, so you know which parts of the code you are expected to write and which parts you must leave alone.

Part 1: Turtlesim Trajectory Planning

Goal:

For this part, you'll need to answer everything in the questions section below and develop a ROS program to drive a turtle in turtlesim from an arbitrary initial pose to a desired location within a tolerance of 0.01 turtle-grid units.

Background:

You need to implement the publish_twist member function in *pa2_student/src/turtlejectory.py*. Do not modify any other files for this part. The three equations in 4.1.1.1 implement a steering angle for the bicycle model. The speed and steering angle must be converted into a twist message that turtlesim listens to. The launch file *turtlejectory.launch* spins up every node needed and queries you for x and then y locations in an x-term window.

Hints:

1. Use roslaunch to start all nodes needed:
   $ roslaunch pa2_student turtlejectory.launch

Questions:

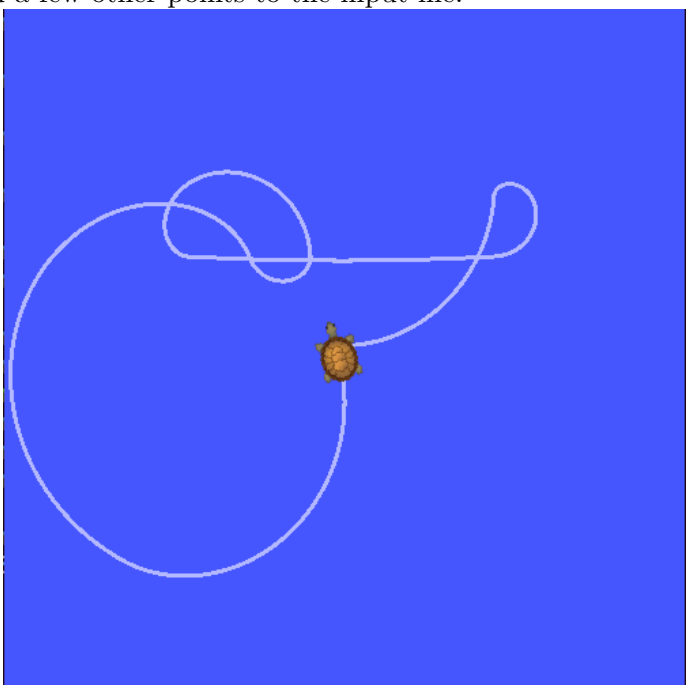
1. What parts of the twist message cause the turtle to move?

2. Let ω_z be the twist component that causes the turtle to rotate. If ω_z is positive, which way does the turtle rotate?

3. Why is python's math.tan2() better than math.tan() for this assignment?
4. How do the values of the proportional gains affect the quality of the turtle's trajectory?
5. Does turtle execute the twist messages that it hears in the world_frame or the turtle_frame. How do you know?

Demonstration:

Run your turtlejectory node and with goal inputs in this order: *[(8,8), (8,7), (3, 7), (5, 7), (4, 7), (2,2)]*. If your code is correct, turtlesim will produce the figure shown below. When your code is graded, we will add a few other points to the input file.



Part 2: R2D2 in Gazebo and Rviz

In the off chance that you didn't know R2D2 was a diff drive robot, check out the pic below. The two side legs/arms contact the ground with wheels (or belts). The third contact on the ground is a castor. Watch this video from 1:58 to 2:00 to see that R2D2 really is a diff drive robot!

Goal:

For this part, you'll gain a better understanding of Gazebo and Rviz by answering everything in the questions section below.

Background:

Do not edit any files for this section. There is no demonstration for this section, only questions to answer. The file useful for answering questions is *pa2_student/urdf/r2d2.gazebo*. r2d2.launch launches rviz and gazebo. The two world options are wall.world and empty.world found in *pa2_student/models/*.

Hints:

1. Install control packages needed to drive a robot in Gazebo:
   $ sudo apt-get install ros-kinetic-ros-control ros-kinetic-ros-controllers
   $ sudo apt-get install ros-kinetic-gazebo-ros-control

2. Launch Gazebo and Rviz (empty.world can be replaced with wall.world):
   $ roslaunch pa2_student r2d2.launch world_name:="empty.world"

3. Information for questions 1 and 2 in the following section can be found in urdf/r2d2.gazebo. This file contains the specifications that Gazebo uses to build and simulate our robot.

4. Help with understanding the Gazebo model file can be found at:
   https://www.generationrobots.com/blog/en/robotic-simulation-scenarios-with-gazebo-and-ros/.

5. If you are stuck on figuring out the moment of inertia read through:
   https://en.wikipedia.org/wiki/List_of_moments_of_inertia

6. If you find Rviz confusing this guide (although old) may be useful:
   http://wiki.ros.org/rviz/UserGuide

Questions:

1. Gazebo only needs a <collision> and <inertial> tag in order to simulate the system. What is the purpose of adding the <visual> tag?

2. Both wheel links have the same starting pose. When viewing the model in Gazebo, they are obviously not in the same place. Where in the r2d2.gazebo file do we specify the actual wheel locations? Use this information to determine the homogeneous transformation matrix between the base_link and the hokuyo_link (sensor).

3. Determine the moment of inertia for a caster wheel of radius 1m with mass 2kg and uniform density. Note that a caster wheel is a solid sphere (be sure to show your work).

4. In Rviz change the following under LaserScan and add a screen-shot to your document: set Style to spheres, set Size to 0.1, and set Channel Name to x.

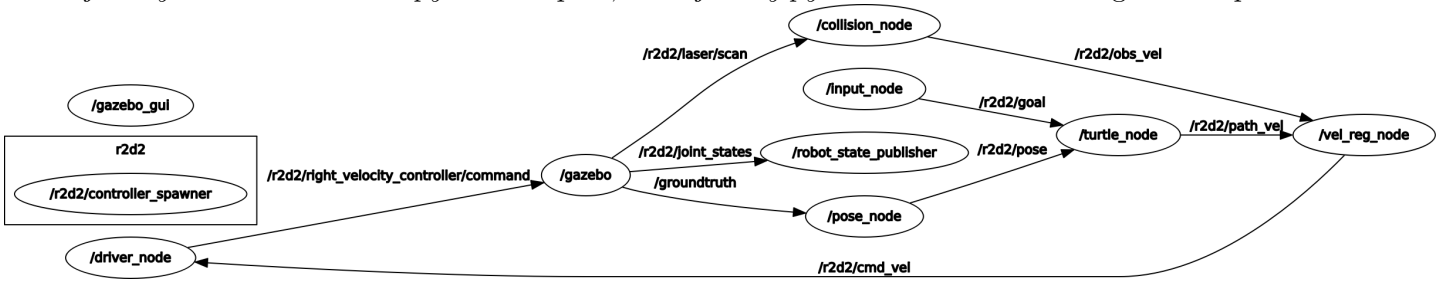
Part 3: R2D2 Trajectory Planning

Goal:

For this part, you'll need to answer everything in the questions section below and develop a demo to drive R2D2 in gazebo from an arbitrary initial pose to a desired location.

Background:

You need to implement all blank member functions in *pa2_student/src/driver.py*. The DriverNode listens for topics on "/r2d2/cmd_vel" and publishes the conversion of those messages into left and right wheel data for gazebo. Note that you will not be publishing directly to the "/r2d2/cmd_vel" topic. You need to remap "/turtle1/cmd_vel" and "/turtle1/pose" topics appropriately in *pa2_student/launch/r2d2jectory.launch*. Use the provided diagram to determine the topic names that each node should be publishing/subscribing to. Reading the diagram is as follows: ovals are nodes (which can be thought of as your python files) and lines with arrows represent topics (arrow in means subscribes to and arrow out publishes). Equation (4.5), and the equations immediately under it in Corke's text, are needed to convert the twist into left and right wheel speeds that R2D2 should use to move. The launch file *r2d2jectory.launch* spins up every node needed and queries you for x and then y locations in an x-term window (similar to part 1). Only modify *r2d2jectory.launch* and *driver.py* for this part; *turtlejectory.py* should remain unchanged from part 1.



Hints:

1. Use roslaunch to start all nodes needed:
   $ roslaunch pa2_student r2d2jectory.launch world_name:="empty.world"
2. Remap syntax can be found at:
   http://wiki.ros.org/roslaunch/XML/remap
3. To find find the topics for left and right wheel data run the launch file and use:
   $ rostopic list

Questions:

1. What topic does R2D2's right wheel subscribe to?
2. What topic does R2D2's left wheel subscribe to?
3. What is the purpose of drive.py?

Demonstration:

Run your r2d2jectory node in an empty.world with same goal inputs as in part 1. When your code is graded, we will view the simulation and add a few other points to the input file. You should not modify any file other than *r2d2jectory.launch* for this demonstration.

Part 4: R2D2 Wall Detection

Goal:

For this part, you'll need to answer everything in the questions section below and expand the demo in Part 3 to stop R2D2 when 1 meter from a wall using the planar lidar sensor already installed on R2D2.

Background:

You need to implement all blank member functions in *pa2_student/src/collision_avoidance.py*. The CollisionNode listens for the planar lidar topic and publishes to "/r2d2/obs_vel" to stop R2D2 if near a wall. When an obstable is detected a Twist() message with 0 linear velocity in the x direction is published. Otherwise, when an obstacle is not present, a NON-ZERO linear velocity in the x direction is published. The launch file *r2d2jectory.launch* spins up every node needed and queries you for x and then y locations in an x-term window (similar to part 1). Only modify *collision_avoidance.py* for this part.

Hints:

1. Use roslaunch to start all nodes needed:
   $ roslaunch pa2_student r2d2jectory.launch world_name:="wall.world"
2. Information about LaserScan messages can be found here:
   http://wiki.ros.org/navigation/Tutorials/RobotSetup/Sensors
3. Find the planar lidar topic with:
   $ rostopic list

Questions:

1. What topic does the planar lidar publish to?
2. What is the x, y, z offset of the planar lidar and how does it effect its range detection?
3. How would you get a list of thetas that correspond to each range detected by the lidar?

Demonstrations:

Run R2D2 in wall.world and drive it to a goal point that must go through the wall (ex: 6, 0). R2D2 is expected to stop 1 meter from the wall and move no further.

Submitting

You will submit a write-up and a ROS package, both in the same zip file and sent to trinkle@gmail.com by the deadline. There is no set length for the write-up. Use as much space as you need to answer all questions thoroughly. Make sure to include an acknowledgment statement describing any resources that you used or assistance that you received. The format of your submission is:

- Please name the zip file you submit as: pa2-*name*, where *name* is replaced by your first name.
- Your name, date, programming assignment title should appear on the write-up and in the comments of your code.
- List the questions asked above and give your answers.
- List and briefly describe useful things you discovered, that go beyond the questions of this assignment.
- Format the written part of your assignment as a PDF file
- Put your pa2 package code and the pdf of your write-up into a zip file and email to trinkle@gmail.com.

Grading

Submissions will be graded according to the following rubric:

Success of demonstrations	60%
Correctness of answers to questions	25%
Clarity of comments and PEP8 compliance of code	15%