W23 CMPUT 412/503 Exercise 2

Ros Development and Kinematics

Primary TA's: Xiao and Montaser
Written report due: Feb 5th
Oral demonstration: Feb 8th/10th

Description

In this exercise, you will learn about different components of ROS, the Basic application of Robotic Kinematics and Odometry.

For this exercise, you will need to work in a **group of 2**. However, the oral demo will still be done **individually,** so please make sure you understand all of the code and work for your submission. The only restriction is that undergrad can only team up with undergrad and grad students can only team up with grad students. Students in different lab sessions can team up with each other if you two can coordinate your schedule.

Please fill out this sign up sheet with your group members
if you don't have a native ubuntu 20 install, we recommend teaming up with someone who does

Objectives

1. Understand the various components of ROS
2. Understand mobile robot kinematics
3. Understand mobile robot odometry
4. Get familiar with the most common and useful ROS commands
5. Learn how to drive DuckieBot with your own code
6. Hands-on practice with ROS nodes, services, and topics

Hints

Familiarizing yourself with these commands will be of great help for this exercise:

```
rostopic/rosnode/rosmsg/rosservice list/info
```

Prerequisites

Make sure your robot is already updated using the command:

```
dts duckiebot update csc229XX
```

If not, please make sure you update docker-compose before running the command:

```
python3 -m pip install docker-compose --upgrade
dts duckiebot update csc229XX
```

Part One Procedure

1) Read over the ROS/Concepts wiki pages related to the following list and understand the concepts. They will be helpful for completing this exercise and *there might be related questions on them during your oral demo:*
    - Node
    - Topic
    - Service
    - Message
    - Bag
2) Fork the template repository to build your code and run in your duckiebot
    - Follow the instructions in Unit C-2.1 to 2.6
    - This will teach you how to use DTROS to create your own node
    - **Write your code to store your robot's hostname in a variable**
        - You should use this variable when publishing and subscribing to topics rather than hard coding your robot's hostname. This is to make sure your code can also run on other robots**.**
    - Create your own subscriber to the camera and print out the size of the image
    - Publish the image or your modified image to another customized topic created by yourself under a certain frequency
    - Submit a screenshot in your report that view the camera image in your customized topic using `rqt_image_view`
    - Submit a screenshot of your code for this part in your report. You don't have to put this part in your repo.
3) Read the instructions in Unit E-3.1, including exercises 27 and 28.
   Go through the following parts and learn how these tasks can be completed at the coding level. You don't need to submit individual code for this part. However, all these behaviors will be some sub-tasks you need to complete in the next part of Exercise 2.
   Please include your answer to the questions below in your written report.
    - The unit of coordinate system used in this exercise will be meters
    - Put your robot on top of DuckieTown at location x = +0.32 and y = + 0.32 we set up in CSC229, facing the positive y direction of the world frame
        - **This is the origin of your initial robot frame**
        - **What is the relation between your initial robot frame and world frame? How do you transform between them?**
    - Straight line task

    - Drive your robot straight forwards for 1.25 meters and then backwards for 1.25 meters. Measure how far your robot has traveled to decide when to stop and keep track of your location with respect to your initial robot frame as well.
    - **How do you convert the location and theta at the initial robot frame to the world frame?**
    - Use a ruler to measure the distance between your actual location and the desired location
    - **Can you explain why there is a difference between actual and desired location?**
    - **Which topic(s) did you use to make the robot move? How did you figure out the topic that could make the motor move?**
    - **Which speed are you using? What happens if you increase/decrease the speed?**
- Rotation task
    - Send a command to have your robot spin 90 degrees (pi/2) clockwise on the spot and track the actual degrees your robot rotates
    - **How did you keep track of the angle rotated?**
    - **Which topic(s) did you use to make the robot rotate?**
    - **How did you estimate/track the angles your DuckieBot has traveled?**
- **Now you know how to do rotational and straight movements, think about how to drive the robot autonomously in a circle**
- Does your program exit properly after you finish all the above tasks? Is it still running even if you shut down the program? If this is the case, think about how to resolve this issue.
- **Bag** is file format in ROS for storing ROS messages. Go through the following instructions to learn how to use a ROS bag to store information and plot the collected data
- Record your odometry information with respect to the world frame.
- Find a useful way to display the information from the ROS bag.
    - For example, load the rosbag in a python script and print the values, plotting x, y, theta vs. time using rqt_plot/matplotlib, or plotting x vs. y using matplotlib. *We might have a related question in the demo.*
- Read the following instructions about how to implement a ROS service.

Part Two Procedure

This Is the coding part of Exercise 2 that you will need to submit in your repo. You will implement a program to complete a multi-state task.

When executing the task at different states, you must implement a ROS service and call it to change the LED light color of your duckiebot. This will serve as a visual indication of different

states. For example, if your LED lights are all set to blue in State1, then they will change and show green when the robot is in State2. You have the freedom to pick any colors you like as long as they are visually distinctive between states. Please be clear in your report the mapping from states to LED colors that you use.

Your implementation must use at least 2 ROS nodes that interact with each other (i.e., your main node and the node that defines your service).

When all the states terminate, all of the nodes you implemented should be shut down.

Steps and robot task to be implemented:

1. Start your robot at the global coordinate (x = 0.32 y = 0.32) and mark it (0, 0, 0) as your robot initial frame (i.e., home position)
    a. Always keep track of your location in both the robot frame (Xr, Yr, Theta) and world frame (Xw, Yw, Thetaw). Continue writing your odometry information with respect to the world frame to a ROS bag until your robot finishes the entire task. **Note that theta will be in the range [0, 2pi] or [-pi, pi]**
2. **State1**: Do not move for 5 seconds (i.e., stay still), indicate this with an LED color of your choice
3. **State2**: Rotate right (clockwise) in position for 90 degrees, then move straight forward 1.25 meters. Rotate left (counterclockwise) in position for 90 degrees, then move straight forward 1.25 meters. Rotate left(counterclockwise) for 90 degrees again, move straight forward 1.25 meters.
    a. Feel free to separate this state into sub-states when you are coding, but they should maintain the same LED color as one state.
4. Wait for 5 seconds (notice this is the same as State1 - change LED color to show this)
5. **State3**: Move back to your robot's initial position and orientation (i.e., home position)
6. Wait for 5 seconds (i.e., State1)
7. **State4**: Perform a clockwise circular movement around the lower half of the duckietown, try to approach your original initial position.
8. Print the total execution time for your entire task and the final location with respect to the world frame. **What is the final location of your robot as shown in your odometry reading? Is it close to your robot's actual physical location in the mat world frame?**
9. Save your odometry (in the world frame) to your rosbag and shut down all your nodes
10. Display the bag file information in a way of your choice and include your written report

Deliverables

Include in your written report:

- Your answers to the questions included in the procedures above
- A screenshot of your robot's camera image view in your own customized topic

- A screenshot of your code that shows how you acquire and send images using ros topics
- A video of your Duckiebot performing the task defined in part two
- A link to your uploaded rosbag file (download your bag file from the dashboard and then upload to your google drive or repo - **make sure it is public to us**)
- A video that captures the print/plot of your odometry information stored in the bag file
- A short write-up on what you implemented, what you learned, what challenges you came across, and how you overcame the challenges

On eClass you will submit:

- To submit your website written report, first publish it to the public then upload a pdf printout of your published report on eClass by the deadline (don't worry if the formatting is weird)
- A link to your course website with the written report for this exercise
- A link to your course Github repository with all the code from this exercise neatly in a subfolder

Extra Reading

For multi-state tasks, it is helpful to understand the concept of finite-state-machine and SMACH.
It is **not mandatory nor recommended** to use SMACH **for this exercise** since it adds unnecessary complexity, but it is a helpful resource for your future work when you need to handle more complex robot behavior. **There might be a question about the basic finite-state-machine concept during your oral demo.**

Resources

You can use any material on the internet as long as you cite it properly and fully understand its logic. You are encouraged to collaborate with your labmates and if you develop a solution together please acknowledge who you worked with in your written report.