Middle East Technical University Department of Computer Engineering

CENG 460

Introduction to Robotics

Fall 2020-2021

Programming Assignment 1 - Robotic Treasure Hunt

Due date: 04.01.2021, Monday, 23:55

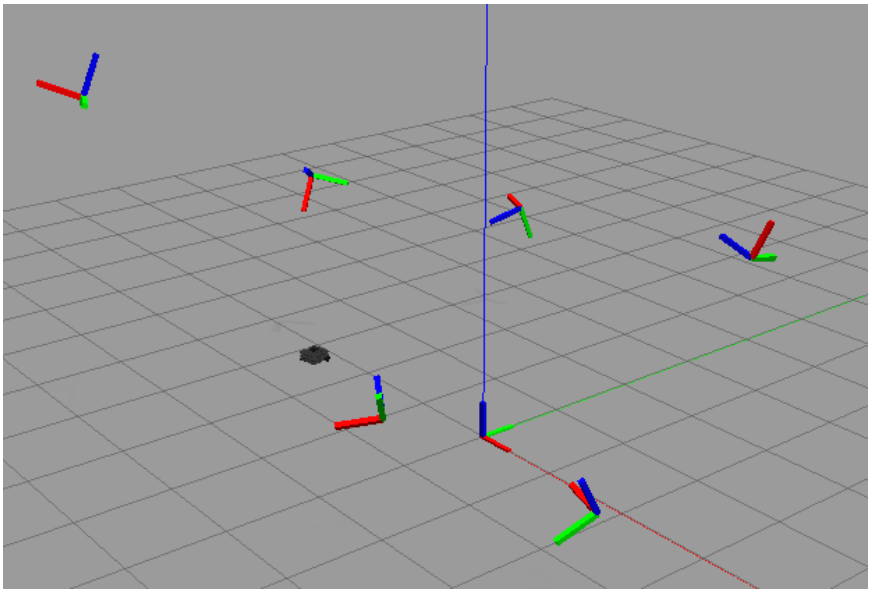

Figure 1: A view of the robot and the treasure after the assignment is complete.

The purpose of this assignment is to get you familiar with ROS and assess your implementation skills of basic control algorithms and coordinate transforms.

1 Overview

In this assignment, you will manipulate a differential drive robot with ROS and look for $N < 15$ *treasures* with global transforms T_0, $T_1 \dots T_N$ scattered around the environment. After physically navigating to each T_i with your robot (such that you are close to it in the xy plane), you will report its transform to a seperate entity called *the referee*. If the transform information is correct and you are indeed sufficiently close to the treasure, the referee will provide you with some *clue* transforms ${}^{g}T_h$. Each ${}^{g}T_h$ represent a transform of treasure h with respect to the pose of treasure g. Note that g and h may not necessarily be the index of the treasure you had traveled to. As you obtain more clues, you will have to come up with a way to infer locations of unfound treasures.

2 Preliminaries

You are **highly encouraged** to read the following material to ensure you have a good understanding of ROS API before you review the later parts of this document:

- A Gentle Introduction to ROS, Jason M. O'Kane: Read the *entire book*. (It is not that long, most of it is code and pictures, you can finish it in a single afternoon. If you are seriously in a hurry, you may skip chapters 4, 5, 9, 10.) The book is in C++ and you will use Python 2.7 in this assignment, but the C++ API is very similar to Python already, so just try to understand concepts at a pseudocode level, that is sufficient.
- rospy: rospy_tutorials: for you to quickly come up with Python versions of examples in the book.
- rospy: Overview: mostly the same as the previous item, but with more depth. Try to make note of things that are *different* from roscpp here, such as subscribers, service handlers creating their own threads (tip: try to understand what is thread-safe and what is not, so that you don't come across hard to solve race-condition bugs).
- tf2: Tutorials: A well-known ROS package that we will use in this assignment. Basic knowledge about listeners is enough.

3 The Setup

You can either use Riders or use ROS in your local machine.

ROS in Local Machine

If you are using ROS in your local machine, it is suggested that you use a virtual machine as ROS loves to fill your PC with unwanted .deb packages. It tends to whine a lot too if you have a distro installed other than Ubuntu 18.04 LTS or your default Python version is 3. Therefore, **make sure you install ROS melodic on Ubuntu 18.04 LTS** using the information on this link (The reason we are not using the latest is for compatibility with Riders). After that go into your workspace `src` directory and then type the following in the terminal (I assume you have git, type `sudo apt install git` in the terminal otherwise):

```
> git clone https://gitlab.com/cemonem/ceng460hw1.git
> cd ..
> catkin_make
> source devel/setup.sh
> roslaunch ceng460hw1 treasure_hunt.launch
```

You should see your robot moving for a while and then stop. After everything is set, start working on your assignment by modifying `ceng460hw1/src/solution.py`. (A big indicator of you not having properly read the preliminaries would be not understanding what these commands were. Read them!)

Riders

Since Riders starts its own gazebo instance, there are custom launchfiles for you to execute your scripts. These are essentially the same with launchfiles for local machines, they just do not start `gazebo_ros` node. Also, you work in a new package called `my_ceng460hw1`, because of how Riders operates (as far as the author of this text understands, unfortunately no documentation is available regarding these):

- Riders essentialy runs a ROS installed docker instance (a virtual machine) for you. Every time you enter an editor, a new instance is started. But what happens to the files you may want to keep from your old docker instance? The answer Riders came up with is that files under `/workspace/src` are kept in a version control system. **Everything else is deleted and reinstalled!** Therefore, you must make sure that you

do your work in `/workspace/src`, and whenever you want to save, make sure you click on the horse sign on the left and then click on "Publish Project". What you do here is essentially commiting files to the internal version control system Riders has. **Otherwise, your files might be potentially lost** (Riders also keeps a temporary commit if you leave the editor and asks you if you want to keep the changes next time you enter. If you say no, your files are deleted for good!). The most surefire way however is to keep a local copy of `/workspace/src/my_ceng460hw1/solution.py` all the time.

- The above item has certain exceptions. One is that `/workspace/src/_lib` directory is not under version control (**so you do not want to keep your files here!**). The purpose of this directory is to keep ROS packages pulled from remote repositories specified in `.ride.yaml` (can be accessed by clicking the horse sign on the left - "Edit Project Configuration..."). The `ceng460hw1` package is also here for you to use the custom messages it defines. Every time you open a new editor instance, this directory will be emptied, filled with packages pulled from remote repositories, and then will be built.

With these warnings out of the way, the setup is as follows:

1. Create a new project in Riders, and open it.
2. Click on the horse sign on the left - "Edit Project Configuration..." and open `.ride.yml`.
3. Copy the contents of `ride.yml` provided to you and paste on `.ride.yml`. Save it by pressing `Ctrl+s`.
4. Click on the horse sign on the left - "Build Project". `/workspace/src/_lib/ceng460hw1` should have been downloaded and built. Click on the file icon to see your workspace (`/workspace`).
5. From the top left menu (≡) click on - "View" - "View Command Palette".
6. Type "ROS: Create Catkin Package" and then press enter. Enter the name as "my_ceng460hw1". For dependencies, do not type anything, just press enter. Your package should have been created in `/workspace/src/my_ceng460hw1`.
7. From the top left menu (≡) click on - "Terminal" - "New Terminal". Enter the following commands to fill your package.
```
> cp src/_lib/ceng460hw1/riders_package.xml src/my_ceng460hw1/package.xml
> cp src/_lib/ceng460hw1/riders_CMakeLists.txt src/my_ceng460hw1/CMakeLists.txt
> cp src/_lib/ceng460hw1/src/ceng460hw1_utils.py src/my_ceng460hw1/
> cp src/_lib/ceng460hw1/src/solution.py src/my_ceng460hw1/
```
8. Click on horse sign - "Build Project". This is to build your new package and source it so that `rosrun` can see it.
9. Click on horse sign - "Start" and start the Gazebo instance of Riders.
10. On the terminal, type `roslaunch ceng460hw1 treasure_hunt_riders.launch` to run your nodes. (The terminal might hang for a little while after you start the simulation, do not worry). You should see your robot moving for a while and then stop.
11. Click on horse sign - "Stop" after you are done. Also kill ROS master by pressing Ctrl-C while focused on terminal.
12. If everything is set, **do not forget to save your project!** Click on horse sign - "Publish Project", name your initial commit and save the changes.
13. As a precaution, click on "Stop Editor" on the topright and then try re-entering your project to see if your changes persisted.

You can start working on `/workspace/src/my_ceng460hw1/solution.py` after everything is set up.

4 Specifications

In order to give you priorities in your implementation, most of the tasks are seperated and given **tentative** percentages. Again, these percentages **are subject to change** and are just given to you so that you can deduce which task should be attacked first.

4.1 Easy Mode (25%)

In this mode, the referee does not require your robot to be near the treasures. You just need to send the correct global transform of each treasure to get their clues and find all the treasures at the end. *The transform of the first treasure is guaranteed to be identity transform and your robot spawns with the same global pose as well.*

To run your solution against easy mode, use launchfiles `treasure_hunt_easy_mode.launch` for running ROS locally and `treasure_hunt_easy_mode_riders.launch` for Riders.

To send your treasure transforms, use the `/spot_announcement` service of the `referee` node. The service uses a custom service message format which can be found in `srv/SpotAnnouncement.srv`. The important fields for request and response messages are as follows:

SpotAnnouncementRequest:

- geometry_msgs/TransformStamped `spot`: should be filled with the information regarding the global transform and the id of the treasure. Use the `child_frame_id` field to indicate which treasure you are sending the transform of (as an integer in $[0,\ N]$ converted to string). All of the other fields are ignored.

SpotAnnouncementResponse:

- `bool success`: is true if the information sent is correct (if not in easy mode, the distance between the robot and the treasure should also be less than 0.05 (5 cm) in the xy plane).
- geometry_msgs/TransformStamped[] `clues`: is filled with m number of clues, $m \in [0, N]$ (yes, you might receive no clues at all). Each clue ${}^{g}T_{h}$ is encoded into each element of `clues` as follows:
  - `header.frame_id`: is assigned to string representation of $g \in [0,\ N]$.
  - `child_frame_id`: is assigned to string representation of $h \in [0,\ N]$.
  - `transform`: is assigned to the value of ${}^{g}T_{h}$.

After you successfully submit a global transform of a treasure, its frame should physically pop up in Gazebo.

4.2 Moving the Robot To A Point with a P Controller (15%)

Fill in the stub `move_to(self, x_goal, y_goal, P_vel, P_angular, rate=20, tolerance=0.02)` with the algorithm described in "Robotics,Vision and Control: Fundamental Algorithms in MATLAB" 2nd edition, p. 102 (Section 4.1.1.1 Moving to a Point), where `x_goal, y_goal` is the goal point, `P_vel` is the velocity gain and `P_angular` is the heading gain. You may terminate the algorithm after the distance of your robot to the goal point is less than `tolerance`. You should send your velocity commands to the robot at the regular speed of `rate` Hz so that you do not cause instability with commands of erratic rate (using `rospy.Rate()` is recommended).

You will use the topic `/cmd_vel` to publish your commands. This topic has messages of type geometry_msgs/Twist, in which `linear.x` and `angular.z` fields are taken for radial and angular velocities of the differential drive. All of the other fields are ignored.

For positional feedback, the `/tf` topic publishes the global transform of the robot as the transform from frame `"odom"` to `"base_footprint"`. It is recommended that you use `tf2` API to listen to this topic.

In Gazebo, wheel slipping and other non-ideal phenomena like static friction do exist. Therefore low gain values may end up with your robot moving too slowly or not moving at all, and high gain values may end up with your robot constantly sliding or completely diverging from the goal point with large arcs. You need to experiment and come up with "nice" gain values. **Report these values in your `solution.py`. Your robot should converge to a point closer than 5 (5 m) in less than 1 minutes in simulation time, with default arguments for `rate` and `tolerance`.**

4.3 Rotating the Robot Towards A Point with a P Controller (5%)

You will notice that rotating the robot towards a point and then going towards it produces slower but much more stable results. Implement the stub `rotate_to` in the same way as section 4.2, but with zero radial velocity (`tolerance` value should correspond to the angular distance in radians for this method). **Your robot should converge in less than 1 minutes in simulation time, with default arguments for `rate` and `tolerance`.**

4.4 Summing It All Up (50%)

Use the implementations of the above tasks to complete the assignment with non-easy launchfiles. If you want a challenge, try to find all the points as fast as possible by tuning your P values or come up with a faster completely different low level travel algorithm of your own. The author of this text found all 7 of the treasures of the example given to you within 3 and a half minutes, without trying too hard. **The definitive time limit for finding all of the treasures is N minutes in simulation time.**

4.5 Neatness of Code (5%)

This homework will be graded with whitebox method, therefore you should pay special attention to comments etc. within your code, especially if you are doing some complicated transform calculation or hard to figure out ROS magic. Encapsulate your code well and guard it against race conditions.

5 Other Regulations and Submission

- You must use Python 2.7 with ROS Melodic for this assignment.
- All of the transforms in this assignment respect the right-hand rule and all of the angles are in radians. Distance values are in meters.
- If you want to use any other Python module in `solution.py` or want to use another ROS package, or any topic other than `/spot_announcement, /tf, /cmd_vel` you must first ask publicly on the course forum before using it, we may or may not allow it.
- Making any other use of `tf` queries other than between `"odom"` and `"base_footprint"` is forbidden. You can register frames to tf and ask about frames it can calculate, but obviously we want you to implement that feature yourself, therefore you are not allowed to use it in that way.
- You may use functions in `tf.transformations` and `numpy` to convert between quaternions and transform matrices. There is also an util file we have provided that fills messages with transform matrices etc.
- The default `solution.py` is filled with some example `rospy` implementations. You may use them however you like, but it is suggested that you review the material in Section 2 at a sufficient level that you already know what these do in a glance.
- You should rename your `solution.py` as `solution_eXXXXXXX.py` and submit it to the platform we announce in the course forum.

- **This is an individual assignment. Using any piece of code that is not your own is strictly forbidden and constitutes as cheating. This includes friends, previous homeworks, or the Internet. The violators will be punished according to the department regulations.**
- Please submit your questions publicly on the course forum if your question does not include parts of your solution. If your question does contain some of the solution, send an email to onem@ceng.metu.edu.tr.