CS460/560: Homework 1 Software Instructions

Aravind Sivaramakrishnan

Fall 2018

1 Getting the software

The repository for this homework is public, and GitHub does not allow the creation of private forks for public repositories. To get access to private repositories on GitHub, sign up for the GitHub student pack. Then, follow the instructions below (*These instructions have been adapted from Jacques Dafflon's gist on GitHub.*).

- `git clone --bare git@github.com:aravindsiv/comprobfall2018-hw1.git`
- Create a new private repository on GitHub and name it `comprobfall2018-hw1`.
- `cd comprobfall2018-hw1.git`
- `git push --mirror git@github.com:<your_username>/comprobfall2018-hw1.git`
- `cd ..`
- `rm -rf comprobfall2018-hw1.git`
- `mkdir -p catkin_ws/src`
- `cd catkin_ws/src`
- `git clone git@github.com:<your_username>/comprobfall2018-hw1.git`
- `git remote add upstream git@github.com:aravindsiv/comprobfall2018-hw1.git`
- `git remote set-url --push upstream DISABLE`

2 High-level overview

The software that we have provided for this assignment consists of the following meta-packages (a *meta-package* is simply a collection of ROS packages grouped by functionality):

- **turtlebot**: Packages pertaining to bringing up the turtlebot.

- **turtlebot_deps**: Packages that are dependencies.
- **turtlebot_simulator**: Packages that load the turtlebot into Gazebo.

Please note that whatever code that you run **must not** be inside any of these packages - any code that you modify inside these packages will incur a grade penalty. We will run a git diff between the last commit that the TAs provided and your code to enforce this. You will have to create a new package and write your code inside that package.

In addition to the Gazebo world and launch files, we have provided map files located inside `turtlebot_maps/map_x.txt`. Each of these files contains the following information, each separated by a $---$.

- The co-ordinates of the endpoints of the world. These co-ordinates define the walls that surround the world. The robot should not attempt to move beyond these walls.
- The co-ordinates of the endpoints of the different polygonal obstacles that are present in the world. The robot must not collide with any of these obstacles.
- The co-ordinates of 10 different start and goal positions that you must run your implementations of A* and DFA* on. Each line corresponds to a single start and goal position pair.

3 Running the software

- Run `roscore` to start the ROS Master.
- In a separate terminal, set the goal state for your experiment as follows: run `rosparam set goal_state [x,y]`. For example, if your goal state is [2.5,6.0], you would run `rosparam set goal_state [2.5,6.0]`. Note that this can also be done using Python or C++, but you will have to rerun the following steps if you do so.
- Launch the launch file using the command `ROBOT_INITIAL_POSE="-x <x of start position> -y <y of start position>" roslaunch turtlebot_gazebo turtlebot_world.launch world_file:= <directory of world_x.world>`. If you are running the experiment corresponding to `map_x.txt`, you must provide the absolute path to the world file `world_x.world`. The world files are located inside the `turtlebot_gazebo` package, under the `worlds` directory.
- In a separate terminal, run the controller service server by running `rosrun turtlebot_ctrl turtlebot_control.py`
- If the above steps executed without any error (usually indicated by the absence of red colored text output to one of your terminal screens), you are good to go.

4 Subscribing to `turtlebot_state`

This section assumes you know what ROS topics are. The ROS tutorials page on ROS topics is a good starting point.

/`turtlebot_state` is a ROS topic that is published to by the node `turtlebot_state_pub` which is provided by us. The message description is defined by `turtlebot_ctrl/TurtleBotState.msg`. You can view the message description by running the command:

```
rosmsg show turtlebot_ctrl/TurtleBotState
```

The message has three components: two float32 values indicating the current x and y co-ordinates of the turtlebot respectively, and a Boolean value indicating whether the goal position specified in the launch file has been reached or not.
You will have to write a ROS subscriber (in C++ or Python) that will subscribe to this topic to query the current position of the turtlebot, and whether the goal position has been reached or not. Your solution path will be treated as invalid if the `goal_reached` parameter does not return the value `True`.

5 Communicating with `turtlebot_control`

This section assumes you know what ROS services are. The ROS tutorials page on ROS services is a good starting point.

`turtlebot_control` is a ROS service provided by us, which uses the service description defined by `turtlebot_ctrl/TurtleBotControl.srv`. You can view the service description by running the command:

```
rossrv show turtlebot_ctrl/TurtleBotControl
```

The srv file is divided into two parts, separated by a $---$. The top part of the srv file is the **request**, while the bottom part of the srv file is the **response**.
You will have to write a ROS client (in C++ or Python) that will send a request in the form of an (x, y) point that you wish the robot to move to (The z co-ordinate that you provide will be ignored).

Once the request is received, the `turtlebot_control` node will turn the turtlebot towards the point that you have provided, and will attempt to steer the turtlebot along a straight line path between its current location and the target point. If the straight line path is collision free, the robot should reach the target in a few seconds, and you will receive a response `True`. If the robot encounters a collision, the `turtlebot_control` node will shut down automatically, and you will have to relaunch it along with Gazebo.

6 Recommended planning pipeline

The planning pipeline recommended for this Homework is as follows:

- Load the world while setting the start and goal positions as described in Section 3.
- Write your code for A* and DFA* that plans a path, defined as a set of discrete waypoints $(x_1, y_1) \cdots (x_k, y_k)$, between the start and goal states.
- Once these waypoints have been obtained, execute them inside the Gazebo environment by calling the `turtlebot_control` service.