BLG456E, Robotics, Fall 2014

Assignment #2: Path Following

Lect. Dr. Damien Jade Duff
Res. Asst. Çağatay Koç

Due: Oct 25 (Saturday) 11:59pm

Assignment summary: The task is to write a node that:

(1) Subscribes to `/waypoint_cmd` to find out the next waypoint in the route that the robot needs to travel. This route is provided by a separate referee package (`trajectory_referee_456`).

(2) Uses the **tf** library to find the current location of the robot according to the robot's odometry.

(3) Sends commands to `/cmd_vel_mux/input/navi` to go to each point in the path.

You do not need to make use of the robot's range sensors. Odometry will be enough for this task.

Submission details: An archive file of the source code. The archive will contain the following files (replacing **{student ID}** with your ID):

```
a2_{student ID}/src/a2_{student ID}_node.cpp
a2_{student ID}/CMakeLists.txt
a2_{student ID}/package.xml
```

It should be possible to compile the package by placing the files into a catkin workspace and running the catkin compilation.

Step-by-step

Basic knowledge

It is advised that you go over the TF tutorials at

http://wiki.ros.org/tf/Tutorials . In particular, “Introduction to TF” and “Write a TF Listener (C++)”.

Also, read up on transforms and reference frames.

Setup your workspace

See the step-by-step instructions in the assignment 1 handout. You will need Ubuntu 14.04, ROS Indigo, the Turtlebot Gazebo simulator and your own workspace. Recall that if you created your catkin workspace in the directory `~/catkin_ws` then you need to run the following command to allow commands like `rosrun` work with it:

```
source ~/catkin_ws/devel/setup.bash
```

You may simplify your life by appending this command to your `~/.bashrc` file so that you don't have to retype it frequently.

Setup the referee

From the assignment description on Ninova, download `trajectory_referee_456.zip` and unzip it into your catkin workspace source directory (e.g. `~/catkin_ws/src` if your workspace is `~/catkin_ws`). Then, run `catkin_make` in your catkin workspace. E.g. if your catkin workspace is `~/catkin_ws`:

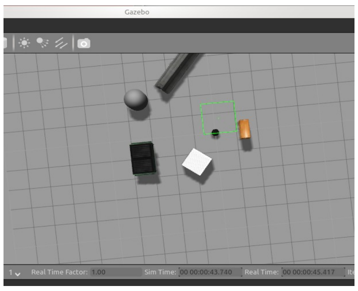
```
cd ~/catkin_ws && catkin_make
```

Run the simulator & run the referee

As with the previous assignment, you will want the simulator running.

```
roslaunch turtlebot_gazebo turtlebot_playground.launch
```

Recall: You should see the Turtlebot robot in the Gazebo GUI amongst a scattering of objects:



Run the referee:

```
rosrun  trajectory_referee_456 referee.py route1
```

The referee has several different routes called `route1`, `route2` `route3`, and `route4`. The last argument on the command line determines which is used. `route4` can be considered too difficult for this assignment, but exists for those who like to be challenged.

The referee will start publishing a destination pose to the topic `/waypoint_cmd` – you can examine what it is sending using `rostopic echo`.

Generally you will want both the simulator and the referee running while you test your code.

Recall: In order to see what your robot sees, in a new terminal window run

```
roslaunch turtlebot_rviz_launchers view_robot.launch
```

In particular, you should be able to make rviz subscribe to a new topic provided by the referee called `/visualization_marker_array`, by clicking Add... Marker Array. This topic allows you to visualize the full route that your robot needs to follow (though `/waypoint_cmd` only receives the next point at any given time).

Recall: In order to drive the robot around in its simulated world:

```
roslaunch turtlebot_teleop keyboard_teleop.launch
```

How to obtain current state and waypoints

TF:

In order to obtain the robot's current pose you will use the TF package to obtain the transform between the robot's original pose (which also happens to be the map frame) and its currently known pose according to the odometry. These poses are known to the TF transform manager as frames `/odom` and `/base_link`. See the file `print_state.cpp` in the `trajectory_referee_456` package for an example of how this is done.

In order to use TF well, you can use the following commands to view what is going on:

```
rosrun tf view_frames && evince frames.pdf
rosrun tf tf_monitor /odom
rosrun tf tf_monitor /base_link
rosrun tf tf_echo /base_link /odom
```

Waypoints:

Run the following commands to find out more about the waypoint message that is being published:

```
rostopic info /waypoint_cmd
rostopic echo /waypoint_cmd
rosmsg show geometry_msgs/Transform
```

The `trajectory_referee_456` package also contains a file called `print_waypoint.cpp`. If you examine this files you will see how your own program can obtain the target pose from the `/waypoint_cmd` topic.

Example code:

You can also run these example programs like this (assuming you have set up your workspace and built the referee):

```
rosrun trajectory_referee_456 print_state
rosrun trajectory_referee_456 print_waypoint
```

Create your package

The package that you create for your assignment will be called `a2_{student ID}` where {student ID} is replaced by your student ID. You can create the package with dependencies using the `catkin_create_pkg` command.

Your package may need the following dependencies:

```
std_msgs roscpp tf geometry_msgs trajectory_msgs
```

Your source code will live inside the file `a2_{student ID}/src/a2_{student ID}_node.cpp` inside the `src` directory of your package. As with assignment 1 you will need to edit the CMakeLists.txt file to add the following lines:

```
add_executable(a2_{student ID}_node src/a2_{student ID}_node.cpp)
target_link_libraries(a2_{student ID}_node ${catkin_LIBRARIES})
```

It is highly recommended that you examine the template code in `print_state.cpp` and `print_waypoint.cpp`. This code allows you to retrieve the current pose of the robot and its desired pose (the next waypoint).

As with assignment 1, your node will publish to twists (geometry_msgs/Twist) on the `/cmd_vel_mux/input/navi` topic.

Building: Inside your workspace (e.g. `~/catkin_ws)` directory, run:

```
catkin_make
```

Running:

```
rosrun a2_{student ID} a2_{student ID}_node
```

Class competition

The referee measures the time it takes to travel the route.

Once the referee does start keeping track, if you are using rviz you will see the reached markers dim, and the referee will also print to the terminal as each way-point is reached.

As a class competition, performance on the routes supplied (route1, route2, route3 and route4) will be collated and the winning student might get some chocolate, or, if not a fan of chocolate, carrots or something.

Marking criteria

- Creation of a package & node.
- Reading and processing waypoints.
- Reading and processing current state.
- Publishing movements that take into account the waypoint and current state.
- Publishing movements that intelligently take into account the waypoint and current state.
- Publishing movements that get the robot to the target position.
- Publishing movements that get the robot to the target orientation.
- Publishing movements that get the robot to the target orientation & position (pose).
- Clear code.
- Lack of errors.

Bonuses avaiable for:

- Use of Proportional (P) control or another advanced controller.
- Figuring out and making use of the `/route_cmd` topic also provided by the referee.
- Doing other cool things.

Assignments not submitted according to requirements will not be evaluated.