Project 3 (100 Points)

This project will walk you through programming your robot to move. You will start by inspecting the Duckiebot to figure out which topics and messages control the wheels. Then, you will create a new package with a few nodes that make your Duckiebot move in different ways.

1. Start your robot and start keyboard control on your laptop. Run
   $ dts start_gui_tools <duckiebot_name>
   and use rqt_graph to inspect the nodes/topics. Use a combination of this graph and rostopic to find the various topics and messages that send commands to the wheels. Record the answers to the following questions:
   1. What node converts joystick commands to robot commands?
   2. What do you think car_cmd_switch_node does? If you're not sure, start the lane following demo as see how the graph changes (or refer to project 2 graphs).
   3. What does the output of car_cmd_switch_node look like? Move the robot around and watch it change. Describe the components of the message (data types) and their purpose.
   4. At some point, the command is converted from linear/angular velocity to commands for the left and right wheels. Which node does this?
   5. During Project 2, you calibrated the wheels. Which node do you think accepts that calibration value and adjusts commands as needed?
2. Start your container and create a new package for this project in the "packages" folder of your repo. Note that you will need to add duckietown_msgs as a dependency. This can be added to the end of the catkin_create_package command that you used before.
3. Create a new node in that package that publishes a command to the output of car_cmd_switch_node. **Do NOT include the vehicle name in the topic**, use a namespace in your launch file as shown below instead. This will allow your code to be run on any vehicle. Select values that make your robot drive in a ~1m diameter circle. You will have to guess in this first trial, we will test in the next few steps. Make sure that you make this node executable (chmod +x <node_name>)
4. Create a launch file for your node such that it looks something like this:

```
<launch>
   <group ns="$(env VEHICLE_NAME)">
      YOUR NODE GOES HERE
   </group>
</launch>
```

   This will start your node inside the namespace for your robot. Make sure to commit these files.

At this point, you can exit the container. We can do the rest of this project from your Linux system.

5. We need to build the container with this new package in it, but we want to build it on the robot this time instead of on your laptop. Run:
   $ dts devel build -H <duckiebot_name>.local
6. Now we need to run the code. Start the container on your robot with:
   $ dts devel run -H <duckiebot_name>.local -s -M --cmd bash
   The -s flag copies your repo to the /code folder on the robot so that it can be mounted by the -M flag and you do not need to rebuild for every small change.
7. Make sure your robot can move using keyboard control. Note that when your code starts running, **the robot will move** and may be unpredictable. Be ready to stop your code with ctrl-c and take over with keyboard control. Your robot will continue doing the last command it was told until you stop it with keyboard control, even if you ctrl-c the launch file. Start your launch file on the robot (same terminal as above command) to test your code.
8. Revise your code until your robot does exactly one circuit of a 1 meter circle. Have your robot stop at the end. Record your best run. You will be graded on precision at the resolution your instructor can see on the video you turn in. Once you are within a precision you find acceptable you can move on.
   There are a few methods to revise your code:
   1. The easiest way to revise your code is to continue editing on your laptop and sync the files on your robot as needed using the same program as dts uses with the -s flag: rsync. To do this, run this command:
      $ rsync --archive <path/to/repo> duckie@<duckiebot>.local:/code/
      NOTE: in <path/to/repo> do NOT include the trailing "/". It should look something like:
      /home/user/my_repo
   2. To avoid using rsync manually, you can exit your container and rerun the dts devel run command from step 6. This is a little tedious but a good way to test that your code works and will be committed to git properly when you are finished.
   3. You can also edit the code directly on your robot. Your repo is transferred to the /code folder on the robot. Note that you will have to sync these changes with your computer to turn in on git, so while this can be fast to test you run the risk of forgetting to turn in completed code.
   4. Finally, you can clone your repo onto your robot and run dts devel build/run through ssh without the -H command, just like you do on your laptop. If you do this and use ssh keys for git, do NOT use the default duckiebot keys because 1) everyone will have complete access to your repo and 2) someone already managed to do that so github will not allow you to upload your public key.
9. Create a second node and launch file that makes your robot move in a 1 meter square. Again, tune it to be as precise as possible and have it stop after one circuit.

Turn in to Dropbox:

Answers to questions:

1. What node converts joystick commands to robot commands? (2 points)
2. What do you think car_cmd_switch_node does? (2 points)
3. What does the output of car_cmd_switch_node look like? Move the robot around and watch it change. Describe the components of the message (data types) and their purpose. (2 points)
4. At some point, the command is converted from linear/angular velocity to commands for the left and right wheels. Which node does this? (2 points)
5. During Project 2, you calibrated the wheels. Which node do you think accepts this calibration value and adjusts commands as needed? (2 points)
6. What difficulties did you have in tuning your robot to make a pattern? (5 points)
7. Do you think you could make your robot reliably drive around a circular/oval track like you have at home without any feedback from the camera? Why or why not? (5 points)

Also turn in:

8. Video of your best attempt at the 1m circle pattern
9. Video of your best attempt at the 1m square pattern
10. Link to your repo and the tag that corresponds to this code

Rubric:

10 points: Package creation, repo and tag link
25 points: Circle node/launch file
10 points: Circle precision (mostly graded on uniformity of shape, not scale)
25 points: Square node/launch file
10 points: Square precision (mostly graded on uniformity of shape, not scale)
20 points: Answers to questions