COMP417
Assignment 1

Posted: Monday, Oct 2, 2017
Due: Thursday, Oct 12 at 11:59pm, 2017
Weight: 12.5% of your final mark

In this assignment you will write a feedback controller that enables a ground robot, which obeys differential drive dynamics, to follow a wall. This assignment aims to make you develop experience with the following concepts:

(1) Robot Operating System: its architecture and its publisher-subscriber model, which abstracts away the details of distributed computation and message passing from the eyes of the robot programmer. Also, its main visualization tool, called rviz.

(2) Controlling the yaw of a ground robot via PID control

(3) Processing 2D laser measurements

(4) The gazebo simulator, which is currently the most popular simulator in the robotics community.

Step 1: Setting up ROS

This depends on your operating system and your environment setup. In this class we are assuming that either you are running Ubuntu 14.04+ on a computer on which you have sudo privileges, or you are working directly from the Trottier machines. If you prefer the former you will have to set up Ubuntu and ROS yourself (Follow the Tutorial: http://wiki.ros.org/ROS/Tutorials for installing and learning the basics of ROS). Note that there is currently no support for compiling ROS on Windows and there is only experimental support for MacOSX. Ubuntu however is fully supported. If you are installing from scratch please choose Ubuntu 16.04 and ROS kinetic. Note that we do not have docker containers for this setup. If you are unsure about the installation process or you need help/pointers, please post your question to mycourses or contact the TA during office hours.

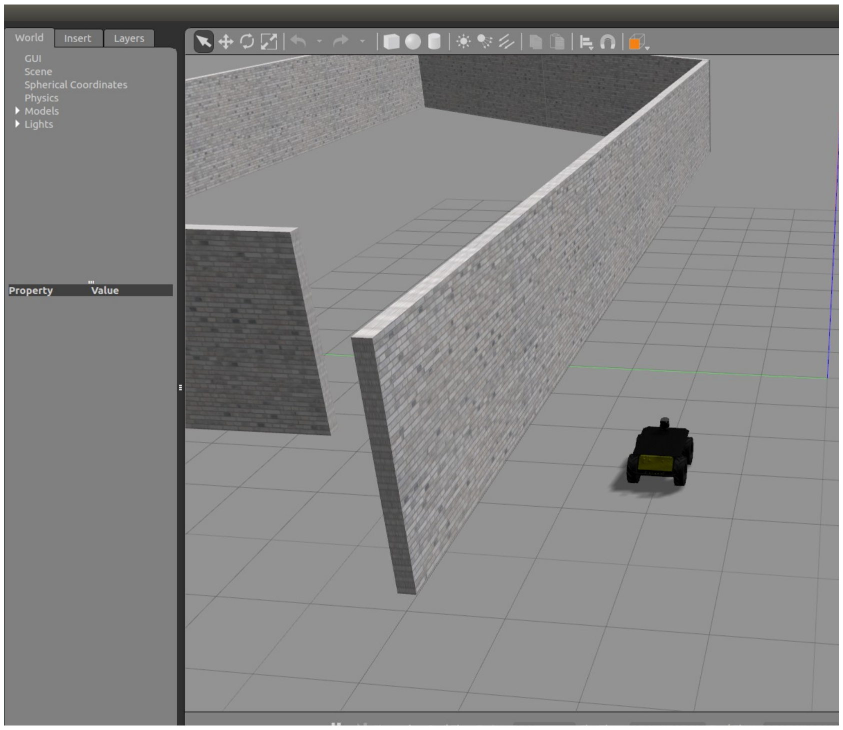
If you are planning to work from the Trottier machines, Ubuntu and ROS are already installed and available to you. We suggest you do NOT try to login remotely to the Trottier machines to run your wall following code because you will need to run the gazebo simulator, which can be CPU-intensive. You could avoid remote generation of the simulator's graphics by separating the gazebo client from the gazebo server, but we have not tested how well this works, so we simply recommend that you do NOT work remotely on the Trottier machines. If you decide to do this despite our suggestions, please make sure whenever you ssh into the Trottier machines that nobody else from the class is running their simulator. You can check this by doing the following after you login to a cs machine:

```
$ ps -aux | grep gz
$ netstat -ntlp | grep gz
```

If any relevant results show up that means someone else is running their gazebo simulation so you should try to login to another machine to avoid interfering with their simulation environment (servers, ports, configuration, parameters etc.)

Step 2: Get and compile the provided starter code

We have created a few worlds consisting of sequences of walls and a simulated ground robot for you for the purposes of this assignment. The gazebo environment looks like this:



We are also providing starter code for this assignment. This starter code is provided in the form of ROS packages in your workspace. Please follow these instructions for setting up your workspace:

Make sure the following lines are at the end of your ~/.bashrc file:

```
export ROS_HOME=~/.ros
source /opt/ros/kinetic/setup.bash
source ~/comp417_ws/devel/setup.bash
```

Then actually create your workspace:

```
$ mkdir -p comp417_ws/src
$ cd comp417_ws/src
$ catkin_init_workspace
```

Download the starter code package ('**assignment1.zip**') and extract all the packages into comp417_ws/src directory.

Then compile it:

```
$ cd comp417_ws
$ catkin_make
```

If this last command results in errors you might need to install additional packages. In the Trottier machines this will most likely NOT be an issue, because we've tested it. If you are working from a personal laptop or desktop, however, you might need to install the following packages:

```
$ sudo apt-get install ros-kinetic-control-toolbox ros-kinetic-joystick-drivers ros-kinetic-realtime-tools ros-kinetic-ros-control ros-kinetic-gazebo-ros-control ros-kinetic-ros-controllers
```

If you still get errors after installing them please post a question on mycourses or email us as soon as possible. If compilation goes smoothly then run the following commands:

Bring up a world with walls in the gazebo simulator

```
$ roslaunch wall_following_assignment gazebo_world.launch world_name:=walls_one_sided
```

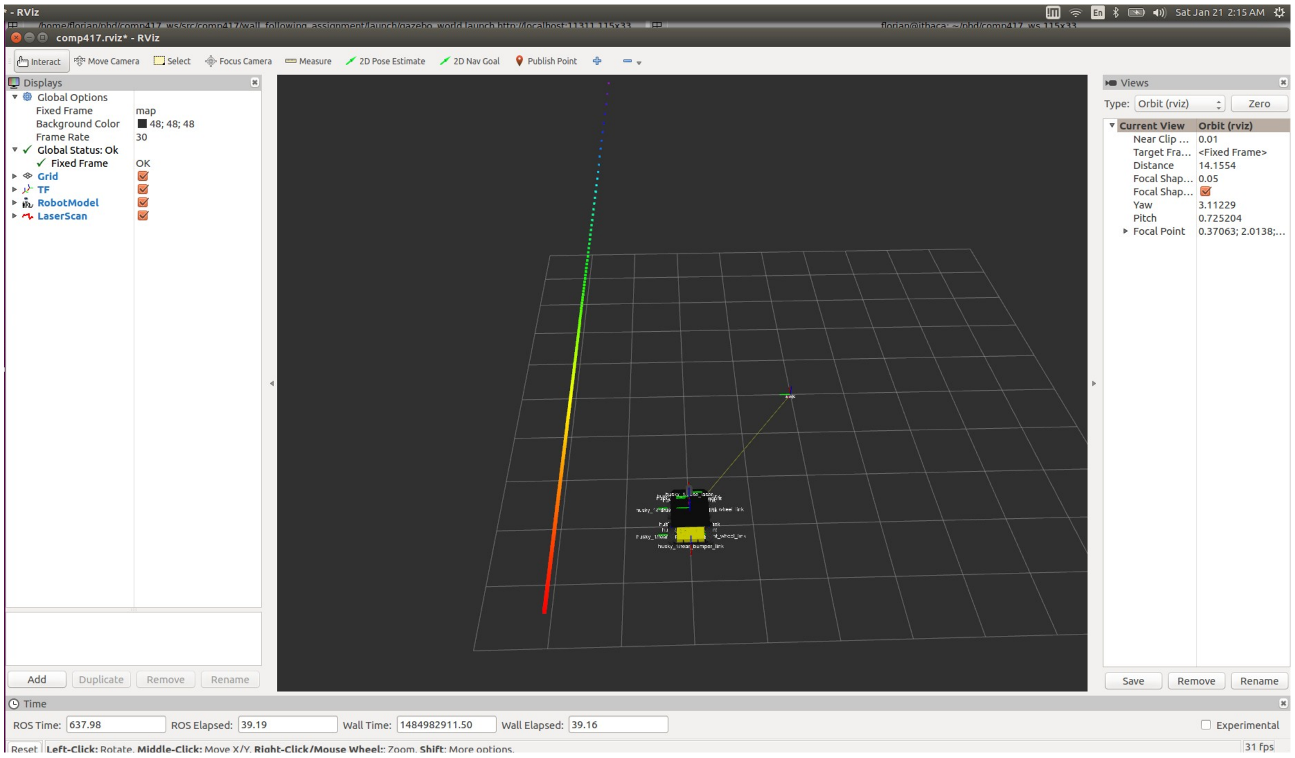
Bring up the robot in that world

```
$ roslaunch wall_following_assignment husky_follower.launch
```

The only error that should appear after these two commands is that no joystick can be found. If another error is printed, please let us know. If these two commands go well this should make the gazebo image shown above appear. Then run the ROS simulator, call rviz:

```
$ rosrun rviz rviz
```

And then go to File > Open config > and select the config file comp417_ws/src/comp417/wall_following_assignment/resources/comp417.rviz
You should see something like this:



Global Options	
Fixed Frame	map
Background Color	■ 48; 48; 48
Frame Rate	30
✓ Global Status: Ok	
✓ Fixed Frame	OK
Grid	✓
TF	✓
RobotModel	✓
LaserScan	✓

Type: Orbit (rviz)

Current View	Orbit (rviz)
Near Clip ...	0.01
Target Fra...	<Fixed Frame>
Distance	14.1554
Focal Shap...	0.05
Focal Shap...	✓
Yaw	3.11229
Pitch	0.725204
Focal Point	0.37063; 2.0138;...

ROS Time: 637.98 ROS Elapsed: 39.19 Wall Time: 1484982911.50 Wall Elapsed: 39.16 ☐ Experimental

Left-Click: Rotate. **Middle-Click:** Move X/Y. **Right-Click/Mouse Wheel:**: Zoom. **Shift:** More options.

The rainbow-colored line is actually a set of points detected by the simulated 2D laser. The other line and frames represent the tree of reference frames that the system is aware of. If all of this goes well then you will be ready to proceed to the next section which is the essence of the assignment and write your controller code to make the robot move.

Step 3: Implement and test a wall following controller [12.5 pts]

The input to the robot will be laser scan messages from the robot's simulated laser scanner. See here http://docs.ros.org/api/sensor_msgs/html/msg/LaserScan.html for the ROS message definition. Also, make sure you have understood these ROS tutorials about the publisher-subscriber model of distributed computing:
http://wiki.ros.org/ROS/Tutorials

We have provided starter code for Python and C++ in the files

```
comp417_ws/src/comp417/wall_following_assignment/python/wall_follower.py
comp417_ws/src/comp417/wall_following_assignment/src/wall_follower_node.cpp
comp417_ws/src/comp417/wall_following_assignment/include/wall_following_assignment
/pid.h
```

Choose your language and edit the appropriate files. Specifically:

(A) [1.5 pt] Compute the cross-track error based on each incoming laser scan and publish it under the topic name /husky_1/cte

(B) [5 pts] Populate the PID class based on the provided API. For each incoming laser scan issue an angular velocity command to the robot at the topic /husky_1/cmd_vel, based on the output of the PID controller. You need to follow the wall on the LEFT of the robot, as it is placed in its initial configuration.

(C) [2 pts] Create a dynamic reconfigure server for tweaking your controller's parameters in real time. Follow the instructions presented here:
http://wiki.ros.org/dynamic_reconfigure/Tutorials
Add at least three parameters for the PID gains and run

```
$ rosrun rqt_reconfigure rqt_reconfigure
```

to tweak the PID parameters manually. A set of parameters is considered good enough and the run is considered successful if the robot does not collide with the wall and completes at least ¾ of a full circuit around the left wall.

(D) [4 pts] For each of the two simple wall worlds in wall_following_assignment/worlds/, namely: walls_one_sided.world, walls_two_sided.world, do the following:

Launch your wall following controller like this:

```
$ roslaunch  wall_following_assignment wall_follower_python.launch
```

or

```
$ roslaunch  wall_following_assignment wall_follower_cpp.launch
```

according to your language of choice.

[1pt] Use a desktop recording program such as recordMyDesktop or something similar to record a video of your robot while it is following the wall. Be sure to include any failure cases.

[1pt] Record the cross-track error published by your node as follows

```
$ rosbag record /husky_1/cte
```

and after your robot's run is done, convert the recorded messages in the bag into a text file like so:

```
$ rostopic echo -b file.bag -p /husky_1/cte  > cross_track_error.csv
$ rosbag play file.bag
```

What to submit

Create a folder by name : Assignment1

Put all the following files into this folder (Assignment 1):

1. Your code in wall_following_assignment, from steps 3A, 3B, 3C. File "wall_follower_node.cpp" or "wall_follower.py".

2. Two videos, one for demonstrating the robot's navigation in the world walls_one_sided.world and another video for the world walls_two_sided.world as explained in step 3D. The videos should be named as follows:

   FirstName_LastName_McGillID_walls_one_sided.[mp4/avi]
   FirstName_LastName_McGillID_walls_two_sided.[mp4/avi]

3. Similarly, two csv files from step 3D. They should be named:
   FirstName_LastName_McGillID_walls_one_sided.csv
   FirstName_LastName_McGillID_walls_two_sided.csv

Compress the Assignment1 folder into Assignment1.zip and upload it under assignment 1 on MyCourses.

The point of including a video of your controller in your submission is not just for us to easily examine your code. It's also so that you can easily show your work later on to classmates/coworkers/employers. It becomes part of your portfolio, as opposed to a forsaken piece of work that collects dust. It is also worth noting that, due to the ROS abstraction layer, if you wanted, you could run your feedback controller on a real Husky robot, without modifying much expect tweaking the parameters a bit.

How we will test your code
We expect your code to run on the Trottier machines. If it does not you will lose marks. We expect to be able to compile it using catkin_make as shown above in step 2. Once we compile your code we will run the following:

```
$ roslaunch wall_following_assignment gazebo_world.launch
world_name:=walls_one_sided
$ roslaunch wall_following_assignment husky_follower.launch
$ roslaunch wall_following_assignment wall_follower_[language].launch
```

and we will see what your code does. We will then terminate all your code and try out the other world:

```
$ roslaunch wall_following_assignment gazebo_world.launch
world_name:=walls_two_sided
$ roslaunch wall_following_assignment husky_follower.launch
$ roslaunch wall_following_assignment wall_follower_[language].launch
```

We reserve the right to try out your code in other similar worlds. We also reserve the right to test it using different desired distances away from the wall. The default desired distance is 1 meter away, and the default forward speed is 1m/s. We will test your code with different nearby parameters. Your code will also be examined for correctness, style and efficiency. If we see something egregiously wrong or error-prone we will subtract marks. We recommend that you come by during office hours or email us if you are unsure of your implementation.