CSC477 – Introduction To Mobile Robotics
assignment 1, 15 points
due: october 7, 2019, at 6pm

Course Page: http://www.cs.toronto.edu/~florian/courses/csc477_fall19

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

- Robot Operating System: its architecture and its publisher-subscriber model, which abstracts away the details of distributed computation and message passing from the robot programmer. Familiarity will also be developed with ROS's main visualization tool, called `rviz`.
- Controlling the yaw of a ground robot via PID control
- Processing 2D laser measurements
- The gazebo simulator, which is currently one of the most popular simulators in the robotics community.

Setting up ROS: If you are planning to work from the MCS lab machines in Deerfield Hall (e.g. DH2020, DH2010, DH2026), Ubuntu and ROS are already installed and available to you. If you want to work from your own machine, we are assuming that you are running Ubuntu 18.04+ on a computer which you have sudo access, then please make sure the following are installed:

- Ubuntu 18.04+
- ROS (melodic) http://wiki.ros.org/melodic/Installation/Ubuntu
- Gazebo 9 from http://gazebosim.org/ or from `sudo apt-get install gazebo9`
- numpy
- scipy
- git
- Python 2 or 3
- any video screen recorder for assignment submission (e.g. recordmydesktop https://wiki.ubuntu.com/ScreenCasts/RecordMyDesktop)

Get and run the starter code: We have created a few simulated worlds consisting of sequences of walls and a simulated ground robot for you for the purposes of this assignment. The Gazebo simulation 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. Make sure the following lines are at the end of your `/home/[username]/.bashrc` file:

```
export ROS_HOME=~/.ros
source /opt/ros/melodic/setup.bash
source ~/csc477_ws/devel/setup.bash
```

Then actually create your workspace:

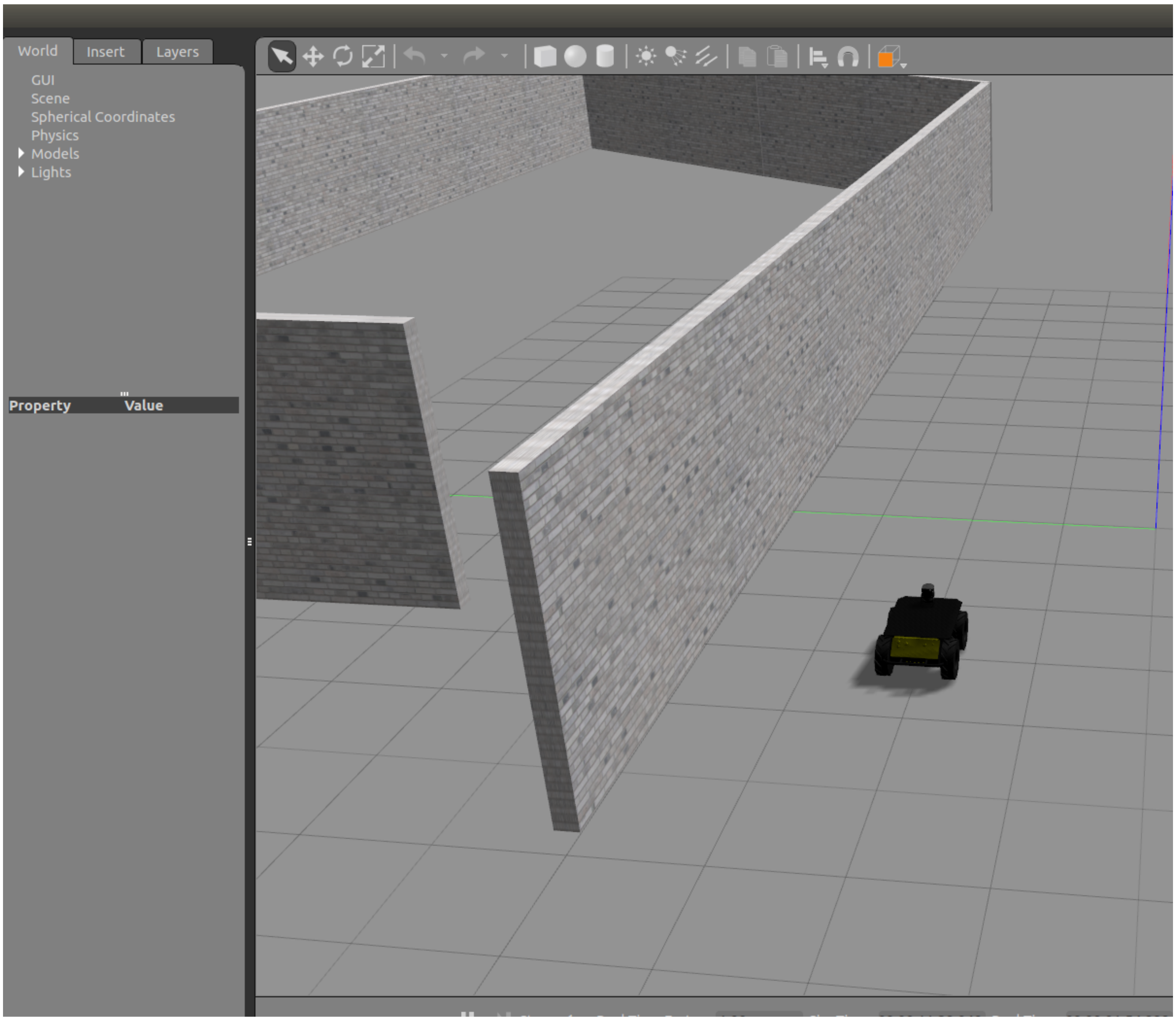


Figure 1: Gazebo environment with walls only on one side of the robot.

```
mkdir -p csc477_ws/src
cd csc477_ws/src
catkin_init_workspace
```

In `csc477_ws/src` download the starter code:

```
git clone https://github.com/florianshkurti/csc477_fall19.git
```

Then compile it:

```
cd csc477_ws
catkin_make
source ~/.bashrc
```

If this last command results in errors, you might need to install additional packages. Please post your questions on Quercus if this is the case and you don't know what to do. In the MCS lab machines this will most likely not be an issue. If you are working from a personal laptop or desktop, however, you might need to install the following packages:

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

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

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 (husky) 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 in Fig 1 shown above appear. Then, call rviz, which is the default visualization system for ROS:

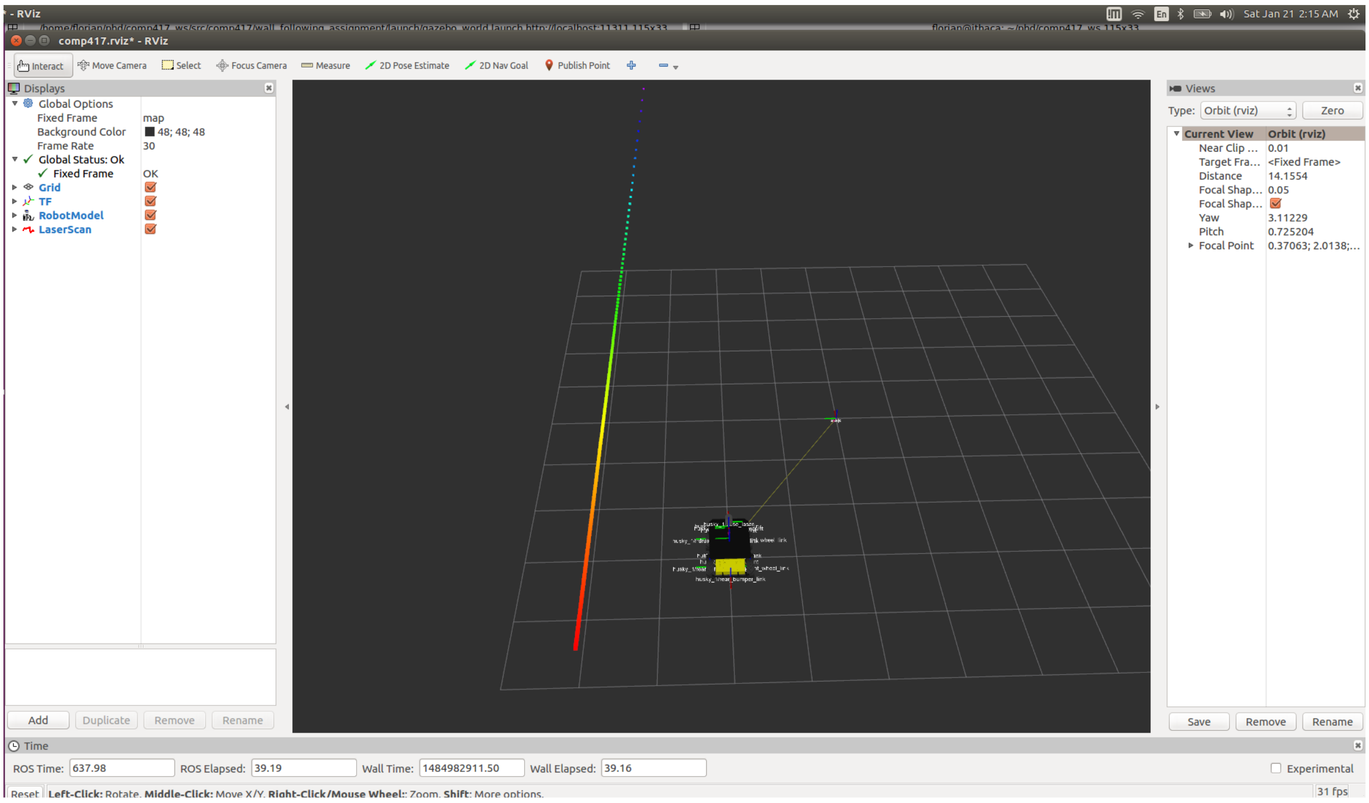
```
rosrun rviz rviz
```

And then go to `File > Open config` and select the config file
`csc477_ws/src/comp417/wall_following_assignment/resources/csc477.rviz`
You should see what's shown in Fig 2:

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 to write your controller code to make the robot move. At this point, you can run a simple last check:

```
rosrun teleop_twist_keyboard teleop_twist_keyboard.py  \
   cmd_vel:=/husky_1/husky_velocity_controller/cmd_vel
```

This command bring up a node publishing on topic `/husky_1/husky_velocity_controller/cmd_vel` and enable you to control the robot through keyboard. If you can successfully move husky around using the keyboard through the command-line interface shown in Fig. 3 then you are ready to proceed with writing your PID controller.



Displays	
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

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

Figure 2: Rviz visualization for the Husky robot in Fig 1. Overlaid on the robot you can see the various reference frames from ROS's `tf` system.

Implement and test a wall-following controller (15 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

```
csc477_ws/src/csc477_fall19/wall_following_assignment/python/wall_follower.py
csc477_ws/src/csc477_fall19/wall_following_assignment/src/wall_follower_node.cpp
csc477_ws/src/csc477_fall19/wall_following_assignment/include/wall_following_assignment/pid.h
```

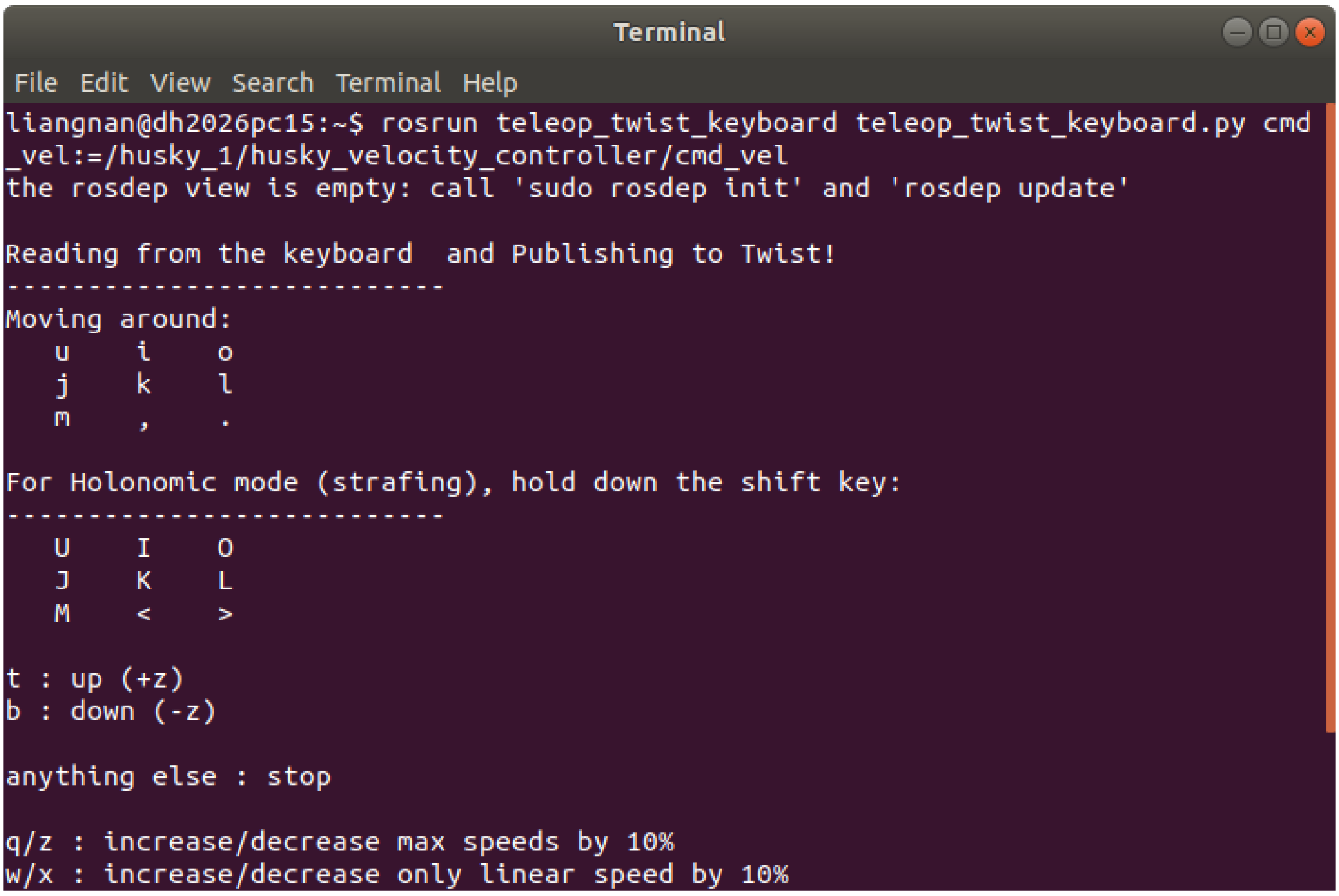
Choose your language and edit the appropriate files. Specifically:

[**Part A, 2 pts**] Compute the cross-track error based on each incoming laser scan and publish it under the topic name `/husky_1/cte` with the ROS message type `std_msgs/Float32`, which can be found here `http://wiki.ros.org/std_msgs`. Tutorials for how to write your own publisher in Python can be found at `http://wiki.ros.org/ROS/Tutorials/WritingPublisherSubscriber%28python%29`.

[**Part B, 7 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.

[**Part 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
```



```
liangnan@dh2026pc15:~$ rosrun teleop_twist_keyboard teleop_twist_keyboard.py cmd
_vel:=/husky_1/husky_velocity_controller/cmd_vel
the rosdep view is empty: call 'sudo rosdep init' and 'rosdep update'

Reading from the keyboard  and Publishing to Twist!
---------------------------
Moving around:
   u    i    o
   j    k    l
   m    ,    .

For Holonomic mode (strafing), hold down the shift key:
---------------------------
   U    I    O
   J    K    L
   M    <    >

t : up (+z)
b : down (-z)

anything else : stop

q/z : increase/decrease max speeds by 10%
w/x : increase/decrease only linear speed by 10%
```

Figure 3: Command-line interface for the node `teleop_twist_keyboard.py` which allows you to drive the robot using your keyboard. If you can do this successfully then you are ready to proceed with designing the controller.

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 3/4 of a full circuit around the left wall.

[**Part 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/world) 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/world) 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
  ```

Submission Instructions Assignment submissions will be done on Quercus. You will submit a zip file containing the following:

1. Your `csc477_fall19/wall_following_assignment` directory for Parts A, B, and C.
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 Part D. The videos should be named as follows:

   ```
   FirstName_LastName_StudentNumber_walls_one_sided.[mp4/avi]
   FirstName_LastName_StudentNumber_walls_two_sided.[mp4/avi]
   ```

   Each video should not exceed 10MB.
3. Similarly, two csv files from Part D. They should be named:

   ```
   FirstName_LastName_StudentNumber_walls_one_sided.csv
   FirstName_LastName_StudentNumber_walls_two_sided.csv
   ```

   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. It is also worth noting that, due to the ROS abstraction layer, if you want, you could run your feedback controller on a real Husky robot, without major modifications.

We expect your code to run on the MCS Lab machines. If it does not you will lose marks. We expect to be able to compile it using `catkin_make`. Once we compile your code we will run the following:

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

We will test your code on other similar worlds to the one provided. We will also test it using different desired distances away from the goal. The default desired distance is 1 meter away, and the default forward speed is 1m/s. We will test your code with different parameters in that neighbourhood. Your code will also be examined for correctness, style and efficiency. We recommend that you come by during office hours or email us if you are unsure of your implementation.