ME 537 - Lab 1 - Robot Kinematics

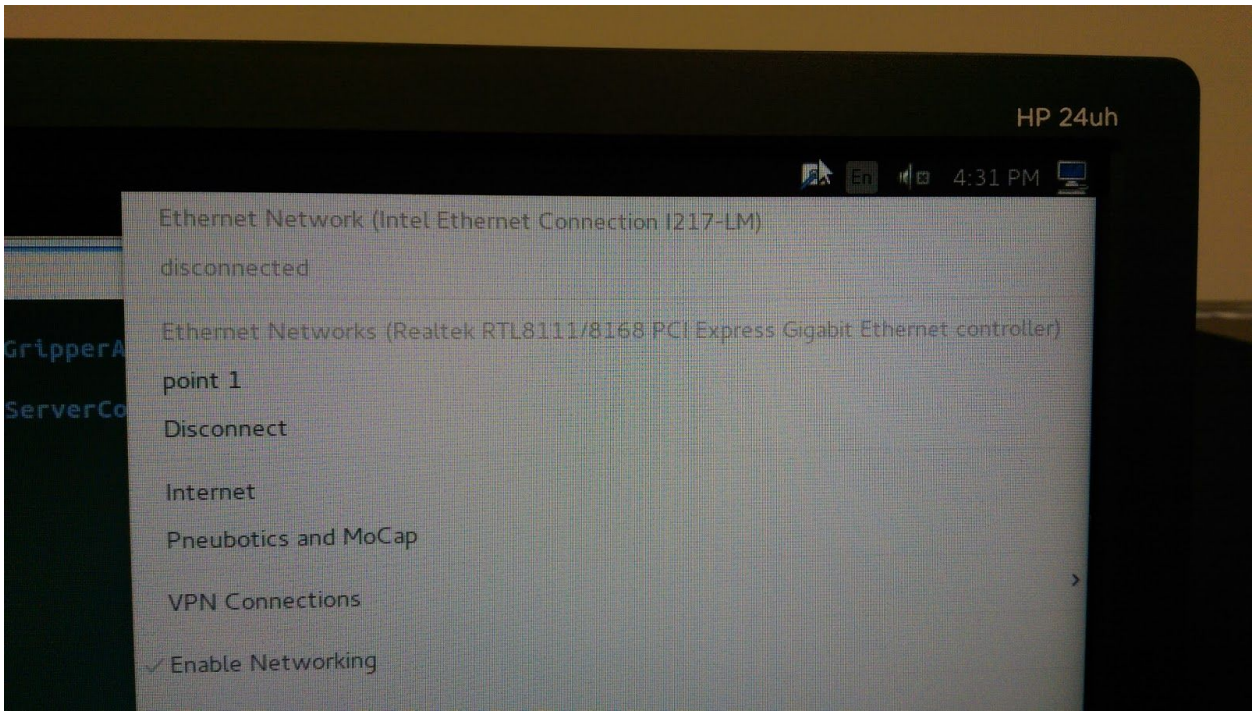
Part I Forward Kinematics with DH Parameterization

The very first thing to do is to identify the big red emergency stop button on the Baxter robot and make sure it is handy in case you accidentally command Baxter to do something bad.

Please don't hesitate to push this switch as it just kills the power and is much better than something worse happening. If you do end up having to push the switch, just follow the instructions in the troubleshooting section below to restart Baxter.

After pushing the “on” button (see below) which is located behind Baxter, wait until the picture on his screen stops moving.

Log in to the desktop computer with the username as “radlab” and the password as “John3:16”. You should first check that your computer is connected to the proper network to communicate with Baxter. Do this by clicking on the icon on the left most icon of the top right of your screen and then clicking on “point 1” (see image):



To confirm that you are connected open a terminal (block box icon on the left hand side of the screen) and type “ping Thing1.local” and hit enter. You should see a series of times that show you are getting data from Baxter. If this does not work, nothing else will and you may need to stop and get help.

If you are connected, you can then go to the terminal and type the following to start Baxter up (the comments below with the # sign are not meant to be typed but to help you understand):

```
cd ~/baxter_ws   #moves you to the correct folder
./baxter.sh          #sets up environment variables correct for baxter to work
                     #the next command turns off the annoying sonar sensors, after running it, you
                     #you may need to hit ctrl+c to stop it
rostopic pub /robot/sonar/head_sonar/set_sonars_enabled std_msgs/UInt16 0
rosrun baxter_tools enable_robot.py -r   #this resets the robot state if you pushed the e-stop
rosrun baxter_tools enable_robot.py -e   #this re-enables the robot, and it may make a noise
rosrun baxter_tools tuck_arms.py -u       #this will untuck his arms
```

After you have untucked Baxter's arms, please grab his left or right wrist on the black rubber part shown with a red dot below and you should be able to move his arm to new positions.

When you push this button, Baxter is entering gravity compensation mode where the weight of his links is compensated by gravity torques applied at the joints, but you are able to otherwise move his arm wherever you want. You should move the right arm out of the way before proceeding as the rest of the code that we'll run is for the left arm. At this point, you can also push on the arm a bit and see that it is back drivable. This is because this arm does not have high gear ratios. We'll talk about this later in the class, but you should be aware that you can fairly safely interact with this robot (unlike most industrial manipulators) although when moving quickly it can still hurt you. So please be careful.

Now, open another terminal window (or a new tab in the same window) where we will execute commands on baxter. We will use "ipython" which is a program that allows you to easily run python commands from the command line, similar to MATLAB. Now type the following:

```
cd ~/baxter_ws
```

The next two commands will have to be run every time you open a new terminal window if you want to command baxter to do anything useful:

```
./baxter.sh
source ~/Desktop/robotics_ws/devel/setup.bash --extend  #this makes the code able to run
```

Now you can start “ipython” and execute a few robot commands as follows:

```
cd ~/Desktop/robotics_ws/src/rad_baxter_limb/src/rad_baxter_limb
ipython
run start_baxter
limb.get_joint_angles()
```

The last command returned the robot’s current joint angles. “start_baxter.py” is not necessary to run every time and in this case just allows us to call functions from the object “limb.” Now move the arm using the wrist button and call that last function again just to see that the angles have changed. Now using the ipython interface again, type the following command:

```
limb.set_joint_positions_mod([0, 0, 0, 0, 0, 0, 0])
```

This will attempt to move the arm towards its neutral position. However, this will not work as the robot times out if you don’t send a command continuously. This is for safety. In the code I provide, we will simply put this command in a loop to make sure the robot actually gets there. Lastly, if you issue the following command, you will also see the robot’s current estimate of its end effector position and orientation using its internal forward kinematics model:

```
limb.get_kdl_forward_position_kinematics()
```

This function returns a vector of length 7. The first 3 numbers are the end effector position. The last four numbers are a quaternion. In my code, I convert this to a rotation matrix and return that matrix to you. The lines of code to get the joint angles, set the joint angles and get the end effector position are the main functions I use in the code that I provide for you.

Now you should copy the three files at ~/Desktop/robotics_ws/src/rad_baxter_limb/src/rad_baxter_limb (part1_new.py, part2_new.py, part3_new.py) to your own files names (such as part1,py, part2.py, and part3.py). You can do this either graphically using the mouse and folders, or you can run the following command:

```
cp part1_new.py part1.py
```

I recommend doing this regardless of what other code already been copied in case people have changed the code at all. There is also a backup of these files on the desktop. Please don’t modify the originals, but make a copy first. Feel free to also look at the code using your favorite text editor or by running “gedit <replace this with name of file>”, like “gedit part1.py”. For this next part, you will be prompted by the code I am providing to move the arm (using the wrist interface) to four different

positions. Please be very careful not to damage the arm. At each of these four positions, record the joint angles and robot's estimated end effector position and orientation which will be printed on the screen after running the following command:

run part1.py

You may need to quit ipython (by doing "quit()") and restart it if this doesn't work the first time and between each "partX.py" file that you run. .

Turn in:

- A sketch of DH parameters you assign to Baxter's arm using the specifications at http://sdk.rethinkrobotics.com/wiki/Hardware_Specifications - make sure to assign the z-axis correctly to match the positive joint angle direction you measure on the real robot.
- Four plots showing your robot in the four positions that you commanded it
- A table with the four positions and orientations that you estimated versus the ones that you got from the code
- A discussion of the discrepancies between your parameterization and forward kinematics and that of the robot's internal code (if your error is quite large, you may want to check your methods and data).

You may want to take pictures of the poses you pick in parts 1-3 so that you can compare against them later when you implement the DH parameters in code.

Part II Robot Repeatability

Pick a new position for the arm to move to. This position should be one that can be reached with the provided measuring device from the ground or table. The arm must be able to move freely between the two positions (yours and the one defined with all the joint angles equal to zero), so please pick your position in free space.

Move the arm to the starting position, then do the following:

- Run the following command in ipython "run part2"
- Measure the vertical distance from the ground or a table to the position on the gripper where you will assign your DH frames after the robot reaches your desired pose.
- Also record the end-effector's position and the final joint angles as printed by the code
- Push enter when you are ready to command the arm back to the initial position
- Repeat this 10 times

Do this again with a different "test" position and orientation and repeat just 5 times. To change the number of times it runs, you will need to open "part2.py" and modfiy the variable "num_times".

Turn in:

- A good way to show or report the variance that you measured in the vertical positioning of the robot arm (although this is just one dimension, we would expect this type of robot to not be very precise).
- A discussion on where the error is coming from (examine both your recorded joint angles vs your commanded angles and the forward kinematics estimate from the robot))
- A report on if and/or why the repeatability or precision of the robot would have been different between the two "test" poses that you picked.

Part III Robot Velocity at the End Effector

Pick two new positions that are spaced fairly far apart and with different end effector orientations as prompted in the next piece of code you'll run. Run the code as follows in ipython:

```
run part3.py
```

Again, please feel free to look at any of the code and try to understand what it is doing. However, I will not require you to understand it for this class. The code will ask you to move the arm to two positions and then will move between them 10 times automatically while recording the joint positions and velocities. In general, the arm will be moving faster than before. The output will be matlab files written on the desktop. If you get bored while the robot arm is moving back and forward, you can check out visualization tools that exist for the robot using the library called ROS. Open a new terminal and do the following if you want to:

```
cd ~/baxter_ws
./baxter.sh
rosrun rviz rviz
```

You should be able to see a graphical representation of Baxter moving. You can also see the video feed from his wrist and if you click on "TF" you can turn on and off all the internal coordinate frames that are used to model Baxter. If for some reason this doesn't work, don't worry.

When you have your data, you will need to write code (which can simply be using a library like the MATLAB Robotics Toolbox) to calculate the Jacobian at the end effector given a configuration and then can estimate the end effector twist.

Turn in:

- A plot of your calculated end effector velocities and angular velocities in the base frame at each time step. If you wish to do things like plot the average velocities and the variance rather than 10 different lines, feel free to do that.
- A discussion of the variation in the velocity from one trial to the next and why it exists.
- Small bonus points (3-5/100) are available if you turn in an animation/video of your robot using the positions. See the MATLAB toolbox for examples of how to that.

When you are finished please copy your velocity data to your own jump drive and delete it from the desktop.

Finally, after quitting ipython, run the following code in the terminal to shut everything down and then push the power button on the back of the robot:

```
rosrun baxter_tools tuck_arms.py -t   # if you've opened a new terminal, you'll need to do the "cd
                                      #~/baxter_ws" and "./baxter.sh" commands again.
```

TROUBLESHOOTING GUIDE

Robot won't respond or we had to use the e-stop:
After disengaging the e-stop, run the following in the terminal and see if it helps:

```
rosrun baxter_tools enable_robot.py -r   #this resets the robot state if you pushed the e-stop
rosrun baxter_tools enable_robot.py -e   #this re-enables the robot, and it may make a noise
```

I don't know how to do X in Linux, python, or ROS:
Start by searching online for answers that may be more obvious than you thought. If the answers don't look obvious or would require you making changes to the computer (such as installing things, etc.) please just skip that part and post your question on Piazza.