Lab 0: Setting up Robot Simulation and Python Refresher

Due date: 09/09/2021 @ 11:59

This exercise is due on **Thursday, September 9, by midnight (11:59 p.m.)**. Submit the brief write up as a PDF and code to Gradescope. Late submissions will be accepted until midnight on **Sunday, September 12**, but they will be penalized by 25% for each partial or full day late. After the late deadline, no further assignments may be submitted; post a private message on Piazza to request an extension if you need one due to a special situation such as illness. This assignment is worth 10 points. You may talk with other students about this assignment, ask the teaching team questions, use a calculator and other tools, and consult outside sources such as the Internet. When you get stuck, post a question on Piazza or go to office hours!

Lab 0 involves much more tedious setup and logistical hurdles than future labs will - that's why we've separated it into its own assignment. It's vital that we work out any setup issues now, so that no one is left behind when Lab 1 is released!

1 Setup Simulation Environment

This semester we have two new state of the art Franka Emika PANDA robot arms ('Panda'), which you will use towards the end of the semester. The Panda arm has 7 Degrees of Freedom (DOF) and sensors at each joint that will allow us to implement controllers and planners for robot manipulation. The purpose of this lab is to get you familiar with the Panda in a ROS+Gazebo simulation environment, which you will use throughout the semester to test the implementations of your algorithms before running them on real robot hardware. The steps outlined below will get you up and running. After thoroughly following the instructions in this handout, if you have issues getting set up, please come to office hours or make a post on Piazza.

1.1 Setup an Ubuntu virtual machine with ROS + Gazebo

Because the simulator is built with ROS and Gazebo, it must run on Ubuntu. Since most students do not have access to a machine running Ubuntu, we will run the simulation on a Virtual Machine.

If you already run an Ubuntu physical machine you are welcome to set up that machine to run the simulation. The VM is considered the standard environment for the labs, so only choose this option if you already feel fairly comfortable with Linux or prepared to deal with any hurdles you encounter! The TA team will do our best to support you, but the standardization of the VM will be a great asset to everyone. The instructions for a native install can be found on the lab git repository: `https://github.com/MEAM520/meam520_labs`

Note: These steps require you to have a computer powerful enough to run the virtual machine (recommended allocation to VM is 10 GB hard-disk space, 8 GB of RAM, 2 CPU cores). If your computer is not able to run the virtual machine, you can use the on-campus computers in the CETS computer labs (Towne M62 and M70). You can sign into any computer using your Pennkey (your normal Penn login) and your personal documents are stored in the "S drive" which is accessible from any lab computer. The name of this drive matches your pennkey.

1. **Install VirtualBox** Install the Oracle VM VirtualBox 6.1.6 for your OS from `https://www.virtualbox.org/`. Versions are available for Windows, Mac and Linux.

2. **Download Virtual Image** Download the MEAM520F21.ova virtual image from:

   `https://drive.google.com/file/d/1eqsxoLSWk9ZSL_x44VJ9PyC1vg8m_VyO`

   Note: This file is quite large (≈6GB) so it may take some time to download. If the image downloads as a .zip, extract the contents. Make a new virtual machine and load the supplied image.

3. **Open VirtualBox**: Tools → Import Appliance → select MEAM520F20.ova → Import. In general, you can leave the defaults. On some machines you may need to disable the USB Controller. By default the VM will have access to 2 CPU cores and 8 GB of RAM, you can reduce these if you are on a less powerful machine (or increase for a more powerful machine).

4. **Enter the VM**: Select MEAM520 and click the green start arrow in the toolbar. Now you should be greeted with a desktop in the VM. If you see a login screen, use the username `student` and password `meam520` to enter the system.

    - **Note**: On Windows or Linux you may get an error about VT-x or AMD-V not being available. If so, you need to enable virtualization for your computer in the BIOS. This is different for every manufacturer, so search "<make> <model> enable virtualization" and follow the instructions. Your manufacturer should have a similar guide and the instructions will be similar, though with varying details. **DO NOT install Microsoft Hyper-V.**

1.2 Get the Code

Git is a code versioning tool, and can be a powerful resource. This year we will use git in a limited fashion to ease the dissemination of code. So that each student can have their own individual private work we are going to have you work on a copy of the original repository in your own GitHub account.

If you have a GitHub account you can skip step 1 and move onto step 2: cloning a new private repository.

1. **Setting up a gitHub account**

    - Got to: `https://github.com`
    - Select signup
    - Follow the prompts and make an account using your Penn email
    - login into your account
    - Go to the top right corner of your account and select the circle icon, navigate to settings, on the left column you will see a list of opitions, select SSH and GPG keys.
    - Next open a terminal in the Virtual Machine (Ctrl-Alt-t), and type the following command:

      ```
      $ ssh-keygen -t ed25519 -C "your_email@seas.upenn.edu"
      ```

    - Once you have finished generating the key (hitting enter for defaults works) type the following commands

      ```
      $ cd ~/.ssh
      $ cat id_ed25519.pub
      ```

      You will see a string output on the terminal starting with: "ssh-ed25519" and ending with your email.
    - In your GitHub account, under "Settings", select "SSH and GPG Keys", "New SSH-key", name the key and copy the entire string on your terminal into the appropriate box. (To copy things off of the terminal highlight the text and click Ctrl-Shift-C. You may also need to enable Devices ¿ Shared Clipboard ¿ Bidirectional in the VirtualBox toolbar.) You have now made it possible to clone your private repository to your virtual machine.
    - Note: GitHub has an educational developers pack which you are eligible for as a Penn student. To learn more about this and to register your account go to: `https://education.github.com/discount_requests/student_application`. To expedite this process it is important that you use your Penn email to make your account (there is a verification process).

2. **Cloning a new private repository** Each of you will be making a deep copy of the TA existing repository which contains code you will use throughout the semester, e.g. unimplemented functions, that you implement for a lab. We will outline a set of steps here so that you are able to keep your code private on your own GitHub account, and get updates as the TAs make updates to the codes, e.g. as new assignments are released. The following steps allow you to copy the `meam520_labs` code base:

    - If you are not already signed in, login to your GitHub account
    - In your terminal do the following:

      ```
      $ cd ~/meam520_ws/src
      $ git clone --bare https://github.com/MEAM520/meam520_labs.git
      ```

This has created a bare repository of the TAs stub code, which we will refer to TA code for the remainder of the instructions. Note that you will not be using this cloned repository and you will see it appear in your src folder as `meam520_labs.git`.

- Next go to your GitHub account and in the top right corner click on your circle user icon, go down to *your repositories*
- At the top of the new page select New Repository, name the repository: `meam520_labs`. This will load to a blank git repository with instructions on how to clone and add things to it, ignore these.

  IMPORTANT: You MUST set your repository to Private (NOT PUBLIC), since publicly posting your assignment solutions online is a violation of Penn's Code of Academic Integrity.
- Next we are going to push the TA code to your newly created git repository.

```
$ cd meam520_labs.git
$ git push --mirror git@github.com:<YOUR_USERNAME>/meam520_labs.git
```

  Note you need to replace `YOUR_USERNAME` with your GitHub username. To check that this step is executed correctly go to your GitHub account and reload your new `meam520_labs` repository page. You should see the README and files loaded into the repository.
- Now we will remove the TA repository from your machine as follows:

```
$ cd ..
$ rm -rf meam520_labs.git
```

  Now if you type `ls` in `/meam520_ws/src` you should not see a directory called `meam520_labs.git`
- We will clone your new private repo as follows (you can get the link for the new repo from the top right corner in the green box labeled Code):

```
$ cd ~/meam520_ws/src
$ git clone git@github.com:<YOUR_USERNAME>/meam520_labs.git
```

  Note you need to replace `YOUR_USERNAME` with your GitHub username.
- You should be able to type `ls` in your terminal and see a new directory created called `meam520_labs`, which now points to *your* github repository.

3. **Getting code updates from the TAs**: Now we are going to make it possible for you to get updates from the TAs main repository. First it is important to understand that git is a tool which is used locally to keep a history of your code. To ensure that code is backed up to an additional location outside of your computer, and to collaborate with others, a *remote* is setup. GitHub is an example of a location which stores remote git repositories and acts as a way to backup code to a secondary location.

   To check that your local git repository is setup correctly type the following command:

```
$ cd ~/meam520_ws/src/meam520_labs
$ git remote -v
```

   You should see the following output:

```
> origin git@github.com/YOUR_USERNAME/meam520_labs.git (fetch)
> origin git@github.com/YOUR_USERNAME/meam520_labs.git (push)
```

   The *origin* is the primary remote location, and this is pointing to the repository you forked to your account.

   Now we are going to add additional information telling your git repository to check updates made at the original repository location (in this case where the TAs will make updates and release new projects). In git language this is called *setting the remote upstream*.

   Do the following:

```
$ git remote add upstream https://github.com/MEAM520/meam520_labs.git
$ git remote set-url --push upstream DISABLE
$ git remote -v
```

You should now see:

```
> origin  git@github.com/YOUR_USERNAME/meam520_labs.git (fetch)
> origin  git@github.com/YOUR_USERNAME/meam520_labs.git (push)
> upstream  https://github.com/MEAM520/meam520_labs.git (fetch)
> upstream  DISABLED (push)
```

Notice that origin still points to your fork, and that upstream is now pointing to the repository that is maintained by the TAs. Now that an upstream is set we will ask that you periodically use the following command:

```
$ git pull upstream master
```

This will ensure that you get updates when TAs make changes. You should run this command before starting each new lab. We will also post on Piazza if any changes need to be released during a given lab.

You can also use your git repository to make it easier to collaborate with your lab group as you work on the assignments (optional but helpful!). To learn more about the basics of git, check out `https://guides.github.com/introduction/git-handbook/`.

1.3 Build and Launch the Simulator

1. **Build the codebase:** To finish the setup of the codebase, you will need to run the following commands

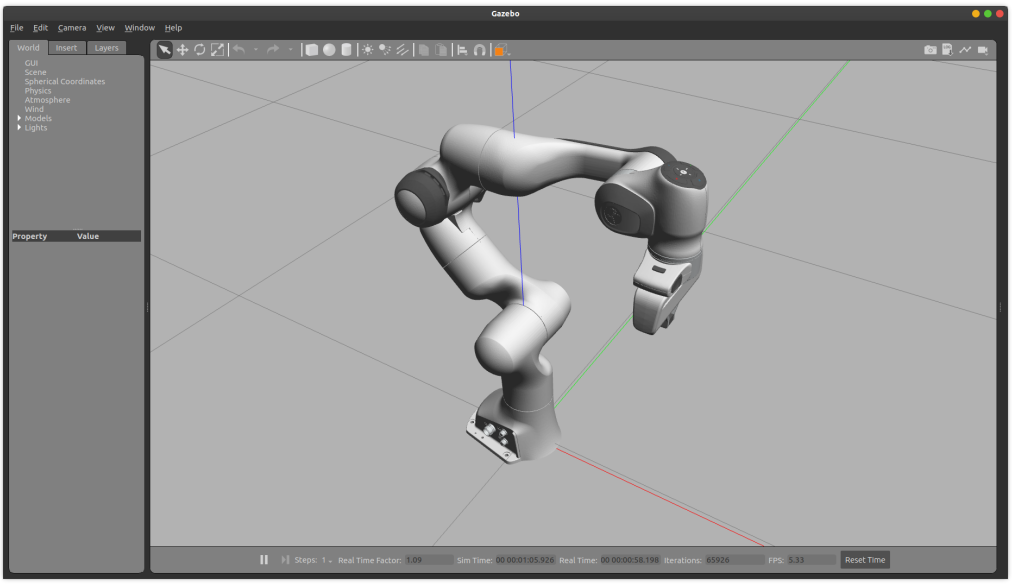
   ```
   $ cd ~/meam520_ws
   $ catkin_make_isolated
   $ source devel_isolated/setup.bash
   ```

   Once this build finishes without errors, you will be ready to launch the simulator.

2. **Start Gazebo:** Run this command in the Terminal:

   ```
   $ roslaunch meam520_labs lab0.launch
   ```

   You should see text indicating that the ROS node has been initialized and eventually a Gazebo window with a robot arm as shown below. Gazebo is a robot simulation tool that will allow you to test your code and view the robot in action. You can zoom with the scroll wheel, pan by left clicking and dragging, and orbit by holding shift while clicking and dragging. Find out more on the Gazebo website at `http://gazebosim.org/`.



3. **Adjust Real Time Factor:** Along the bottom toolbar, Gazebo will report the *Real Time Factor*, which tells you how fast the simulator is running relative to the real world time. All time-based operations in your code should follow the ROS clock, which matches Gazebo's clock when running in simulation:

```
from core.utils import time_in_seconds
...
t = time_in_seconds() # matches Gazebo clock
```

Depending on your computer and VM resource allocations, your Real Time Factor may be anywhere from 0.3 to 1.0 or higher. The compute power available to the simulator will put an upper limit on the Real Time Factor that is possible to achieve, but you can **try** to adjust some settings by editing the following file:

```
$ atom ~/meam520_ws/src/meam520_labs/ros/meam520_labs/worlds/empty.world
```

and following the instructions in the comments.

2 Control Arm from Python

While the simulator runs on ROS+Gazebo, we can interface with it through code you write in Python, allowing us to send commands to the arm and receive data about its state in return. In your terminal, type

```
$ cd ~/meam520_ws/src/meam520_labs
```

to enter repository you just forked. Inside are several directories which you will use throughout the semester:

- `core`: This directory contains Python utilities which we provide to you to e.g. send commands to the robot. You will **not** need to modify code in this directory, but **will** want to look at it to see what functionality is available to you.
- `labs`: This directory contains a subfolder for each labs, containing scripts that you can run to test your code. Some of these we will provide to you, others you will write yourself.
- `lib`: This directory is where you will store functions that you write to implement abstract algorithms for the robot, some of which will be reused in future labs. These will be imported by test scripts in the `labs` folder.
- `ros`: This directory contains code that runs the simulator. You will **not** need to modify this or read through it closely, but you can if you're interested!

We will now walk through a simple example of controlling the arm from a Python script.

1. **Open an editor:** Open this lab's test script in your editor of choice. For example,

   ```
   $ atom ~/meam520_ws/src/meam520_labs/labs/lab0/demo.py
   ```

   will open the script in Atom. Read through the simple script and see what commands we are sending to the arm.

2. **Control the arm:** Make sure the simulator is still running. If not, relaunch it using the same command from the previous section. Open a new terminal and run the test script:

   ```
   $ cd ~/meam520_ws/src/meam520_labs/labs/lab0
   $ python demo.py
   ```

   The terminal will print out some information when the script connects to the simulated arm. Then, the arm will "untuck" to a neutral position, open its gripper, and then move to a new commanded joint configuration. Finally, it will close the gripper and the script will terminate.

3. **Examine Different Configurations:** Try changing the joint angles we command the arm to move to, which are given in units of radians. If you command a joint angle outside its permitted range, an error will print on your terminal and the value will be constrained to within the allowed range. Rerun the `demo.py` script to see the results!

   If you see a `Trajectory Server Message: error code -4` or if the terminal running the simulator reports that you've `Exceeded Error Threshold`, this may be because you've asked the robot to move though a configuration where it is in self-collision. This can trigger the failsafe which stops trajectory execution when the error between desired and actual configurations exceeds a threshold. Try turning on View > Collisions in Gazebo to visualize the simplified geometry Gazebo uses to model collision, and and View > Contacts to visualize collision events.

4. **Turn Off Simulation:** Close all running terminals so that Gazebo will shut down automatically. You can do this by pressing Ctrl-C in each terminal. You can turn off the VM as well.

5. **Congratulations on your first simulation run!**

3 Report

The report for this lab should consist of screenshots of the arm in 5 different interesting configurations, including a list of what each of the joint configurations are.

4 Python Practice

As part of Lab 0 we have included a simple coding assignment to help students learn about Numpy and other basic Python functionalities, and get used to submitting code assignments to be autograded.

4.1 Code Assignment

For the code assignment you will be competing 3 different code assignments. The stub code is located in `/meam520_ws/src/meam520_labs/labs/lab0/lab0Utils.py`. The stub code includes a simple test script to make sure your code runs; to run your code use:

```
$ cd ~/meam520_ws/src/meam520_labs/labs/lab0
$ python lab0Utils.py
```

4.1.1 linear_solver

The first function is a linear equation solver. The inputs are A, b and your job is to solve for x in $Ax = b$. You should assume that A is invertible.

- **Hint**: Numpy has a number of useful tools for matrix and vector manipulation

4.1.2 angle_solver

The 2nd function solves for the magnitude (in radians) of the angle between two vectors in $\mathbb{R}^2$. The inputs are v_1, v_2. Since there are two different angles between the vectors, you should return the smallest of the two options

- **Hint** one way to calculate the angle is to use $\theta = \arccos \frac{v_1 \cdot v_2}{||v_1|| ||v_2||}$
- **Hint** the range of $\arccos$ is $[0, \pi]$

4.1.3 linear_euler_integration

The 3rd function should use Euler's Method to approximate the solution to a linear differential equation. The inputs are

- A: matrix describing ode $\dot{x} = Ax$
- x_0: vector describing initial condition
- d_t: time step for integration
- nSteps: number of times steps

And the output is the state after nSteps.

- **Hint** During Euler integration $x_{k+1} = x_k + d_t A x_k$
- **Hint** Consider using a for loop to apply the above equation iteratively to the initial condition
- **Hint** for more information on Euler's Method checkout `https://en.wikipedia.org/wiki/Euler_method`

4.2 Helpful Python and Coding Resources

There will be a decent amount of Python programming done this semester. If you would like to refresh your skills or learn Python basics, please check out the Python Primer on Canvas under `Files/Resources/Python_primer.zip` to help introduce key features used in Python. In addition, here are links full of tips and tricks to help you get going with Python:

- Introduction to Python: `https://developers.google.com/edu/python/introduction`
- Quick resources for using Numpy: `https://www.dataquest.io/blog/numpy-cheat-sheet/`
- Additional resources for using Numpy: `https://docs.scipy.org/doc/numpy/user/quickstart.html`
- Tutorials on matplotlib: `https://matplotlib.org/3.2.1/tutorials/index.html`
- **Chosing a text editor:** Since you will be writing your Python code in the VM, you will need to choose a text editor to run in the VM. Some good options include Atom, emacs, or vim (all installed on the VM already).

5 Submission to Gradescope

Please submit `lab0Utils.py` and any other files needed to run your code to the Lab0 Code assignment on Gradescope. Once submitted we will run automated tests to grade your code. You may submit as many times as you want, but any attempts to try to figure out what test cases we are running will result in your code getting a 0.

Please submit your written report as a PDF to the Lab0 Report assignment on Gradescope.

IMPORTANT, please submit the correct thing to each assignment. Don't submit the code to the report assignment. Don't submit the report to the code assignment.