Assignment Guide

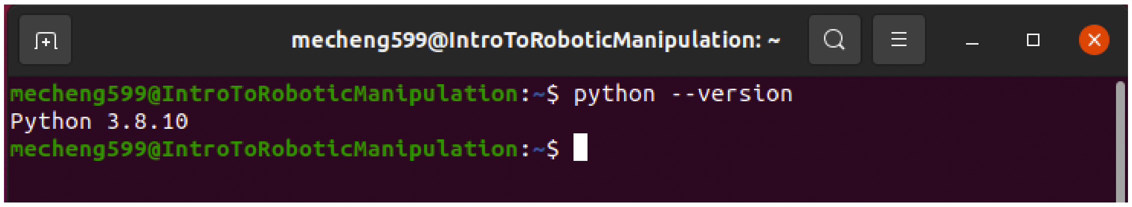
Assignments Overview

Along this course, you will be asked to submit on 7 assignments. These assignments will be related to the topics discussed in lectures, and will enhance and strengthen your manipulation expertise by directly implementing them. Each of these assignments is graded out of 100 and they contribute to 65% of the final grade.

In each assignment, you will be asked you to fill out missing pieces of code in a Python script. Python is an important programming language and very prevalent in robotics. It is also freely available, cross-platform, and great for rapid prototyping. Code developed with Python is accessible to anyone with a computer, making it a great choice for development and deployment. This is contrast to languages such as MATLAB that require licences not easily available to everyone. Moreover, python is an interpreted language, making it easier and faster to debug than compilated languages like C++. In this course, we will use `Python3`. More precisely, we recommend you to use a Python version 3.8 or newer. You can check which python version you have by typing on the terminal:

```
$ python3 --version
```

If you are using Ubuntu 20.04, the native python is `Python3`, and therefore `python` is the same as `python3`.



```
mecheng599@IntroToRoboticManipulation:~$ python --version
Python 3.8.10
mecheng599@IntroToRoboticManipulation:~$ 
```

Figure 1. Displaying the python version on Ubuntu 20.04

Assignment Submission

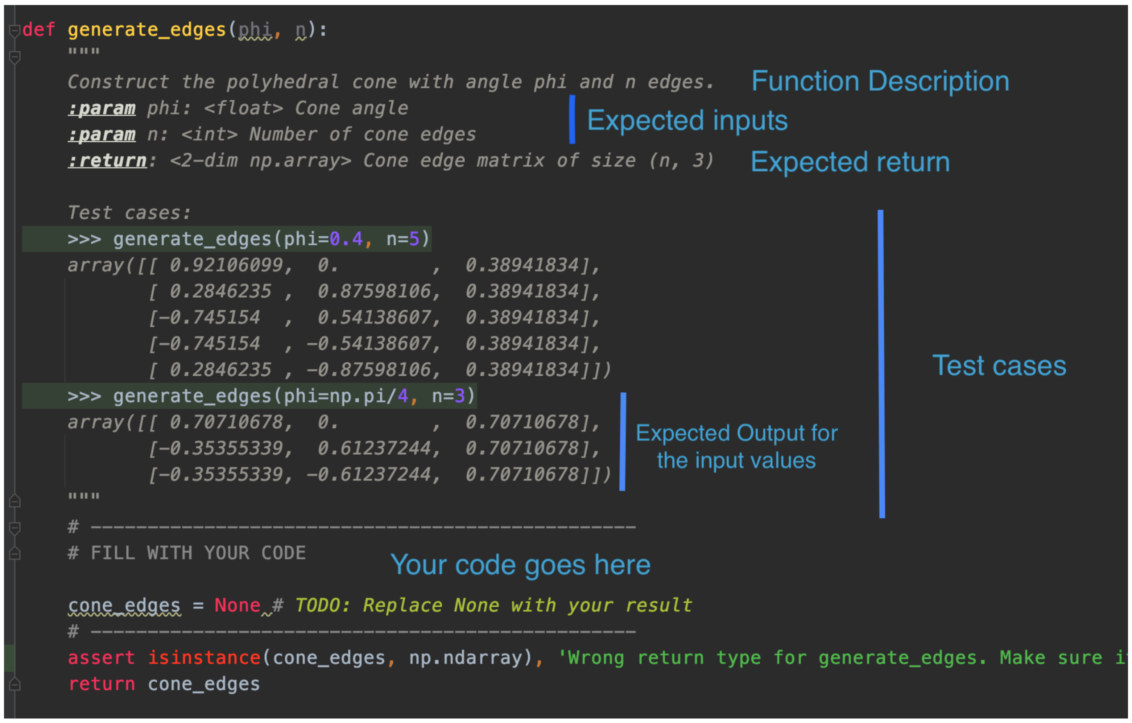
For each assignment, you will have to submit one single `.py` file to autograder. You will have unlimited submissions before the due date. After the due data the submission will be graded by the auto-grader and the score will be made visible on autograder. Scores are also provided for earlier submissions (before the deadline), but only the final grade will be kept. Please, comment out any print or debugging trace you may have added on the turn-in function.

Assignment Grading

Your code will be passed to an auto-grader that will verify the correctness of your work and assign you a score out of 100. Assignments are divided into different parts. Parts may have different weights depending on their complexity and previous dependencies. Some parts may use functions and methods implemented in previous parts, so be special careful and test your implementations. Each assignment has the same contribution towards the final grade. All functions you will have to implement contain a description, and their expected input and output types. Figure 2 shows an example of a function skeleton we will provide you. Make sure that the return type matches the desired. The auto-grader times each function it evaluates. If it reaches the timeout, that part will receive no points. The timeouts are designed so they should only be triggered if your code reaches an infinite loop. We recommend you to test and debug your code before submitting.

Debugging

Each assignment has functionality implemented to test and debug your code. This tests can be either visual, e.g. making sure that the robot ends where it is supposed to go, or doctest-based, i.e. if you run the file



```
def generate_edges(phi, n):
    """
    Construct the polyhedral cone with angle phi and n edges.
    :param phi: <float> Cone angle
    :param n: <int> Number of cone edges
    :return: <2-dim np.array> Cone edge matrix of size (n, 3)

    Test cases:
    >>> generate_edges(phi=0.4, n=5)
    array([[ 0.92106099,  0.        ,  0.38941834],
           [ 0.2846235 ,  0.87598106,  0.38941834],
           [-0.745154  ,  0.54138607,  0.38941834],
           [-0.745154  , -0.54138607,  0.38941834],
           [ 0.2846235 , -0.87598106,  0.38941834]])
    >>> generate_edges(phi=np.pi/4, n=3)
    array([[ 0.70710678,  0.        ,  0.70710678],
           [-0.35355339,  0.61237244,  0.70710678],
           [-0.35355339, -0.61237244,  0.70710678]])
    """
    # ------------------------------------------------
    # FILL WITH YOUR CODE

    cone_edges = None # TODO: Replace None with your result
    # ------------------------------------------------
    assert isinstance(cone_edges, np.ndarray), 'Wrong return type for generate_edges. Make sure i
    return cone_edges
```

Figure 2. Example of a function to be implemented.

it will provide feedback on terminal. Figure 3 shows an example of debugging functions. The auto-grader tests your code with more exhaustive cases than the cases provided for debugging. **Therefore, passing the debug test cases we provide does not necessary imply that you will receive full points.** Passing the test cases means that you are probably on the right track. You can create your own test values to debug your code on the `__main__` function. Anything on the `__main__` will not be evaluated, so you are free to edit it as you will. To run the main:

```
$ python assignment_X.py
```

Each function has an assertion to make sure that the return type is the correct one. Python will raise errors if the return does not match the desired. Figure 4 shows an example of an error due to the wrong return type. If automated testing is provided, it will compare your return with the expected one. If they match, it will output `ok`, as figure 5. Otherwise it will produce an output like the one shown on figure 6.

Requirements and Set Up

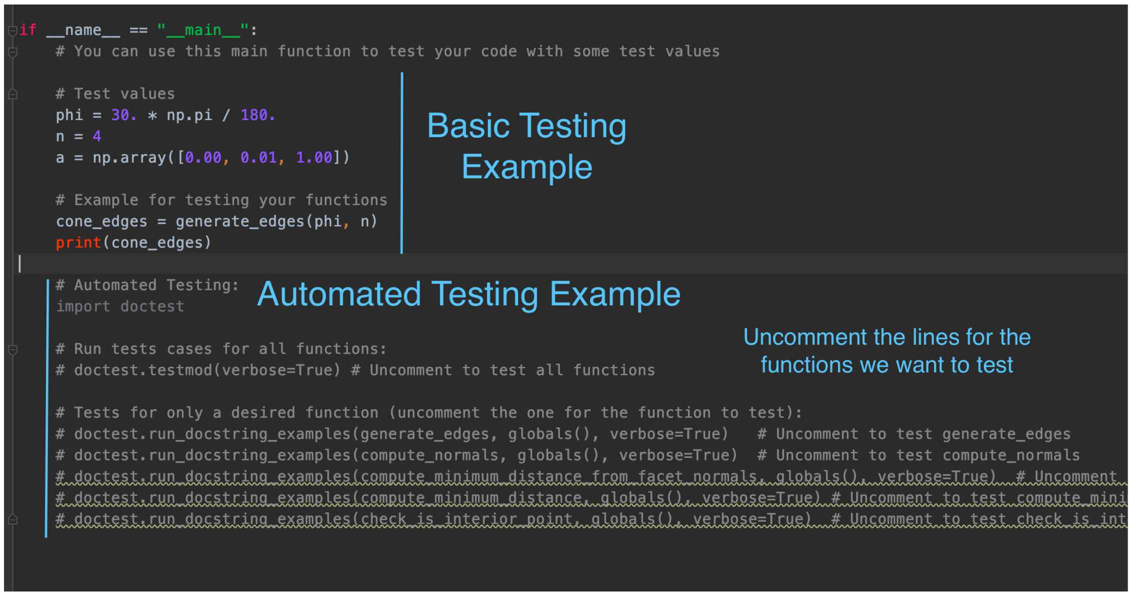
Operative System

You are free to work on your preferred OS, we will only evaluate the submitted code. Our recommended OS is Ubuntu 20.04, since the native python is `Python3`. You can download install Ubuntu 20.04 on a new partition, or you can just use a virtual machine. If you are interested on how to dual boot your machine with Ubuntu, check this tutorial. If you have problems setting up python and the required packages, we provide a Ubuntu 20.04 virtual machine that has everything installed for you. It also has PyCharm installed, which is our recommended IDE. Below you can find a guide how to set up the virtual machine.

Python Packages

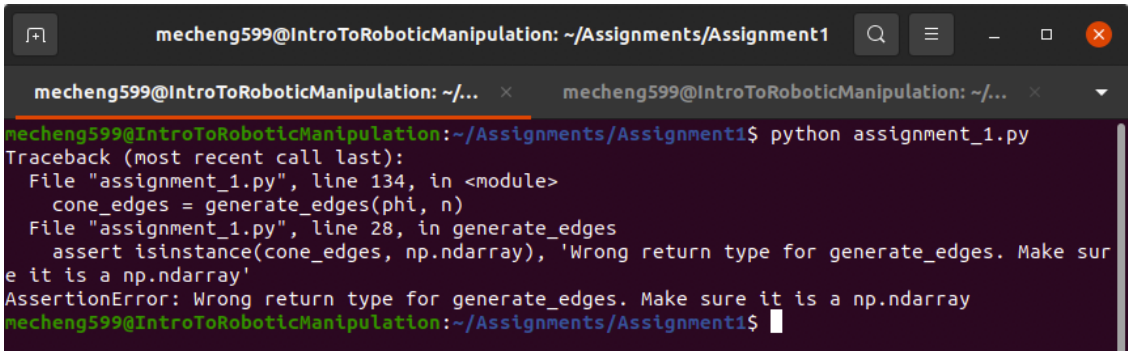
You will need `python3` and the following python packages:

- `numpy` >= 1.21.2


Basic Testing Example
Automated Testing Example
Uncomment the lines for the functions we want to test

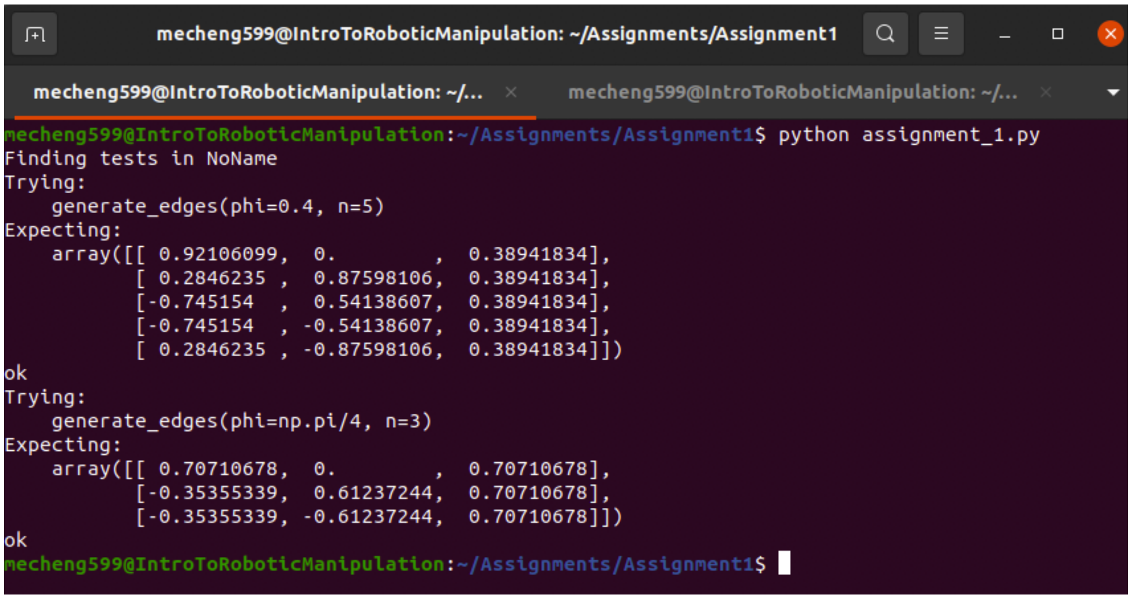
```
if __name__ == "__main__":
    # You can use this main function to test your code with some test values

    # Test values
    phi = 30. * np.pi / 180.
    n = 4
    a = np.array([0.00, 0.01, 1.00])

    # Example for testing your functions
    cone_edges = generate_edges(phi, n)
    print(cone_edges)

    # Automated Testing:
    import doctest

    # Run tests cases for all functions:
    # doctest.testmod(verbose=True) # Uncomment to test all functions

    # Tests for only a desired function (uncomment the one for the function to test):
    # doctest.run_docstring_examples(generate_edges, globals(), verbose=True)   # Uncomment to test generate_edges
    # doctest.run_docstring_examples(compute_normals, globals(), verbose=True)  # Uncomment to test compute_normals
    # doctest.run_docstring_examples(compute_minimum_distance_from_facet_normals, globals(), verbose=True)  # Uncomment
    # doctest.run_docstring_examples(compute_minimum_distance, globals(), verbose=True) # Uncomment to test compute_mini
    # doctest.run_docstring_examples(check_is_interior_point, globals(), verbose=True)  # Uncomment to test check_is_int
```

Figure 3. Example of debugging functions.


mecheng599@IntroToRoboticManipulation: ~/Assignments/Assignment1
mecheng599@IntroToRoboticManipulation: ~/...
mecheng599@IntroToRoboticManipulation: ~/...

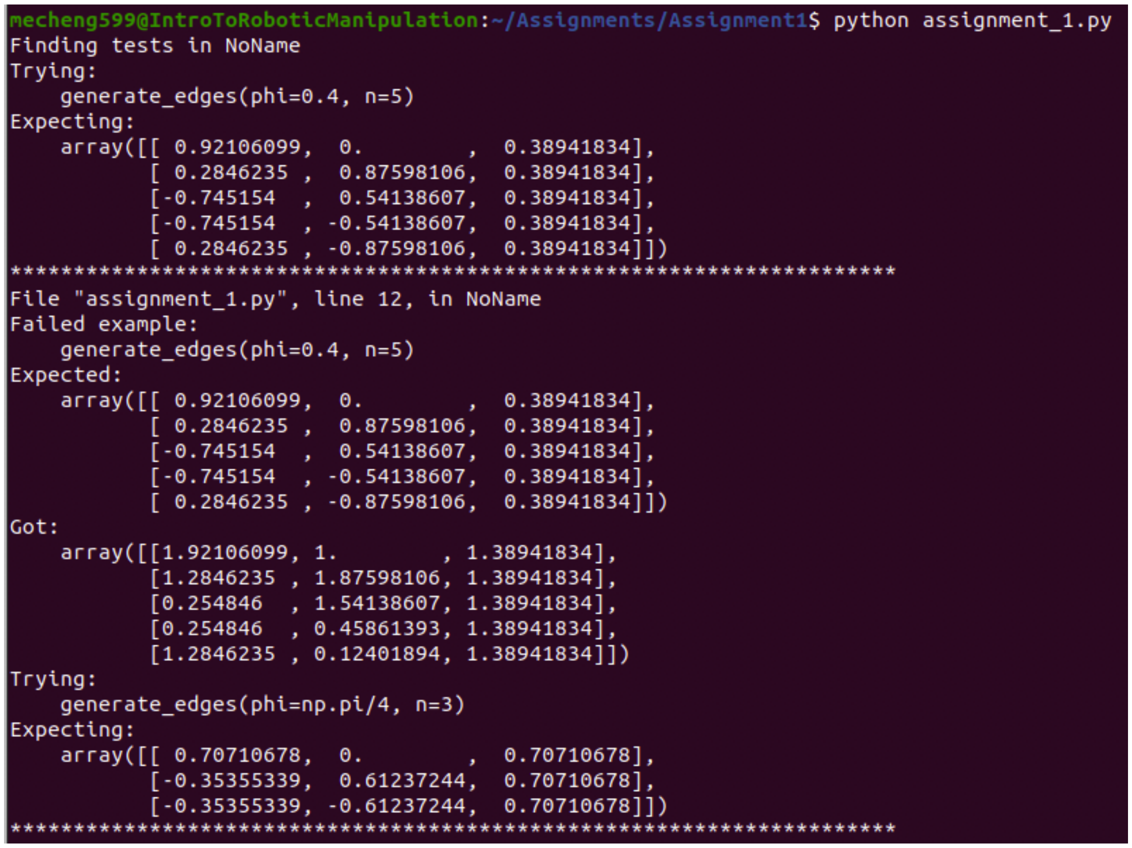
```
mecheng599@IntroToRoboticManipulation:~/Assignments/Assignment1$ python assignment_1.py
Traceback (most recent call last):
  File "assignment_1.py", line 134, in <module>
    cone_edges = generate_edges(phi, n)
  File "assignment_1.py", line 28, in generate_edges
    assert isinstance(cone_edges, np.ndarray), 'Wrong return type for generate_edges. Make sur
e it is a np.ndarray'
AssertionError: Wrong return type for generate_edges. Make sure it is a np.ndarray
mecheng599@IntroToRoboticManipulation:~/Assignments/Assignment1$ 
```

Figure 4. Example of error on return type. Make sure that all returns are of the type specified on the function's description.


mecheng599@IntroToRoboticManipulation: ~/Assignments/Assignment1
mecheng599@IntroToRoboticManipulation: ~/...
mecheng599@IntroToRoboticManipulation: ~/...

```
mecheng599@IntroToRoboticManipulation:~/Assignments/Assignment1$ python assignment_1.py
Finding tests in NoName
Trying:
    generate_edges(phi=0.4, n=5)
Expecting:
    array([[ 0.92106099,  0.        ,  0.38941834],
           [ 0.2846235 ,  0.87598106,  0.38941834],
           [-0.745154  ,  0.54138607,  0.38941834],
           [-0.745154  , -0.54138607,  0.38941834],
           [ 0.2846235 , -0.87598106,  0.38941834]])
ok
Trying:
    generate_edges(phi=np.pi/4, n=3)
Expecting:
    array([[ 0.70710678,  0.        ,  0.70710678],
           [-0.35355339,  0.61237244,  0.70710678],
           [-0.35355339, -0.61237244,  0.70710678]])
ok
mecheng599@IntroToRoboticManipulation:~/Assignments/Assignment1$ 
```

Figure 5. Example of successful automated testing. All test cases got ok as a return.



```
mecheng599@IntroToRoboticManipulation:~/Assignments/Assignment1$ python assignment_1.py
Finding tests in NoName
Trying:
    generate_edges(phi=0.4, n=5)
Expecting:
    array([[ 0.92106099,  0.        ,  0.38941834],
           [ 0.2846235 ,  0.87598106,  0.38941834],
           [-0.745154  ,  0.54138607,  0.38941834],
           [-0.745154  , -0.54138607,  0.38941834],
           [ 0.2846235 , -0.87598106,  0.38941834]])
**********************************************************************
File "assignment_1.py", line 12, in NoName
Failed example:
    generate_edges(phi=0.4, n=5)
Expected:
    array([[ 0.92106099,  0.        ,  0.38941834],
           [ 0.2846235 ,  0.87598106,  0.38941834],
           [-0.745154  ,  0.54138607,  0.38941834],
           [-0.745154  , -0.54138607,  0.38941834],
           [ 0.2846235 , -0.87598106,  0.38941834]])
Got:
    array([[1.92106099, 1.        , 1.38941834],
           [1.2846235 , 1.87598106, 1.38941834],
           [0.254846  , 1.54138607, 1.38941834],
           [0.254846  , 0.45861393, 1.38941834],
           [1.2846235 , 0.12401894, 1.38941834]])
Trying:
    generate_edges(phi=np.pi/4, n=3)
Expecting:
    array([[ 0.70710678,  0.        ,  0.70710678],
           [-0.35355339,  0.61237244,  0.70710678],
           [-0.35355339, -0.61237244,  0.70710678]])
**********************************************************************
```

Figure 6. Example of error on automated testing.

- `matplotlib` >= 3.4.3
- `open3D` >= 0.13.0
- `cvxpy` >= 1.1.15
- `pybullet` >= 3.1.7

You can install them using your preferred package manager. We recommend `pip`, e.g. `pip install numpy` We recommend you to work on a python virtual environment, so there is no problem on dependencies between the installed packages. We provide a package requirement list `requirements.txt`

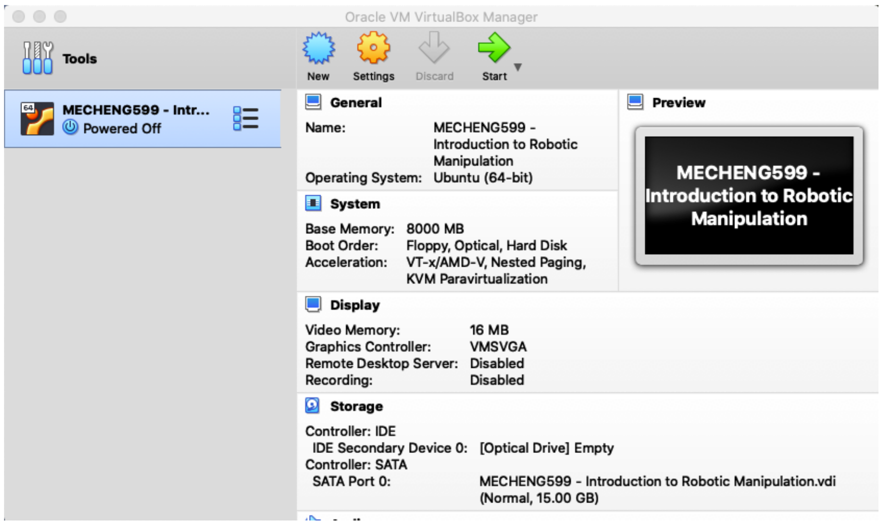
VirtualBox Setup Guide

We provide a virtual machine with everything you will need for this course. Therefore, there is no need to set up a virtual environment if you use this approach. Here you can find a guide on how to install and set up a virtual machine on VirtualBox, our recommended Virtual Machine client:

To install our Virtual Machine, do the following steps:

1. Download and Install Virtual Box
2. Download the file `MECHENG599 - Introduction to Robotic Manipulation.ova`
3. Open VirtualBox and select **File** > **Import Appliance...**
4. Select the downloaded `.ova` file. Press **Continue**.
5. Verify the setting. You may need to tailor RAM and CPU to fit your machine characteristics. Setting up to much RAM can make your native machine run slower.
6. Click **Import**

Start the virtual machine by pressing the green arrow Start.


Oracle VM VirtualBox Manager
Tools
New
Settings
Discard
Start
MECHENG599 - Intr...
Powered Off
Preview
MECHENG599 - Introduction to Robotic Manipulation

General

Name: MECHENG599 - Introduction to Robotic Manipulation
Operating System: Ubuntu (64-bit)

System

Base Memory: 8000 MB
Boot Order: Floppy, Optical, Hard Disk
Acceleration: VT-x/AMD-V, Nested Paging, KVM Paravirtualization

Display

Video Memory: 16 MB
Graphics Controller: VMSVGA
Remote Desktop Server: Disabled
Recording: Disabled

Storage

Controller: IDE
IDE Secondary Device 0: [Optical Drive] Empty
Controller: SATA
SATA Port 0: MECHENG599 - Introduction to Robotic Manipulation.vdi (Normal, 15.00 GB)

Figure 7. Virtual box manager window with the course virtual machine

Troubleshooting:

- If it runs too slow, you may need to assign more resources to it (CPU and/or RAM).
- If you can't maximize the screen size, you may need to install Guest Additions

Virtual Environment Guide

Here you can find a great guide for setting up a virtual environment using `virtualenv`: python-virtualenv-guide.

To set up a virtual environment for this class do the following:

1. Install virtualenv:

   ```
   $ pip install virtualenv
   ```

2. Create the working directory and move there:

   ```
   $ mkdir -p path/to/your/working/directory/Assignments
   $ cd path/to/your/working/directory/Assignments
   ```

3. Create the virtual environment:

   ```
   $ virtualenv -p python3 venv
   ```

4. Activate the virtual environment. You will need to soure this file every time you open a new terminal:

   ```
   $ source venv/bin/activate
   ```

   If you correctly activated the virtual env, `(venv)` should appear in front of your command line prompt.

5. Download the requirements.txt from Canvas.

6. Install all required packages

   ```
   $ pip install -r path/to/requirements.txt
   ```

7. Once you are done with the virtual environemnt, you can deactivate it (close it) as follows:

   ```
   $ deactivate
   ```

Assignment 1

Assignment Objective

Benchmark Python coding skills. Python is an important programming language and very prevalent in robotics. It is also freely available, cross-platform, and great for rapid prototyping. Code developed with Python is accessible to anyone with a computer, making it a great choice for development and deployment. This is contrast to languages such as MATLAB that require licences not easily available to everyone. We will use the Numpy library primarily for numerical computations. This assignment will introduce you to some basic functionality of this library. If you are new to numpy, this Numpy Tutorial is a great introduction. Also, we recommend you to take a look at the post Look Ma, No For-Loops: Array Programming With Numpy which will help you avoid unnecessary `for` loops and really boost your code.

Instructions

In this assignment, we will ask you to fill out missing pieces of code in a Python script. You will submit the completed code to autograder. Your code is passed to an auto-grader that will verify the correctness of your work and assign you a score out of 100.

- Download the assignment script `assignment_1.py` from the Files menu of Canvas.
- For each "Part" of the HW assignment, fill in the missing code as indicated by the comments.
- Submit the completed code to autograder.
- Each part has an equal contribution to your score and the total is equal to 100.
- The assignment is due on September 30^{th}, 2024.
- You will need the `numpy` library.

Part 1

Consider the polyhedral cone depicted in Fig. 8. The cone is constructed by N edges defined as:

$$\boldsymbol{v}_i = \begin{bmatrix} \cos\theta\cos\phi \\ \sin\theta\cos\phi \\ \sin\phi \end{bmatrix}$$

where $\theta = (2\pi i)/N$ for $i = 0, \cdots, N-1$. The cone defined by these edges is written as:

$$\mathbf{V} = \begin{bmatrix} \boldsymbol{v}_0 & \cdots & \boldsymbol{v}_{N-1} \end{bmatrix}$$

The polyhedral cone is a common approximation to the Coulomb friction cone (covered in the course notes). Complete the function `generate_edges` in `assignment_1.py`. This function generates a cone given its parameters:

$$\mathbf{V} = \text{generate\_edges}(N, \phi)$$

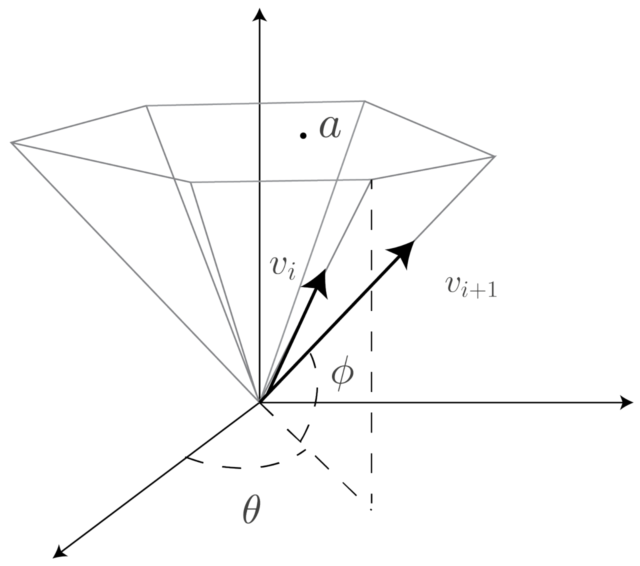


Figure 8. Polyhedral cone parameterized by ϕ and N.

Part 2

We can compute the facet normals (pointing inwards of the cone) using the cross-product of the edge vectors:

$$\boldsymbol{s}_i = \boldsymbol{v}_i \times \boldsymbol{v}_{i+1}$$

The cone generated by facet normals can be written as:

$$\mathbf{S} = \begin{bmatrix} \boldsymbol{s}_0 & \cdots & \boldsymbol{s}_{N-1} \end{bmatrix}$$

Complete the function `compute_normals` in `assignment_1.py`. This function generates the facet normal cone given the edge cone:

$$\mathbf{S} = \text{compute\_normals}(\mathbf{V})$$

Part 3

We can compute the distance of any point $\boldsymbol{a} = (a_x, a_y, a_z)$ in the interior of the polyhedral cone to facet i using:

$$d = \frac{\boldsymbol{s}_i \cdot \boldsymbol{a}}{||\boldsymbol{s}_i||}$$

Complete the function `compute_minimum_distance_from_facet_normals` in `assignment_1.py`. This function computess the minimum distance d^* of a given point $\boldsymbol{a}$ to the facet normal cone:

$$d^* = \text{compute\_minimum\_distance\_from\_facet\_normals}(a, \mathbf{S})$$

Part 4

We're going to string together the functions above. Complete the function `compute_minimum_distance.py` in `assignment_1.py` that computes the minimum distance d^* of a given point $\boldsymbol{a}$ to a polyhedral cone parameterized by N and ϕ:

$$d^* = \text{compute\_minimum\_distance}(\boldsymbol{a}, N, \phi)$$

Using the functions previously written for this purpose.

Part 5

Complete the function `check_is_interior_point` in `assignment_1.py` to check if a given point $\boldsymbol{a}$ is in the interior of a polyhedral cone parameterized by N and ϕ. The function should return true if the point is inside the cone, and false if it is outside. Hint: Think about what it means for a point to be in the interior of a polyhedral cone in terms of distances.