EECS C106A HW 0: Python Intro

EECS C106A: Introduction to Robotics, Fall 2019

Introduction

We will be using the Python programming language for labs in EECS C106a. Some hw assignments will entail matrix calculations where Python will come in handy, but you are welcome to use something like Matlab instead. This hw is meant to be a mini bootcamp on Python for students who have had experience programming in another language already (e.g. Matlab) or need a quick refresher on Python. We will be using a few popular libraries (numpy, scipy) that are very useful. If you have experience with Matlab but not Python, we recommend checking out the numpy for Matlab users guide (https://docs.scipy.org/doc/numpy/user/numpy-for-matlab-users.html).

Table of Contents

How to Submit the Notebook

Every place in the notebook that you are required to answer a question will be marked with "TODO". When you are completed with the notebook, `cltr + p (or cmd + p)` and save as a pdf. Submit the pdf file to Gradescope. Make sure to periodically save your notebook process by clicking File/Save and Checkpoint.

IPython Basics

For those who have never used IPython, it is a command shell for interactive Python programming. You can imagine it as allowing you to run blocks of Python code that you would execute in a single python script (python [script_name].py) in the terminal. Benefits of using IPython include easier visualization and debugging. The purpose of this bootcamp in IPython is to give you an idea of basic Python syntax and semantics. For labs you are still expected to write and execute actual Python scripts to work with ROS.

Executing Cells

ipython notebooks are constituted of cells, that each have text, code, or html scripts. To run a cell, click on the cell and press Shift+Enter or the run button on the toolbar. Run the cell below, you should expect an output of 6:

In [1]:
```
1+2+3
```
Out[1]: 6

Stopping or Restarting Kernel

To debug, or terminate a process, you can interupt the kernel by clicking Kernel/interrupt on the toolbar. If interuppting doesn't work, or you would like to restart all the processes in the notebook, click Kernel/restart. Try interrupting the following block:

In [2]:
```
import time

while True:
    print("bug")
    time.sleep(1.5)
```

```
bug
bug
bug

---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call las
t)
<ipython-input-2-eaaf4ade5b30> in <module>
      3 while True:
      4     print("bug")
----> 5     time.sleep(1.5)

KeyboardInterrupt:
```

Import a library

To import a certain library `x`, just type `import x` Calling function `y` from that library is simply `x.y` To give the library a different name (e.g. abbreviation), type `import x as z`

In [3]:
```
import numpy as np
np.add(3, 4)
```
Out[3]: 7

Python

Data Types

Integers and Floats

In Python2, integer division returns the floor. In Python3, there is no floor unless you specify using double slahes. The differences between Python2 and Python3 you can check out (https://wiki.python.org/moin/Python2orPython3), but we will be using Python2 in this class.

In [4]:
```
5 / 4
```
Out[4]:
```
1.25
```

In [5]:
```
5.0 / 4
```
Out[5]:
```
1.25
```

Booleans

Python implements all usual operators for Boolean logic. English, though, is used rather than symbols (no &, ||, etc.). Pay attention to the following syntax, try to guess what the output for each print statement should be before running the cell.

In [6]:
```
print(0 == False)

t = True
print(1 == t)

print(0 != t)

print(t is not 1)

if t is True:
    print(0 != 0)
```
```
True
True
True
True
False
```

Strings

Strings are supported very well. To concatenate strings we can do the following:

In [7]:
```
hello = 'hello'
robot = 'robot'

print(hello + ' ' + robot + str(1))
```
```
hello robot1
```

To find the length of a string use `len(...)`

In [8]:
```
print(len(hello + robot))
```

```
10
```

Lists

A list is a mutable array of data, meaning we can alter it after insantiating it. To create a list, use the square brackets [] and fill it with elements.

Key operations:

- `'+'` appends lists
- `len(y)` to get length of list y
- `y[0]` to index into 1st element of y **Python indices start from 0
- `y[1:6]` to slice elements 1 through 5 of y

In [9]:
```
y = ["Robots are c"] + [0, 0, 1]
y
```

Out[9]:
```
['Robots are c', 0, 0, 1]
```

In [10]:
```
len(y)
```

Out[10]:
```
4
```

In [11]:
```
y[0]
```

Out[11]:
```
'Robots are c'
```

In [12]:
```
# TODO: slice the first three elements of list 'y' and
# store in a new list, then print the 2nd element of this
# new list
new_list = y[:3]
print(new_list[1])
```

```
0
```

Loops

You can loop over the elements of a list like this:

In [13]:
```
robots = ['baxter', 'sawyer', 'turtlebot']
for robot in robots:
    print(robot)
# Prints "baxter", "sawyer", "turtlebot", each on its own line.
```

```
baxter
sawyer
turtlebot
```

If you want access to the index of each element within the body of a loop, use the built-in `enumerate` (https://docs.python.org/2.7/library/functions.html#enumerate) function:

In [14]:
```
robots = ['baxter', 'sawyer', 'turtlebot']

# TODO: Using a for loop and the python built-in enumerate function,
# Print "#1: baxter", "#2: sawyer", "#3: turtlebot",
# each on its own line
for i, robot in enumerate(robots):
    print('#' + str(i + 1) + ': ' + robot)
```

```
#1: baxter
#2: sawyer
#3: turtlebot
```

Numpy Array

The numpy array is like a list with multidimensional support and more functions (which can all be found here (https://docs.scipy.org/doc/numpy/reference/index.html)).

NumPy arrays can be manipulated with all sorts of arithmetic operations. You can think of them as more powerful lists. Many of the functions that already work with lists extend to numpy arrays.

To use functions in NumPy, we have to import NumPy to our workspace. by declaring `import numpy` , which we have done previously above in this notebook already. We typically rename `numpy` as `np` for ease of use.

Making a Numpy Array

In [15]:
```
x = np.array([[1, 2, 3], [4 , 5, 6], [7, 8, 9]])
print(x)
# x is a 3x3 matrix here
```

```
[[1 2 3]
 [4 5 6]
 [7 8 9]]
```

Finding the shape of a Numpy Array

In [16]:
```
x.shape # returns the dimensions of the numpy array
```

Out[16]:
```
(3, 3)
```

Elementwise operations

Arithmetic operations on numpy arrays correspond to elementwise operations.

In [17]:
```
print(x)
print
print(x * 5) # numpy carries operation on all elements!
```

```
[[1 2 3]
 [4 5 6]
 [7 8 9]]
[[ 5 10 15]
 [20 25 30]
 [35 40 45]]
```

Matrix multiplication

In [18]:
```
print(np.dot(x, x))
```

```
[[ 30  36  42]
 [ 66  81  96]
 [102 126 150]]
```

Slicing numpy arrays

Numpy uses pass-by-reference semantics so it creates views into the existing array, without implicit copying. This is particularly helpful with very large arrays because copying can be slow. Although be wary that you may be mutating an array when you don't intend to, so make sure to make a copy in these situations.

In [19]:
```
orig = np.array([0, 1, 2, 3, 4, 5])
print(orig)
```

```
[0 1 2 3 4 5]
```

Slicing an array is just like slicing a list

In [20]:
```
sliced = orig[1:4]
print(sliced)
```

```
[1 2 3]
```

Note, since slicing does not copy the array, mutating `sliced` mutates `orig`. Notice how the 4th element in `orig` changes to 9 as well.

In [21]:
```
sliced[2] = 9
print(orig)
print(sliced)
```

```
[0 1 2 9 4 5]
[1 2 9]
```

We should use `np.copy()` to actually copy `orig` if we don't want to mutate it.

In [22]:
```
orig = np.array([0, 1, 2, 3, 4, 5])
copy = np.copy(orig)
sliced_copy = copy[1:4]
sliced_copy[2] = 9
print(orig)
print(sliced_copy)
```

```
[0 1 2 3 4 5]
[1 2 9]
```

In [23]:
```
A = np.array([[5, 6, 8], [2, 4, 5], [3, 1, 10]])
B = np.array([[3, 5, 0], [3, 1, 1]])
# TODO: multiply matrix A with matrix B padded with 1's to the
# same dimensions as A; sum this result with the identiy matrix
# (you may find np.concatenate, np.vstack, np.hstack, or np.eye useful)
# Make sure you don't alter the original contents of B. Print the resul
B_copy = np.copy(B)
row_1s = np.array([1, 1, 1])
B_pad = np.vstack((B_copy, row_1s))
C = np.dot(A, B_pad) + np.eye(3)
print(C)
```

```
[[42. 39. 14.]
 [23. 20.  9.]
 [22. 26. 12.]]
```

Handy Numpy function: arange

We use `arange` to instantiate integer sequences in numpy arrays. It's similar to the built-in range function in Python for lists. However, it returns the result as a numpy array, rather a simple list.

`arange(0,N)` instantiates an array listing every integer from 0 to N-1.

`arange(0,N,i)` instantiates an array listing every `i` th integer from 0 to N-1 .

In [24]:
```
print(np.arange(-3,4)) # every integer from -3 ... 3
```

```
[-3 -2 -1  0  1  2  3]
```

In [25]:
```
# TODO: print every other integer from 0 ... 6 multiplied by 2
# as a list
print(np.arange(0, 7, 2) * 2)
```

```
[ 0  4  8 12]
```

Functions

Python functions are defined using the `def` keyword. For example:

In [26]:
```
def hello_robot(robot_name, yell=True):
    if yell:
        print('HELLO, %s!' % robot_name.upper())
    else:
        print('hello, %s' % robot_name)
```

In [27]:
```
hello_robot('Baxter') # Prints "HELLO, BAXTER!"
hello_robot('Sawyer', yell=False)  # Prints "hello, Sawyer"
```

```
HELLO, BAXTER!
hello, Sawyer
```

Rodrigues' Formula

The Rodrigues' Formula is a useful formula that allows us to calculate the corresponding rotation matrix R when given an axis ω an angle θ of rotation:

$$R = I_3 + \frac{\hat{\omega}}{\|\omega\|}\sin\left(\|\omega\|\,\theta\right) + \frac{\hat{\omega}^2}{\|\omega\|^2}(1 - \cos\left(\|\omega\|\,\theta\right))$$

where

$$\hat{\omega} = \widehat{\begin{bmatrix} \omega_1 \\ \omega_2 \\ \omega_3 \end{bmatrix}} = \begin{bmatrix} 0 & -\omega_3 & \omega_2 \\ \omega_3 & 0 & -\omega 1 \\ -\omega_2 & \omega 1 & 0 \end{bmatrix}$$

$\hat{\omega}$ is known as the skey-symmetric matrix form of ω. For now, you don't have to worry about the exact details and derivation of this formula since it will be discussed in class and the given formula alone should be enough to complete this problem. A sanity check for your rodrigues implementation is provided for your benefit.

In [28]:
```
# TODO: define a function that converts a rotation vector in 3D
# of shape (3,) to its coressponding skew-symmetric representation
# of shape (3, 3). This funciton will prove useful in the next questior
def skew_3d(omega):
    """
    Converts a rotation vector in 3D to its corresponding skew-symmetri

    Args:
    omega - (3,) ndarray: the rotation vector

    Returns:
    omega_hat - (3,3) ndarray: the corresponding skew symmetric matrix
    """
    if not omega.shape == (3,):
        raise TypeError('omega must be a 3-dim column vector')

    # YOUR CODE HERE
    x, y, z = omega[0], omega[1], omega[2]
    omega_hat = np.array([[0,-z,y], [z,0,-x], [-y,x,0]])

    return omega_hat
```

In [29]:

```
# TODO: define a function that when given an axis of rotation omega
# and angle of rotation theta, uses the Rodrigues' Formula to compute
# and return the corresponding 3D rotation matrix R.
# The Function has already been partially defined out for you below.
def rodrigues(omega, theta):
    """
    Computes a 3D rotation matrix given a rotation axis and angle of ro

    Args:
    omega - (3,) ndarray: the axis of rotation
    theta: the angle of rotation

    Returns:
    R - (3,3) ndarray: the resulting rotation matrix
    """
    if not omega.shape == (3,):
        raise TypeError('omega must be a 3-dim column vector')

    # YOUR CODE HERE
    omega_skew = skew_3d(omega)
    omega_norm = np.linalg.norm(omega)
    R = np.eye(3) + (omega_skew / omega_norm) * np.sin(omega_norm * \
        theta) + (np.dot(omega_skew, omega_skew) / omega_norm ** 2) * \
        (1 - np.cos(omega_norm * theta))

    return R
```

In [30]:

```
arg1 = np.array([2.0, 1, 3])
arg2 = 0.587
ret_desired = np.array([[-0.1325, -0.4234,  0.8962],
                        [ 0.8765, -0.4723, -0.0935],
                        [ 0.4629,  0.7731,  0.4337]])
print("sanity check for rodrigues:")
if np.allclose(rodrigues(arg1, arg2), ret_desired, rtol=1e-2):
    print("passed")
```

```
sanity check for rodrigues:
passed
```

References

- [1] EE 120 lab1
- [2] EECS 126 Lab01
- [3] EE 16a Python Bootcamp
- [4] CS 231n Python Numpy Tutorial. [Link (http://cs231n.github.io/python-numpy-tutorial/)](http://cs231n.github.io/python-numpy-tutorial/)

In [ ]:

```

```