Introdução a Robótica com Python

Jhonatan da Silva

Instituto Federal de Santa Catarina, Janeiro 3, 2017

Sumário

1 Breve introdução ao Numpy

1.1 Arrays

1.1.1 Operações básicas

Podemos usar as operações normais de adição, subtração, multiplicação e divisão com os vetores declarados no numpy. Primeiro importa-se a biblioteca numpy como np.

```
import numpy as np
```

Depois podemos declarar os nossos vetores com a função np.array([..])

```
u = np.array([3,2,1])
v = np.array([1,2,3])
```

Agora pode-se fazer as operações básicas normalmente

```
z = u + v
z = u - v
z = u * v
z = u / v
```

O que retornará respectivamente :

$$\begin{bmatrix} 4 & 4 & 4 \end{bmatrix}$$
$$\begin{bmatrix} 2 & 0 & -2 \end{bmatrix}$$
$$\begin{bmatrix} 3 & 4 & 3 \end{bmatrix}$$
$$\begin{bmatrix} 3 & 1 & \frac{1}{3} \end{bmatrix}$$

1.1.2 Algumas propriedades dos arrays

Fazendo uso de arange, reshape, shape, itemsize :

```
x = np.arange(0,9)
print(x)
print(x.shape)
print(x.itemsize)

y = x.reshape((3,3))
print(y)
print(y.shape)
print(y.itemsize)
```

$$x = \begin{bmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \end{bmatrix}$$
$$x.shape = (9\,,)$$
$$x.itemsize = 8$$
$$y = \begin{bmatrix} 0 & 1 & 2 \\ 3 & 4 & 5 \\ 6 & 7 & 8 \end{bmatrix}$$
$$y.shape = (3\,,\,3)$$
$$y.itemsize = 8$$

1.1.3 Produto Interno e Vetorial

Para o produto interno, usa-se a função np.inner(array) :

```
# Usando inner para produto interno

u = np.array([3,2,1])
v = np.array([1,2,3])

z = np.inner(v,u)

# retorna z = 10
```

Para o produto vetorial usa-se a função np.cross(array) :

```python
# Usando cross para produto vetorial

i = [1,0,0]
j = [0,1,0]

k = np.cross(i,j)
```

$$k = \begin{bmatrix} 0 & 0 & 1 \end{bmatrix}$$

1.1.4 Somando elementos de um array

Podemos somar elementos de um array com np.sum(vetor)

```python
x = np.array([1,1,1])

soma = sum(x)
print(soma)
```

$$x = \begin{bmatrix} 1 & 1 & 1 \end{bmatrix}$$

$$soma = 3$$

1.2 Matrizes

Podemos declarar um array normalmente apenas com mais dimensões, ou podemos usar a função np.reshape((m,n)) para transformar um array de uma dimensão em uma matriz qualquer. Desde que a quantidade de elementos seja compatível.

```python
A = np.array([[1,1,1], [2,2,2], [3,3,3]])
print(A)

x = np.array([1,2,3,4,5,6,7,8,9])

B = x.reshape((3,3))
print(B)
```

$$A = \begin{bmatrix} 1 & 1 & 1 \\ 2 & 2 & 2 \\ 3 & 3 & 3 \end{bmatrix}$$

$$x = \begin{bmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \end{bmatrix}$$

$$B = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

1.2.1 Algumas matrizes características

Fazendo uso da biblioteca matlib do numpy pode-se fazer diversos tipos de matrizes vazias, com zeros, uns, identidade, aleatórias, com números da distribuição normal. Primeiro devemos importar matlib

```python
import numpy.matlib
```

Agora com a função np.matlib.tipodematriz(dim) podemos criar alguns tipos de matrizes

```python
# Criando uma matriz vazia

A = np.matlib.zeros((3,3))

# Criando uma matriz Identidade

I = np.matlib.identity(3)

# Criando matrizes com random

B = np.matlib.rand((3,3))

# Criando matriz com random mas usando valores da tabela de distribuicao normal

N = np.matlib.randn((3,3))
```

$$A = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

$$I = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

1.2.2 Soma de matrizes

```
import numpy as np
import numpy.matlib

# soma das matrizes
A = np.array([[1,0],[0,2]])
B = np.array([[0,1],[1,0]])
C = A + B
print(C)
```

$$C = \begin{bmatrix} 1 & 1 \\ 1 & 2 \end{bmatrix}$$

1.2.3 Somando os elementos das linhas

```
# soma das linhas
A = np.array([[1,0],[0,2]])
B = np.array([[0,1],[1,0]])
s_linha = sum(A)
print(s_linha)
```

$$s\_linha = \begin{bmatrix} 1 & 2 \end{bmatrix}$$

1.2.4 Soma de todos elementos

```
# soma dos elementos
A = np.array([[1,0],[0,2]])
B = np.array([[0,1],[1,0]])
soma = sum(sum(A))
print(soma)
```

$$soma = 5$$

1.2.5 Subtração

```
A = np.array([[1,0],[0,2]])
B = np.array([[0,1],[1,0]])
C = A - B
print(C)
```

$$C = \begin{bmatrix} 1 & -1 \\ -1 & 2 \end{bmatrix}$$

1.2.6 Multiplicação

```
A = np.array([[1,0],[0,2]])
B = np.array([[0,1],[1,0]])
C = np.matmul(A,B)
print(C)
```

$$C = \begin{bmatrix} 0 & 1 \\ 2 & 0 \end{bmatrix}$$

1.2.7 Transposta

```
A = np.array([[1,0],[0,2]])
A_transposta = A.T
print(A_transposta)
```

$$A\_transposta = \begin{bmatrix} 1 & 0 \\ 0 & 2 \end{bmatrix}$$

1.2.8 Inversa

Para computar a inversa primeiro temos que importar do módulo de algebra linear do numpy :

```
from numpy.linalg import *
from numpy import linalg as LA
```

Agora podemos usar a função inv, que tem argumento uma matriz e retorna sua inversa.

```
A = np.array([[1,3],[2,0]])
A_inv = inv(A)
print(A_inv)
I = np.matmul(A,A_inv)
print(I)
```

$$A\_inv = \begin{bmatrix} 0.00 & 0.50 \\ 0.33 & -0.16 \end{bmatrix}$$

$$I = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

1.2.9 Determinante

Usando o módulo de algebra linear como LA :

```
A = ([2,2],[4,8])
A_det = LA.det(A)
print(A_det)
```

1.2.10 Potência

Novamente usando o módulo como LA :

```
A = ([[1,2],[1,2]])
A_n = LA.matrix_power(A, 2)
```

2 Matrizes de Transformação

2.1 Computação Numérica

2.1.1 Reflexão

2.1.2 Rotação

2.1.3 Translação

2.1.4 Transformação homogênea

3 Usando PyDy

3.1 Sistema de Referência

Às vezes queremos criar um eixo de referência novo, por exemplo quando se quer fazer a rotação de um vetor com um sistema de referência. Com python podemos criar esse novo eixo de referência com a biblioteca PyDy que foi desenvolvida para simular dinâmica de muitos corpos com python.

Primeiro temos que importar as bibliotecas necessárias, será importado a biblioteca sympy que é usada para computação simbólica e da biblioteca sympy será importado o módulo mechanics. O * depois de import significa que será importado todos os módulos do sympy

```python
from sympy import *
from sympy.physics.mechanics import *
import numpy as np
```

Agora temos que criar nosso primeiro sistema de referência usando a função ReferenceFrame('(nome do sistema de referência)') e a partir dele criar um novo vetor, o qual podemos fazer diversas operações.

```python
N = ReferenceFrame('N')
v = 1 * N.x + 2 * N.y + 3 * N.z

print(v)
print(2 * v)
print(np.log(int(v.magnitude())))
print(v.normalize())

u = v.normalize()
print(u.magnitude())
```

$$v = N.x + 2N.y + 3N.z$$

$$2v = 2N.x + 4N.y + 6N.z$$

$$np.\log|magnitude(v)| = 1.0986123$$

$$v.normalize() = \frac{\sqrt{14}}{14N.x} + \frac{\sqrt{14}}{7N.y} + \frac{3\sqrt{14}}{14N.z}$$

$$u.magnitude() = 1.0$$