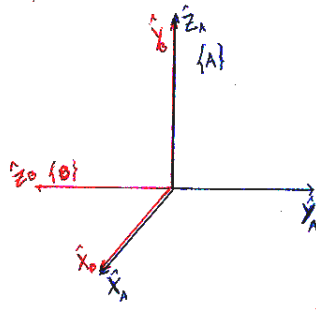


Sumário

1 Matrizes de Rotação e Translação

Quando temos um frame de referência A e outro frame de referência B e queremos saber qual relação por exemplo do eixo x em A, \hat{X}^A com o eixo x em B, \hat{X}^B , como na imagem abaixo,



podemos utilizar de matrizes de rotação, geralmente uma matriz de rotação R , tem a seguinte forma geral.

$$R := \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix}$$

Assim com matrizes de rotação podemos escrever a relação entre \hat{X}^A e \hat{X}^B em termos de ${}^A_B R$, onde

$${}^A \hat{X}_B = {}^A_B R {}^B \hat{X}_B$$

e ${}^A_B R$ é definido como

$${}^A_B R = [{}^A \hat{X}_B \quad {}^A \hat{Y}_B \quad {}^A \hat{Z}_B]$$

Nos resta determinar ${}^A \hat{X}_B$ ${}^A \hat{Y}_B$ ${}^A \hat{Z}_B$,

$${}^A \hat{X}_B = \begin{bmatrix} \hat{X}_B \hat{X}_A \\ \hat{X}_B \hat{Y}_A \\ \hat{X}_B \hat{Z}_A \end{bmatrix}$$

1.1 Computação simbólica com python

Como faríamos se quiséssemos definir uma matriz simbólica tal qual igual a R , Em uma linguagem de programação como python? O python possui uma biblioteca em específico chamada sympy que permite que o usuário compute matemática simbólica, um simples código segue abaixo,

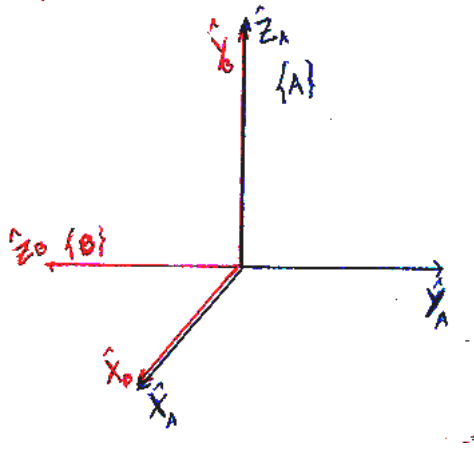
```
1 from sympy import symbols, Matrix
2
3 (r11, r12, r13,
4  r21, r22, r23,
5  r31, r32, r33) = symbols('r11 r12 r13 r21 r22 r23 r31 r32 r33')
6
7 R = Matrix([[r11, r12, r13], [r21, r22, r23], [r31, r32, r33]])
```

Que retornará

$$R = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix}$$

1.2 Definindo a matriz de rotação

Definindo os dois frames de referência pela imagem abaixo



Como poderíamos determinar ${}^A_B R$ tal que $\hat{X}^A \mapsto \hat{X}^B$ Pela definição anterior de que

$${}^A_B R = [{}^A\hat{X}_B \quad {}^A\hat{Y}_B \quad {}^A\hat{Z}_B]$$

Pela matriz podemos ver que cada coluna na verdade é a cordenada de B em relação a A, como vemos na imagem as coordenadas de \hat{X}^A e \hat{X}^B são as mesmas, já, \hat{Y}^B tem coordenadas em \hat{Z}^A enquanto \hat{Z}^B tem coordenadas em \hat{Y}^A . Então podemos escrever ${}^A_B R$ como

$${}^A_B R = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & -1 \\ 0 & 1 & 0 \end{bmatrix}$$

E como R é ortonormal sabemos que ${}^A_B R^T = {}^B_A R = {}^A_B R^{-1}$

1.3 Exemplo de rotação com python

Quando lida-se com computação numérica ao contrário da simbólica usa-se o numpy, usualmente importa-se o numpy como np como se vê no código abaixo

```
1 import numpy as np
2
3 R = np.matrix('1 0 0; 0 0 -1; 0 1 0')
```

Que retornará

$$R = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & -1 \\ 0 & 1 & 0 \end{bmatrix}$$

1.4 Usando C++

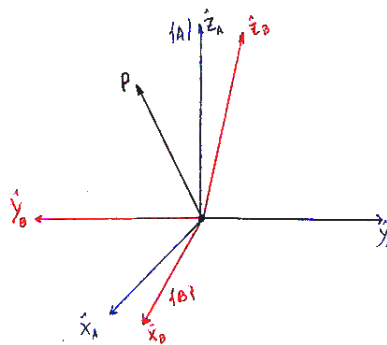
Como ficaria a sintaxe do programa escrito em python em C++ ? Para o código em questão foi usado a biblioteca Eigen que permite trabalhar com algebra linear em C++

```
1 #include <iostream>
2 #include <Eigen/Dense>
3
4 using Eigen::MatrixXd;
5 using namespace std;
6
7 int main()
8 {
9     MatrixXd R(3,3);
10    R << 1, 0, 0,
11         0, 0, -1,
12         0, 1, 0;
13    cout << R;
14 }
```

Esse código tem o mesmo retorno que o anterior.

1.5 Rotação geral

E quando temos um ponto P em termos de B e queremos saber como definir o mesmo em termos de A, por exemplo na imagem

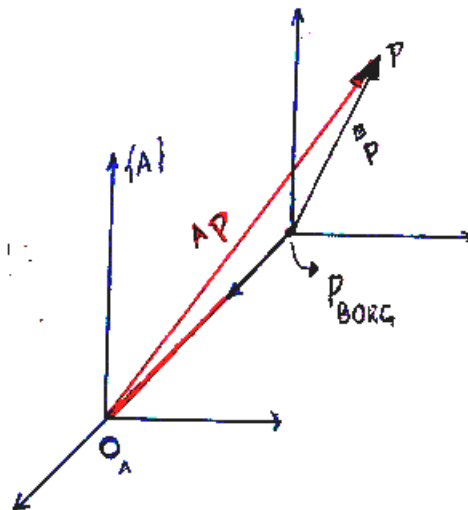


Podemos definir o ponto P em A como ${}^A P$ assim temos

$${}^A P = {}^A_B R {}^B P$$

1.6 Translações

Quando temos um ponto P em um frame A e queremos transladá-lo sem rotacionar ou mudar de frame como mostra a imagem a seguir

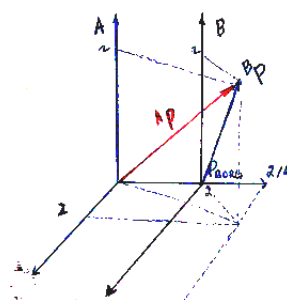


Podemos definir ${}^A P$ como

$${}^A P = {}^B P + {}^A P_{BORG}$$

1.7 Exemplo de translação em python

Como nenhum exemplo prático foi definido na seção anterior antes de fazer o código precisamos de algumas definições, segundo a imagem



Então temos o primeiro frame de referência A e o segundo B , que está deslocado duas casas no eixo y , então nosso ponto de origem para B é $(0,2,0)$, pode-se notar que em relação ao frame B , a posição y é 2 mas em relação a A é 4, agora temos algumas definições

$${}^B P = \begin{bmatrix} 2 \\ 2 \\ 2 \end{bmatrix}$$

$$P_{BORG} = \begin{bmatrix} 0 \\ 2 \\ 0 \end{bmatrix}$$

$${}^A P = P_b + P_{borg}$$

E agora podemos fazer o código para computar ${}^A P$

```
1 import numpy as np
2
3 Pborg = np.matrix('0; 2; 0')
4 Pb = np.matrix('2; 2; 2')
5
6 Pa = Pb + Pborg
```

Que irá retornar

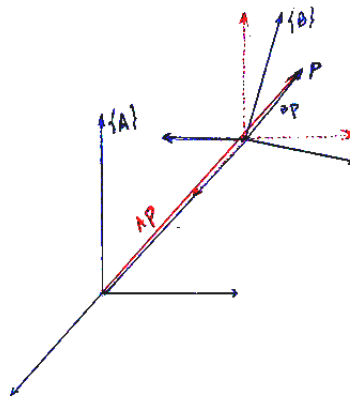
$${}^A P = \begin{bmatrix} 2 \\ 4 \\ 2 \end{bmatrix}$$

1.8 Exemplo de translação em C++

```
1 #include <iostream>
2 #include <Eigen/Dense>
3
4 using Eigen::MatrixXd;
5 using namespace std;
6
7 int main()
8 {
9     MatrixXd Pborg(3,1);
10    Pborg << 0,
11            2,
12            0;
13    cout << Pborg << endl;
14    MatrixXd Pb(3,1);
15    Pb << 2,
16         2,
17         2;
18    cout << Pb << endl;
19    cout << "\n\n Ponto final" << endl;
20    MatrixXd Pa(3,1);
21    Pa << Pborg + Pb;
22    cout << Pa;
23 }
```

1.9 Transformação Geral

Se temos um caso onde foi feita uma translação e uma rotação em relação a um novo frame, e queremos saber qual é o novo ponto em relação ao frame original, como na imagem abaixo



Primeiro temos que fazer a matriz de rotação ${}^A_B R$ pois o novo ponto está em um frame diferente do original, então não podemos apenas fazer a soma, depois de fazer ${}^A_B R$ aplicado em ${}^B P$ podemos fazer a translação somando com ${}^A P_{BORG}$, em uma forma geral podemos escrever

$${}^A P = {}^A_B R {}^B P + {}^A P_{BORG}$$

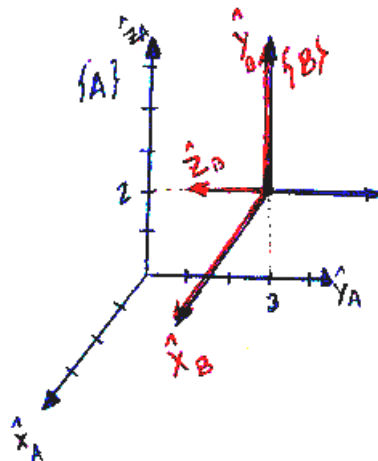
1.9.1 A Transformação Homogênea

Seria interessante ter uma matriz que resolvesse o problema geral, onde temos a rotação e a translação, porque computacionalmente falando, é mais interessante ter uma multiplicação de uma matriz com um vetor que ter várias operações. A transformação homogênea resolve esse problema, expandindo para uma matriz (4,4) podemos reescrever a expressão ${}^A P = {}^A_B R {}^B P + {}^A P_{BORG}$ em forma matricial

$$\begin{bmatrix} {}^A P \\ 1 \end{bmatrix} = \begin{bmatrix} {}^A_B R & {}^A P_{BORG} \\ 0 & 0 \end{bmatrix} \begin{bmatrix} {}^B P \\ 1 \end{bmatrix}$$

1.9.2 Exemplo

Um exemplo prático de como funciona a transformação homogênea, pela imagem abaixo,



Temos um frame de referência A e outro B, pelo que se vê na imagem o frame de referência continuou no mesmo plano xy, deslocou 3 casas em direção ao eixo y e 2 em relação ao eixo z, definindo a translação como q temos que

$$q = \begin{bmatrix} 0 \\ 3 \\ 2 \end{bmatrix}$$

Agora precisamos definir ${}^A_B R$, que tem a mesma rotação do primeiro exemplo de rotações

$${}^A_B R = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 0 & 1 \\ 0 & -1 & 0 \end{bmatrix}$$

Agora podemos definir uma transformação homogênea ${}^A_B T$ de tal modo

$${}^A_B T = \begin{bmatrix} {}^A_B R & {}^A P_{BORG} \\ 00 & 0 \end{bmatrix}$$

$${}^A_B T = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & -1 & 3 \\ 0 & 1 & 0 & 2 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Se definirmos um ponto ${}^B P$

$${}^B P = \begin{bmatrix} 0 \\ 1 \\ 1 \\ 1 \end{bmatrix}$$

Onde a quarta cordenada de ${}^B P$ não é levada em conta no final. Podemos saber o novo ponto ${}^A P$ fazendo

$${}^A P = {}^A_B T {}^B P$$

$${}^A P = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & -1 & 3 \\ 0 & 1 & 0 & 2 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \\ 1 \\ 1 \end{bmatrix} \quad {}^A P = \begin{bmatrix} 0 \\ 2 \\ 3 \\ 1 \end{bmatrix}$$

$$\therefore {}^A P = \begin{bmatrix} 0 \\ 2 \\ 3 \end{bmatrix}$$

1.10 Exemplo de transformação homogênea com python

Precisamos definir a matriz de transformação ${}^A_B T$ o ponto ${}^B P$ e agora que temos essas duas informações podemos computar ${}^A P$ com a multiplicação dos dois com `np.dot(${}^A_B T$, ${}^B P$)`, e do resultado final pegar apenas as 3 primeiras coordenadas que resultam no ponto desejado.

```
1 import numpy as np
2
3 T = np.matrix('1 0 0 0; 0 0 -1 3; 0 1 0 2; 0 0 0 1')
4
5 Pb = np.matrix('0;1;1;1')
6
7 Pa = np.dot(T,Pb)
8
9 Pa = Pa[0:3]
```

O código retornará

$${}^A P = \begin{bmatrix} 0 \\ 2 \\ 3 \end{bmatrix}$$

1.11 Exemplo de transformação homogênea com C++

```
1 #include <iostream>
2 #include <Eigen/Dense>
3
4 using Eigen::MatrixXd;
5 using namespace std;
6
7 int main()
8 {
9     MatrixXd T(4,4);
10    T << 1, 0, 0, 0,
11         0, 0, -1, 3,
12         0, 1, 0, 2,
13         0, 0, 0, 1;
14    MatrixXd Pb(4,1);
15    Pb << 0,
16         1,
```



```

17         1,
18         1;
19     MatrixXd Pa(4,1);
20     Pa << T * Pb;
21     MatrixXd P(3,1);
22     int i = 0;
23     for(i = 0; i < 3; i++)
24     {
25         P(i) = Pa(i);
26     }
27     cout << P;
28 }

```