ch02_transforms

January 19, 2022

0.1 Problem 1: Basic concentric frame transforms

Most of the frames have the same origin. The code in chap2/transforms.py calculates the rotation matrices used to transform between these frames. Correctly implement the following functions: * rot_x: calculate elementary rotation matrix about x-axis * rot_y: calculate elementary rotation matrix about y-axis * rot_z: calculate elementary rotation matrix about z-axis * rot_v_to_v1: calculates the rotation from frame v to v1 * rot_v1_to_v2: calculates the rotation from frame v1 to v2 * rot_v2_to_b: calculates the rotation from v2 to body frame * rot_b_to_s: calculates the rotation from body to stability frame * rot_s_to_w: calculates the rotation from stability to wind frame

Hint: You should only compute the cosine and sine of the angle in *rot_x*, *rot_y*, and *rot_z*. All the remaining functions should call those functions (i.e., one line change from what they currently are)

Use these function to compute the following. Assume that $\psi = \frac{\pi}{4}$, $\theta = 0.3$, $\phi = 0.25$, $\alpha = 0.1$, and $\beta = 0.15$. Display the results in the exported pdf. * Compute p_1^{v1} given $p_1^v = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$ * Compute p_1^w * Compute p_2^s given $p_2^{v2} = \begin{bmatrix} 4 \\ 5 \\ 6 \end{bmatrix}$ * Compute p_2^v

[1]:

```
import numpy as np # Imports the numpy library and creates the alias np

from IPython.display          import display # Used to display variables nicely␣
 ↪in Jupyter
from mav_sim.chap2.transforms import rot_v_to_v1, rot_v1_to_v2, rot_v2_to_b,␣
 ↪rot_b_to_s, rot_s_to_w

# Calculate the required rotation matrices
psi   = np.pi/4
theta = 0.3
phi   = 0.25
alpha = 0.1
beta  = 0.15

# Initialize p1 and p2
p_1_v  = np.array([[1],[2],[3]])
```

```
p_2_v2 = np.array([[4],[5],[6]])

# Calculate p_1^v1
p_1_v1 = rot_v_to_v1(psi)@p_1_v
print('p_1^v1 = ')
display(p_1_v1)

# Calculate p_1^w
#       rot_v2_to_b(phi)@rot_v1_to_v2(theta)@rot_v_to_v1(psi)
p_1_w =␣
 ↪rot_s_to_w(beta)@rot_b_to_s(alpha)@rot_v2_to_b(phi)@rot_v1_to_v2(theta)@rot_v_to_v1(psi)@p_
print('p_1^w = ')
display(p_1_w)

# Calculate p_2^s
p_2_s = rot_b_to_s(alpha)@rot_v2_to_b(phi)@p_2_v2
print('p_2^s = ')
display(p_2_s)

# Calculate p_2^v
p_2_v = np.transpose(rot_v1_to_v2(theta)@rot_v_to_v1(psi))@p_2_v2
print('p_2^v = ')
display(p_2_v)
```

```
p_1^v1 =

array([[2.12132034],
       [0.70710678],
       [3.        ]])

p_1^w =

array([[1.66990864],
       [1.31449478],
       [3.07953058]])

p_2^s =

array([[4.43689977],
       [6.32898586],
       [4.15425786]])

p_2^v =

array([[0.42035179],
       [7.4914196 ],
       [4.54993811]])
```

0.2 Problem 2: Compound rotation matrices

The transform from the vehicle frame to the body frame can be written as a compound of three rotation matrices (and so can the inverse transform). However, these matrices are used so often that it is nice to avoid multiplying these three matrices each time the transform is needed.

Implement the following functions:

- rot_v_to_b: calculates the rotation from vehicle to body frame
- rot_b_to_v: calculates the rotation from body frame to vehicle frame

Hint: You really only need to implement one of them and then use a transpose for the other

Using the same values as above, show that your implementation produces the same rotation matrices as three elementary matrices multiplied together. Display the difference in the exported pdf.

[2]:

```
from mav_sim.chap2.transforms import rot_v_to_b, rot_b_to_v

# Calculate the rotation matrices as compound rotation matrices (i.e., matrix␣
↪multiplication)
R_v_to_b_mult = (rot_v2_to_b(phi)@rot_v1_to_v2(theta)@rot_v_to_v1(psi))
R_b_to_v_mult = np.transpose(R_v_to_b_mult)

# Calculate the rotation matrices using the functions
R_v_to_b_func = rot_v_to_b(psi,theta,phi)
R_b_to_v_func = rot_b_to_v(psi,theta,phi)

# Calculate and display the difference
R_v_to_b_diff = R_v_to_b_mult - R_v_to_b_func
print("R_v_to_b_diff")
display(R_v_to_b_diff)

R_b_to_v_diff = R_b_to_v_mult - R_b_to_v_func
print("R_b_to_v_diff")
display(R_b_to_v_diff)
```

```
R_v_to_b_diff

array([[0., 0., 0.],
       [0., 0., 0.],
       [0., 0., 0.]])

R_b_to_v_diff

array([[0., 0., 0.],
       [0., 0., 0.],
       [0., 0., 0.]])
```

0.3 Problem 3: Tranform to vehicle frame

Converting to and from the inertial frame requires translation. Implement the following functions: * trans_i_to_v: transforms a point from inertial frame to the vehicle frame * trans_v_to_i:

transforms a point from vehicle frame to the inertial frame * trans_i_to_b: transforms a point from inertial frame to the body frame * trans_b_to_i: transforms a point from the body frame to the inertial frame

Note that the transform between inertial and the vehicle frame is purely translational. The transform between the vehicle and body frame is purely rotational. Thus, you can use the functions already implemented to make the *trans_i_to_b* and *trans_b_to_i* functions quite simple.

Given that the UAV is in the position $p_n = 1$, $p_e = 2$, and $p_d = 3$ with the angles defined as before, transform the following points to the body frame using the implemented functions:

$$p_3^i = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$$

$$p_4^i = \begin{bmatrix} 3 \\ 2 \\ 1 \end{bmatrix}$$

Transform the following point in the body frame to the inertial frame

$$p_5^b = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$$

Transform the following point in the wind frame to the inertial frame

$$p_6^w = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$$

Display the results in the exported pdf.

[3]:

```python
from mav_sim.chap2.transforms import trans_i_to_b, trans_b_to_i

# Create the pose of the aircraft
class Pose:
    def __init__(self) -> None:
        self.north: float     = 1.      # north position
        self.east: float      = 2.      # east position
        self.altitude: float  = 3.      # altitude
        self.phi: float       = phi     # roll angle
        self.theta: float     = theta   # pitch angle
        self.psi: float       = psi     # yaw angle
pose = Pose()

# Initialize the points
p_3_i = np.array([[1],[2],[3]])
p_4_i = np.array([[3],[2],[1]])
p_5_b = np.array([[1],[2],[3]])
p_6_w = np.array([[1],[2],[3]])
```

```
# Calculate p_3^b
p_3_b = trans_i_to_b(pose, p_3_i)
print("p_3^b = ")
display(p_3_b)

# Calculate p_4_b
p_4_b = trans_i_to_b(pose, p_4_i)
print("p_4^b = ")
display(p_4_b)

# Calcualte p_5^i
p_5_i = trans_b_to_i(pose, p_5_b)
print("p_5^i")
display(p_5_i)

# Calculate p_6^i
p_6_b = np.transpose(rot_s_to_w(beta)@rot_b_to_s(alpha))@p_6_w
p_6_i = trans_i_to_b(pose, p_6_b)
print("p_6^i")
display(p_6_i)
```

```
p_3^b =

array([[0.],
       [0.],
       [0.]])

p_4^b =

array([[ 1.94209023],
       [-1.73955994],
       [-1.09645645]])

p_5^i

array([[1.54090053],
       [4.2317526 ],
       [5.95410003]])

p_6^i

array([[-0.34427829],
       [ 0.4946215 ],
       [-0.17799618]])
```

0.4 Simple code checking

The following code does not need to change. It should just be used as a sanity check so that you know the code is implemented properly. The output should not have any lines reading `Failed test!`

[4]:
```python
from mav_sim.unit_tests.ch2_transforms_tests import run_all_tests
run_all_tests()
```

```
Starting rot_x test

Calculated output:
[[ 1.0000000e+00  0.0000000e+00  0.0000000e+00]
 [ 0.0000000e+00 -1.0000000e+00  1.2246468e-16]
 [ 0.0000000e+00 -1.2246468e-16 -1.0000000e+00]]
Expected output:
[[ 1  0  0]
 [ 0 -1  0]
 [ 0  0 -1]]
Passed test
Calculated output:
[[ 1.          0.          0.        ]
 [ 0.          0.0707372   0.99749499]
 [ 0.         -0.99749499  0.0707372 ]]
Expected output:
[[ 1.          0.          0.        ]
 [ 0.          0.0707372   0.99749499]
 [ 0.         -0.99749499  0.0707372 ]]
Passed test
End of test

Starting rot_y test

Calculated output:
[[-1.0000000e+00  0.0000000e+00 -1.2246468e-16]
 [ 0.0000000e+00  1.0000000e+00  0.0000000e+00]
 [ 1.2246468e-16  0.0000000e+00 -1.0000000e+00]]
Expected output:
[[-1  0  0]
 [ 0  1  0]
 [ 0  0 -1]]
Passed test
Calculated output:
[[ 0.0707372   0.         -0.99749499]
 [ 0.          1.          0.        ]
 [ 0.99749499  0.          0.0707372 ]]
Expected output:
[[ 0.0707372   0.         -0.99749499]
 [ 0.          1.          0.        ]
 [ 0.99749499  0.          0.0707372 ]]
Passed test
End of test
```

```
Starting rot_z test

Calculated output:
[[-1.0000000e+00  1.2246468e-16  0.0000000e+00]
 [-1.2246468e-16 -1.0000000e+00  0.0000000e+00]
 [ 0.0000000e+00  0.0000000e+00  1.0000000e+00]]
Expected output:
[[-1  0  0]
 [ 0 -1  0]
 [ 0  0  1]]
Passed test
Calculated output:
[[ 0.0707372   0.99749499  0.        ]
 [-0.99749499  0.0707372   0.        ]
 [ 0.          0.          1.        ]]
Expected output:
[[ 0.0707372   0.99749499  0.        ]
 [-0.99749499  0.0707372   0.        ]
 [ 0.          0.          1.        ]]
Passed test
End of test

Starting rot_v_to_v1 test

Calculated output:
[[-0.70710678  0.70710678  0.        ]
 [-0.70710678 -0.70710678  0.        ]
 [ 0.          0.          1.        ]]
Expected output:
[[-0.70710678  0.70710678  0.        ]
 [-0.70710678 -0.70710678  0.        ]
 [ 0.          0.          1.        ]]
Passed test
Calculated output:
[[-0.80114362  0.59847214  0.        ]
 [-0.59847214 -0.80114362  0.        ]
 [ 0.          0.          1.        ]]
Expected output:
[[-0.80114362  0.59847214  0.        ]
 [-0.59847214 -0.80114362  0.        ]
 [ 0.          0.          1.        ]]
Passed test
End of test

Starting rot_v1_to_v2 test

Calculated output:
[[-0.70710678  0.         -0.70710678]
```

```
 [ 0.          1.          0.        ]
 [ 0.70710678  0.         -0.70710678]]
Expected output:
[[-0.70710678  0.         -0.70710678]
 [ 0.          1.          0.        ]
 [ 0.70710678  0.         -0.70710678]]
Passed test
Calculated output:
[[-0.80114362  0.         -0.59847214]
 [ 0.          1.          0.        ]
 [ 0.59847214  0.         -0.80114362]]
Expected output:
[[-0.80114362  0.         -0.59847214]
 [ 0.          1.          0.        ]
 [ 0.59847214  0.         -0.80114362]]
Passed test
End of test

Starting rot_v2_to_b test

Calculated output:
[[ 1.          0.          0.        ]
 [ 0.         -0.70710678  0.70710678]
 [ 0.         -0.70710678 -0.70710678]]
Expected output:
[[ 1.          0.          0.        ]
 [ 0.         -0.70710678  0.70710678]
 [ 0.         -0.70710678 -0.70710678]]
Passed test
Calculated output:
[[ 1.          0.          0.        ]
 [ 0.         -0.80114362  0.59847214]
 [ 0.         -0.59847214 -0.80114362]]
Expected output:
[[ 1.          0.          0.        ]
 [ 0.         -0.80114362  0.59847214]
 [ 0.         -0.59847214 -0.80114362]]
Passed test
End of test

Starting rot_b_to_s test

Calculated output:
[[-0.70710678  0.          0.70710678]
 [ 0.          1.          0.        ]
 [-0.70710678  0.         -0.70710678]]
Expected output:
[[-0.70710678  0.          0.70710678]
```

```
 [ 0.          1.          0.        ]
 [-0.70710678  0.         -0.70710678]]
Passed test
Calculated output:
[[-0.80114362  0.          0.59847214]
 [ 0.          1.          0.        ]
 [-0.59847214  0.         -0.80114362]]
Expected output:
[[-0.80114362  0.          0.59847214]
 [ 0.          1.          0.        ]
 [-0.59847214  0.         -0.80114362]]
Passed test
End of test

Starting rot_s_to_w test

Calculated output:
[[-0.70710678  0.70710678  0.        ]
 [-0.70710678 -0.70710678  0.        ]
 [ 0.          0.          1.        ]]
Expected output:
[[-0.70710678  0.70710678  0.        ]
 [-0.70710678 -0.70710678  0.        ]
 [ 0.          0.          1.        ]]
Passed test
Calculated output:
[[-0.80114362  0.59847214  0.        ]
 [-0.59847214 -0.80114362  0.        ]
 [ 0.          0.          1.        ]]
Expected output:
[[-0.80114362  0.59847214  0.        ]
 [-0.59847214 -0.80114362  0.        ]
 [ 0.          0.          1.        ]]
Passed test
End of test

Starting rot_v_to_b test

Calculated output:
[[ 0.5        -0.5         0.70710678]
 [-0.14644661 -0.85355339 -0.5       ]
 [ 0.85355339  0.14644661 -0.5       ]]
Expected output:
[[ 0.5        -0.5         0.70710678]
 [-0.14644661 -0.85355339 -0.5       ]
 [ 0.85355339  0.14644661 -0.5       ]]
Passed test
Calculated output:
```

```
[[ 0.7502363  -0.56044324  0.35078323]
 [-0.53071095 -0.82688153 -0.18604522]
 [ 0.39432396 -0.04658662 -0.9177899 ]]
Expected output:
[[ 0.7502363  -0.56044324  0.35078323]
 [-0.53071095 -0.82688153 -0.18604522]
 [ 0.39432396 -0.04658662 -0.9177899 ]]
Passed test
End of test

Starting rot_b_to_v test

Calculated output:
[[ 0.5        -0.14644661  0.85355339]
 [-0.5        -0.85355339  0.14644661]
 [ 0.70710678 -0.5        -0.5       ]]
Expected output:
[[ 0.5        -0.14644661  0.85355339]
 [-0.5        -0.85355339  0.14644661]
 [ 0.70710678 -0.5        -0.5       ]]
Passed test
Calculated output:
[[ 0.7502363  -0.53071095  0.39432396]
 [-0.56044324 -0.82688153 -0.04658662]
 [ 0.35078323 -0.18604522 -0.9177899 ]]
Expected output:
[[ 0.7502363  -0.53071095  0.39432396]
 [-0.56044324 -0.82688153 -0.04658662]
 [ 0.35078323 -0.18604522 -0.9177899 ]]
Passed test
End of test

Starting trans_i_to_v test

Calculated output:
[[ -27.]
 [  16.]
 [-146.]]
Expected output:
[[ -27]
 [  16]
 [-146]]
Passed test
End of test

Starting trans_v_to_i test

Calculated output:
```

```
[[ 57.]
 [180.]
 [242.]]
Expected output:
[[ 57]
 [180]
 [242]]
Passed test
End of test

Starting trans_i_to_b test

Calculated output:
[[-97.73759005]
 [ 55.3890873 ]
 [-98.3890873 ]]
Expected output:
[[-97.73759005]
 [ 55.3890873 ]
 [-98.3890873 ]]
Passed test
End of test

Starting trans_b_to_i test

Calculated output:
[[ 89.82233047]
 [-16.17766953]
 [179.60660172]]
Expected output:
[[ 89.82233047]
 [-16.17766953]
 [179.60660172]]
Passed test
End of test
```