

ME449 Homework 1

Sean Morton, 10/14/22

Part 1B

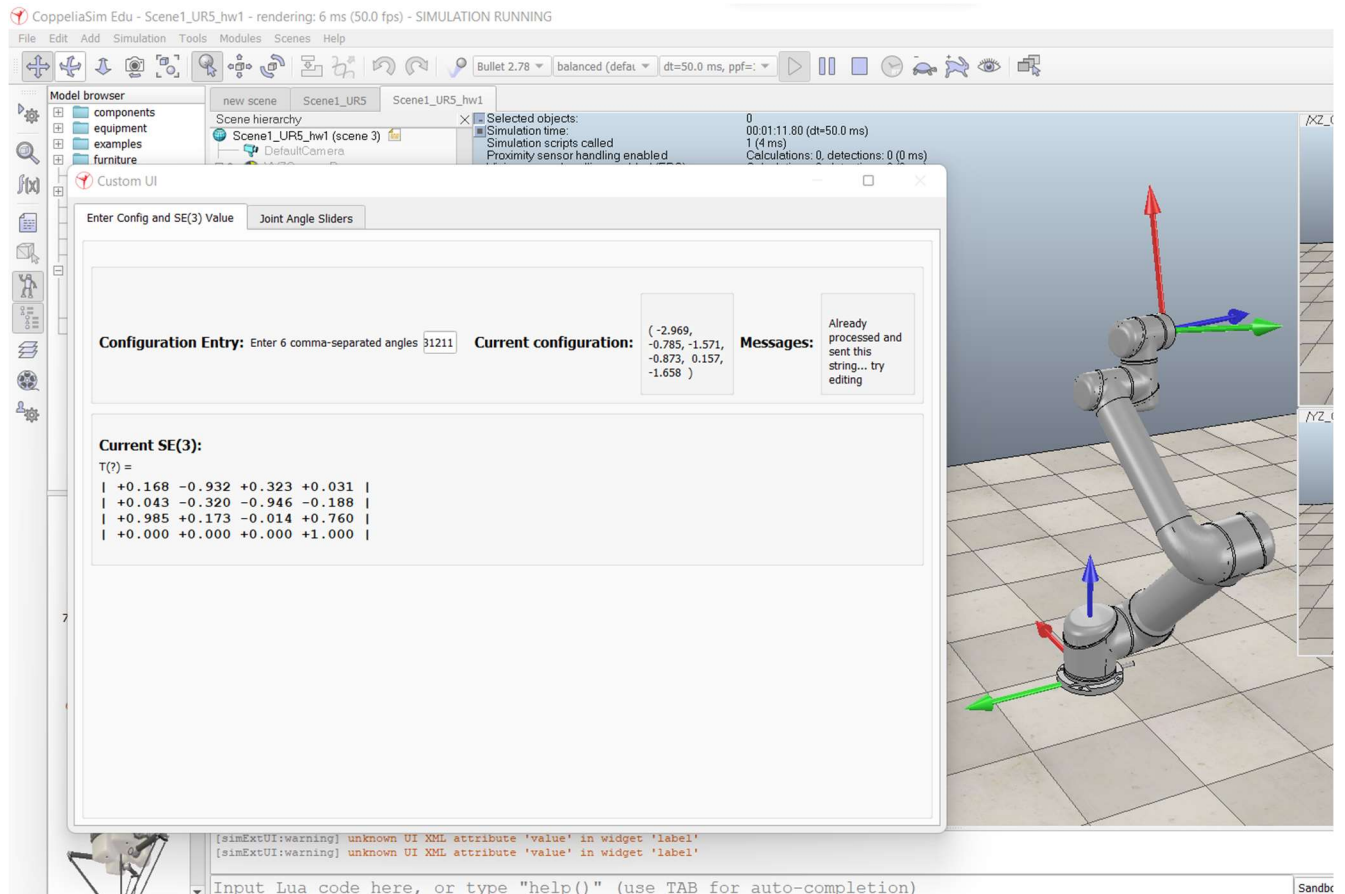
```

238     sim.addStatusbarMessage('Window '..h..' is closing...')
239     simUI.hide(h)
240 end
241
242 if (sim_call_type==sim.syscb_init) then
243     -- joint limits:
244     -- 1: (-360, 360)
245     -- 2: (-360, 360)
246     -- 3: (-360, 360)
247     -- 4: (-360, 360)
248     -- 5: (-360, 360)
249     -- 6: (-360, 360)
250
251     xml = {}
252     <ui closeable="false" on-close="closeEventHandler" resizable="true">
253         <tabs>
254
255             <tab title="Enter Config and SE(3) Value">
256                 <group>
257                     <group layout="hbox">
258                         <label text="<big> Configuration Entry:</big>" wordwrap="false"
259                         <label text="Enter 6 comma-separated angles" />
260                         <edit value="" id="7006" oneditingfinished="fullJointEntry" />
261                         <label value="" id="5006" wordwrap="false" />
262                         <label text="<big> Current configuration:</big>" id="6007" word
263                     <group layout="hbox">
264                         <label value="" id="1237" wordwrap="true" />
265                     </group>
266                     <label text="<big> Messages:</big>" id="6006" wordwrap="false"
267                     <group layout="hbox">
268                         <label value="" id="1236" wordwrap="true" />
269                     </group>
270                 </group>

```

This is the UI script for Scene 1 in CoppeliaSim. I changed the group layouts of the UI so that all the vertical boxes “vbox” were changed to horizontal boxes “hbox”.

Parts 1B + 2



This is a screenshot of my CoppeliaSim Scene, showing:

- The changes to the UI I made
- The position of the robot after moving to the joint angles calculated in my Python script
- The SE(3) matrix corresponding to the orientation of the robot. The rotation matrix within the SE(3) representation is approximately equal to the rotation matrix R_{sb} I calculated in Python, within 2 decimal places. Below: comparison between CoppeliaSim and Python output

Current SE(3):
 $T(?) = \begin{bmatrix} +0.168 & -0.932 & +0.323 & +0.031 \\ +0.043 & -0.320 & -0.946 & -0.188 \\ +0.985 & +0.173 & -0.014 & +0.760 \\ +0.000 & +0.000 & +0.000 & +1.000 \end{bmatrix}$

Rotation matrix R_{sb} :
 $\begin{bmatrix} 0.1676 & -0.9308 & 0.3250 \\ 0.0434 & -0.3224 & -0.9456 \\ 0.9849 & 0.1726 & -0.0136 \end{bmatrix}$

See attached for Python code + full outputs

Modern Robotics HW1 - Part 2

Sean Morton, 10/14/22

My method of calculating the joint angles given the rotation matrices was as follows:

1. calculate the rotation matrices for frame $\{i+1\}$ relative to frame $\{i\}$ using the rotation matrices provided
2. convert rotation matrices to $so(3)$ representation
3. convert $so(3)$ representation to axis-and-angle representation
4. if output of step [3] gave an axis opposite to our reference axes, multiply the axis and the angle by -1
5. calculate R_{sb} through 2 methods to verify matrix calculations worked

Calculated joint angles: [-2.969482157066879, -0.7853926894212007, -1.5707661989213484, -0.8726096667837093, 0.15704051490320972, -1.658121567631211]

For rotation matrix R_{sb} : see code output below

```
In [14]: import core as mr
import numpy as np
```

Define axes of rotation for each joint:

```
In [15]: axes_ref = [
    [0, 0, 1],
    [0, 1, 0],
    [0, 1, 0],
    [0, 1, 0],
    [0, 0, -1],
    [0, 1, 0],
]
```

Matrix calculations:

```
In [16]: #define existing rotation matrices
R13 = np.matrix([[ -0.7071, 0, -0.7071], [0, 1, 0], [0.7071, 0, -0.7071]])
Rs2 = np.matrix([[ -0.6964, 0.1736, 0.6964], [ -0.1228, -0.9848, 0.1228], [0.7071, 0, 0.7071]])
R15 = np.matrix([[ -0.9839, -0.1558, 0.0872], [ -0.1564, 0.9877, 0], [ -0.0861, -0.0136, 0.9961]])
R12 = np.matrix([[0.7071, 0, -0.7071], [0, 1, 0], [0.7071, 0, 0.7071]])
R34 = np.matrix([[0.6428, 0, -0.7660], [0, 1, 0], [0.7660, 0, 0.6428]])
Rs6 = np.matrix([[ -0.1676, 0.3250, -0.9308], [ -0.0434, -0.9456, -0.3224], [ -0.9849, -0.1736, 0.6964]])
R6b = np.matrix([[ -1, 0, 0], [0, 0, 1], [0, 1, 0]])

#define new R matrices in terms of old ones
Rs1 = Rs2 * R12.T
#R12 given
R23 = R12.T * R13
#R34 given
```

```

R45 = R34.T * R13.T * R15
R56 = R15.T * R12 * Rs2.T * Rs6

#rotation matrix Rsb
Rsb = Rs6 * R6b

```

Print results of matrix calculations:

```

In [17]: #source for formatting: https://tinyurl.com/2m6w7r8n
np.set_printoptions(formatter={'float_kind':'{: .4f}'.format})

R_array = [Rs1, R12, R23, R34, R45, R56] #b is fixed relative to 6, so this isn't a j
for i, R in enumerate(R_array):

    if i == 0:
        print("Rs1:")
    else:
        print(f"R{i}{i+1}:")
    print(R.round(4), end = "\n\n")

```

```

Rs1:
[[-0.9848  0.1736  0.0000]
 [-0.1737 -0.9848 -0.0000]
 [0.0000  0.0000  1.0000]]

```

```

R12:
[[0.7071  0.0000 -0.7071]
 [0.0000  1.0000  0.0000]
 [0.7071  0.0000  0.7071]]

```

```

R23:
[[-0.0000  0.0000 -1.0000]
 [0.0000  1.0000  0.0000]
 [1.0000  0.0000  0.0000]]

```

```

R34:
[[0.6428  0.0000 -0.7660]
 [0.0000  1.0000  0.0000]
 [0.7660  0.0000  0.6428]]

```

```

R45:
[[0.9876  0.1564 -0.0001]
 [-0.1564  0.9877  0.0000]
 [0.0001 -0.0000  1.0000]]

```

```

R56:
[[-0.0872  0.0001 -0.9963]
 [-0.0000  1.0000  0.0001]
 [0.9962 -0.0000 -0.0871]]

```

Matrix manipulation to get angles:

```

In [18]: #take matrix Log of R, then turn so(3) matrix into [axis, angle]
J_vec_array = [mr.so3ToVec(mr.MatrixLog3(R.tolist())) for R in R_array]
J_ax_array = [mr.AxisAng3(vec)[0] for vec in J_vec_array]
J_ang_array = [mr.AxisAng3(vec)[1] for vec in J_vec_array]

```

```

for i, ax in enumerate(J_ax_array):

    #if calculated axis is opposite from reference axis, flip the axis + angle
    if (np.allclose(-ax, axes_ref[i], rtol = 0.001, atol = 0.001)):
        J_ax_array[i] = -ax
        J_ang_array[i] = -J_ang_array[i]

    #expected: T, T, T, T, F, T

```

Print results of axis/angle calculations:

In [19]:

```

for i, (ax, ang) in enumerate(zip(J_ax_array, J_ang_array)):

    if i == 0:
        print("Axis + angle J1:")
    else:
        print(f"Axis + angle J{i}{i+1}:")
    print(ax)
    print(ang, end = '\n\n')
    #print(f"{round(J*360/2/3.14159, 2)} degrees", end = "\n\n")

#coppeliastm takes in 6 angles in radians, one for each joint
print("Joint angle vector (C-x C-v into CoppeliaSim:)")
print(J_ang_array, end = '\n\n')

Axis + angle J1:
[-0.0000 -0.0000 1.0000]
-2.969482157066879

Axis + angle J12:
[-0.0000 1.0000 -0.0000]
-0.7853926894212007

Axis + angle J23:
[-0.0000 1.0000 -0.0000]
-1.5707661989213484

Axis + angle J34:
[-0.0000 1.0000 -0.0000]
-0.8726096667837093

Axis + angle J45:
[-0.0001 -0.0004 -1.0000]
0.15704051490320972

Axis + angle J56:
[0.0001 1.0000 0.0000]
-1.658121567631211

Joint angle vector (C-x C-v into CoppeliaSim:)
[-2.969482157066879, -0.7853926894212007, -1.5707661989213484, -0.8726096667837093,
0.15704051490320972, -1.658121567631211]

```

Check value of final rotation matrix:

```
In [20]: print(f"Rotation matrix Rsb:")  
print(Rsb, end = '\n\n')
```

Rotation matrix Rsb:

```
[[0.1676 -0.9308 0.3250]  
 [0.0434 -0.3224 -0.9456]  
 [0.9849 0.1726 -0.0136]]
```