ME449 Homework 1

Sean Morton, 10/14/22

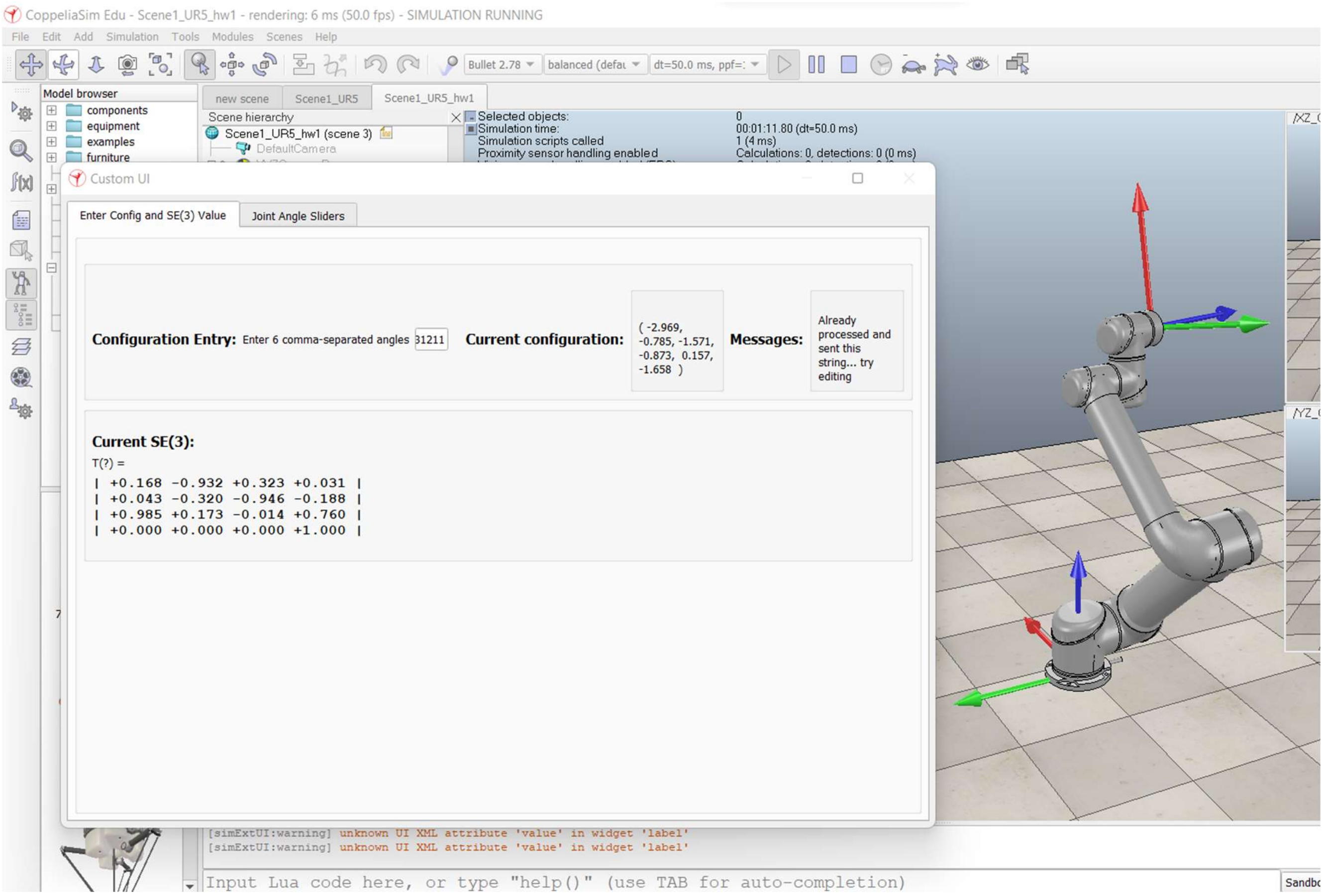
Part 1B

Child script "/UI_Script"

```
    sim.addStatusbarMessage('Window '..h..' is closing...')
    simUI.hide(h)
end

if (sim_call_type==sim.syscb_init) then
   -- joint limits:
   --     1: (-360, 360)
   --     2: (-360, 360)
   --     3: (-360, 360)
   --     4: (-360, 360)
   --     5: (-360, 360)
   --     6: (-360, 360)

   xml = [[
   <ui closeable="false" on-close="closeEventHandler" resizable="true">
     <tabs>

       <tab title="Enter Config and SE(3) Value">
         <group>
           <group layout="hbox">
             <label text="<big> Configuration Entry:</big>" wordwrap="false"
             <label text="Enter 6 comma-separated angles" />
             <edit value="" id="7006" oneditingfinished="fulljointEntry" />
             <label value="" id="5006" wordwrap="false" />
             <label text="<big> Current configuration:</big>" id="6007" wordw
             <group layout="hbox">
               <label value="" id="1237" wordwrap="true" />
             </group>
             <label text="<big> Messages:</big>" id="6006" wordwrap="false"
             <group layout="hbox">
               <label value="" id="1236" wordwrap="true" />
             </group>
           </group>
```

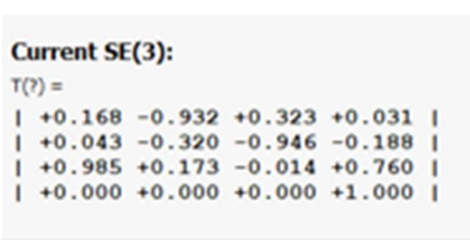
This is the UI script for Scene 1 in CoppeliaSim. I changed the group layouts of the UI so that all the vertical boxes “vbox” were changed to horizontal boxes “hbox”.

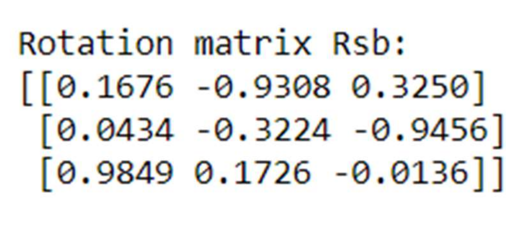
Parts 1B + 2



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 Rsb I calculated in Python, within 2 decimal places. Below: comparison between CoppeliaSim and Python output





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 Rsb through 2 methods to verify matrix calculations worked

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

In [1]:
```
import core as mr
import numpy as np
```

Define axes of rotation for each joint:

In [13]:
```
axes_ref = [
    [0, 0, 1],
    [0, 1, 0],
    [0, 1, 0],
    [0, 1, 0],
    [0, 0, -1],
    [0, 1, 0],
]
```

Matrix calculations:

In [4]:
```
#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.
R15 = np.matrix([[-0.9839, -0.1558, 0.0872], [-0.1564, 0.9877, 0], [-0.0861, -0.0136,
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
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 [9]:
```
#source for formatting: https://tinyurl.com/2m6w7r8n
np.set_printoptions(formatter={'float_kind':'{:.4f}'.format})

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]:
```
R_array = [Rs1, R12, R23, R34, R45, R56] #b is fixed relative to 6, so this isn't a jc

#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 [20]:

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

    if i == 0:
        print("Axis + angle Js1:")
    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")

#coppeliasim 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 Js1:
[-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 conformity of results for final rotation matrix:

In [21]:
```python
print(f"Rotation matrix Rsb:")
print(Rsb, end = '\n\n')

R_test = Rs1 * R12 * R23 * R34 * R45 * R56 * R6b
print("Test of calculated Rsb:")
print(R_test, end = '\n\n')
```

```
Rotation matrix Rsb:
[[0.1676 -0.9308 0.3250]
 [0.0434 -0.3224 -0.9456]
 [0.9849 0.1726 -0.0136]]

Test of calculated Rsb:
[[0.1676 -0.9307 0.3249]
 [0.0434 -0.3224 -0.9456]
 [0.9848 0.1726 -0.0136]]
```