

Part 1

Import Libraries

```
In [2]: import numpy as np
import modern_robotics as mr
import matplotlib.pyplot as plt
import random

# Set print options for numpy arrays
np.set_printoptions(precision=3, suppress=True)
```

Util Functions

```
In [11]: def pi2pi(angle):
    while angle > np.pi:
        angle -= 2*np.pi
    while angle < -np.pi:
        angle += 2*np.pi
    return angle

def print_iteration(M, B, V_b, index, joint_vector):
    joint_vector = np.array(joint_vector)
    joint_vector = joint_vector.tolist()

    print(f"Iteration {index}:\n")

    print("joint vector:")
    joint_vector_to_print = joint_vector
    for i in range(len(joint_vector_to_print)):
        joint_vector_to_print[i] = round(joint_vector_to_print[i], 3)
    print(f"{str(joint_vector_to_print)[1:-1]}\n")

    print("SE(3) end-effector config:")
    Tsb = mr.FKinBody(M, B, joint_vector)
    decimal_places = 3
    width = 6
    print('\n'.join(' '.join(f"{{cell:{width}.{{decimal_places}}f}}" for cell in
    print()

    decimal_places = 3
    Vb_to_print = V_b.tolist()
    for i in range(len(Vb_to_print)):
        Vb_to_print[i] = round(Vb_to_print[i], decimal_places)
    Vb_to_print = tuple(Vb_to_print)
    print(f"          error twist V_b: {Vb_to_print}")
    print(f"angular error ||omega_b||: {np.linalg.norm([V_b[0], V_b[1], V_b[2]])}")
    print(f"linear error |v_b|: {np.linalg.norm([V_b[3], V_b[4], V_b[5]])}")
    print("=====
```

```
def save_csv(filename, data):
```

```

np.savetxt(filename, data, delimiter=', ')

def Vb(M, B, T_sd, joint_vector):
    Vb = mr.se3ToVec(mr.MatrixLog6(np.dot(mr.TransInv(mr.FKinBody(M, B, joint_vector)), joint_vector)))
    return Vb

def check_convergence(V_b, eps_w, eps_v):
    return (np.linalg.norm([V_b[0], V_b[1], V_b[2]]) > eps_w \
            or np.linalg.norm([V_b[3], V_b[4], V_b[5]]) > eps_v)

def IKinBodyIterates(M, B, T_sd, theta_0, traj_filename, max_iterations=100, printing=False):
    index = 1
    joint_vector = np.array(theta_0)
    joint_vector = joint_vector.tolist()
    V_b = Vb(M, B, T_sd, theta_0)

    # For saving later
    traj = np.array(joint_vector)
    angular_error = np.array(np.linalg.norm([V_b[0], V_b[1], V_b[2]]))
    linear_error = np.array(np.linalg.norm([V_b[3], V_b[4], V_b[5]]))

    iterating = check_convergence(V_b, eps_w, eps_v)

    while iterating and index < max_iterations:
        joint_vector = joint_vector + np.dot(np.linalg.pinv(mr.JacobianBody(M, B, joint_vector)), [np.arctan2(np.sin(theta), np.cos(theta)) for theta in theta_0])
        V_b = Vb(M, B, T_sd, joint_vector)
        if(printing):
            print_iteration(M, B, V_b, index, joint_vector)
        traj = np.vstack([traj, joint_vector])
        angular_error = np.vstack([angular_error, np.array(np.linalg.norm([V_b[0], V_b[1], V_b[2]]))])
        linear_error = np.vstack([linear_error, np.array(np.linalg.norm([V_b[3], V_b[4], V_b[5]]))])
        iterating = check_convergence(V_b, eps_w, eps_v)

        if(iterating == False):
            if(printing):
                print(f"Converged. It took {index} iterations.")
                print("Trajectory:")
                print(traj)
                save_csv(traj_filename+'.csv', traj)
                save_csv(traj_filename+'_angular_error.csv', angular_error)
                save_csv(traj_filename+'_linear_error.csv', linear_error)
            break

        index += 1

    return index

def joint_traj2position_traj(joint_traj):
    joint_vector = joint_traj[0]
    pose_matrix = mr.FKinBody(M, B, joint_vector)
    position = pose_matrix[:3, 3]
    position_traj = np.array(position)
    for idx in range(1, joint_traj.shape[0]):
        joint_vector = joint_traj[idx]
        pose_matrix = mr.FKinBody(M, B, joint_vector)
        position = pose_matrix[:3, 3]
        position_traj = np.vstack([position_traj, position])
    return position_traj

```

```

        position = pose_matrix[:3, 3]
        position_traj = np.vstack([position_traj, position])
    return position_traj

def plot_position_traj(long_filename, short_filename):
    long_angle_traj = np.genfromtxt(long_filename+'.csv', delimiter=',')
    short_angle_traj = np.genfromtxt(short_filename+'.csv', delimiter=',')
    long_position_traj = joint_traj2position_traj(long_angle_traj)
    short_position_traj = joint_traj2position_traj(short_angle_traj)

    # Plot two trajectories in 3D space
    fig = plt.figure()
    ax = fig.add_subplot(111, projection='3d')
    ax.plot(long_position_traj[0,0], long_position_traj[0,1], long_position_traj[0,2])
    ax.plot(long_position_traj[:, 0], long_position_traj[:, 1], long_position_traj[:, 2])
    ax.plot(long_position_traj[-1,0], long_position_traj[-1,1], long_position_traj[-1,2])

    ax.plot(short_position_traj[0,0], short_position_traj[0,1], short_position_traj[0,2])
    ax.plot(short_position_traj[:, 0], short_position_traj[:, 1], short_position_traj[:, 2])
    ax.plot(short_position_traj[-1,0], short_position_traj[-1,1], short_position_traj[-1,2])

    ax.set_xlabel('X')
    ax.set_ylabel('Y')
    ax.set_zlabel('Z')
    ax.legend()
    plt.show()

def plot_angular_error(long_filename, short_filename):
    long_angular_error = np.genfromtxt(long_filename+'_angular_error.csv', delimiter=',')
    short_angular_error = np.genfromtxt(short_filename+'_angular_error.csv', delimiter=',')
    plt.plot(long_angular_error, label='long_traj')
    plt.plot(short_angular_error, label='short_traj')
    plt.xlabel('Iterations')
    plt.ylabel('Angular Error')
    plt.legend()
    plt.show()

def plot_linear_error(long_filename, short_filename):
    long_linear_error = np.genfromtxt(long_filename+'_linear_error.csv', delimiter=',')
    short_linear_error = np.genfromtxt(short_filename+'_linear_error.csv', delimiter=',')
    plt.plot(long_linear_error, label='long_traj')
    plt.plot(short_linear_error, label='short_traj')
    plt.xlabel('Iterations')
    plt.ylabel('Linear Error')
    plt.legend()
    plt.show()

def filter_initial_guess(M, B, T_sd, lb, ub):
    # give a random initial guess
    ites = 0
    while (ites>=ub or ites<=lb):
        random_initial_guess = [random.uniform(-np.pi, np.pi) for i in range(3)]
        # round to 3 decimal places
        random_initial_guess = [round(i, 3) for i in random_initial_guess]
        random_traj_name = "random_traj"
        ites = IKinBodyIterates(M, B, T_sd, random_initial_guess, random_traj_name)

```

```
print(f"Found initial guess: {random_initial_guess} with {ites} iterations")
return random_initial_guess
```

Constants

```
In [4]: L1 = 0.425
L2 = 0.392
H1 = 0.089
H2 = 0.095
W1 = 0.109
W2 = 0.082

# Home Configuration:
M = np.array([[[-1, 0, 0, L1+L2],
              [0, 0, 1, W1+W2],
              [0, 1, 0, H1-H2],
              [0, 0, 0, 1]]])

# B:
B_list = []
B_list.append(np.array([0, 1, 0, W1+W2, 0, L1+L2]))
B_list.append(np.array([0, 0, 1, H2, -L1-L2, 0]))
B_list.append(np.array([0, 0, 1, H2, -L2, 0]))
B_list.append(np.array([0, 0, 1, H2, 0, 0]))
B_list.append(np.array([0, -1, 0, -W2, 0, 0]))
B_list.append(np.array([0, 0, 1, 0, 0, 0]))
B = np.column_stack(B_list)

# T_sd:
T_sd = np.array([[1, 0, 0, 0.3],
                 [0, 1, 0, 0.3],
                 [0, 0, 1, 0.4],
                 [0, 0, 0, 1]])
```

Find Appropriate Initial Theta Lists

```
In [5]: # converges after 2–4 Newton-Raphson steps (short iterates), not more or less
short_traj_initial_guess = filter_initial_guess(M, B, T_sd, 2, 4)

# converges after 10 Newton-Raphson steps, or never converges (long iterates)
long_traj_initial_guess = filter_initial_guess(M, B, T_sd, 11, np.inf)
```

Found initial guess: [-1.554, 2.462, 1.464, 0.288, 1.801, 0.389] with 3 iterations to converge.

Found initial guess: [0.834, -0.491, 3.134, -1.729, -0.532, 2.921] with 22 iterations to converge.

Solve IK With the Initial Guess Resulting in **Long** Trajectory

```
In [16]: long_traj_name = "long iterates"
IKinBodyIterates(M, B, T_sd, long_traj_initial_guess, long_traj_name)
```

Iteration 1:

```
joint vector:  
-2.47, -2.957, -3.093, 2.928, -0.025, 1.527  
  
SE(3) end-effector config:  
-0.020 0.798 0.602 0.144  
-0.014 0.602 -0.798 -0.129  
-1.000 -0.024 -0.000 0.171  
0.000 0.000 0.000 1.000  
  
error twist V_b: (-0.705, -1.459, 0.739, -0.178, 0.52, 0.09)  
angular error ||omega_b||: 1.781107191540879  
linear error ||v_b||: 0.5573832230715222
```

=====

Iteration 2:

```
joint vector:  
0.863, 1.843, 2.831, 1.255, -2.59, -0.519  
  
SE(3) end-effector config:  
0.909 0.259 0.328 -0.119  
0.362 -0.097 -0.927 -0.078  
-0.209 0.961 -0.182 -0.033  
0.000 0.000 0.000 1.000  
  
error twist V_b: (-1.688, -0.479, -0.092, 0.357, 0.661, 0.104)  
angular error ||omega_b||: 1.7572528705011987  
linear error ||v_b||: 0.757821864184124
```

=====

Iteration 3:

```
joint vector:  
-1.232, -0.326, 0.048, -0.741, 0.916, -1.345  
  
SE(3) end-effector config:  
0.420 0.562 0.713 0.447  
-0.656 0.730 -0.190 -0.791  
-0.627 -0.388 0.675 0.338  
0.000 0.000 0.000 1.000  
  
error twist V_b: (0.125, -0.842, 0.766, -0.585, 0.972, 0.003)  
angular error ||omega_b||: 1.1454603223849076  
linear error ||v_b||: 1.1343751666568116
```

=====

Iteration 4:

```
joint vector:  
-0.453, -3.005, 2.133, -0.638, 1.634, -0.806  
  
SE(3) end-effector config:  
0.952 -0.304 0.027 -0.017  
0.305 0.949 -0.083 0.124
```

```
-0.000  0.088  0.996  0.523
 0.000  0.000  0.000  1.000

      error twist V_b: (-0.087, -0.014, -0.309, 0.342, 0.121, -0.127)
angular error ||omega_b||: 0.3216527672596053
    linear error ||v_b||: 0.3842815880566039
```

```
=====
Iteration 5:
```

```
joint vector:
1.408, 2.501, 1.999, 0.483, 1.31, -2.981
```

```
SE(3) end-effector config:
 0.977 -0.000 -0.213 -0.179
 0.065  0.953  0.297 -0.285
 0.203 -0.304  0.931  0.269
 0.000  0.000  0.000  1.000
```

```
      error twist V_b: (0.307, 0.212, -0.034, 0.504, 0.555, 0.168)
angular error ||omega_b||: 0.3748876422480335
    linear error ||v_b||: 0.7682236713147313
```

```
=====
Iteration 6:
```

```
joint vector:
2.405, 2.254, -2.1, -1.477, 1.835, 2.347
```

```
SE(3) end-effector config:
 1.000 -0.007  0.000 -0.230
 0.006  0.936  0.352  0.090
 -0.002 -0.352  0.936 -0.247
 0.000  0.000  0.000  1.000
```

```
      error twist V_b: (0.36, -0.002, -0.006, 0.53, 0.089, 0.679)
angular error ||omega_b||: 0.36050466495917344
    linear error ||v_b||: 0.8653642517998467
```

```
=====
Iteration 7:
```

```
joint vector:
-0.519, -2.52, 2.071, -0.367, 0.86, -1.074
```

```
SE(3) end-effector config:
 0.550 -0.312  0.774  0.184
 0.102  0.946  0.309  0.082
 -0.829 -0.091  0.552  0.487
 0.000  0.000  0.000  1.000
```

```
      error twist V_b: (0.239, -0.959, -0.248, 0.171, 0.208, 0.006)
angular error ||omega_b||: 1.0194319799551061
    linear error ||v_b||: 0.269193625412744
```

Iteration 8:

joint vector:
2.684, -1.831, 2.487, -0.428, -2.56, 2.513

SE(3) end-effector config:
0.275 0.451 0.849 -0.140
-0.631 -0.582 0.513 0.024
0.726 -0.677 0.124 0.178
0.000 0.000 0.000 1.000

error twist V_b: (1.626, -0.169, 1.478, 0.17, 0.284, 0.519)
angular error ||omega_b||: 2.203660220481679
linear error ||v_b||: 0.615730495534247

=====

Iteration 9:

joint vector:
1.472, 3.132, -2.488, 2.215, 1.449, 0.056

SE(3) end-effector config:
-0.973 0.082 -0.215 -0.140
0.229 0.265 -0.937 -0.203
-0.020 -0.961 -0.277 -0.082
0.000 0.000 0.000 1.000

error twist V_b: (0.297, 2.399, -1.809, 1.088, -0.311, -0.492)
angular error ||omega_b||: 3.0188432600470803
linear error ||v_b||: 1.2335653472027712

=====

Iteration 10:

joint vector:
1.674, -1.674, 3.094, 1.809, -1.406, 2.8

SE(3) end-effector config:
-0.906 -0.332 -0.264 -0.133
-0.278 -0.007 0.961 0.091
-0.320 0.943 -0.086 0.212
0.000 0.000 0.000 1.000

error twist V_b: (0.671, -2.185, -2.096, -0.011, -0.386, 0.667)
angular error ||omega_b||: 3.1016566843180366
linear error ||v_b||: 0.7710985577089325

=====

Iteration 11:

joint vector:
2.127, -1.86, 2.396, 2.494, 2.166, 1.314

SE(3) end-effector config:
-0.161 0.381 0.911 -0.126
-0.140 0.905 -0.403 0.084

```

-0.977 -0.192 -0.092  0.383
 0.000  0.000  0.000  1.000

      error twist V_b: (-0.187, -1.674, 0.461, 0.255, 0.322, 0.331)
angular error ||omega_b||: 1.7460721283841358
    linear error ||v_b||: 0.527797904970423
=====
```

Iteration 12:

```

joint vector:
0.426, -2.375, -2.1, 2.99, -1.821, 2.609
```

```

SE(3) end-effector config:
-0.822  0.569  0.027 -0.319
 0.543  0.798 -0.260 -0.048
-0.169 -0.199 -0.965 -0.084
 0.000  0.000  0.000  1.000
```

```

      error twist V_b: (-0.894, -2.88, 0.369, -0.59, 0.754, 0.724)
angular error ||omega_b||: 3.0383396277112946
    linear error ||v_b||: 1.2004161739935544
=====
```

Iteration 13:

```

joint vector:
0.888, 1.779, 2.736, -1.903, 2.474, 2.883
```

```

SE(3) end-effector config:
 0.959 -0.076  0.272 -0.196
 0.231 -0.344 -0.910 -0.171
 0.163  0.936 -0.313  0.114
 0.000  0.000  0.000  1.000
```

```

      error twist V_b: (-1.898, -0.113, -0.315, 0.574, 0.519, -0.197)
angular error ||omega_b||: 1.9269984582887372
    linear error ||v_b||: 0.7983816415495258
=====
```

Iteration 14:

```

joint vector:
2.074, -2.967, -2.724, -0.496, 1.826, -0.387
```

```

SE(3) end-effector config:
-0.880 -0.409 -0.243 -0.066
-0.260 -0.015  0.965 -0.063
-0.398  0.913 -0.093 -0.158
 0.000  0.000  0.000  1.000
```

```

      error twist V_b: (0.726, -2.124, -2.037, -0.311, -0.197, 0.901)
angular error ||omega_b||: 3.0311864415604486
    linear error ||v_b||: 0.973389846034667
=====
```

Iteration 15:

```
joint vector:  
-0.306, -3.139, 2.181, 1.018, 1.668, -2.07  
  
SE(3) end-effector config:  
-0.238 0.317 0.918 -0.088  
-0.425 0.816 -0.392 0.134  
-0.874 -0.483 -0.060 0.311  
0.000 0.000 0.000 1.000  
  
error twist V_b: (0.085, -1.675, 0.693, 0.141, 0.276, 0.384)  
angular error ||omega_b||: 1.8144148324657048  
linear error ||v_b||: 0.49302623573459875
```

=====

Iteration 16:

```
joint vector:  
-0.681, 2.522, -2.904, -1.079, 0.567, -1.548  
  
SE(3) end-effector config:  
0.778 0.249 0.577 0.203  
-0.615 0.490 0.618 0.065  
-0.129 -0.836 0.534 0.022  
0.000 0.000 0.000 1.000  
  
error twist V_b: (0.919, -0.447, 0.545, -0.048, 0.054, 0.474)  
angular error ||omega_b||: 1.1578766648413341  
linear error ||v_b||: 0.4792205903660268
```

=====

Iteration 17:

```
joint vector:  
-2.785, 1.457, 1.848, 0.963, 1.958, -1.783  
  
SE(3) end-effector config:  
-0.927 0.287 0.241 0.295  
-0.137 -0.859 0.493 0.026  
0.348 0.424 0.836 -0.160  
0.000 0.000 0.000 1.000  
  
error twist V_b: (0.454, 0.707, 2.795, -0.105, 0.047, 0.635)  
angular error ||omega_b||: 2.9185858191554925  
linear error ||v_b||: 0.6454991586603375
```

=====

Iteration 18:

```
joint vector:  
-2.265, 2.548, 2.313, -0.141, 2.327, 0.748  
  
SE(3) end-effector config:  
0.843 0.091 -0.531 0.168  
0.179 0.883 0.435 0.119
```

```
0.508 -0.461  0.727  0.298
0.000  0.000  0.000  1.000
```

```
error twist V_b: (0.494, 0.573, -0.048, 0.166, 0.152, 0.103)
angular error ||omega_b||: 0.7580659643891672
linear error ||v_b||: 0.24780753933581995
```

Iteration 19:

joint vector:
-1.751, 2.461, 1.918, -0.183, 1.575, -0.033

SE(3) end-effector config:
0.978 0.188 0.084 0.182
-0.205 0.849 0.486 0.390
0.020 -0.493 0.870 0.310
0.000 0.000 0.000 1.000

```
error twist V_b: (0.515, -0.034, 0.207, 0.127, -0.099, 0.068)
angular error ||omega_b||: 0.5564965404269925
linear error ||v_b||: 0.17435573296645318
```

Iteration 20:

joint vector:
-2.09, 2.807, 1.709, 0.196, 1.793, 0.471

SE(3) end-effector config:
0.980 0.058 -0.191 0.269
-0.037 0.993 0.110 0.287
0.196 -0.100 0.975 0.414
0.000 0.000 0.000 1.000

```
error twist V_b: (0.106, 0.195, 0.047, 0.029, 0.014, -0.016)
angular error ||omega_b||: 0.22714814415109005
linear error ||v_b||: 0.03646046948008749
```

Iteration 21:

joint vector:
-2.103, 2.786, 1.65, 0.285, 1.571, 0.534

SE(3) end-effector config:
1.000 -0.002 -0.005 0.302
0.002 1.000 -0.007 0.298
0.005 0.007 1.000 0.399
0.000 0.000 0.000 1.000

```
error twist V_b: (-0.007, 0.004, -0.001, -0.002, 0.002, 0.001)
angular error ||omega_b||: 0.008523650641431324
linear error ||v_b||: 0.003055527820815518
```

```
Iteration 22:
```

```
joint vector:
```

```
-2.096, 2.785, 1.651, 0.276, 1.571, 0.526
```

```
SE(3) end-effector config:
```

```
1.000 -0.001 0.000 0.300
0.001 1.000 0.000 0.300
-0.000 -0.000 1.000 0.400
0.000 0.000 0.000 1.000
```

```
error twist V_b: (0.0, 0.0, -0.0, 0.0, 0.0, -0.0)
```

```
angular error ||omega_b||: 6.221323477282383e-05
```

```
linear error ||v_b||: 1.2429527304356725e-05
```

```
=====
Converged. It took 22 iterations.
```

```
Trajectory:
```

```
[[ 0.834 -0.491 3.134 -1.729 -0.532 2.921]
 [-2.47 -2.957 -3.093 2.928 -0.025 1.527]
 [ 0.863 1.843 2.831 1.255 -2.59 -0.519]
 [-1.232 -0.326 0.048 -0.741 0.916 -1.345]
 [-0.453 -3.005 2.133 -0.638 1.634 -0.806]
 [ 1.408 2.501 1.999 0.483 1.31 -2.981]
 [ 2.405 2.254 -2.1 -1.477 1.835 2.347]
 [-0.519 -2.52 2.071 -0.367 0.86 -1.074]
 [ 2.684 -1.831 2.487 -0.428 -2.56 2.513]
 [ 1.472 3.132 -2.488 2.215 1.449 0.056]
 [ 1.674 -1.674 3.094 1.809 -1.406 2.8 ]
 [ 2.127 -1.86 2.396 2.494 2.166 1.314]
 [ 0.426 -2.375 -2.1 2.99 -1.821 2.609]
 [ 0.888 1.779 2.736 -1.903 2.474 2.883]
 [ 2.074 -2.967 -2.724 -0.496 1.826 -0.387]
 [-0.306 -3.139 2.181 1.018 1.668 -2.07 ]
 [-0.681 2.522 -2.904 -1.079 0.567 -1.548]
 [-2.785 1.457 1.848 0.963 1.958 -1.783]
 [-2.265 2.548 2.313 -0.141 2.327 0.748]
 [-1.751 2.461 1.918 -0.183 1.575 -0.033]
 [-2.09 2.807 1.709 0.196 1.793 0.471]
 [-2.103 2.786 1.65 0.285 1.571 0.534]
 [-2.096 2.785 1.651 0.276 1.571 0.526]]
```

```
Out[16]: 22
```

Solve IK With the Initial Guess Resulting in **Short** Trajectory

```
In [17]: short_traj_name = "short_iterates"
IKinBodyIterates(M, B, T_sd, short_traj_initial_guess, short_traj_name)
```

```
Iteration 1:
```

```
joint vector:  
-1.991, 2.845, 1.655, 0.261, 1.686, 0.434
```

```
SE(3) end-effector config:  
0.992 -0.011 -0.125 0.250  
0.011 1.000 0.003 0.316  
0.125 -0.004 0.992 0.425  
0.000 0.000 0.000 1.000
```

```
error twist V_b: (0.004, 0.125, -0.011, 0.048, -0.016, -0.028)  
angular error ||omega_b||: 0.12534507476104986  
linear error ||v_b||: 0.057951035716249905
```

```
Iteration 2:
```

```
joint vector:  
-2.102, 2.792, 1.651, 0.281, 1.566, 0.529
```

```
SE(3) end-effector config:  
1.000 0.002 -0.002 0.301  
-0.002 1.000 -0.012 0.296  
0.002 0.012 1.000 0.402  
0.000 0.000 0.000 1.000
```

```
error twist V_b: (-0.013, 0.002, 0.002, -0.001, 0.004, -0.002)  
angular error ||omega_b||: 0.013039491663839251  
linear error ||v_b||: 0.004693607444187299
```

```
Iteration 3:
```

```
joint vector:  
-2.096, 2.785, 1.651, 0.276, 1.571, 0.526
```

```
SE(3) end-effector config:  
1.000 -0.001 0.000 0.300  
0.001 1.000 0.000 0.300  
-0.000 -0.000 1.000 0.400  
0.000 0.000 0.000 1.000
```

```
error twist V_b: (0.0, 0.0, -0.0, 0.0, -0.0, -0.0)  
angular error ||omega_b||: 6.943420225727724e-05  
linear error ||v_b||: 2.316914360262802e-05
```

```
Converged. It took 3 iterations.
```

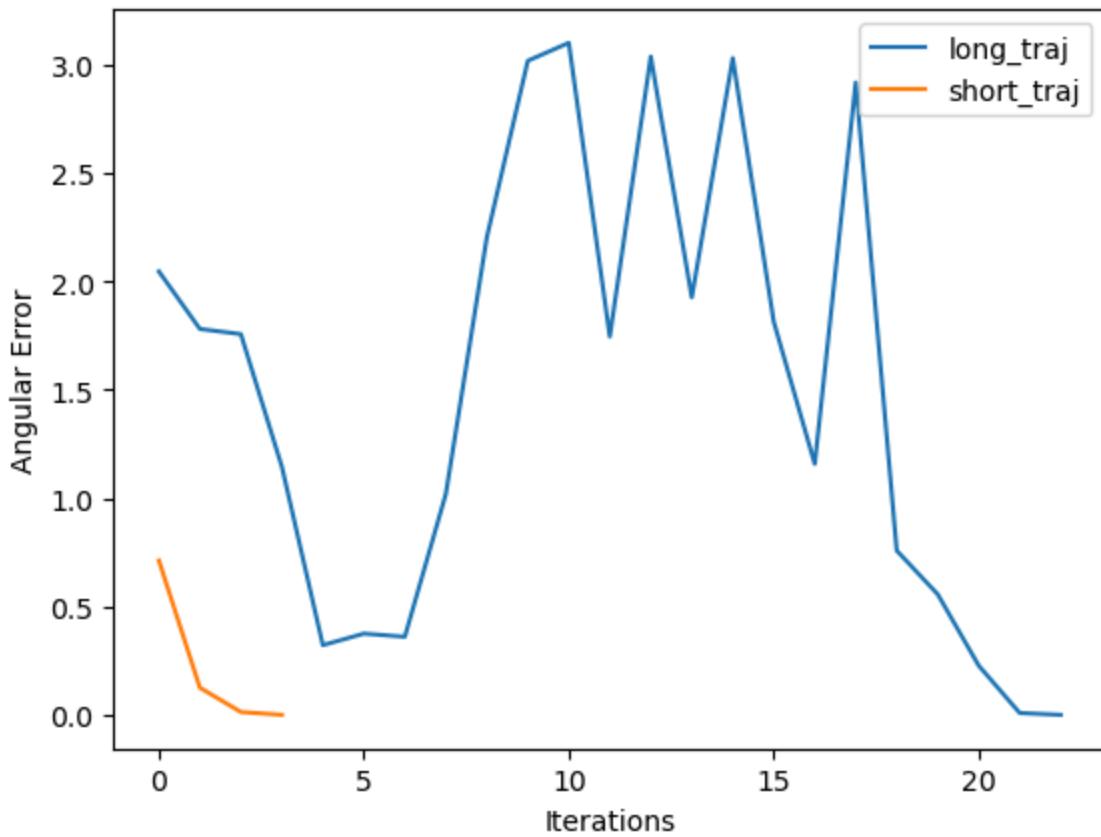
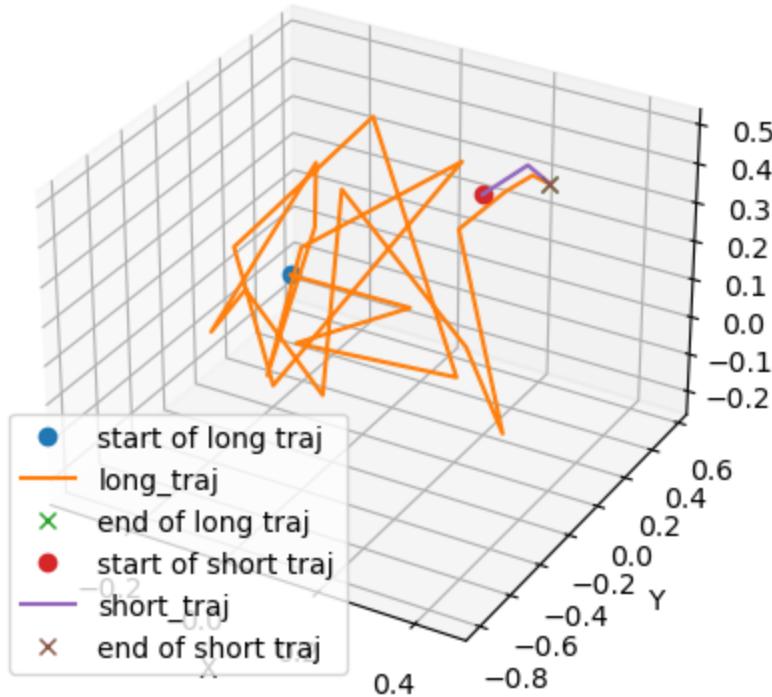
```
Trajectory:
```

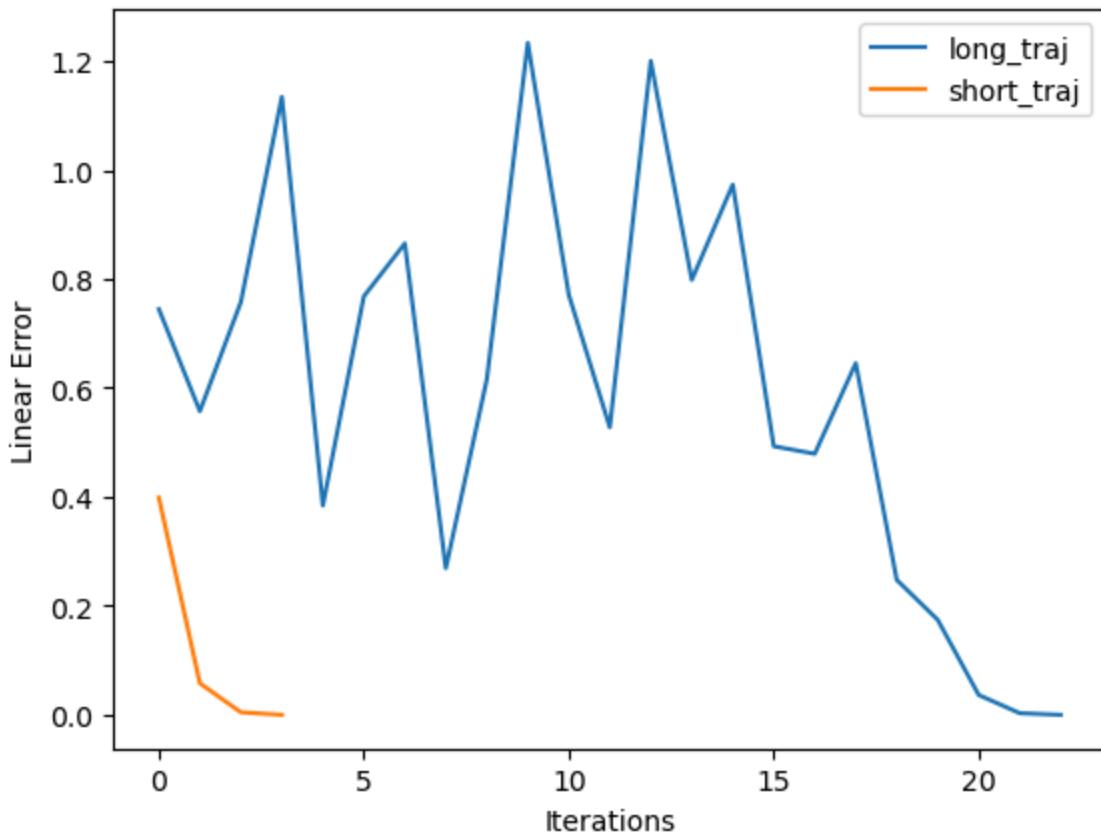
```
[[ -1.554 2.462 1.464 0.288 1.801 0.389]  
[-1.991 2.845 1.655 0.261 1.686 0.434]  
[-2.102 2.792 1.651 0.281 1.566 0.529]  
[-2.096 2.785 1.651 0.276 1.571 0.526]]
```

```
Out[17]: 3
```

Plot the results

```
In [18]: plot_position_traj(long_traj_name, short_traj_name)
plot_angular_error(long_traj_name, short_traj_name)
plot_linear_error(long_traj_name, short_traj_name)
```





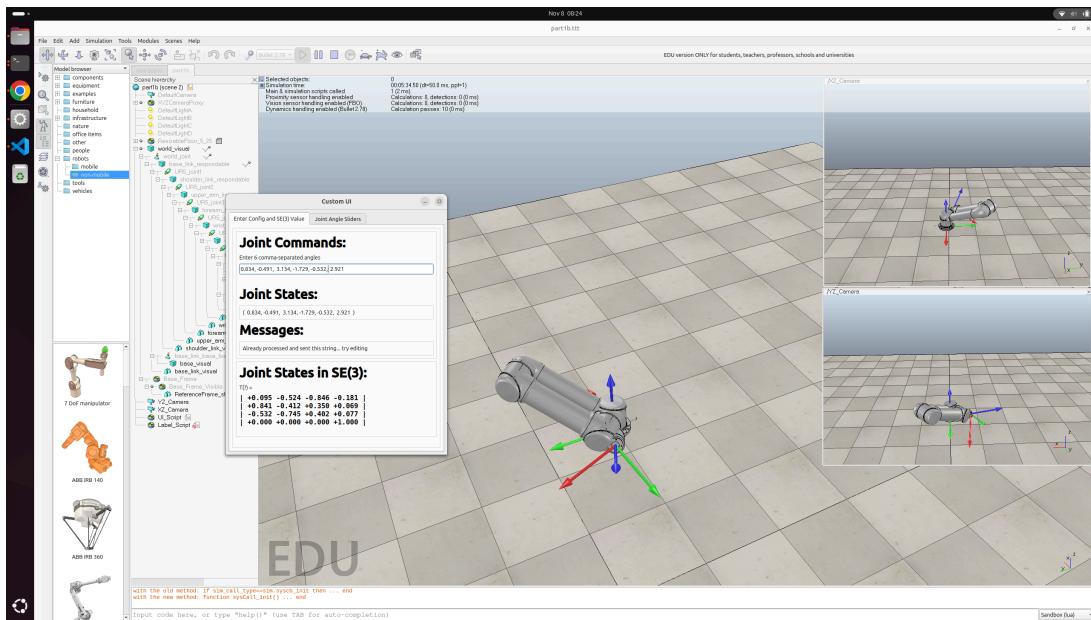
Explanation

When the initial guess is far from the actual solution, the algorithm might take long steps in the wrong direction, struggle to find the correct gradient, or even diverge.

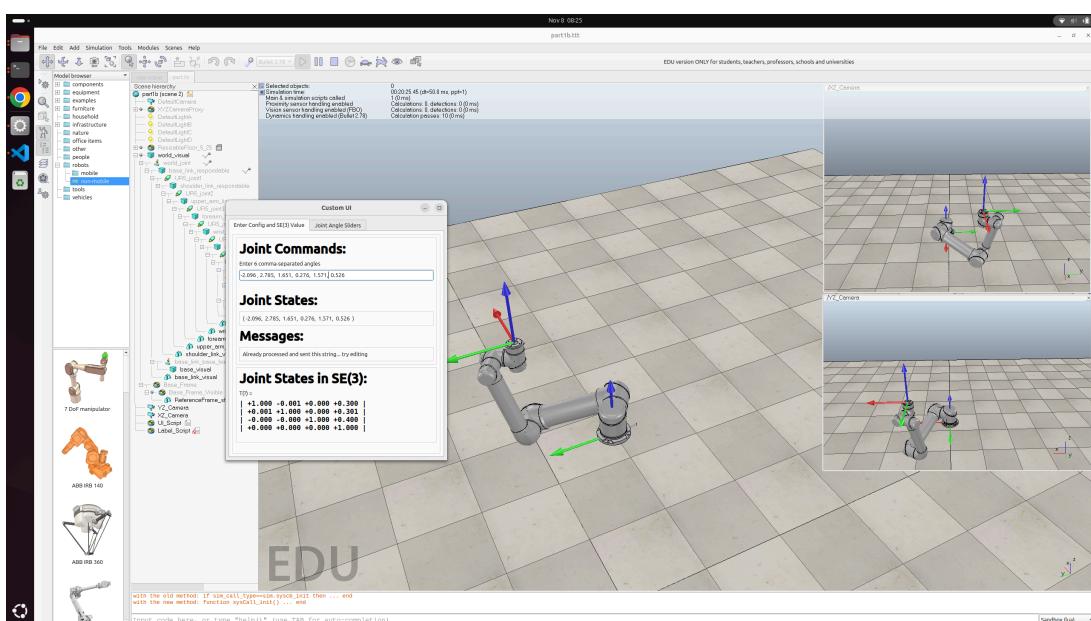
Visualization

Long Trajectory IK in Simulation Visualization

- Initial Configuration

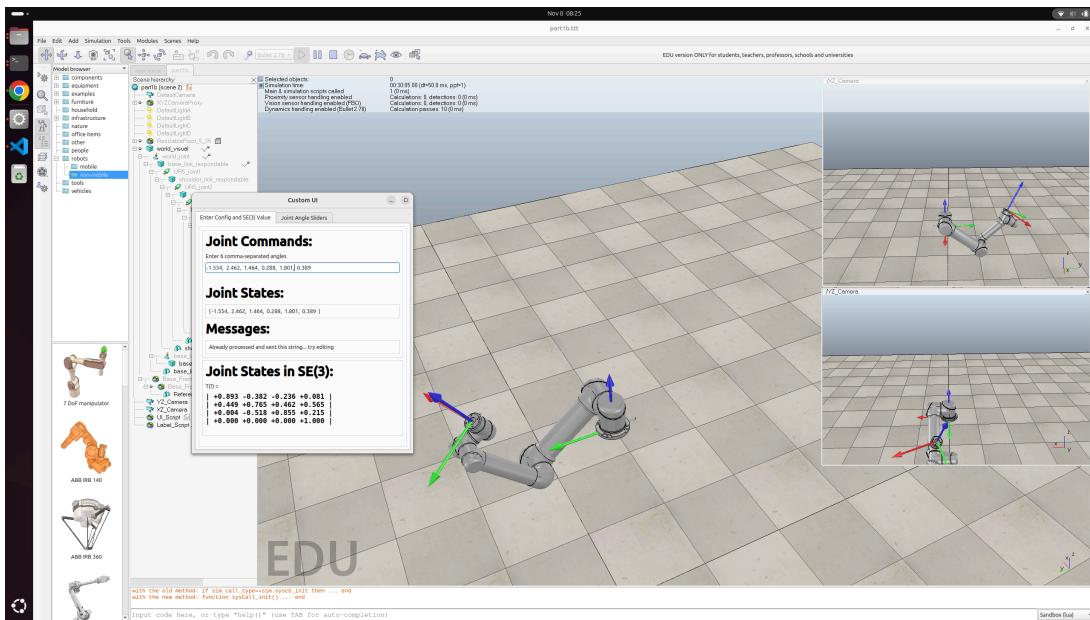


- End

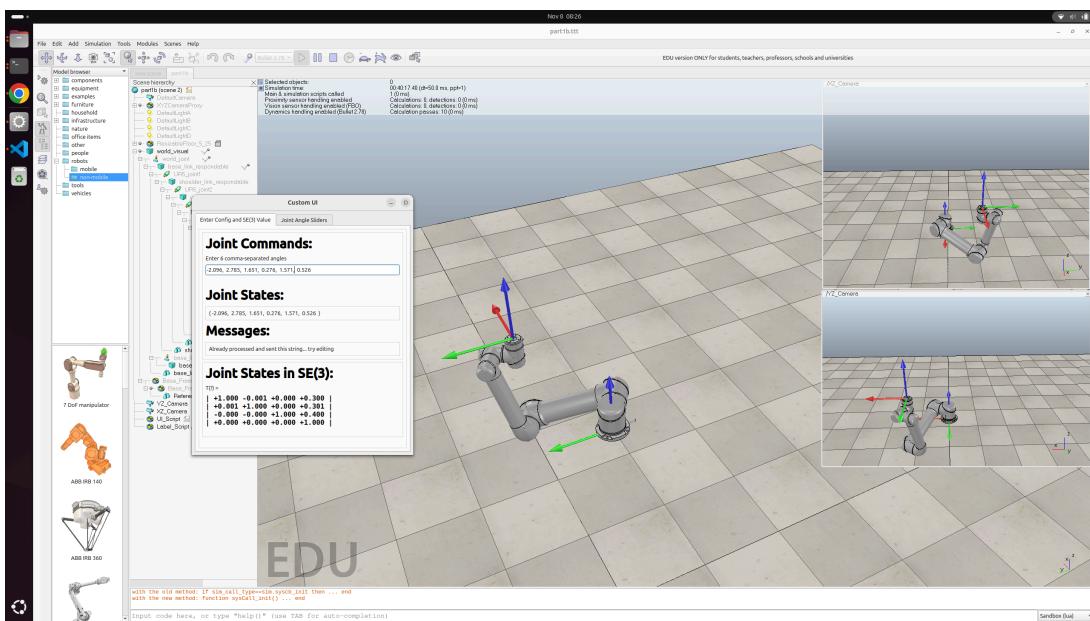


Short Trajectory IK in Simulation Visualization

- Initial Configuration



- End



Part 2

Imagine the robot is powered up at a random configuration θ^0 such that $T_{sb}(\theta^0) = T^0$, and its first task is to move to a θ^* satisfying $T_{sb}(\theta^*) = T_{sd}$. Part 1 of this assignment deals with calculating an appropriate θ^* . In this part, you will consider how to actually control the robot to move from rest at θ^0 to rest at $\dot{\theta}^*$ in t_f seconds.

- (a) If you decide to move at constant joint speeds, what joint speed vector $\dot{\theta}$ do you command to the joints? Your answer should be symbolic in terms of relevant variables

Answer:

$$\Delta\theta = \theta^* - \theta^0 = \dot{\theta} \cdot t_f$$

$$\text{Thus, } \dot{\theta} = \frac{\theta^* - \theta^0}{t_f}$$

(b) If you decide to make the last link of the robot follow a single constant twist from T^0 to R_{sd} , what joint speed vector $\dot{\theta}(0)$ do you command to the joints at time $t = 0$? What is the commanded joint speed vector $\dot{\theta}(t_f/2)$, halfway through the motion? Your answers should be symbolic in terms of relevant variables. Use the function $\text{vec}([\mathcal{V}])$ to convert $[\mathcal{V}] \in se(3)$ to the twist $\mathcal{V} \in \mathbb{R}^6$, and assume you have access to the body Jacobian $J_b(\theta)$.

Answer:

- Determine the Twist \mathcal{V} to move from T^0 to T_{sd}

$$\mathcal{V} = \frac{\text{vec}(\log((T^0)^{-1}T_{sd}))}{t_f}$$

- Determine $\dot{\theta}(0)$

$$\begin{aligned}\mathcal{V} &= J_b(\theta^0)\dot{\theta}(0) \\ \dot{\theta}(0) &= J_b(\theta^0)^{-1}\mathcal{V}\end{aligned}$$

$$\text{Thus, } \dot{\theta}(0) = \frac{J_b(\theta^0)^{-1}\text{vec}(\log((T^0)^{-1}T_{sd}))}{t_f}$$

- Determine $\dot{\theta}(t_f/2)$

$$\dot{\theta}(t_f/2) = J_b(\theta^{t_f/2})^{-1}\mathcal{V}$$

$$\text{Thus, } \dot{\theta}(t_f/2) = \frac{J_b(\theta^{t_f/2})^{-1}\text{vec}(\log((T^0)^{-1}T_{sd}))}{t_f}$$

(c) Do you see any advantages to using the approach in (a) relative to (b), or vice-versa?

Answer:

- (a) is more simple in control strategy and has lower computational cost.
- (a) lacks direct cartesian path control for end effector, which can be dangerous in real world. (b) directly uses the information in cartesian space.