ME 572: Fall 2022 Computer Project

TJ Wiegman

2022-11-21

In [1]:
```
# Read text file input
with open("RRR.txt") as file:
    RR = file.readlines()

for i in range(len(RR)):
    RR[i] = RR[i].rstrip() # rstrip removes newline characters

    # For documentation purposes: print contents of input file here
    print(RR[i])
```

```
.24,.015,2,3,RRR
4,1.5,13
4.2,-0.5,0
```

In [2]:
```
# Parse the input
## First line
timeTotal, timeStep, Sacc, Sdec, Title = RR[0].split(",")
timeTotal, timeStep = float(timeTotal), float(timeStep)
Sacc, Sdec = int(Sacc), int(Sdec)

## Second line
P0 = []
for coord in RR[1].split(","): P0.append(float(coord))

## Third line
PF = []
for coord in RR[2].split(","): PF.append(float(coord))

## For documentation purposes: print results
print(
    f"{Title}: Moving from {P0} to {PF} "
    f"over {timeTotal} seconds in {timeStep} second steps"
)
```

```
RRR: Moving from [4.0, 1.5, 13.0] to [4.2, -0.5, 0.0] over 0.24 seconds in 0.015 second steps
```

In [3]:
```
# Import inverse kinematics and jacobian based on input file
from importlib import import_module
robot = import_module(Title)
IK = robot.inverseKinematics
jac = robot.jacobian

# For documentation purposes: print out the code from that file here
fileTitle = Title + ".py"
print(fileTitle)
print("==========")
print(open(fileTitle).read())
```

```
RRR.py
==========
from math import sqrt, atan2, acos, sin, cos

def inverseKinematics(px, py, pz):
    L1, L2, L3, D1 = 5, 5, 4, 4
    e = (px*px + py*py)
    th1 = atan2(py, px) - acos(D1 / sqrt(e))
    a = px - (D1 * cos(th1))
    b = py - (D1 * sin(th1))
    c = pz - L1
    d = (px * sin(th1)) - (py*cos(th1))
    th3 = acos((a*a + b*b + c*c - L2*L2 - L3*L3) / (2*L2*L3))
    num = (d * (L2 + L3*cos(th3))) - (c * L3 * sin(th3))
    den = (c * (L2 + L3*cos(th3))) + (d * L3 * sin(th3))
    th2 = atan2(num,den)
    return [th1, th2, th3]

from numpy import matrix
from numpy.linalg import inv

def jacobian(th1, th2, th3, vx, vy, vz, px, py, pz):
    L1, L2, L3, D1 = 5, 5, 4, 4
    th23 = th2 + th3

    j11 = -py
    j12 = (L3 * sin(th1) * cos(th23)) + (L2 * sin(th1) * cos(th2))
    j13 = L3 * sin(th1) * cos(th23)

    j21 = px
    j22 = (-L3 * cos(th1) * cos(th23)) - (L2 * cos(th1) * cos(th2))
    j23 = -L3 * cos(th1) * cos(th23)

    j31 = 0
    j32 = (-L3 * sin(th23)) - (L2 * sin(th2))
    j33 = -L3 * sin(th23)

    J = matrix([[j11, j12, j13],
                [j21, j22, j23],
                [j31, j32, j33]])
    Jinv = inv(J)
    vxyz = matrix([[vx], [vy], [vz]])
    output = Jinv @ vxyz # the "@" is matrix multiplication

    th1d = output[0,0]
    th2d = output[1,0]
    th3d = output[2,0]
    return [th1d, th2d, th3d]
```

In [4]:

```
# Calculate top velocities
assert (timeTotal / timeStep) % 1 == 0, "Total time must divide evenly by timestep!"
S = int(timeTotal / timeStep)
assert (Sacc + Sdec) <= S, "Cannot accelerate + decelerate longer than total time!"
Smod = S - 0.5*Sacc - 0.5*Sdec

vMax = [0, 0, 0]
for i in range(3):
    vMax[i] = (PF[i] - P0[i]) / (Smod * timeStep)
```

```
print(vMax)
```

[0.9876543209876553, -9.876543209876544, -64.19753086419753]

In [5]:

```
# Calculate velocity curves
velocity = [[0], [0], [0]]
for i in range(3):
    # Acceleration
    for j in range(Sacc):
        velocity[i].append(vMax[i] * ((j+1) / Sacc))

    # Steady state
    for j in range(S - (Sacc + Sdec)):
        velocity[i].append(vMax[i])

    # Deceleration
    for j in range(Sdec):
        velocity[i].append(vMax[i] * ((Sdec - j - 1)/Sdec))
```

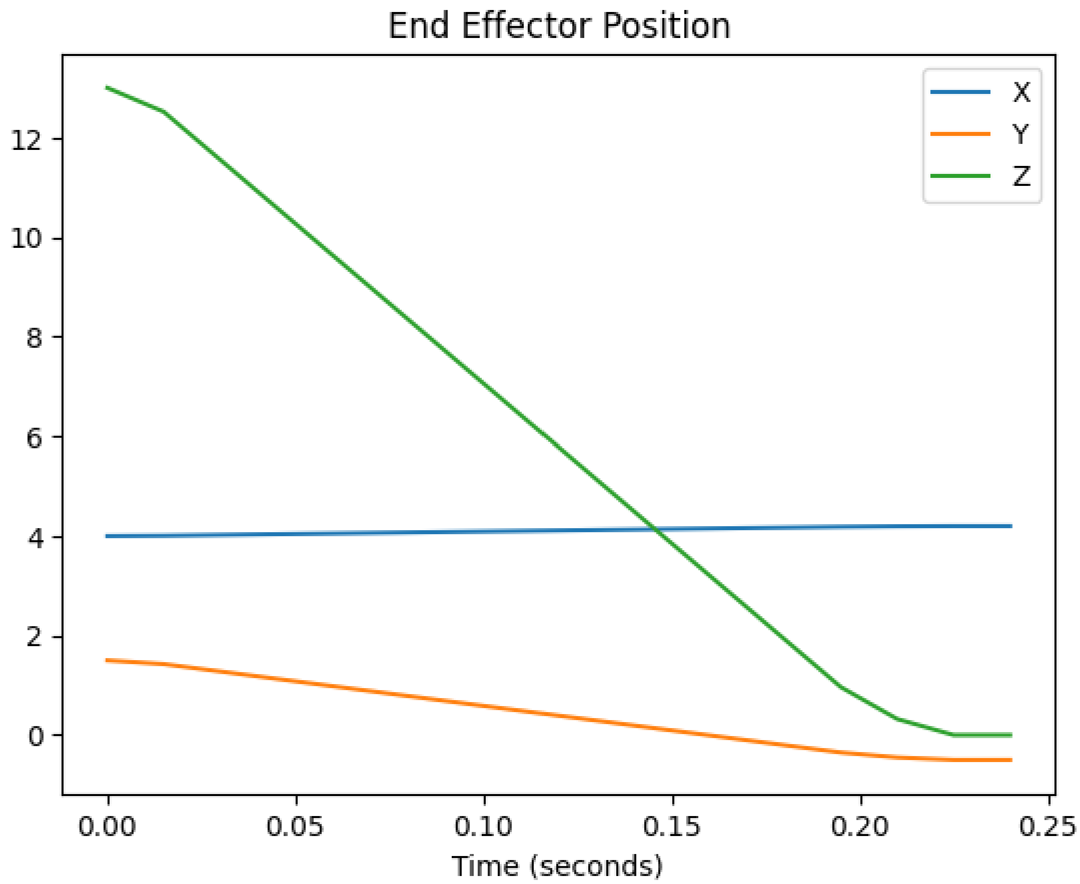
In [6]:

```
# Calculate position curves
position = [[P0[0]], [P0[1]], [P0[2]]]
for i in range(3):
    for t in range(1,S):
        position[i].append(position[i][-1] + velocity[i][t]*timeStep)
    position[i].append(PF[i])
```

In [7]:

```
# Make position plot
import matplotlib.pyplot as plt

time = [0]
for _ in range(S): time.append(time[-1]+timeStep)

for i in range(3): plt.plot(time, position[i])

plt.title("End Effector Position")
plt.legend(["X", "Y", "Z"])
plt.xlabel("Time (seconds)")
```

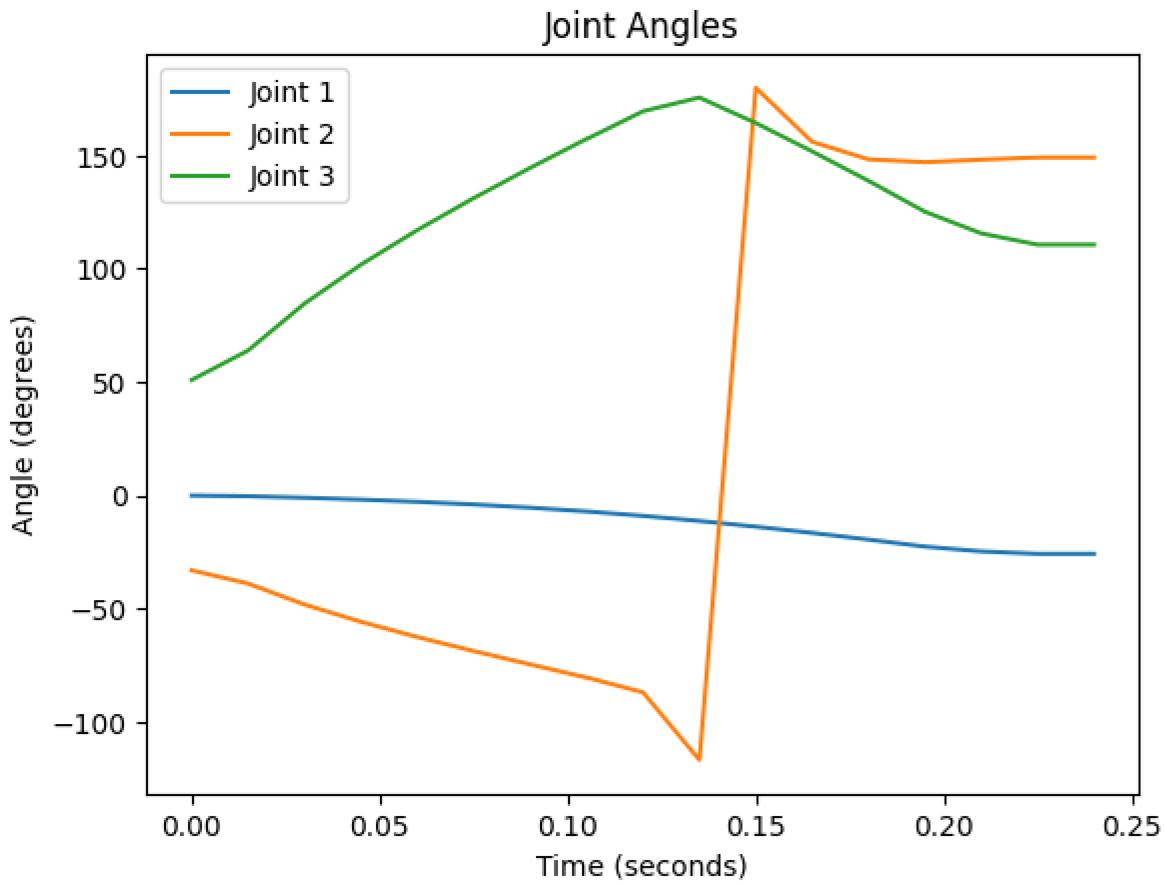
Out[7]: Text(0.5, 0, 'Time (seconds)')



In [8]:
```
# Calculate joint angles
from math import degrees

th1, th2, th3 = IK(P0[0], P0[1], P0[2])
thetas = [[degrees(th1)], [degrees(th2)], [degrees(th3)]]

for i in range(S):
    th1, th2, th3 = IK(position[0][i+1], position[1][i+1], position[2][i+1])
    thetas[0].append(degrees(th1))
    thetas[1].append(degrees(th2))
    thetas[2].append(degrees(th3))
```

In [9]:
```
# Make angle plots
for i in range(3): plt.plot(time, thetas[i])

plt.title("Joint Angles")
plt.legend(["Joint 1", "Joint 2", "Joint 3"])
plt.ylabel("Angle (degrees)")
plt.xlabel("Time (seconds)")
```

Out[9]:
```
Text(0.5, 0, 'Time (seconds)')
```



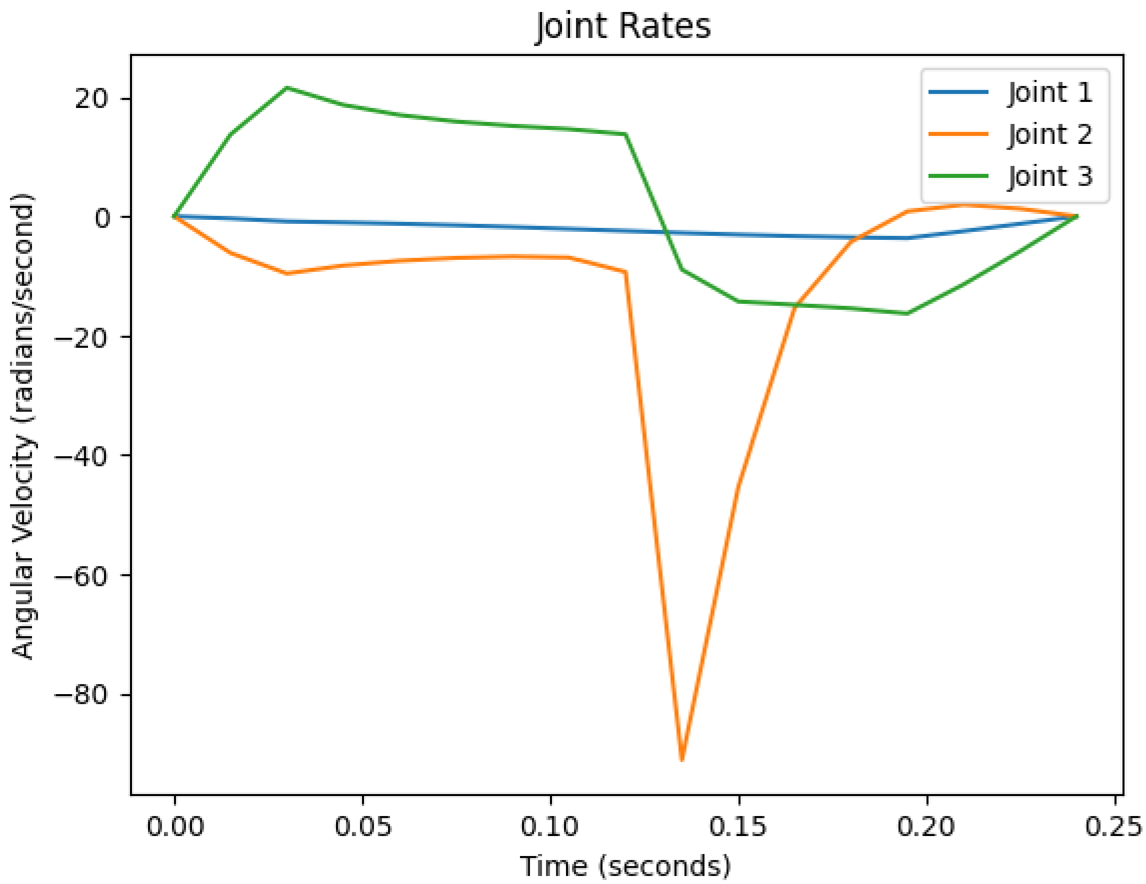
In [10]:

```
# Get joint rates
from math import radians
dotThetas = [[0], [0], [0]]
for i in range(S):
    dth1, dth2, dth3 = jac(
        radians(thetas[0][i+1]),
        radians(thetas[1][i+1]),
        radians(thetas[2][i+1]),
        velocity[0][i+1],
        velocity[1][i+1],
        velocity[2][i+1],
        position[0][i+1],
        position[1][i+1],
        position[2][i+1]
    )
    dotThetas[0].append(dth1)
    dotThetas[1].append(dth2)
    dotThetas[2].append(dth3)
```

In [11]:

```
# Plot joint rates
for i in range(3): plt.plot(time, dotThetas[i])

plt.title("Joint Rates")
plt.legend(["Joint 1", "Joint 2", "Joint 3"])
plt.ylabel("Angular Velocity (radians/second)")
plt.xlabel("Time (seconds)")
```

Out[11]: `Text(0.5, 0, 'Time (seconds)')`



In [12]:
```
# Generate output report
report = [",".join([
    "Time",
    "Px",
    "Py",
    "Pz",
    "Theta 1",
    "Theta 2",
    "Theta 3",
    "Theta-dot 1",
    "Theta-dot 2",
    "Theta-dot 3"
])]

for i in range(len(time)):
    report.append(",".join([
        str(time[i]),             # Time
        str(position[0][i]),      # Px
        str(position[1][i]),      # Py
        str(position[2][i]),      # Pz
        str(thetas[0][i]),        # Theta 1
        str(thetas[1][i]),        # Theta 2
        str(thetas[2][i]),        # Theta 3
        str(dotThetas[0][i]),     # Theta-dot 1
        str(dotThetas[1][i]),     # Theta-dot 2
        str(dotThetas[2][i])      # Theta-dot 3
    ]))
```

In [13]:
```
# Save the report to a file
outputTitle = Title + ".csv"
with open(outputTitle, mode = "w") as file:
```

```python
    file.write("\n".join(report))

# For documentation purposes: print the file here
with open(outputTitle, mode = "r") as file:
    print(file.read())
```

```
Time,Px,Py,Pz,Theta 1,Theta 2,Theta 3,Theta-dot 1,Theta-dot 2,Theta-dot 3
0,4.0,1.5,13.0,6.3611093629270335e-15,-33.02444851992605,50.85759440358768,0,0,0
0.015,4.007407407407407,1.4259259259259258,12.518518518518519,-0.2955001720920082,-38.8611117
7596609,63.86309632502407,-0.3589785649011977,-6.151358153814702,13.723165919586569
0.03,4.022222222222222,1.2777777777777777,11.555555555555555,-0.9706160512484265,-48.11476133
866482,84.56839815428263,-0.8581351036043056,-9.606824456889303,21.571756308221076
0.045,4.037037037037037,1.1296296296296295,10.592592592592592,-1.78002829328194,-55.741226925
38416,101.75513083838469,-1.0314727131170045,-8.26779961724698,18.7022516951567
0.06,4.051851851851852,0.9814814814814814,9.629629629629628,-2.7546834687141133,-62.477559347
092445,117.04196683949884,-1.2434723172890523,-7.471730057755569,17.004127741818916
0.075,4.066666666666667,0.8333333333333333,8.666666666666664,-3.929447539745731,-68.671147750
46886,131.15386020018374,-1.497360653580192,-6.984190139883228,15.911182480981642
0.09,4.081481481481482,0.6851851851851851,7.7037037037037015,-5.340025738435087,-74.553478973
22473,144.49357665802802,-1.79136879167061,-6.749348386045408,15.17811849413948
0.105,4.096296296296297,0.537037037037037,6.740740740740739,-7.017461485354663,-80.3803876186
1297,157.300912998035,-2.1160743889233213,-6.914923446887003,14.642671388923977
0.12,4.1111111111111125,0.38888888888888884,5.777777777777776,-8.981072902237337,-86.95704352
078741,169.597887634983,-2.4538033565375272,-9.343558315110757,13.766393079990117
0.135,4.1259259259259276,0.24074074074074067,4.814814814814813,-11.23244542342781,-116.813384
30874393,175.64457765776515,-2.7818201504611535,-91.18077594710411,-8.944659402511258
0.15000000000000002,4.140740740740743,0.0925925925925925,3.8518518518518503,-13.7536123139590
92,179.95239827656752,164.40503904041816,-3.0784581660135726,-45.398062753192356,-14.29936020
8369805
0.16500000000000004,4.155555555555558,-0.055555555555555663,2.8888888888888875,-16.5105360917
95267,156.0891305940098,151.85868972343113,-3.3286608898681056,-15.360700256073937,-14.860779
391886009
0.18000000000000005,4.170370370370373,-0.20370370370370383,1.9259259259259247,-19.45993665008
3222,148.34401670100712,138.8434850268885,-3.525970760263695,-4.2783935394511525,-15.45913270
7377359
0.19500000000000006,4.185185185185188,-0.35185185185185197,0.9629629629629618,-22.55623415938
44,147.094044328456,125.2095795249082,-3.671063439118202,0.8217051284869183,-16.3256160758617
6
0.21000000000000008,4.195061728395064,-0.4506172839506174,0.3209876543209865,-24.680630433515
29,148.20184995569,115.63465360770826,-2.4940860693160944,1.960758890201734,-11.4235721848604
95
0.2250000000000001,4.200000000000003,-0.5000000000000001,-1.1102230246251565e-15,-25.75647067
4270656,149.17156741350155,110.65560994147053,-1.2562930059974466,1.2713185131223803,-5.87916
1121643312
0.2400000000000001,4.2,-0.5,0.0,-25.75647067427056,149.17156741350144,110.65560994147059,0.0,
0.0,0.0
```