ME 572: Fall 2022 Computer Project

TJ Wiegman

2022-11-21

In [1]:
```
# Read text file input
with open("RR_HW5.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])
```

```
.25,.01,2,2,RR_HW5
3,.01,0
-2,.01,0
```

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"
)
```

```
RR_HW5: Moving from [3.0, 0.01, 0.0] to [-2.0, 0.01, 0.0] over 0.25 seconds in 0.01 second st
eps
```

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())
```

```
RR_HW5.py
==========
from math import sqrt, atan2, acos, sin, cos

def inverseKinematics(px, py, pz):
    L1, L2 = 20, 20
    a = (px*px + py*py + L1*L1 - L2*L2) / (2*L1)
    b = sqrt(px*px + py*py)
    th1 = atan2(py, px) + acos(a/b)
    num = py - (L1 * sin(th1))
    den = px - (L1 * cos(th1))
    th2 = atan2(num, den) - th1
    th3 = 0
    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 = 20, 20
    th12 = th1 + th2

    A = (-L1 * sin(th1))  + (-L2 * sin(th12))
    B = (-L2) * sin(th12)
    C = (L1 * cos(th1)) + (L2 * cos(th12))
    D = L2 * cos(th12)

    J = matrix([[A, B],
                [C, D]])
    Jinv = inv(J)
    vxy = matrix([[vx], [vy]])
    output = Jinv @ vxy # the "@" is matrix multiplication

    th1d = output[0,0]
    th2d = output[1,0]
    th3d = 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)
```

```
[-21.73913043478261, 0.0, 0.0]
```

In [5]:

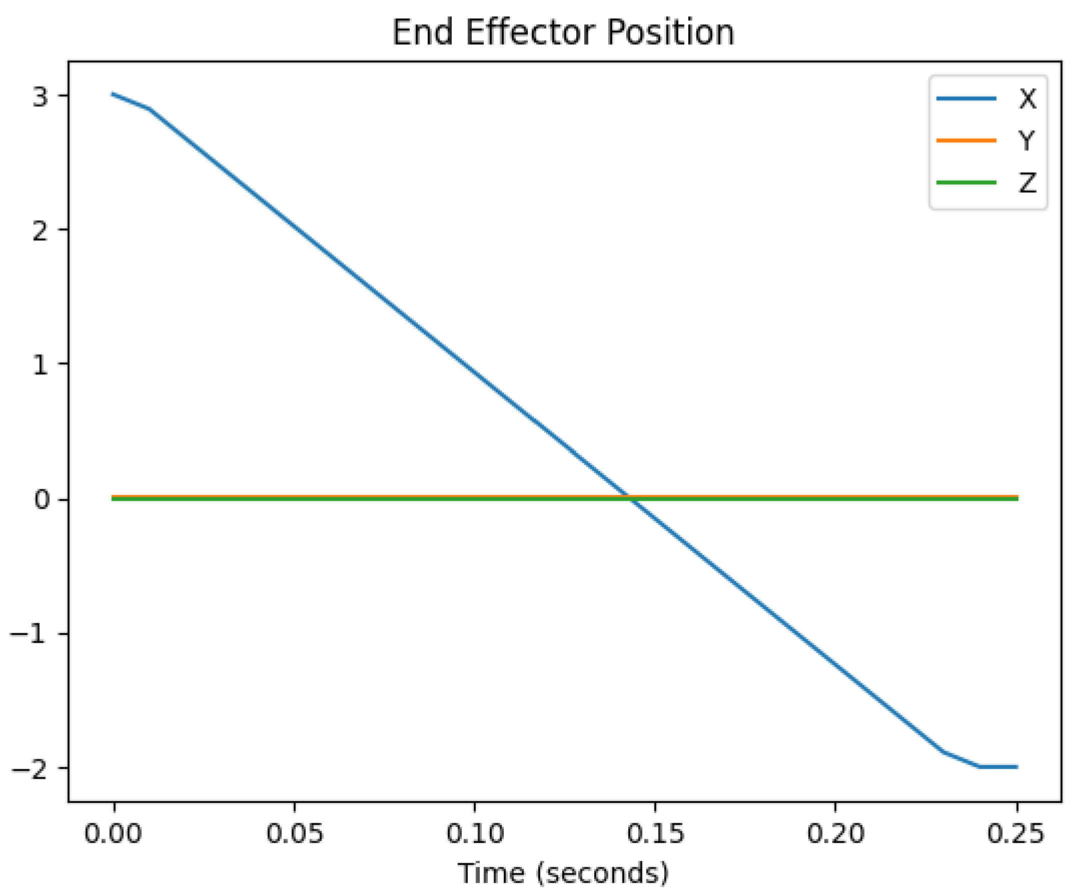
```
# Calculate velocity curves
velocity = [[0], [0], [0]]
for i in range(3):
    # Acceleration
    for j in range(Sacc):
        velocity[i].append(vMax[i] * ((j+1) / Sacc))
```

```
    # 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)')
```

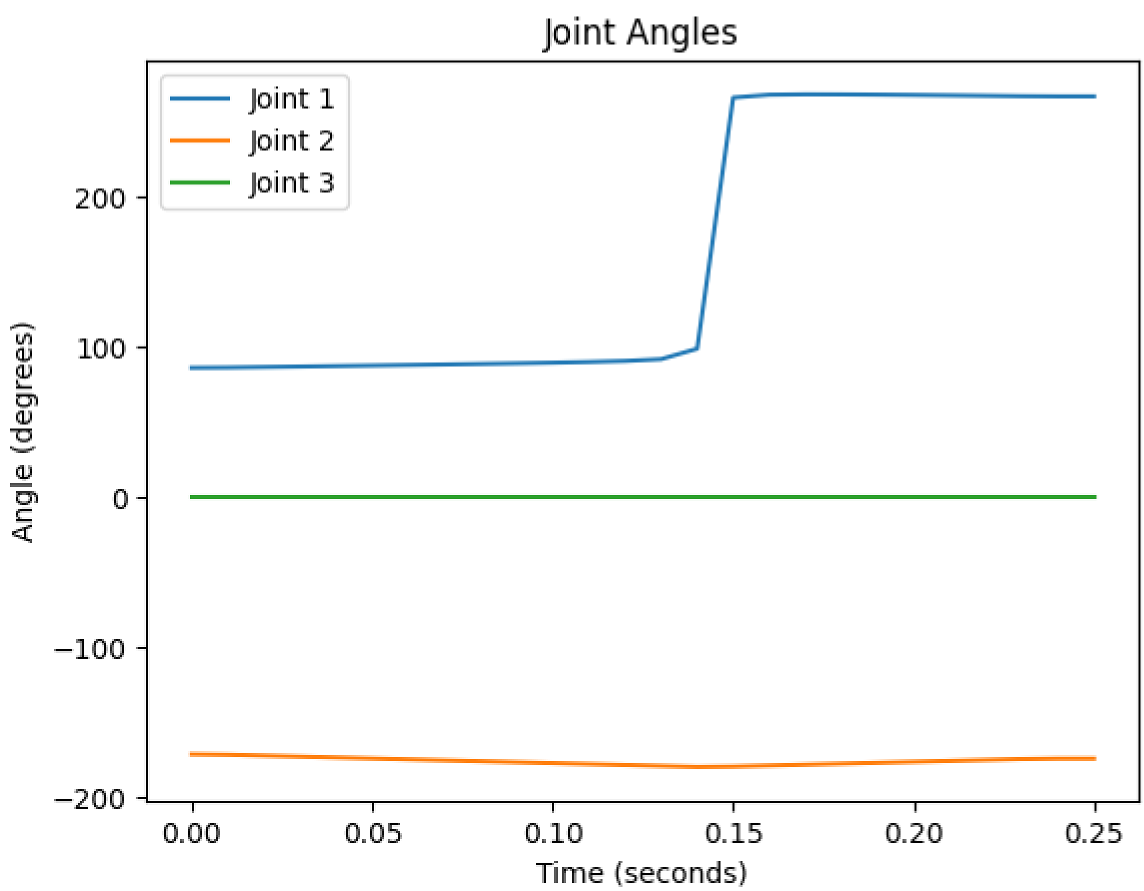


In [8]:
```
# Calculate joint angles
from math import degrees
```

```
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)')
```

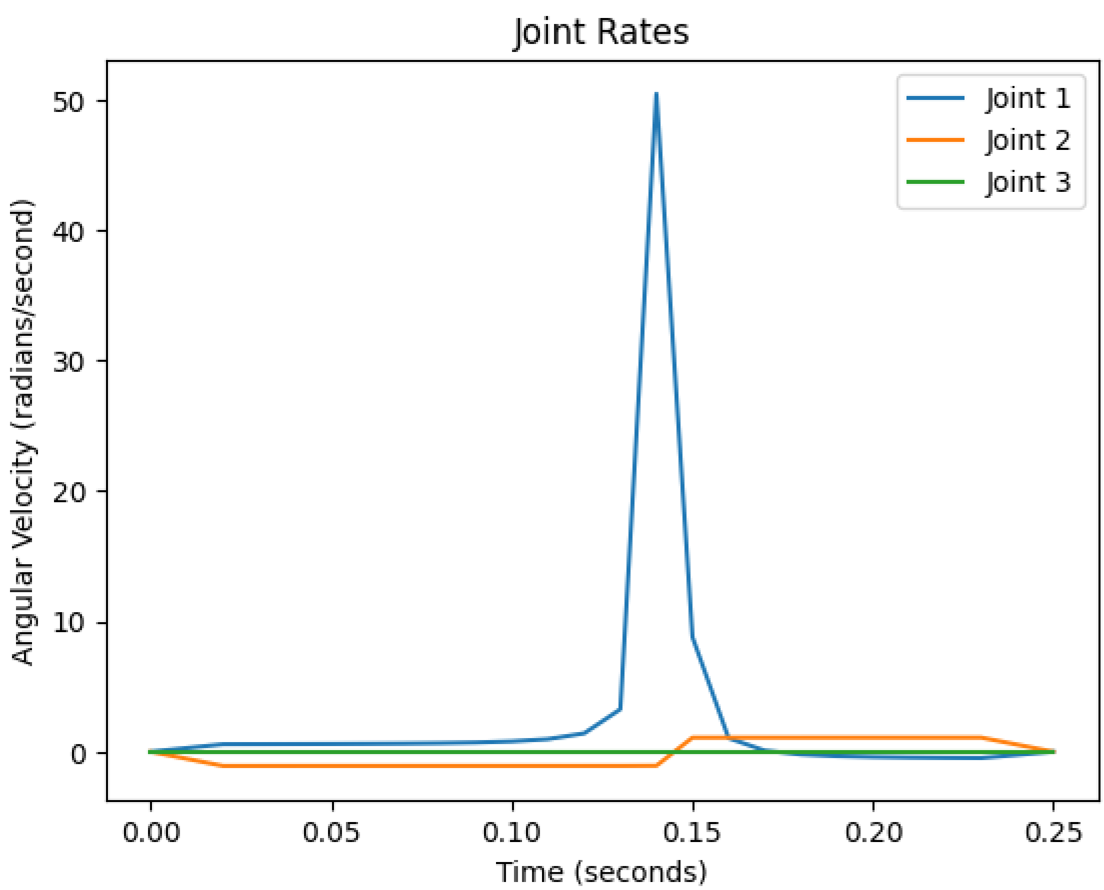


In [10]:
```
# Get joint rates
from math import radians
dotThetas = [[0], [0], [0]]
for i in range(S):
    dth1, dth2, dth3 = jac(
        radians(thetas[0][i+1]),
        radians(thetas[1][i+1]),
        radians(thetas[2][i+1]),
        velocity[0][i+1],
        velocity[1][i+1],
        velocity[2][i+1],
```

```
        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"
])]
```

```python
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]:

```python
# Save the report to a file
outputTitle = Title + ".csv"
with open(outputTitle, mode = "w") as file:
    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
0,3.0,0.01,0.0,85.8897389790859,-171.3975075094524,0.0,0,0,0
0.01,2.891304347826087,0.01,0.0,86.05303692087429,-171.70974371416503,0.0,0.2854524666942258,
-0.5449003826825275,0
0.02,2.6739130434782608,0.01,0.0,86.38129244970918,-172.33403309939018,0.0,0.575097612737642
7,-1.0893857117792778,0
0.03,2.4565217391304346,0.01,0.0,86.71228515974431,-172.95809398334686,0.0,0.580525706008401
5,-1.0890031110864664,0
0.04,2.2391304347826084,0.01,0.0,87.04685481133592,-173.5819448974027,0.0,0.5876849154541973,
-1.0886527368690893,0
0.05,2.0217391304347823,0.01,0.0,87.38619828195013,-174.2056042274859,0.0,0.5973511889758886,
-1.0883343015828213,0
0.060000000000000005,1.804347826086956,0.01,0.0,87.73208470599124,-174.8290901653967,0.0,0.61
0794767129871,-1.0880474029656548,0
0.07,1.58695652173913,0.01,0.0,88.08724742768845,-175.45242061586956,0.0,0.6302124156581418,-
1.0877914158970188,0
0.08,1.3695652173913038,0.01,0.0,88.45614920697545,-176.07561301027067,0.0,0.659674660307004
1,-1.0875652554863033,0
0.09,1.1521739130434776,0.01,0.0,88.8466135897912,-176.69868390367833,0.0,0.7074304103862107,
-1.087366800742035,0
0.09999999999999999,0.9347826086956514,0.01,0.0,89.27373221645989,-177.32164800281865,0.0,0.7
923503001158292,-1.087191283288173,0
0.10999999999999999,0.7173913043478253,0.01,0.0,89.77087442779911,-177.94451541789812,0.0,0.9
658367206583467,-1.0870258081112059,0
0.11999999999999998,0.49999999999999917,0.01,0.0,90.42940370565952,-178.56728173496867,0.0,1.
4126296009038877,-1.0868241410493538,0
0.12999999999999998,0.28260869565217306,0.01,0.0,91.62147975483518,-179.18987228711234,0.0,3.
2616416657611866,-1.0863038336627175,0
0.13999999999999999,0.06521739130434695,0.01,0.0,98.62294810083199,-179.8109822724644,0.0,50.
47423650866489,-1.074401194835588,0
0.15,-0.15217391304347916,0.01,0.0,266.02181080366495,-179.56311251933772,0.0,8.8050771199573
92,1.0846250397665111,0
0.16,-0.36956521739130527,0.01,0.0,267.92045734783756,-178.94087104171322,0.0,1.0472283327240
108,1.086605228335021,0
0.17,-0.5869565217391314,0.01,0.0,268.1830386272387,-178.318189030623,0.0,0.0873603373404054
2,1.0869158653850544,0
0.18000000000000002,-0.8043478260869574,0.01,0.0,268.1354006809083,-177.69537978292811,0.0,-
0.20758642683166742,1.0870923728960848,0
0.19000000000000003,-1.0217391304347836,0.01,0.0,267.9754878864581,-177.0724743718893,0.0,-0.
3354105527240865,1.0872592592221162,0
0.20000000000000004,-1.2391304347826098,0.01,0.0,267.7623576077636,-176.44946912529116,0.0,-
0.4021487305840946,1.0874430710139784,0
0.21000000000000005,-1.456521739130436,0.01,0.0,267.5198079385294,-175.82635153283874,0.0,-0.
44135824000381785,1.0876522420020187,0
0.22000000000000006,-1.673913043478262,0.01,0.0,267.25927057922695,-175.2031059647499,0.0,-0.
4663630940066193,1.0878901561873666,0
0.23000000000000007,-1.8913043478260883,0.01,0.0,266.9869173426723,-174.57971544340467,0.0,-
0.48330669394514697,1.0881584111526907,0
0.24000000000000007,-2.0000000000000013,0.01,0.0,266.8475036526389,-174.26796032583206,0.0,-
0.24490281644795678,0.5441521003211889,0
0.25000000000000006,-2.0,0.01,0.0,266.847503652639,-174.26796032583206,0.0,0.0,0.0,0
```

RR_HW5 results in a prettier table:

Time	Px	Py	Pz	Theta 1	Theta 2	Theta 3	Theta-dot 1	Theta-dot 2	Theta-dot 3
0.00	3.0000	0.0100	0.0000	85.8897	-171.3975	0.0000	0.0000	0.0000	0.0000
0.01	2.8913	0.0100	0.0000	86.0530	-171.7097	0.0000	0.2855	-0.5449	0.0000
0.02	2.6739	0.0100	0.0000	86.3813	-172.3340	0.0000	0.5751	-1.0894	0.0000
0.03	2.4565	0.0100	0.0000	86.7123	-172.9581	0.0000	0.5805	-1.0890	0.0000
0.04	2.2391	0.0100	0.0000	87.0469	-173.5819	0.0000	0.5877	-1.0887	0.0000
0.05	2.0217	0.0100	0.0000	87.3862	-174.2056	0.0000	0.5974	-1.0883	0.0000
0.06	1.8043	0.0100	0.0000	87.7321	-174.8291	0.0000	0.6108	-1.0880	0.0000
0.07	1.5870	0.0100	0.0000	88.0872	-175.4524	0.0000	0.6302	-1.0878	0.0000
0.08	1.3696	0.0100	0.0000	88.4561	-176.0756	0.0000	0.6597	-1.0876	0.0000
0.09	1.1522	0.0100	0.0000	88.8466	-176.6987	0.0000	0.7074	-1.0874	0.0000
0.10	0.9348	0.0100	0.0000	89.2737	-177.3216	0.0000	0.7924	-1.0872	0.0000
0.11	0.7174	0.0100	0.0000	89.7709	-177.9445	0.0000	0.9658	-1.0870	0.0000
0.12	0.5000	0.0100	0.0000	90.4294	-178.5673	0.0000	1.4126	-1.0868	0.0000
0.13	0.2826	0.0100	0.0000	91.6215	-179.1899	0.0000	3.2616	-1.0863	0.0000
0.14	0.0652	0.0100	0.0000	98.6229	-179.8110	0.0000	50.4742	-1.0744	0.0000
0.15	-0.1522	0.0100	0.0000	266.0218	-179.5631	0.0000	8.8051	1.0846	0.0000
0.16	-0.3696	0.0100	0.0000	267.9205	-178.9409	0.0000	1.0472	1.0866	0.0000
0.17	-0.5870	0.0100	0.0000	268.1830	-178.3182	0.0000	0.0874	1.0869	0.0000
0.18	-0.8043	0.0100	0.0000	268.1354	-177.6954	0.0000	-0.2076	1.0871	0.0000
0.19	-1.0217	0.0100	0.0000	267.9755	-177.0725	0.0000	-0.3354	1.0873	0.0000
0.20	-1.2391	0.0100	0.0000	267.7624	-176.4495	0.0000	-0.4021	1.0874	0.0000
0.21	-1.4565	0.0100	0.0000	267.5198	-175.8264	0.0000	-0.4414	1.0877	0.0000
0.22	-1.6739	0.0100	0.0000	267.2593	-175.2031	0.0000	-0.4664	1.0879	0.0000
0.23	-1.8913	0.0100	0.0000	266.9869	-174.5797	0.0000	-0.4833	1.0882	0.0000
0.24	-2.0000	0.0100	0.0000	266.8475	-174.2680	0.0000	-0.2449	0.5442	0.0000
0.25	-2.0000	0.0100	0.0000	266.8475	-174.2680	0.0000	0.0000	0.0000	0.0000

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)
```

```python
print(vMax)
```

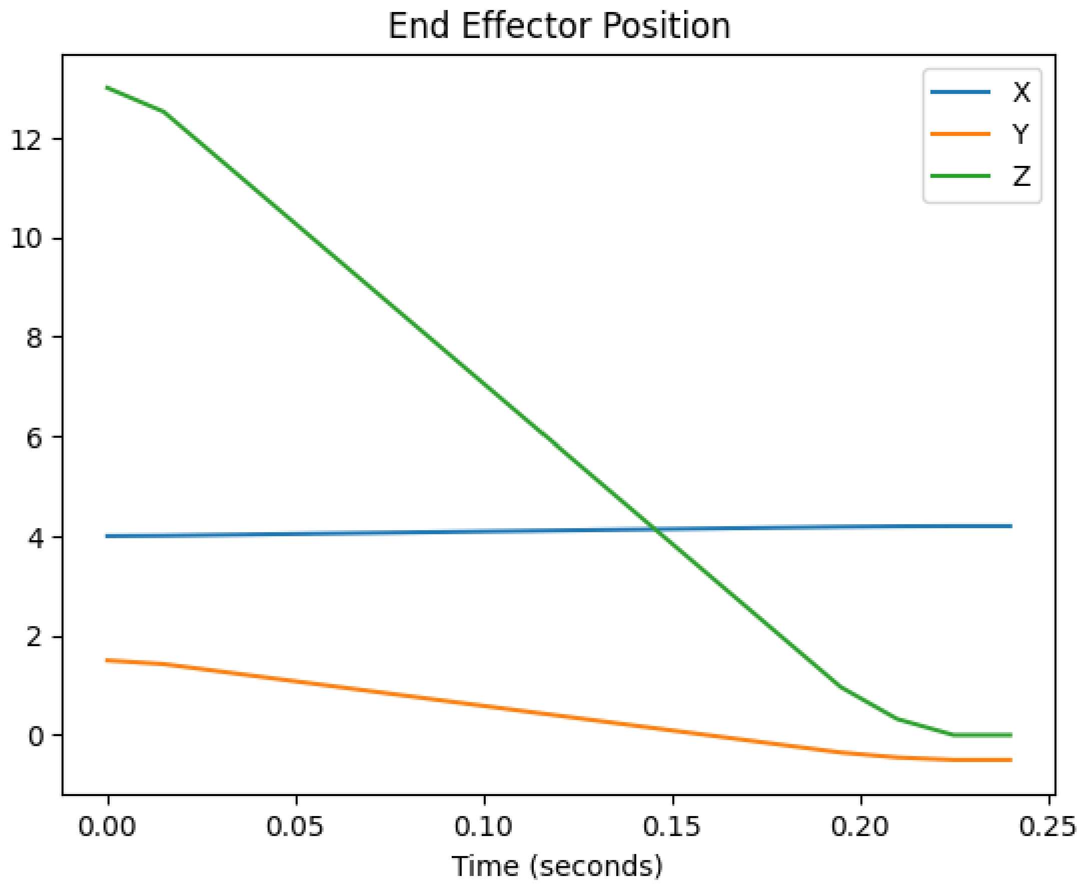
```
[0.9876543209876553, -9.876543209876544, -64.19753086419753]
```

In [5]:
```python
# 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]:
```python
# 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]:
```python
# 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)")
```

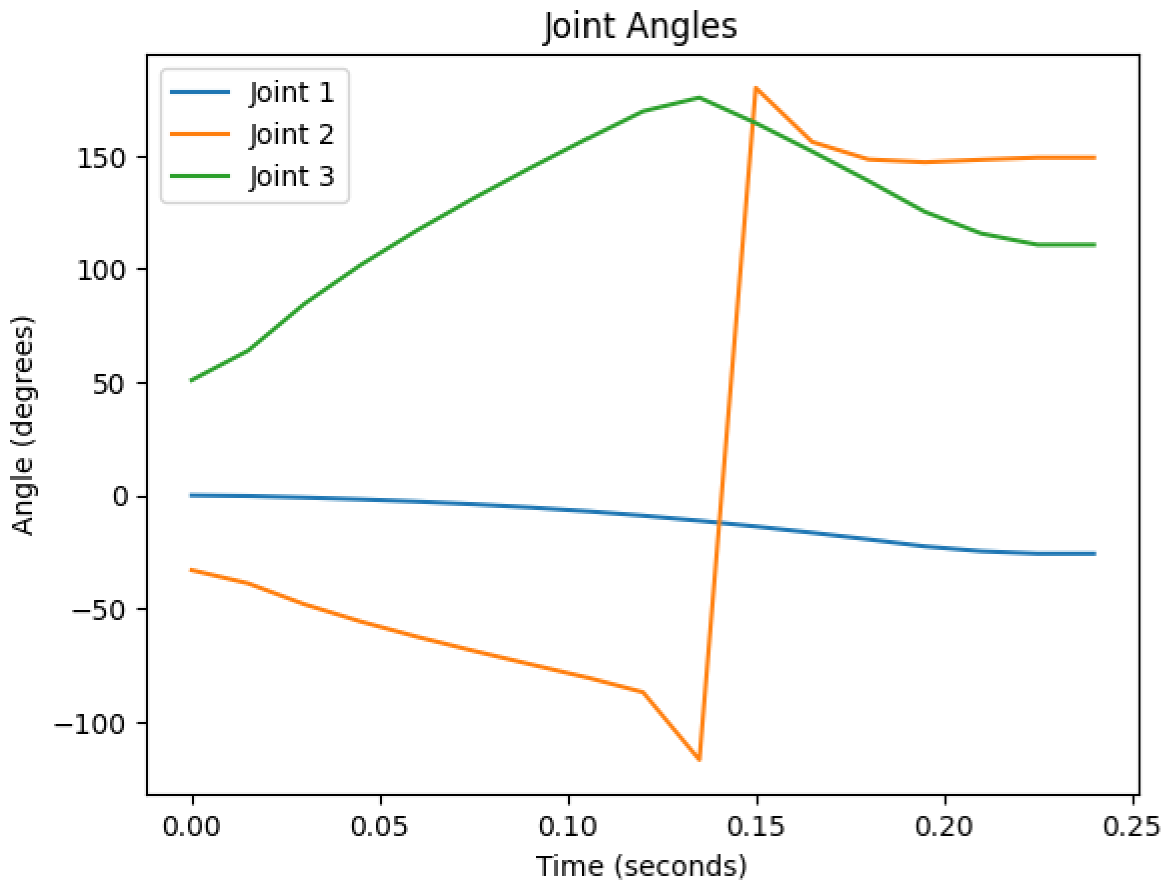
Out[7]:
```
Text(0.5, 0, 'Time (seconds)')
```



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)")
```

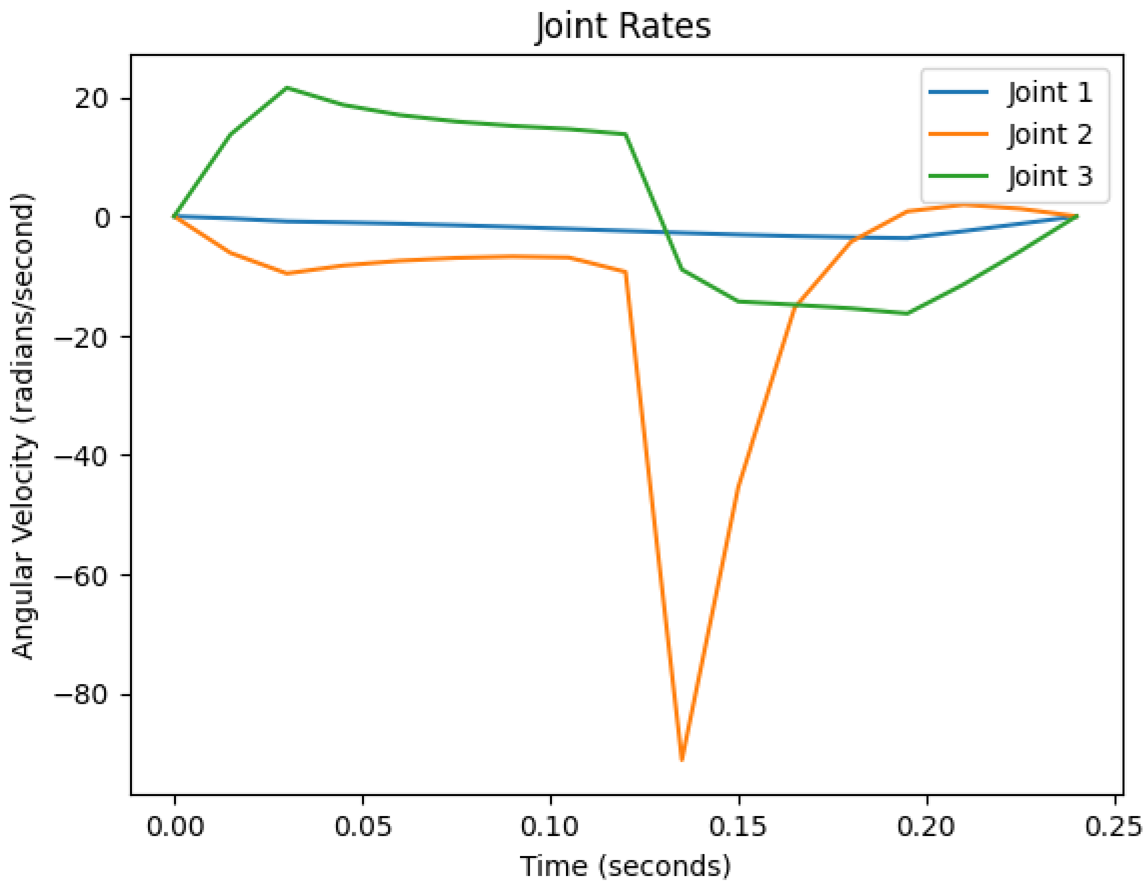
Out[9]:
```
Text(0.5, 0, 'Time (seconds)')
```



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
0.09,4.081481481481482,0.6851851851851851,7.703703703703015,-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
0.12,4.111111111111125,0.3888888888888884,5.777777777777776,-8.981072902237337,-86.95704352
078741,169.597887634983,-2.4538033565375272,-9.343558315110757,13.766393079990117
0.135,4.125925925925276,0.24074074074074067,4.814814814814813,-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,-0.05555555555555663,2.888888888888875,-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
```

RRR results in a prettier table:

Time	Px	Py	Pz	Theta 1	Theta 2	Theta 3	Theta-dot 1	Theta-dot 2	Theta-dot 3
0.000	4.0000	1.5000	13.0000	0.0000	-33.0244	50.8576	0.0000	0.0000	0.0000
0.015	4.0074	1.4259	12.5185	-0.2955	-38.8611	63.8631	-0.3590	-6.1514	13.7232
0.030	4.0222	1.2778	11.5556	-0.9706	-48.1148	84.5684	-0.8581	-9.6068	21.5718
0.045	4.0370	1.1296	10.5926	-1.7800	-55.7412	101.7551	-1.0315	-8.2678	18.7023
0.060	4.0519	0.9815	9.6296	-2.7547	-62.4776	117.0420	-1.2435	-7.4717	17.0041
0.075	4.0667	0.8333	8.6667	-3.9294	-68.6711	131.1539	-1.4974	-6.9842	15.9112
0.090	4.0815	0.6852	7.7037	-5.3400	-74.5535	144.4936	-1.7914	-6.7493	15.1781
0.105	4.0963	0.5370	6.7407	-7.0175	-80.3804	157.3009	-2.1161	-6.9149	14.6427
0.120	4.1111	0.3889	5.7778	-8.9811	-86.9570	169.5979	-2.4538	-9.3436	13.7664
0.135	4.1259	0.2407	4.8148	-11.2324	-116.8134	175.6446	-2.7818	-91.1808	-8.9447
0.150	4.1407	0.0926	3.8519	-13.7536	179.9524	164.4050	-3.0785	-45.3981	-14.2994
0.165	4.1556	-0.0556	2.8889	-16.5105	156.0891	151.8587	-3.3287	-15.3607	-14.8608
0.180	4.1704	-0.2037	1.9259	-19.4599	148.3440	138.8435	-3.5260	-4.2784	-15.4591
0.195	4.1852	-0.3519	0.9630	-22.5562	147.0940	125.2096	-3.6711	0.8217	-16.3256
0.210	4.1951	-0.4506	0.3210	-24.6806	148.2018	115.6347	-2.4941	1.9608	-11.4236
0.225	4.2000	-0.5000	0.0000	-25.7565	149.1716	110.6556	-1.2563	1.2713	-5.8792
0.240	4.2000	-0.5000	0.0000	-25.7565	149.1716	110.6556	0.0000	0.0000	0.0000