Linkage Analysis Project

2022 Fall

B08502006

Shiu-Cheng, Wang

王旭承

Linkage analysis project

Formulation of problem

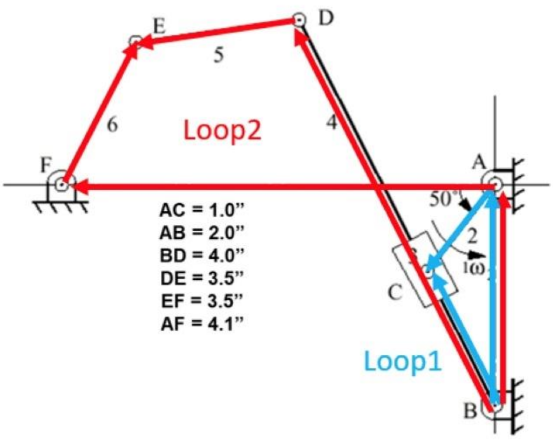


Figure 1

First define:

$$r1 = \overrightarrow{BA}$$

$$r2 = \overrightarrow{AC}$$

$$r3 = \overrightarrow{BC}$$

$$r4 = \overrightarrow{BD}$$

$$r5 = \overrightarrow{DE}$$

$$r6 = \overrightarrow{EF}$$

$$r7 = \overrightarrow{BG}$$

$$r8 = \overrightarrow{GF}$$

The respective θ from 1 to 8 is defined as the angle from the positive x-axis, with counter-clockwise being the positive direction.

Given the inputs, the constants are r1, r2, r3, r4, r5, r6, r7, r8, θ1, θ7, θ8. The unknowns are r3, θ2, θ3, θ4, θ5, θ6.

1. Position equations

(a) Loop 1:

$$r1 + r2 - r3 = 0$$

x-direction:

$$\mathrm{r1cos\theta1 + r2cos\theta2 - r3cos\theta3 = 0}$$

y-direction:

$$\mathrm{r1sin\theta1 + r2sin\theta2 - r3sin\theta3 = 0}$$

Plug in the condition that $\theta 1 = 90$ degree:

x-direction:

$$\mathrm{r2cos\theta2 - r3cos\theta3 = 0}$$

y-direction:

$$\mathrm{r1 + r2sin\theta2 - r3sin\theta3 = 0}$$

(b) Loop 2:

$$r4 + r5 + r6 - r7 - r8 = 0$$

x-direction:

$$\mathrm{r4cos\theta4 + r5cos\theta5 + r6cos\theta6 - r7cos\theta7 - r8cos\theta8 = 0}$$

y-direction:

$$\mathrm{r4sin\theta4 + r5sin\theta5 + r6sin\theta6 - r7sin\theta7 - r8sin\theta8 = 0}$$

Plug in the conditions that $\theta 3 = \theta 4, \theta 7 = 180$ degree, $\theta 8 = 90$ degree

x-direction:

$$\mathrm{r4cos\theta3 + r5cos\theta5 + r6cos\theta6 - r7 = 0}$$

y-direction:

$$\mathrm{r4sin\theta3 + r5sin\theta5 + r6sin\theta6 - r8 = 0}$$

First solve Loop 1 to yield r3, $\theta 3$, then solve for $\theta 5$ and $\theta 6$.

2. Velocity equations

By differentiating the position equations used above:

(a) Loop 1:

x-direction:

$$- r2\sin\theta 2\dot{\theta}2 \; - \; \dot{r}3\cos\theta 3 + r3\sin\theta 3\dot{\theta}3 = \; 0$$

y-direction:

$$r2\cos\theta 2\dot{\theta}2 \; - \; \dot{r}3\sin\theta 3 \; - \; r3\cos\theta 3\dot{\theta}3 = \; 0$$

(b) Loop 2:

x-direction:

$$- r4\sin\theta 3\dot{\theta}3 - \; r5\sin\theta 5\dot{\theta}5 \; - \; \dot{r}6\sin\theta 6\dot{\theta}6 = \; 0$$

y-direction:

$$r4\cos\theta 3\dot{\theta}3 + \; r5\cos\theta 5\dot{\theta}5 \; + \; r6\cos\theta 6\dot{\theta}6 = \; 0$$

3. Acceleration equations

(a) Loop 1:

x-direction:

$$- r2\cos\theta 2\dot{\theta}2 \; - \; r2\sin\theta 2\ddot{\theta}2 - \ddot{r}3\cos\theta 3 + 2\dot{r}3\sin\theta 3\dot{\theta}3 + r3\cos\theta 3\dot{\theta}3^2 + r3\sin\theta 3\ddot{\theta}3$$
$$= \; 0$$

y-direction:

$$-r2\cos\theta 2\dot{\theta}2^2 + r2\cos\theta 2\ddot{\theta}2 - \ddot{r}3\sin\theta 3 - \; 2\dot{r}3\cos\theta 3\dot{\theta}3 + r3\sin\theta 3\dot{\theta}3^2$$
$$- \; r3\cos\theta 3\ddot{\theta}3 = \; 0$$

(b) Loop 2:

x-direction:

$$-r4\cos\theta 3\dot{\theta}3^2 - \; r4\sin\theta 3\ddot{\theta}3 - \; r5\cos\theta 5\dot{\theta}5^2 - \; r5\sin\theta 5\ddot{\theta}5 \; - \; r6\cos\theta 6\dot{\theta}6^2$$
$$- \; r6\sin\theta 6\ddot{\theta}6 = \; 0$$

y-direction:

$$-r4\sin\theta 3\dot{\theta}3^2 + \; r4\cos\theta 3\ddot{\theta}3 - r5\sin\theta 5\dot{\theta}5^2 + \; r5\cos\theta 5\ddot{\theta}5 - r6\sin\theta 6\dot{\theta}6^2$$
$$+ \; r6\cos\theta 6\ddot{\theta}6 = \; 0$$

Computer program

1. Parameters:

```
#the symbols needed later, theta is represented as t, delxx is the calibrating value of the unknowns (xx) we want to estimate
#v stands for velocity, w stands for angular velocity (rad/s), a is the angular acceleration, except for ar3 being the acceleration of vector r3
r3, t3, t5, t6, delr3, delt3, delt5, delt6, v3, w3, w5, w6, ar3, at3, a5, a6, t, x, y= sym.symbols('r3, t3, t5, t6, delr3, delt3, delt5, delt6, v3, w3, w5, w6, ar3, at3, a5, a6, t, x, y')

# It is important to distinguish between the symbols and values in a symbolic calculation. The line above are the symbols, which are the unknows we aim to solve or we solved in the way;
# the line below is the ones being numeric.
rv2, rv1, rv4, rv5, rv6, rv7, wv2, av2 = [float(i) for i in input("Enter parameters, in the sequence of r2, r1, r4, r5, r6, r7, w2, a2: ").split()] #Input of the given parameters
which = int(input("enter 1 if you want the position relation plot; enter 2 if you want velocity relation plot; else if you want acceleration plot: "))
rv8 = rv1
```

2. Numerical method:

(a) Initial values

```
#find the desired initial values for t5 and t6
phi = sym.atan(rv2/rv1)
xright = rv4*sym.sin(phi)
yright = rv4*sym.cos(phi) - rv1
xleft = -rv7
yleft = 0
b = ((xright - xleft)**2 + yright**2)**0.5
eq1 = sym.Eq((x+b)**2 + y**2 - rv6**2,0)
eq2 = sym.Eq(x**2 + y**2 - rv5**2,0)
result = sym.solve([eq1,eq2],(x,y))
```

```
for i in result:
    if i[0].is_real and i[1].is_real:
        x = i[0]
        y = abs(i[1])
tp1 = sym.atan(y/x)

#There are two circumstances. If the first method won't work, the
second will
try:
    if x*y <0:
        tp1 = sym.N(tp1+sym.pi)
        tp2 = sym.atan(y/(x+b))
    if y/(x+b) <0:
        tp2 = sym.N(tp2+sym.pi)
    theta = sym.atan(yright/(xright-xleft))
    tp1 += theta
    tp2 += theta
    tp2 += sym.N(sym.pi)

except:
    t3min56 = sym.N(sym.pi - sym.atan(rv1 / rv7) - sym.acos((rv1**2 +
rv7**2 + rv4**2 - (rv5 + rv6)**2) / (2 * sym.sqrt(rv1**2 + rv7**2) *
rv4)))
    xright = rv4 * sym.cos(t3min56)
    yright = rv4 * sym.sin(t3min56) - rv1
    tp1 = sym.atan(yright/(xright + rv7))
    tp1 = sym.N(sym.pi + tp1)
    tp2 = tp1 + 0.01
#The initial guesses of the positions in Loop one are all ones.
rv3 = rv1
tv3 = 1.0
tv5 = tp1
tv6 = tp2
```

(b) Iterations

The rationale of using symbolic calculation is that $\Delta\theta 3$ and $\Delta r3$ can be seen as functions of the estimates $\widehat{\theta 3}$ and $\widehat{r3}$. Each time we plug in new values of $\widehat{\theta 3}$ and $\widehat{r3}$, we get a new set of values of $\Delta\theta3$ and $\Delta r3$. Repeat this calculation until f1 and f2 is close enough to zero.

```
#take 35 values of input angles with an equal interval of 10 degree for
the for loop
for i in range(1,37):
    tv2 = 10*i/180*sym.pi
    print("tv2 =", 10*i)

    eqt = sym.Eq(0.5 * av2 * t**2 + wv2 * t - sym.N(10/180*sym.pi),0)
    resultt = sym.solve(eqt,t)
    tv = max(resultt)
    wv2 = wv2 + av2 * tv

    #Loop1
    #x-direction, we want to find solutions for f1 == 0
    f1 = rv2 * sym.cos(tv2) - r3 * sym.cos(t3) # eq1
    #y-direction
    f2 = rv1 + rv2 * sym.sin(tv2) - r3 * sym.sin(t3) #eq2
    eq1 = sym.Eq(sym.diff(f1, r3) * delr3 + sym.diff(f1, t3) * delt3, -
f1)
    eq2 = sym.Eq(sym.diff(f2, r3) * delr3 + sym.diff(f2, t3) * delt3, -
f2)
    result1 = sym.solve([eq1, eq2], (delr3, delt3)) # the result contain
the symbolic representation of delr3 and delt3
    delr3v = 0 #in each input angle, delta is set to 0
    delt3v = 0

    #we calculate the delr3v and delt3v and add them to r3v and
t3v.Everytime we adjust the value of r3 and t3, we plug them into eq1
and eq2 to find if f1 and f2 are both lesser than or equal to 0.0001
    while abs(sym.N(f1.subs({t3: tv3, r3: rv3}))) > 0.0001 or
abs(sym.N(f2.subs({t3: tv3, r3: rv3}))) > 0.0001:
        delr3v = sym.N(result1[delr3].subs({t3:tv3, r3: rv3}))
        delt3v = sym.N(result1[delt3].subs({t3:tv3, r3: rv3}))
        rv3 += delr3v
        tv3 += delt3v
```

```
    #if the resulted values of r3 or t3 are not in the limits, it cannot
occur.
    if round(tv3, 4) > round(maxt3,4) or round(tv3,4) < round(mint3,4):
#the round() is used to avoid the wrong result from the calculation
above
        print(f"This input angle {10*i} degree cannot occur!")
        table.append([10*i,"N","N","N","N","N","N","N","N","N"])
        continue
    else:
        t2plot6[i-1] = np.float64(10*i)
        t3plot[i-1] = np.float64(tv3/sym.N(sym.pi)*180)
        r3plot[i-1] = np.float64(rv3*100)
        table.append([10*i])

    #Loop 2, basically the same as above
    f3 = rv4 * sym.cos(tv3) + rv5 * sym.cos(t5) + rv6 * sym.cos(t6) +
rv7
    f4 = rv4 * sym.sin(tv3) + rv5 * sym.sin(t5) + rv6 * sym.sin(t6) -
rv8
    eq3 = sym.Eq(sym.diff(f3, t5) * delt5 + sym.diff(f3, t6) * delt6, -
f3)
    eq4 = sym.Eq(sym.diff(f4, t5) * delt5 + sym.diff(f4, t6) * delt6, -
f4)
    result2 = sym.solve([eq3, eq4], (delt5, delt6))
    delt5v = 0
    delt6v = 0
    while abs(sym.N(f3.subs({t5: tv5, t6: tv6}))) > 0.0001 or
abs(sym.N(f4.subs({t5: tv5, t6: tv6}))) > 0.0001:
        delt5v = sym.N(result2[delt5].subs({t5:tv5, t6: tv6}))
        delt6v = sym.N(result2[delt6].subs({t5:tv5, t6: tv6}))
        tv5 += delt5v
        tv6 += delt6v

        #let the angles yielded always be in 0 to 2*pi
        while tv5 < 0:
            tv5 += sym.N(2 * sym.pi)
        while tv6 < 0:
            tv6 += sym.N(2 * sym.pi)
```

```
        while tv5 > 2 * sym.pi:
            tv5 -= sym.N(2 * sym.pi)
        while tv6 > 2 * sym.pi:
            tv6 -= sym.N(2 * sym.pi)
    t5plot[i-1] = np.float64(tv5/sym.N(sym.pi)*180)
    t6plot[i-1] = np.float64(tv6/sym.N(sym.pi)*180)
```

3. The complete code:

```
#This program uses symbolic calculation, while the iterative numerical
methods to find the solution of non-linear equations are used.
import sympy as sym
import matplotlib.pyplot as plt
import numpy as np
from tabulate import tabulate

#the symbols needed later, theta is represented as t, delxx is the
calibrating value of the unknowns (xx) we want to estimate
#v stands for velocity, w stands for angular velocity (rad/s), a is the
angular acceleration, except for ar3 being the acceleration of vector
r3
r3, t3, t5, t6, delr3, delt3, delt5, delt6, v3, w3, w5, w6, ar3, at3,
a5, a6, t, x, y= sym.symbols('r3, t3, t5, t6, delr3, delt3, delt5,
delt6, v3, w3, w5, w6, ar3, at3, a5, a6, t, x, y')

# It is important to distinguish between the symbols and values in a
symbolic calculation. The line above are the symbols, which are the
unknows we aim to solve or we solved in the way;
# the line below is the ones being numeric.
rv2, rv1, rv4, rv5, rv6, rv7, wv2, av2 = [float(i) for i in
input("Enter parameters, in the sequence of r2, r1, r4, r5, r6, r7, w2,
a2: ").split()] #Input of the given parameters
which = int(input("enter 1 if you want the angle position relation
plot; enter 2 if you want the position of point D & E; enter 3 if you
want velocity relation plot; else if you want acceleration plot: "))
rv8 = rv1
# a = np.full( (10,1),0.00001)
t2plot6 = np.full((36,1),0.00001)
```

```
t3plot = np.full((36,1),0.00001)
r3plot = np.full((36,1),0.00001)
t5plot = np.full((36,1),0.00001)
t6plot = np.full((36,1),0.00001)
dxplot = np.full((36,1),0.00001)
dyplot = np.full((36,1),0.00001)
explot = np.full((36,1),0.00001)
eyplot = np.full((36,1),0.00001)
evxplot = np.full((36,1),0.00001)
evyplot = np.full((36,1),0.00001)
eaxplot = np.full((36,1),0.00001)
eayplot = np.full((36,1),0.00001)
table = list()

#find the desired initial values for t5 and t6
phi = sym.atan(rv2/rv1)
xright = rv4*sym.sin(phi)
yright = rv4*sym.cos(phi) - rv1
xleft = -rv7
yleft = 0
b = ((xright - xleft)**2 + yright**2)**0.5
eq1 = sym.Eq((x+b)**2 + y**2 - rv6**2,0)
eq2 = sym.Eq(x**2 + y**2 - rv5**2,0)
result = sym.solve([eq1,eq2],(x,y))

for i in result:
    if i[0].is_real and i[1].is_real:
        x = i[0]
        y = abs(i[1])
tp1 = sym.atan(y/x)

#There are two circumstances.  If the first method won’t work, the
second will
try:
    if x*y <0:
        tp1 = sym.N(tp1+sym.pi)
        tp2 = sym.atan(y/(x+b))
    if y/(x+b) <0:
```

```
        tp2 = sym.N(tp2+sym.pi)
    theta = sym.atan(yright/(xright-xleft))
    tp1 += theta
    tp2 += theta
    tp2 += sym.N(sym.pi)

except:
    t3min56 = sym.N(sym.pi - sym.atan(rv1 / rv7) - sym.acos((rv1**2 +
rv7**2 + rv4**2 - (rv5 + rv6)**2) / (2 * sym.sqrt(rv1**2 + rv7**2) *
rv4)))
    xright = rv4 * sym.cos(t3min56)
    yright = rv4 * sym.sin(t3min56) - rv1
    tp1 = sym.atan(yright/(xright + rv7))
    tp1 = sym.N(sym.pi + tp1)
    tp2 = tp1 + 0.1
#The initial guesses of the positions in Loop one are all ones.
rv3 = 1.0
tv3 = 1.0
tv5 = tp1
tv6 = tp2

#Whether the input link r2 can revolute 360 degree is dependent on
maximum and minimum angle of r3
#The maximum and minimum angles have two restricts: one by Loop1 and
one by Loop2
#t3max123 is the maximum angle allowed by Loop 1, same for t3min123
t3max123 = sym.N(sym.pi / 2 + sym.asin(rv2 / rv1))
t3min123 = sym.N(sym.pi / 2 - sym.asin(rv2 / rv1))

#t3max56 is the maximum angle allowed by Loop 2, same for t3min56
if rv5 == rv6:
    t3max56 = sym.N(sym.pi - sym.atan(rv1 / rv7))
else:
    t3max56 = sym.N(sym.pi - sym.atan(rv1 / rv7) - sym.acos((rv1**2 +
rv7**2 + rv4**2 - (rv5 - rv6)**2) / (2 * sym.sqrt(rv1**2 + rv7**2) *
rv4)))
```

```python
t3min56 = sym.N(sym.pi - sym.atan(rv1 / rv7) - sym.acos((rv1**2 +
rv7**2 + rv4**2 - (rv5 + rv6)**2) / (2 * sym.sqrt(rv1**2 + rv7**2) *
rv4)))

# this yields the possible moving range of r3
#Because of the numbers in sympy are all complex, the complex ones
should be excluded
if t3max56.is_real:
    maxt3 = min(t3max123, t3max56)
else:
    maxt3 = t3max123

if t3min56.is_real:
    mint3 = max(t3min123, t3min56)
else:
    mint3 = t3min123
# print("maxt3:", maxt3, "mint3:", mint3)

#Note that if and only if maxt3 == t3max123 and mint3 ==t3min123, r3
can revolute 360 degree!

#take 35 values of input angles with an equal interval of 10 degree for
the for loop
for i in range(1,37):
    tv2 = 10*i/180*sym.pi
    print("tv2 =", 10*i)
#     t2plot[i-1] = np.float64(10*i)

    eqt = sym.Eq(0.5 * av2 * t**2 + wv2 * t - sym.N(10/180*sym.pi),0)
    resultt = sym.solve(eqt,t)
    tv = max(resultt)
    wv2 = wv2 + av2 * tv
    # print("f")
    #Loop1
    #x-direction, we want to find solutions for f1 == 0
    f1 = rv2 * sym.cos(tv2) - r3 * sym.cos(t3) # eq1
    #y-direction
    f2 = rv1 + rv2 * sym.sin(tv2) - r3 * sym.sin(t3) #eq2
```

```
    eq1 = sym.Eq(sym.diff(f1, r3) * delr3 + sym.diff(f1, t3) * delt3, -
f1)
    eq2 = sym.Eq(sym.diff(f2, r3) * delr3 + sym.diff(f2, t3) * delt3, -
f2)
    result1 = sym.solve([eq1, eq2], (delr3, delt3)) # the result contain
the symbolic representation of delr3 and delt3
    delr3v = 0
    delt3v = 0
    # print("s")
    #everytime we adjust the value of r3 and t3, we plug them into eq1
and eq2 to find if f1 and f2 are both lesser than or equal to 0.0001
    while abs(sym.N(f1.subs({t3: tv3, r3: rv3}))) > 0.0001 or
abs(sym.N(f2.subs({t3: tv3, r3: rv3}))) > 0.0001:
        delr3v = sym.N(result1[delr3].subs({t3:tv3, r3: rv3}))
        delt3v = sym.N(result1[delt3].subs({t3:tv3, r3: rv3}))
        rv3 += delr3v
        tv3 += delt3v

    #if the resulted values of r3 or t3 are not in the limits, it cannot
occur.
    if round(tv3, 4) > round(maxt3,4) or round(tv3,4) < round(mint3,4):
#the round() is used to avoid the wrong result from the calculation
above
        print(f"This input angle {10*i} degree cannot occur!")
        table.append([10*i,"N","N","N","N","N","N","N","N","N"])
        continue
    else:
        t2plot6[i-1] = np.float64(10*i)
        t3plot[i-1] = np.float64(tv3/sym.N(sym.pi)*180)
        r3plot[i-1] = np.float64(rv3*100)
        table.append([10*i])
#         print("rv3", rv3,"tv3:", tv3)
    #Loop 2, basicallly the same as above
    f3 = rv4 * sym.cos(tv3) + rv5 * sym.cos(t5) + rv6 * sym.cos(t6) +
rv7
    f4 = rv4 * sym.sin(tv3) + rv5 * sym.sin(t5) + rv6 * sym.sin(t6) -
rv8
```

```
    eq3 = sym.Eq(sym.diff(f3, t5) * delt5 + sym.diff(f3, t6) * delt6, -
f3)
    eq4 = sym.Eq(sym.diff(f4, t5) * delt5 + sym.diff(f4, t6) * delt6, -
f4)
    result2 = sym.solve([eq3, eq4], (delt5, delt6))
    delt5v = 0
    delt6v = 0
    while abs(sym.N(f3.subs({t5: tv5, t6: tv6}))) > 0.0001 or
abs(sym.N(f4.subs({t5: tv5, t6: tv6}))) > 0.0001:
        delt5v = sym.N(result2[delt5].subs({t5:tv5, t6: tv6}))
        delt6v = sym.N(result2[delt6].subs({t5:tv5, t6: tv6}))
        tv5 += delt5v
        tv6 += delt6v

        #let the angles yielded always be in 0 to 2*pi
        while tv5 < 0:
            tv5 += sym.N(2 * sym.pi)
        while tv6 < 0:
            tv6 += sym.N(2 * sym.pi)
        while tv5 > 2 * sym.pi:
            tv5 -= sym.N(2 * sym.pi)
        while tv6 > 2 * sym.pi:
            tv6 -= sym.N(2 * sym.pi)
    t5plot[i-1] = np.float64(tv5/sym.N(sym.pi)*180)
    t6plot[i-1] = np.float64(tv6/sym.N(sym.pi)*180)
    # print("tv5 = ", tv5, "tv6 = ", tv6)

#     print("tv5:",tv5,"tv6:", tv6)
#     print(f"position
D:({sym.N(rv4*sym.cos(tv3))},{sym.N(rv4*sym.sin(tv3)-rv1)})",
f"position
E:({sym.N(rv4*sym.cos(tv3)+rv5*sym.cos(tv5))},{sym.N(rv4*sym.sin(tv3)+r
v5*sym.sin(tv5)-rv1)})")
    dx = round(sym.N(rv4*sym.cos(tv3)),4)
    dy = round(sym.N(rv4*sym.sin(tv3)-rv1), 4)
    ex = round(sym.N(rv4*sym.cos(tv3)+rv5*sym.cos(tv5)), 4)
    ey = round(sym.N(rv4*sym.sin(tv3)+rv5*sym.sin(tv5)-rv1), 4)
    dxplot[i-1] = np.float64(dx)
```

```
    dyplot[i-1] = np.float64(dy)
    explot[i-1] = np.float64(ex)
    eyplot[i-1] = np.float64(ey)
    table[i-1].extend((dx, dy, ex, ey))
#velocity
    eq1v = sym.Eq(sym.N(-rv2 * sym.sin(tv2) * wv2) - v3 * sym.cos(tv3) +
rv3 * sym.sin(tv3) * w3, 0)
    eq2v = sym.Eq(sym.N(rv2 * sym.cos(tv2) * wv2) - v3 * sym.sin(tv3) -
rv3 * sym.cos(tv3) * w3, 0)
    resultv1 = sym.solve([eq1v,eq2v],(v3,w3))
    wv3 = resultv1[w3]
    vv3 = resultv1[v3]
    eq3v = sym.Eq(-rv4 * sym.sin(tv3) * wv3 - rv5 * sym.sin(tv5) * w5 -
sym.N(rv6 * sym.sin(tv6)) * w6, 0)
    eq4v = sym.Eq(rv4 * sym.cos(tv3) * wv3 + rv5 * sym.cos(tv5) * w5 +
rv6 * sym.N(sym.cos(tv6)) * w6, 0)
    resultv2 = sym.solve([eq3v,eq4v],(w5,w6))
    wv5 = resultv2[w5]
    wv6 = resultv2[w6]
#     print(f"Ve:({-rv4 * sym.sin(tv3) * wv3 - rv5 * sym.sin(tv5) *
wv5},{rv4 * sym.cos(tv3) * wv3 + rv5 * sym.cos(tv5) * wv5})")
    table[i-1].extend((round(-rv4 * sym.sin(tv3) * wv3 - rv5 *
sym.sin(tv5) * wv5,4), round(rv4 * sym.cos(tv3) * wv3 + rv5 *
sym.cos(tv5) * wv5,4)))
    if -rv4 * sym.sin(tv3) * wv3 - rv5 * sym.sin(tv5) * wv5 == 0:
        evxadd = 0.00001
        evxplot[i-1] = np.float64(evxadd)
    else:
        evxplot[i-1] = np.float64(round(-rv4 * sym.sin(tv3) * wv3 - rv5
* sym.sin(tv5) * wv5,4))
    if rv4 * sym.cos(tv3) * wv3 + rv5 * sym.cos(tv5) * wv5 == 0:
        evyadd = 0.00001
        evyplot[i-1] = np.float64(evyadd)
    else:
        evyplot[i-1] = np.float64(round(rv4 * sym.cos(tv3) * wv3 + rv5 *
sym.cos(tv5) * wv5,4))
#acceleration
```

```
    eq1a = sym.Eq(-rv2 * sym.N(sym.cos(tv2)) * wv2 ** 2 - rv2 *
sym.N(sym.sin(tv2)) * av2 - ar3 * sym.cos(tv3) + 2 * (vv3 *
sym.N(sym.sin(tv3)) * wv3) + rv3 * sym.N(sym.cos(tv3)) * wv3 ** 2 + rv3
* sym.sin(tv3) * at3, 0)
    eq2a = sym.Eq(-rv2 * sym.N(sym.sin(tv2)) * wv2 ** 2 + rv2 *
sym.N(sym.cos(tv2)) * av2 - ar3 * sym.sin(tv3) - 2 * (vv3 *
sym.N(sym.cos(tv3)) * wv3) + rv3 * sym.N(sym.sin(tv3)) * wv3 ** 2 - rv3
* sym.cos(tv3) * at3, 0)
    resulta1 = sym.solve([eq1a,eq2a],(ar3,at3))
    atv3 = resulta1[at3]
    arv3 = resulta1[ar3]
    eq3a = sym.Eq(-rv4 * sym.cos(tv3) * wv3 ** 2 - rv4 * sym.sin(tv3) *
atv3 - rv5 * sym.cos(tv5) * wv5 ** 2 - rv5 * sym.sin(tv5) * a5 - rv6 *
sym.cos(tv6) * wv6 ** 2 - rv6 * sym.sin(tv6) * a6, 0)
    eq4a = sym.Eq(-rv4 * sym.sin(tv3) * wv3 ** 2 + rv4 * sym.cos(tv3) *
atv3 - rv5 * sym.sin(tv5) * wv5 ** 2 - rv5 * sym.cos(tv5) * a5 - rv6 *
sym.sin(tv6) * wv6 ** 2 + rv6 * sym.cos(tv6) * a6, 0)
    resultav2 = sym.solve([eq3a,eq4a],(a5,a6))
    av5 = resultav2[a5]
    av6 = resultav2[a6]
#     print(f"Ae:({-rv4 * sym.cos(tv3) * wv3 ** 2 - rv4 * sym.sin(tv3) *
atv3 - rv5 * sym.cos(tv5) * wv5 ** 2 - rv5 * sym.sin(tv5) * av5},{-rv4
* sym.sin(tv3) * wv3 ** 2 + rv4 * sym.cos(tv3) * atv3 - rv5 *
sym.sin(tv5) * wv5 ** 2 - rv5 * sym.cos(tv5) * av5})")
    table[i-1].extend((round(-rv4 * sym.cos(tv3) * wv3 ** 2 - rv4 *
sym.sin(tv3) * atv3 - rv5 * sym.cos(tv5) * wv5 ** 2 - rv5 *
sym.sin(tv5) * av5,4),round( -rv4 * sym.sin(tv3) * wv3 ** 2 + rv4 *
sym.cos(tv3) * atv3 - rv5 * sym.sin(tv5) * wv5 ** 2 - rv5 *
sym.cos(tv5) * av5, 4)))
    table[i-1].append(round(wv6, 4))
    eaxplot[i-1] = np.float64(round(-rv4 * sym.cos(tv3) * wv3 ** 2 - rv4
* sym.sin(tv3) * atv3 - rv5 * sym.cos(tv5) * wv5 ** 2 - rv5 *
sym.sin(tv5) * av5,4))
    eayplot[i-1] = np.float64(round( -rv4 * sym.sin(tv3) * wv3 ** 2 +
rv4 * sym.cos(tv3) * atv3 - rv5 * sym.sin(tv5) * wv5 ** 2 - rv5 *
sym.cos(tv5) * av5, 4))
    # print("seven")
t2plot6 = t2plot6[t2plot6 != 0.00001]
```

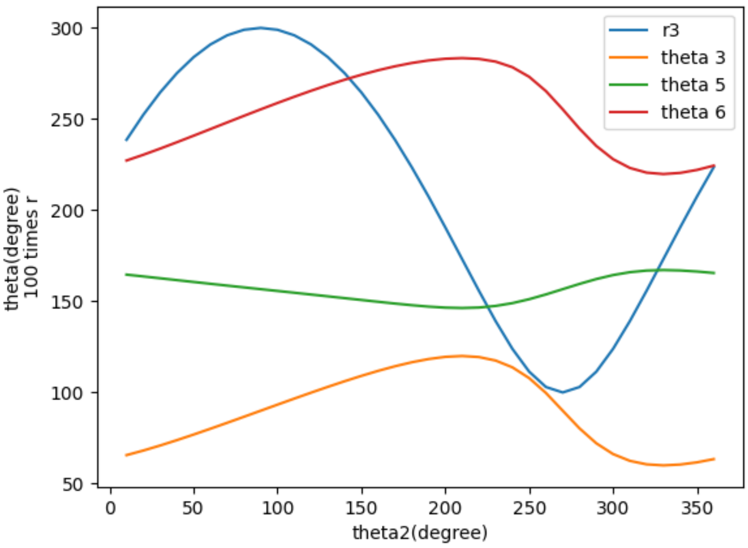
```python
t3plot = t3plot[t3plot != 0.00001]
r3plot = r3plot[r3plot != 0.00001]
t5plot = t5plot[t5plot != 0.00001]
t6plot = t6plot[t6plot != 0.00001]
dxplot = dxplot[dxplot != 0.00001]
dyplot = dyplot[dyplot != 0.00001]
explot = explot[explot != 0.00001]
eyplot = eyplot[eyplot != 0.00001]
evxplot = evxplot[evxplot != 0.00001]
evyplot = evyplot[evyplot != 0.00001]
eaxplot = eaxplot[eaxplot != 0.00001]
eayplot = eayplot[eayplot != 0.00001]
if which == 1:
    plt.plot(t2plot6, r3plot, label = "r3")
    plt.plot(t2plot6, t3plot, label = "theta 3")
    plt.plot(t2plot6, t5plot, label = "theta 5")
    plt.plot(t2plot6, t6plot, label = "theta 6")
    plt.xlabel("theta2(degree)")
    plt.ylabel("theta(degree)\n100 times r")
    plt.legend()
    plt.show()
else:
    fig = plt.figure()
    ax = fig.add_subplot(projection = "3d")
    if which == 2:
        ax.plot(t2plot6, dxplot, dyplot, label='dp')
        ax.plot(t2plot6, explot, eyplot, label='ep')
    elif which == 3:
        ax.plot(t2plot6, evxplot, evyplot, label='ev')
    else:
        ax.plot(t2plot6, eaxplot, eayplot, label='ea')
    ax.plot
    ax.set_xlabel("theta2(degree)")
    ax.set_ylabel("x")
    ax.set_zlabel("y")
    ax.legend()
    plt.show()
```

```
print(tabulate(table, headers = ["theta2","Dpx","Dpy", "Epx", "Epy",
"Evx", "Evy", "Eax", "Eay", "w6"],floatfmt=".4f"))
```

Plots and tables

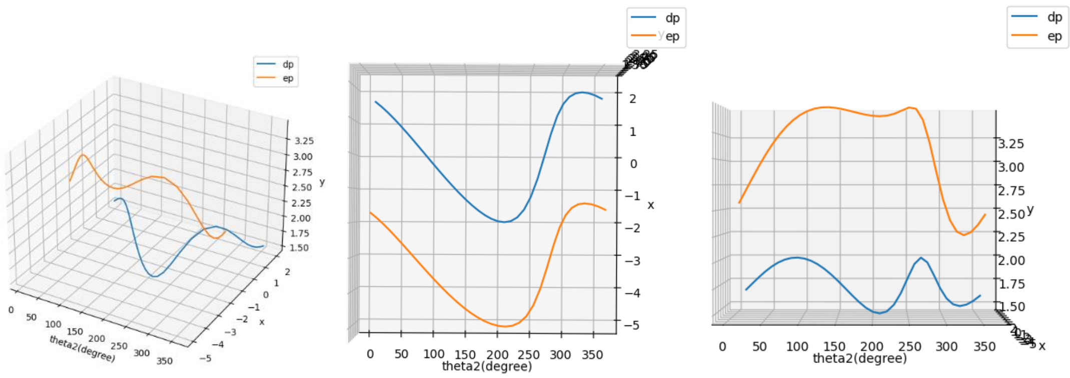
The following results are calculated from the parameters rv2, rv1, rv4, rv5, rv6, rv7 equal to 1, 2, 4, 3.5, 3.5, 4.1 respectively.

1. angular position plots

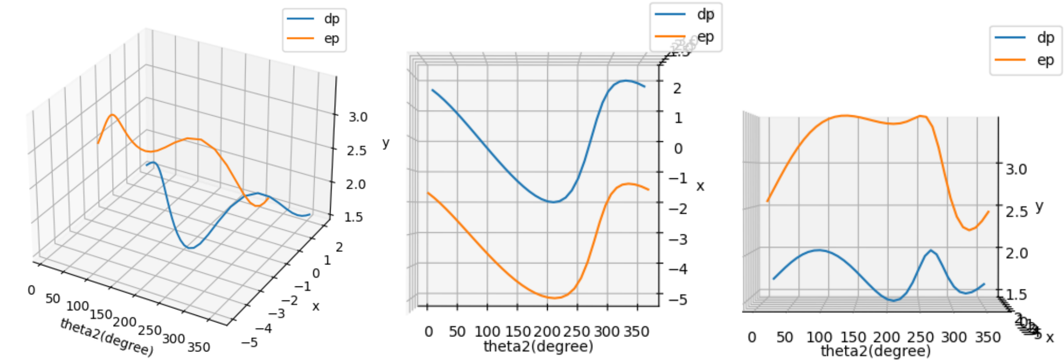


2. Position D and E

(a) w2 = 10, a2 = 0

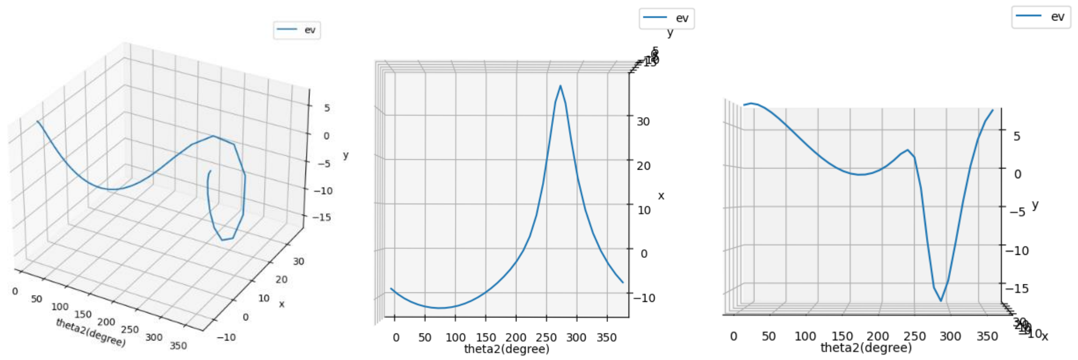


(b) w2 = 0, a2 = 2

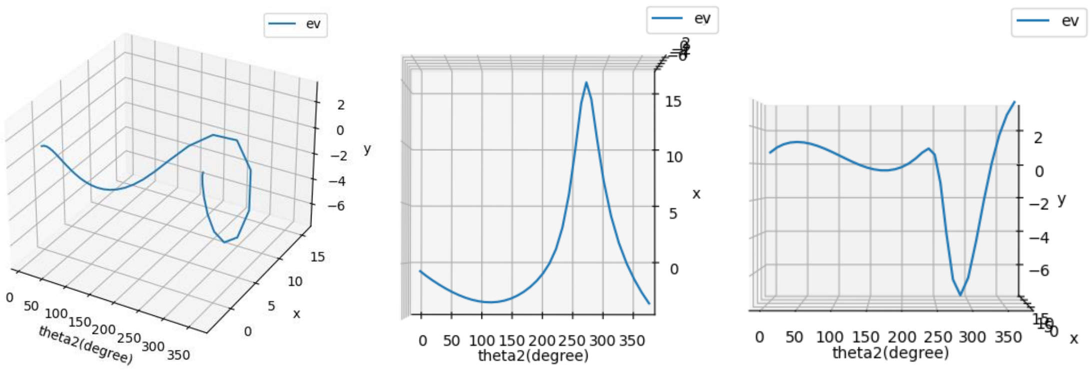


3. velocity plots

(a) w2 = 10, a2 = 0

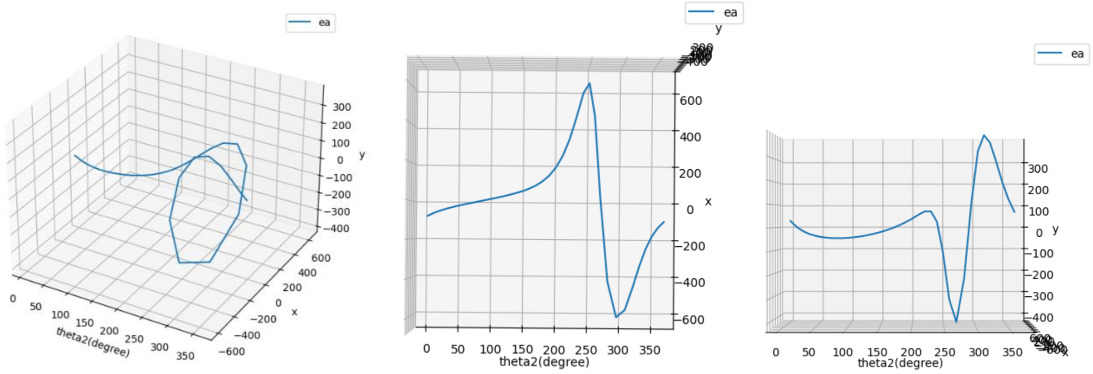


(b) w2 = 0, a2 = 2

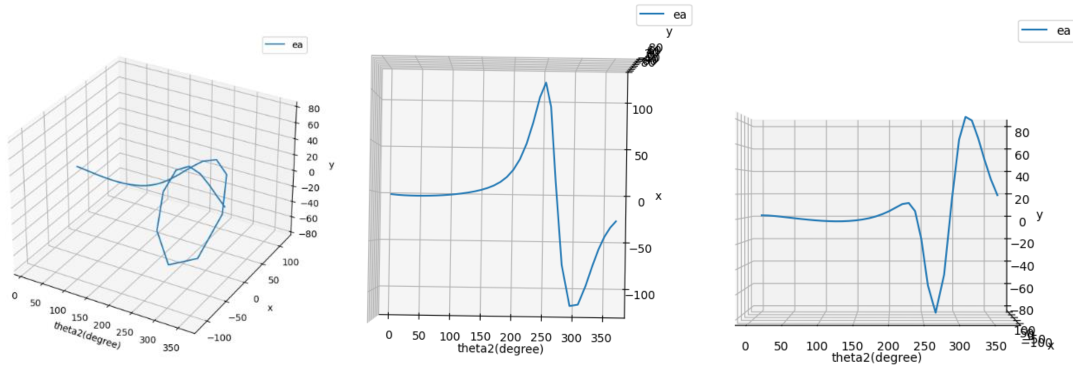


4. acceleration plots

(a) w2 = 10, a2 = 0



(2) w2 = 0, a2 = 2



5. Tables

Theta2 is the input angle in degree, Dpx is the position of D in the x direction, v indicates velocity, a represents acceleration, and w6 is the angular velocity of link 6.

(a) w2 = 10, a2 = 0

theta2	Dpx	Dpy	Epx	Epy	Evx	Evy	Eax	Eay	w6
10	1.6507	1.6435	-1.7244	2.5703	-7.7305	7.1449	-78.0727	29.0471	3.0076
20	1.4895	1.7123	-1.8691	2.6969	-8.8307	7.3046	-61.0041	1.7621	3.2744
30	1.3093	1.7796	-2.0315	2.8233	-9.7387	7.1351	-47.7633	-16.6298	3.4494
40	1.1136	1.8419	-2.2081	2.9446	-10.4788	6.7325	-36.9631	-28.9348	3.5586

50	0.9054	1.8962	-2.3963	3.0574	-11.0651	6.1658	-27.6940	-37.0490	3.6192
60	0.6875	1.9405	-2.5935	3.1592	-11.5055	5.4864	-19.3654	-42.1952	3.6419
70	0.4623	1.9732	-2.7971	3.2485	-11.8039	4.7342	-11.5936	-45.1414	3.6337
80	0.2323	1.9932	-3.0047	3.3242	-11.9624	3.9413	-4.1241	-46.3615	3.5986
90	0.0000	2.0000	-3.2139	3.3860	-11.9812	3.1355	3.2221	-46.1418	3.5385
100	-0.2323	1.9932	-3.4221	3.4337	-11.8598	2.3412	10.5878	-44.6491	3.4539
110	-0.4623	1.9732	-3.6271	3.4679	-11.5969	1.5816	18.1124	-41.9725	3.3441
120	-0.6875	1.9405	-3.8261	3.4893	-11.1908	0.8783	25.9625	-38.1498	3.2072
130	-0.9054	1.8962	-4.0168	3.4990	-10.6389	0.2528	34.3682	-33.1833	3.0406
140	-1.1136	1.8419	-4.1966	3.4987	-9.9387	-0.2745	43.6714	-27.0483	2.8407
150	-1.3093	1.7796	-4.3629	3.4901	-9.0869	-0.6845	54.3952	-19.6959	2.6036
160	-1.4895	1.7123	-4.5129	3.4756	-8.0799	-0.9600	67.3470	-11.0479	2.3248
170	-1.6508	1.6435	-4.6440	3.4575	-6.9125	-1.0876	83.7710	-0.9818	1.9993
180	-1.7889	1.5777	-4.7532	3.4385	-5.5744	-1.0590	105.5715	10.6926	1.6212
190	-1.8985	1.5208	-4.8374	3.4214	-4.0392	-0.8706	135.6193	24.2404	1.1806
200	-1.9723	1.4799	-4.8927	3.4091	-2.2376	-0.5203	178.1233	39.8695	0.6564
210	-2.0000	1.4641	-4.9130	3.4043	0.0000	0.0000	238.9068	57.0560	0.0000
220	-1.9661	1.4834	-4.8881	3.4102	3.0336	0.7010	324.8330	72.2242	-0.8896
230	-1.8480	1.5475	-4.7989	3.4295	7.4844	1.5253	439.5294	72.5636	-2.1824
240	-1.6137	1.6600	-4.6142	3.4620	14.0822	2.0917	566.7430	25.6345	-4.0676
250	-1.2277	1.8069	-4.2940	3.4946	22.8850	1.2705	629.6027	-115.3755	-6.5487
260	-0.6744	1.9427	-3.8146	3.4883	31.7277	-2.5954	470.4578	-328.9906	-9.0953
270	0.0003	2.0000	-3.2136	3.3859	35.9443	-9.4100	28.8683	-415.2913	-10.6159
280	0.6742	1.9428	-2.6055	3.1649	32.6008	-15.3941	-419.2092	-212.7398	-10.3008
290	1.2280	1.8068	-2.1049	2.8757	24.3442	-16.8902	-593.6519	106.5869	-8.4656
300	1.6138	1.6600	-1.7575	2.6005	15.6561	-14.1030	-546.5235	321.5687	-6.0204

310	1.8479	1.5476	-1.5479	2.3952	8.7046	-9.2747	-431.2887	391.9879	-3.6342
320	1.9661	1.4834	-1.4426	2.2777	3.6438	-4.2513	-323.1520	363.2612	-1.5998
330	2.0000	1.4641	-1.4124	2.2421	0.0000	0.0000	-238.9857	286.4679	0.0000
340	1.9723	1.4799	-1.4370	2.2713	-2.6928	3.1572	-177.1524	200.1211	1.1856
350	1.8985	1.5208	-1.5028	2.3462	-4.7578	5.2669	-132.7419	125.4718	2.0279
360	1.7889	1.5777	-1.6007	2.4502	-6.3968	6.5252	-101.0101	68.9594	2.6108

(b) w2 = 0, a2 = 2

theta2	Dpx	Dpy	Epx	Epy	Evx	Evy	Eax	Eay	w6
--------	-------	------	-------	------	-------	-------	---------	--------	-------
10	1.6507	1.6435	-1.7244	2.5703	-0.6459	0.5970	-2.5680	2.0725	0.2513
20	1.4895	1.7123	-1.8691	2.6969	-1.0435	0.8631	-3.1389	1.9165	0.3869
30	1.3093	1.7796	-2.0315	2.8233	-1.4094	1.0326	-3.4981	1.4817	0.4992
40	1.1136	1.8419	-2.2081	2.9446	-1.7511	1.1251	-3.6969	0.9040	0.5947
50	0.9054	1.8962	-2.3963	3.0574	-2.0673	1.1520	-3.7611	0.2639	0.6762
60	0.6875	1.9405	-2.5935	3.1592	-2.3548	1.1229	-3.7022	-0.3888	0.7454
70	0.4623	1.9732	-2.7971	3.2485	-2.6094	1.0466	-3.5241	-1.0199	0.8033
80	0.2323	1.9932	-3.0047	3.3242	-2.8270	0.9314	-3.2258	-1.6024	0.8504
90	0.0000	2.0000	-3.2139	3.3860	-3.0032	0.7859	-2.8035	-2.1125	0.8870
100	-0.2323	1.9932	-3.4221	3.4337	-3.1336	0.6186	-2.2501	-2.5270	0.9126
110	-0.4623	1.9732	-3.6271	3.4679	-3.2137	0.4383	-1.5547	-2.8215	0.9267
120	-0.6875	1.9405	-3.8261	3.4893	-3.2391	0.2542	-0.6991	-2.9704	0.9283
130	-0.9054	1.8962	-4.0168	3.4990	-3.2051	0.0762	0.3455	-2.9457	0.9160
140	-1.1136	1.8419	-4.1966	3.4987	-3.1072	-0.0858	1.6267	-2.7166	0.8881
150	-1.3093	1.7796	-4.3629	3.4901	-2.9406	-0.2215	3.2218	-2.2489	0.8425
160	-1.4895	1.7123	-4.5129	3.4756	-2.7004	-0.3208	5.2568	-1.5033	0.7770
170	-1.6508	1.6435	-4.6440	3.4575	-2.3814	-0.3747	7.9360	-0.4322	0.6888

180	-1.7889	1.5777	-4.7532	3.4385	-1.9761	-0.3754	11.5866	1.0245	0.5747
190	-1.8985	1.5208	-4.8374	3.4214	-1.4711	-0.3171	16.7256	2.9430	0.4300
200	-1.9723	1.4799	-4.8927	3.4091	-0.8361	-0.1944	24.1493	5.3991	0.2453
210	-2.0000	1.4641	-4.9130	3.4043	0.0000	0.0000	35.0256	8.3648	0.0000
220	-1.9661	1.4834	-4.8881	3.4102	1.1889	0.2747	50.8661	11.3182	-0.3486
230	-1.8480	1.5475	-4.7989	3.4295	2.9991	0.6112	72.8746	12.1201	-0.8745
240	-1.6137	1.6600	-4.6142	3.4620	5.7643	0.8562	99.0103	4.8969	-1.6650
250	-1.2277	1.8069	-4.2940	3.4946	9.5607	0.5308	116.0505	-19.7946	-2.7358
260	-0.6744	1.9427	-3.8146	3.4883	13.5174	-1.1058	93.5373	-60.3826	-3.8750
270	0.0003	2.0000	-3.2136	3.3859	15.6056	-4.0855	14.4594	-80.6414	-4.6090
280	0.6742	1.9428	-2.6055	3.1649	14.4137	-6.8062	-73.7557	-45.4530	-4.5542
290	1.2280	1.8068	-2.1049	2.8757	10.9538	-7.5998	-113.9696	17.2638	-3.8091
300	1.6138	1.6600	-1.7575	2.6005	7.1649	-6.4542	-110.3769	63.6679	-2.7552
310	1.8479	1.5476	-1.5479	2.3952	4.0495	-4.3147	-91.0283	82.3714	-1.6907
320	1.9661	1.4834	-1.4426	2.2777	1.7223	-2.0094	-71.2158	80.0134	-0.7561
330	2.0000	1.4641	-1.4124	2.2421	0.0000	0.0000	-55.0583	65.9975	0.0000
340	1.9723	1.4799	-1.4370	2.2713	-1.3119	1.5382	-42.7722	48.3487	0.5776
350	1.8985	1.5208	-1.5028	2.3462	-2.3519	2.6035	-33.7036	32.0629	1.0024
360	1.7889	1.5777	-1.6007	2.4502	-3.2069	3.2712	-27.0775	19.0562	1.3088

Discussion

From the position figure, we can see in the span of 360 input degree all variables begin and ends in the same value, which is expected. In the first set of input parameters, since the angular velocity of the input link is zero, the motion should be continuous. This is shown in the velocity and acceleration curves of the point E: the starting and ending points of the curve is of the same value; whereas in the second set of parameters, the two points of both curves do not have the same values. In the velocity figures, the cyclic phenomenon is absent in the non-zero acceleration condition. As the angular velocity won't be the same after one revolution, the velocity of the points D and E will be a function of the input angular velocity. It is the same for the acceleration plots.

The curve of angles t3 and t6 are steeper in the right side of the figure. These two links both have one end anchored to the ground link. Combine with the steeper curve in the D & E position plot, and higher velocity of the latter half rotation, this means this is a quick return mechanism. The velocity in both x and y direction peaked around 270 degree, while the velocity in x is in the positive direction and negative in y.

The derivative relations of the position, velocity, acceleration curves are also evident.

Bonus

1. The range of input link motion

Whether the input link could perform a whole rotation depends on the link geometries. Basically, the problem is that the links in Loop 2 could pose a constraint on the range of motion of link 3. If the range of motion of link 3 is limited, then it would not allow link 2 to have a full rotation.

We can find the range which link 3 "want", which is the minimum range for full rotation of link 2:

$$\theta 3\mathrm{max}1 = sin^{-1}(\frac{r2}{r1}) + 90$$

and

$$\theta 3\mathrm{min}1 = -sin^{-1}(\frac{r2}{r1}) + 90$$

The limitations given by Loop 2:

$$\theta 3max2 = 180 - tan^{-1}(\frac{r1}{r7}) - cos^{-1}(\frac{r1^2 + r7^2 + r4^2 - (r5 - r6)^2}{2 \times r4 \times \sqrt{r1^2 + r7^2}})$$

$$\theta 3min2 = 180 - tan^{-1}(\frac{r1}{r7}) - cos^{-1}(\frac{r1^2 + r7^2 + r4^2 - (r5 + r6)^2}{2 \times r4 \times \sqrt{r1^2 + r7^2}})$$

If $\theta 3max1 < \theta 3max2$ and $\theta 3min1 > \theta 3min2$, then link 2 will have a full rotation. This condition is implemented in the code.

The above is a fool-proved way to find which input angles are possible given the parameters. But what about some examples? We can vary the length of r5 and r6 to yield three conditions that the input link in the mechanism cannot fully rotate.

Condition 1: When the parameters rv2, rv1, rv4, rv5, rv6, rv7 equal to 1, 2, 4, 1.5, 1.5, 4.1 respectively, $\theta 3min1 < \theta 3min2$. This means the input link will be constrained so that r2 cannot be perpendicular to r3 in the right side (which is a necessary condition for a full rotation).

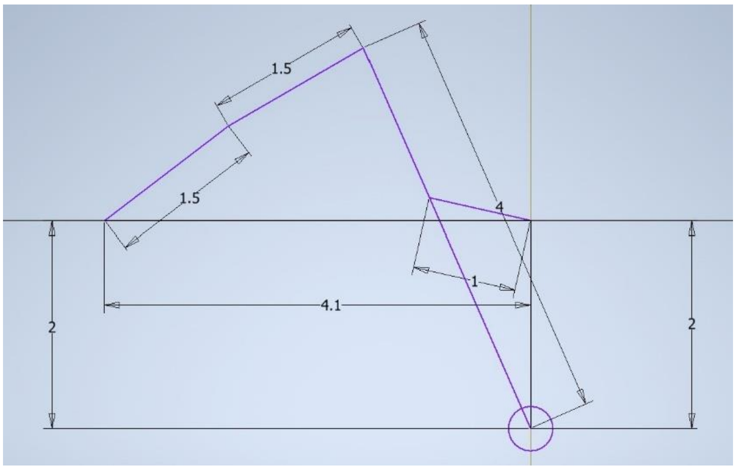


Figure 2

Condition 2: When the parameters rv2, rv1, rv4, rv5, rv6, rv7 equal to 1, 2, 4, 4.5, 7.5, 4.1 respectively, $\theta 3max1 > \theta 3max2$. This means the input link will be constrained so that r2 cannot be perpendicular to r3 in the left side (which is a necessary condition for a full rotation). The possible motion of this mechanism is thus separated into to two ranges.

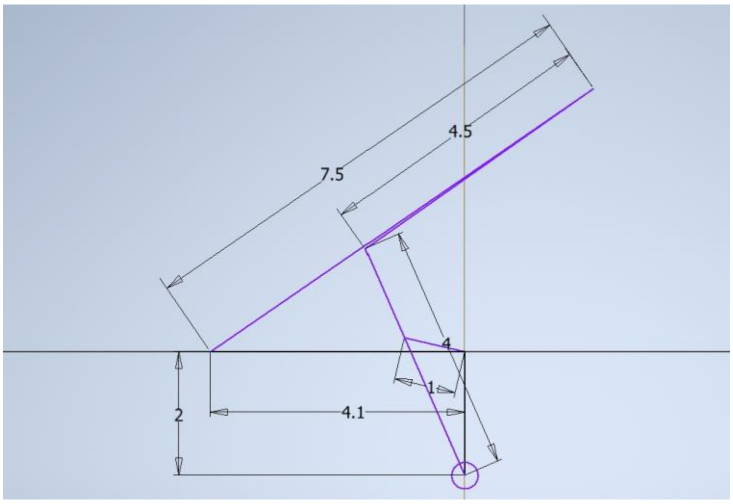


Figure 3

Condition 3: When the parameters rv2, rv1, rv4, rv5, rv6, rv7 equal to 1, 2, 4, 1.5, 4.5, 4.1 respectively, $\theta 3min1 < \theta 3min2$ and $\theta 3max1 > \theta 3max2$. This means the input link will be constrained so that r2 cannot be perpendicular to r3 in both right and right sides (which is a necessary condition for a full rotation). The possible motion of this mechanism is thus separated into to two ranges.

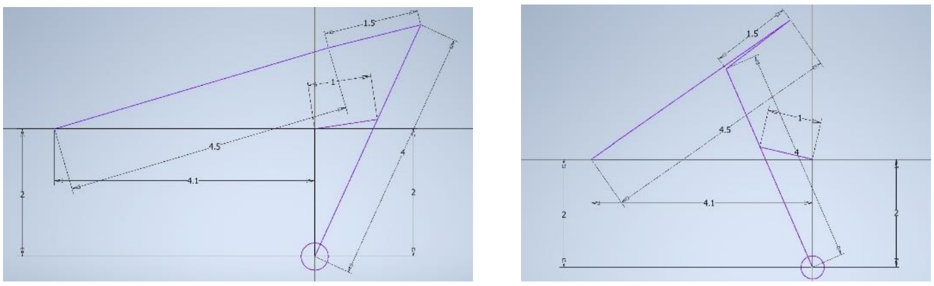


Figure 4, 5

From these examples we can get a feeling of what kinds of length of r5 and r6 will result in which kind of conditions that may be prohibit the input link to rotate: if the summation of the length of r5 and r6 is too small, condition 1 will occur; when the difference of the length r5 and r6 is too large, then will lead to condition 2; if both, will result in condition 3. These findings are also consistent with what the constraint equations suggest.

Note that which condition will happen does not wholly depend on the length of r5 and r6, as seen in the equations. Easier method to discriminate without much calculation is left for readers to derive.

2. A note on initial values using Newton's method

Newton's method, also called Newton–Raphson method, is a root-finding algorithm used in numerical analysis. The basic form is the following:

$$\mathrm{X_{n+1}} = \mathrm{X_n} - \frac{f(X_n)}{f'(X_n)}$$

The rationale is using the first order Taylor's expansion to get a better estimation of the root of the function, until the criterion is meet.

There are several types of problems that may appear to obstruct the convergence of the calculation [1]. Including the derivative of the function cannot be found, the function is complex, the initial guess is not local enough to the true root, cyclic calculation that cannot converge due to the nature of the function, and so on.

The problems I encountered in this linkage analysis is that the wrong roots being yielded. I set the all the initial values to 1.0 at first, including r3, t3, t5, and t6. The calculation in Loop 1 worked out as a dream, but not the same for the calculation in Loop 2. This may due to the fact that it is lucky that the geometry is happen to be close to the guesses, and that the angle of input link and the position of r3 has a one-to-one relation, which provide only one possible root.

The case with t5 and t6 are the opposite. r5 and r6 can almost always have two configurations, one peaks upward and one downward. So the issue is how to find the one that peaks upward?

The first method I tried is to set the values of t5 and t6 in 90 degrees increment. Varying both in 360 degrees. The result is a disaster, multiple roots are found according to the algorithm, while the accuracy is very low. This is clearly not a good way to set the initial values, let alone the efficiency of this algorithm.

The second and the last method I used is to perform a calculation for the geometry when the input link is at 0 degree. First find the angles in triangle DEF when input link is at zero degree. There are two triangles, to solve the problem, I intentionally choose the upper one. Calibrate the result by angle DFA, a precise angle of r5 and r6 is yielded. However, some parameters do not allow the mechanism to exist if the input link is at 0 degree. In these instances, set the initial values of r5 and r6 close to the results calculated when r5 and r6 is in a sequential line (as shown in figure 2), and set t6 to be larger than r6 to ensure it peaks upward in the following iteration. This way we can ensure the configuration is always the desired one.

Good initial guess is crucial when using Newton's method. Even when the problem is more difficult that the precise initial values cannot be attained, using maximum likelihood is a must to have good initial values.

Reference

[1] (2022). Retrieved 22 November 2022, from
http://www.cas.mcmaster.ca/~cs4te3/notes/newtons_method.pdf