DEPARTMENT OF MECHANICAL ENGINEERING

INDIAN INSTITUTE OF ENGINEERING, SCIENCE AND TECHNOLOGY, SHIBPUR

HOWRAH-711103

Project on:-

Numerical Position Analysis of a Quick Return Mechanism and its Simulation

Given by: Prof. Shyamal Chatterjee

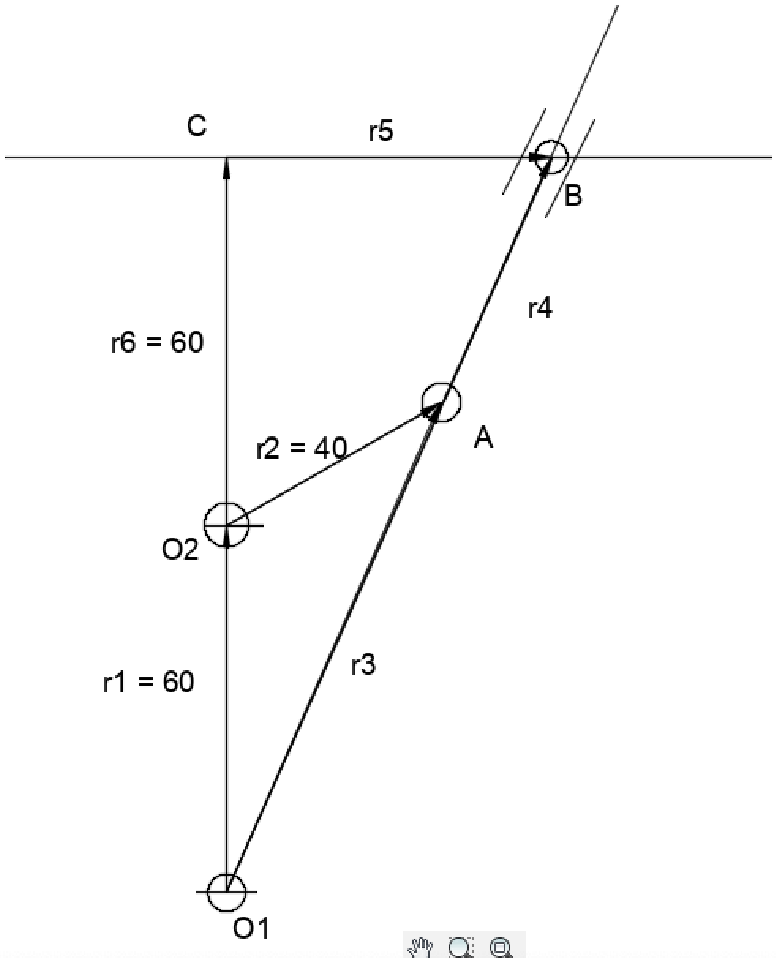
Submitted by

Ankan Man (Enroll. No. – 511019014 – 4rd Sem)

Date of submission : 14th May, 2021

Problem Statement:-

- **For a Quick Return Mechanism calculate the unknowns taking appropriate lengths for the const. length links and simulate it using Newton Raphson method. Compare results with analytical solutions.**



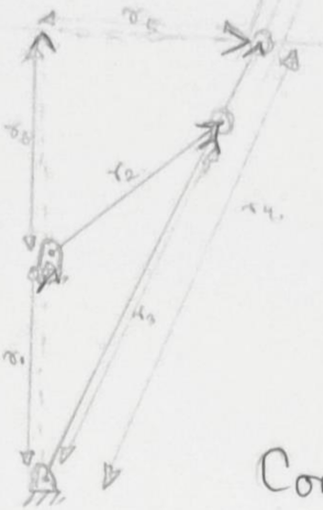
r1 = O2O1 r2 = O2A r3 = O1A r4 = O1B

r5 = CB r6 = O2C o2 = angle of r2 with horizontal

o3 = angle of r3 with horizontal

Analytical Solutions

Analytical Solution $\Rightarrow \theta_2 = 30^\circ$ (taken)



According to 3RIP
Grashof Condition.

i) $r_2 < r_1$

ii) $r_1 + r_2 < r_1 + r_6$

$\Rightarrow r_2 < r_6$

Considering the above inequalities we take:

$r_2 = 40mm \qquad r_1 = 60mm$

$r_6 = 60\ mm$

$\theta_2 = 30^\circ$

Required to find: r_3, r_4, r_5, θ_3

Loop Closure equation

$$\vec{r_1} + \vec{r_2} - \vec{r_3} = \vec{0}$$

$$\vec{r_1} + \vec{r_6} + \vec{r_5} - \vec{r_4} = \vec{0}$$

$$\theta_3 = \theta_4$$

$\therefore \quad r_1 + r_2 \sin\theta_2 = r_3 \sin\theta_3 \quad \text{-----} \quad (I)$

$r_2 \cos\theta_2 = r_3 \cos\theta_3 \quad \text{-----} \quad (II)$

$r_1 + r_6 = r_4 \sin\theta_3 \quad \left[\because \theta_3 = \theta_4\right] \quad \text{---} \quad (III)$

$r_5 = r_4 \cos\theta_3 \quad \text{---} \quad (IV)$

$(I)^2 + (II)^2$

$$(r_1 + r_2 \sin\theta_2)^2 + (r_2 \cos\theta_2)^2 = r_3^2$$

$$\Rightarrow r_1^2 + r_2^2 + 2 r_1 r_2 \sin\theta_2 = r_3^2$$

$$\Rightarrow r_3 = (r_1^2 + r_2^2 + 2 r_1 r_2 \sin\theta_2)^{1/2} \therefore \text{(Ans)} \quad r_3 = 87.17$$

Now:-

$(I)/(II)$

$$\tan\theta_3 = \frac{r_1 + r_2 \sin\theta_2}{r_2 \cos\theta_2}$$

for $\theta_2 \in [-\pi/2, \pi/2]$

$$\theta_3 = \tan^{-1}\left(\left|\frac{r_1 + r_2 \sin\theta_2}{r_2 \cos\theta_2}\right|\right) \Rightarrow \theta_3 = 66.58 \text{ (Ans)}$$

$\theta_2 \in [\pi/2, 3\pi/2]$

$$\theta_3 = \pi - \tan^{-1}\left(\left|\frac{r_1 + r_2 \sin\theta_2}{r_2 \cos\theta_2}\right|\right)$$

Now

$$\sin\theta_3 = \frac{r_1 + r_2 \sin\theta_2}{r_3}$$

From (III)

$$r_4 = \frac{r_1 + r_6}{\sin\theta_3} = \frac{(r_1 + r_6)\, r_3}{(r_1 + r_2 \sin\theta_3)}$$

$$= 130.75 \text{ (Ans)}$$

From (IV)

$$r_5 = r_4 \cos\theta_3$$

$$= \frac{(r_1 + r_6)\, \cancel{r_3}}{(r_1 + r_2 \sin\theta_2} \times \frac{r_2 \cos\theta_2}{\cancel{r_3}}$$

$$\cos\theta_3 = \frac{r_2 \cos\theta_2}{r_3}$$

$$\therefore r_5 = \frac{r_2 (r_1 + r_6)\cos\theta_2}{r_1 + r_2 \sin\theta_2}$$

$$= 51.96 \text{ (Ans)}$$

$\therefore$ At $\theta_2 = 30^\circ$

$r_3 = 87.17\text{mm}$ $r_4 = 130.75 \text{ mm}$ $r_5 = 51.96 \text{ mm}$

$\theta_3 = 66.58^\circ$

So, the values of the unknowns at $\theta_2 = 30^o$ are :

r3 = 87.17

r4 = 130.75

r5 = 51.96

$\theta_3 = 66.58^o$

Numerical Solution

- **I have used Python as the programing Language.**
- **The platform used is Jupiter Notebook.**
- **Concept : We have used multivariable Newton Raphson algorithm to determine the roots [4 unknowns] from the 4 non linear simultaneous equations. The Crank angle θ_2 is varied from 0 to 360 degrees and the links are plotted using matplotlib library. The plots were saved as image file and then the animation was created by successive playing of the images.**
- **Libraries used**
    - **NumPy as np (use pip install numpy to install it)**
    - **Jacobian from numdifftools.nd_algopy (use pip install numdifftools to install it)**
    - **Math**
    - **Matplotlib.pyplot (use pip install matplotlib to install it)**
- The initial values are taken as follows as per Grashof criteria :
    - R1 = 60 mm
    - R2 = 40 mm
    - R6 = 60 mm
    - Θ_2 = varied for getting animation [value of 30 degrees considered for analytical verification]

- ## Comparing with the Analytical Solution

```
In [*]: #Itering through theta to find all solutions
for i in range(360):
    o2 = np.radians(i)
    x_new = root_finder()
    #co-ordinates
    o3 = x_new[3][0]
    r4 = x_new[1][0]
    s0 = [0,0]
    s1 = [0,r1]
    s2 = [s1[0]+r2*math.cos(o2),s1[1]+r2*math.sin(o2)]
    s3 = [r4*math.cos(o3),r4*math.sin(o3)]
    if i ==30 :
        print(x_new)
    clf()
    ct = ploter(ct)
```

```
<ipython-input-5-e7567b2c9fca>:12: RuntimeWarning: More than 20 figures have been opened. Figures created through t
he pyplot interface (`matplotlib.pyplot.figure`) are retained until explicitly closed and may consume too much memo
ry. (To control this warning, see the rcParam `figure.max_open_warning`).
  figure()
[[ 87.17797887]
 [130.76696831]
 [ 51.96152423]
 [ 13.72852909]]
```

It can be seen that the values are nearly same as the analytical one.

- r3 = 87.1779
- r4 = 130.766
- r5 = 51.961
- $\theta_3 = 66.58^o$

The analytical Solution gives the values at $\theta_2 = 30^o$ as :

r3 = 87.17

r4 = 130.75

r5 = 51.96

$\theta_3 = 66.58^o$

The Jupiter Notebook Solution :

Ankan/Simulation Of Natural Processes/ | Assignment 2 KMR - Jupyter Notebook

jupyter Assignment 2 KMR Last Checkpoint: Yesterday at 00:21 (unsaved changes) Logout

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

In [1]:
```python
import numpy as np # Since we will define the equation variables and coefficients in array form
from numdifftools.nd_algopy import Jacobian # For use of grad operator and finding jacobian of a matrix with ease
import math # for using mathematical funcs
from matplotlib.pyplot import * #for plotting the links for animation
```

In [2]:
```python
ct = 0
# Given data [taken as per Student's wish as Instructed]
r1 = 60
o2 = math.radians(0) #varries
r2 = 40
r6 = 60
```

In [3]:
```python
# Defining Func in Python
#r3,r4,r5,o3 = 87.17,130.75,51.96,math.radians(66.58) for theta = 30 degrees
eqn = []
eq1 = lambda x : 40*np.sin(o2)-x[0]*np.sin(x[3])+60
eqn.append(eq1)
eq2 = lambda x : 40*np.cos(o2)-x[0]*np.cos(x[3])
eqn.append(eq2)
eq3 = lambda x : 60-x[1]*np.sin(x[3])+60
eqn.append(eq3)
eq4 = lambda x : x[2]-x[1]*np.cos(x[3])
eqn.append(eq4)
```

In [4]:
```python
jacob1 = Jacobian(eq1)
jacob2 = Jacobian(eq2)
jacob3 = Jacobian(eq3)
jacob4 = Jacobian(eq4)
```

- **The necessary libraries are inputted.**
- **The given lengths are initialized.**
- **Ct variable used for naming the pictures created.**
- **The equations have been initialized using lambda functions.**
- **The Jacobians have been found out using the Jacobian function.**

In [5]:
```python
#plot
def ploter(ct):
    #Naming file
    if ct<10 :
        filename = 'QuickReturn00'+str(ct)+'.jpg'
    elif ct<100 :
        filename = 'QuickReturn0'+str(ct)+'.jpg'
    else :
        filename = 'QuickReturn'+str(ct)+'.jpg'

    #Plotting
    figure()
    xlim(-200, 200)
    ylim(0, 125)
    plot((s0[0],s3[0]),(s0[1],s3[1]),linewidth='4')
    plot((s1[0],s2[0]),(s1[1],s2[1]),linewidth='4')
    plot((-200,200),(120,120),linestyle='dashed')
    plot((0,0),(0,120),linestyle='dashed')
    savefig(filename)
    ct=ct+1
    return ct
```

- **We define the ploter function to plot the links 2 and 4 and describe its motion.**
- **We use matplotlib for the purpose.**
- **We limit the axis using x_lim and y_lim.**
- **And the plot function for plotting the link with minor customizations.**

```
In [6]: # Newton Rampson Method for Multivariable systems of Non-Linear equation to be implemented

        def root_finder():
            i = 0
            er = 100
            tol = 0.000001
            maxiter = 100
            M = 4
            N = 4
            x0 = np.array([1,1,1,1],dtype=float).reshape(N,1)
            while np.any(abs(er)>tol) and i< maxiter:
                func_eval = np.array([eq1(x0),eq2(x0),eq3(x0),eq4(x0)]).reshape(M,1)
                flat_x0 = x0.flatten()
                jac = np.array([jacob1(flat_x0),jacob2(flat_x0),jacob3(flat_x0),jacob4(flat_x0)])
                jac = jac.reshape(N,M)
                x_new = x0 - np.linalg.inv(jac)@func_eval
                er = x_new - x0
                x0 = x_new
                i+=1
            return x_new
```

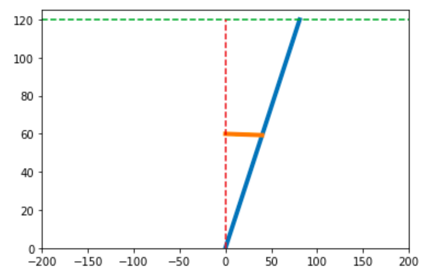
- **We define the function of find the roots using Newton Raphson Method.**
- **The tolerance is taken as 0.000001 and max iterations taken 100.**
- **We take the initial guess in a NumPy array as [1,1,1,1] of type float as NumPy works with float type values.**
- **We use a while loop to test if the max iterations been reached or the tolerance for all the values been reached.**
- **We use the initial guess and the Jacobian and the function evaluated at the guess to calculate the new tentative solution as per the Newton Raphson Algorithm**
- **The new solution is considered as the new guess and we continue unless the about stooping criteria is reached.**

In [8]:

```
#Itering through theta to find all solutions
for i in range(360):
    o2 = np.radians(i)
    x_new = root_finder()
    #co-ordinates
    o3 = x_new[3][0]
    r4 = x_new[1][0]
    s0 = [0,0]
    s1 = [0,r1]
    s2 = [s1[0]+r2*math.cos(o2),s1[1]+r2*math.sin(o2)]
    s3 = [r4*math.cos(o3),r4*math.sin(o3)]
    if i ==30 :
        print(x_new)
    clf()
    ct = ploter(ct)
```

```
<Figure size 432x288 with 0 Axes>

<Figure size 432x288 with 0 Axes>
```



- **Here we iterate over θ_2 from 0 to 360 degrees and get the solutions for each value of θ_2 using the root_finder() function.**
- **We print the specific value for $\theta = 30$ and plot the links using the ploter() function.**

The Programme

```
import numpy as np # Since we will define the equation variables and coefficients in array form

from numdifftools.nd_algopy import Jacobian # For use of grad operator and finding jacobian of a matrix with ease

import math # for using mathematical funcs

from matplotlib.pyplot import * #for plotting the links for animation

ct = 0

# Given data [taken as per Student's wish as Instructed]

r1 = 60

o2 = math.radians(0) #varries

r2 = 40

r6 = 60

# Defining Func Eqns in Python
```

```
#r3,r4,r5,o3 = 87.17,130.75,51.96,math.radians(66.58) for theta = 30
degrees

eqn = []

eq1 = lambda x : 40*np.sin(o2)-x[0]*np.sin(x[3])+60

eqn.append(eq1)

eq2 = lambda x : 40*np.cos(o2)-x[0]*np.cos(x[3])

eqn.append(eq2)

eq3 = lambda x : 60-x[1]*np.sin(x[3])+60

eqn.append(eq3)

eq4 = lambda x : x[2]-x[1]*np.cos(x[3])

eqn.append(eq4)

#Jacobians Defined

jacob1 = Jacobian(eq1)

jacob2 = Jacobian(eq2)

jacob3 = Jacobian(eq3)

jacob4 = Jacobian(eq4)

#plot

def ploter(ct):

    #Naming file

    if ct<10 :

        filename = 'QuickReturn00'+str(ct)+'.jpg'

    elif ct<100 :

        filename = 'QuickReturn0'+str(ct)+'.jpg'

    else :

        filename = 'QuickReturn'+str(ct)+'.jpg'


    #Plotting

    figure()

    xlim(-200, 200)

    ylim(0, 125)

    plot((s0[0],s3[0]),(s0[1],s3[1]),linewidth='4')
```

```
    plot((s1[0],s2[0]),(s1[1],s2[1]),linewidth='4')
    plot((-200,200),(120,120),linestyle='dashed')
    plot((0,0),(0,120),linestyle='dashed')
    savefig(filename)
    ct=ct+1
    return ct
# Newton Raphson Method for Multivariable systems of Non-Linear
equation to be implemented


def root_finder():
    i = 0
    er = 100
    tol = 0.000001
    maxiter = 100
    M = 4
    N = 4
    x0 = np.array([1,1,1,1],dtype=float).reshape(N,1)
    while np.any(abs(er)>tol) and i< maxiter:
        func_eval =
np.array([eq1(x0),eq2(x0),eq3(x0),eq4(x0)]).reshape(M,1)
        flat_x0 = x0.flatten()
        jac =
np.array([jacob1(flat_x0),jacob2(flat_x0),jacob3(flat_x0),jacob4(flat_
x0)])
        jac = jac.reshape(N,M)
        x_new = x0 - np.linalg.inv(jac)@func_eval
        er = x_new - x0
        x0 = x_new
        i+=1
    return x_new
#Itering through theta to find all solutions
for i in range(360):
```

```
o2 = np.radians(i)
x_new = root_finder()
#co-ordinates
o3 = x_new[3][0]
r4 = x_new[1][0]
s0 = [0,0]
s1 = [0,r1]
s2 = [s1[0]+r2*math.cos(o2),s1[1]+r2*math.sin(o2)]
s3 = [r4*math.cos(o3),r4*math.sin(o3)]
if i ==30 :
    print(x_new)
clf()
ct = ploter(ct)
```

Conclusion

- **Finding numerical solutions to the Quick Return mechanism successfully carried out and Animation performed.**
- **The calculated value has been verified with the analytically obtained solution.**