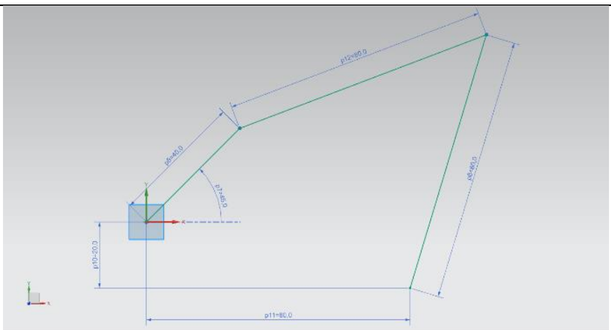
Lab_7

GitHub: https://github.com/jjroemerjj/VP_project

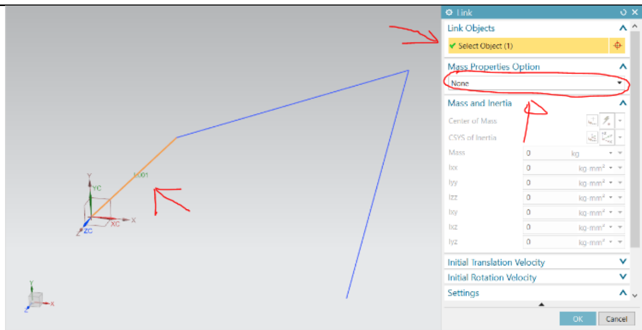
Skill: Motion

Tools: Link, Joint, Solution, Solve, Animation, XY Result View

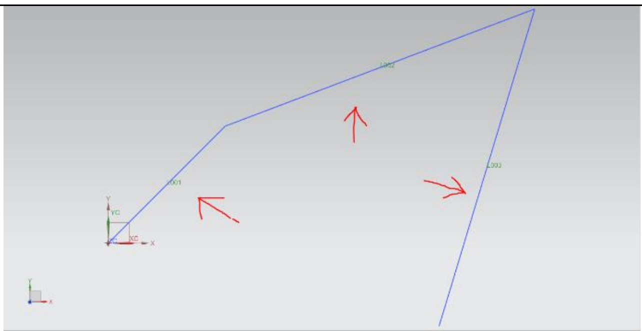
1. Create sketch with given dimensions (Figure 1)
2. File -> Motion
3. Home -> New Simulation: save file to your working directory -> Analysis Type: Kinematics, Joint Wizard: unclick if needed (do not use Wizard)
4. Home -> Link -> create first link (Figure 2). Link has to be massless
5. Create two more links (Figure 3). Complete mechanism consists of three links.
6. Home -> Joint -> Type: Revolute -> create first joint (Figure 4 -6). Joint origin has to be at the end (base) of the mechanism. Joint vector has to be normal to the sketch created in point 1 (Z-axis if the instruction is followed strictly)
7. Create second joint -> Base: Snap Links -> chose first link (Figure 7)
8. Create third joint snapped to second one (Figure 8).
9. Create fourth joint (Figure 9). Joint is not snapped
10. Motion Navigator -> edit first joint (Figure 10 - 11). Create Driver: rotational, polynomial
11. Home -> Solution -> Solution type: Normal Run, Analysis type: Kinematics/Dynamics, Time: 4s, Steps: 360
12. Home -> Solve
13. Analysis -> Animation -> Check the results
14. Motion Navigator -> Click on Joint 3 (J003) -> Open XY Result View -> Absolute -> Displacement -> double click on 'X' -> Viewpoint: Create New Window (Fig 12 - 14)
15. Crete plot for 'Y' (Fig 15)
16. Create 'RZ' displacement plots for Joint 2 and 3 (Fig 16 and 17)
17. Write program in Python (Appendix 1). Reed carefully all comments.



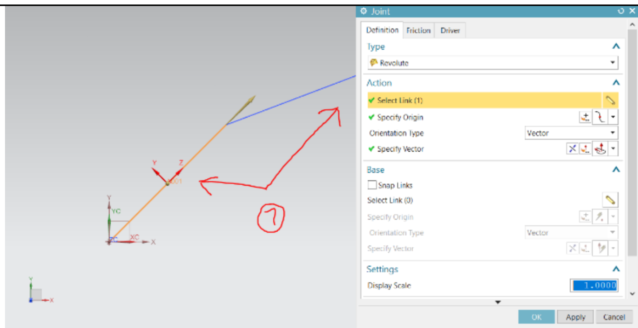
Picture 1



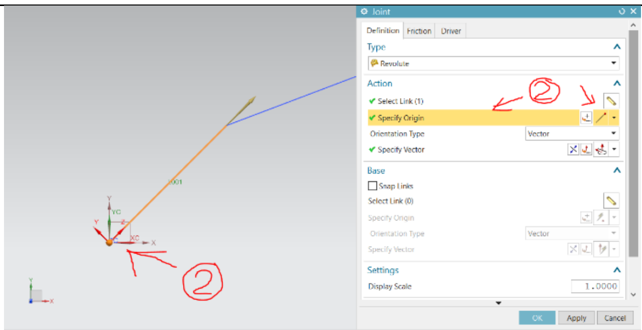
Picture 2



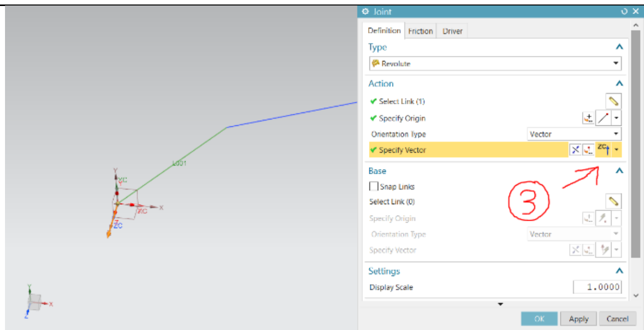
Picture 3



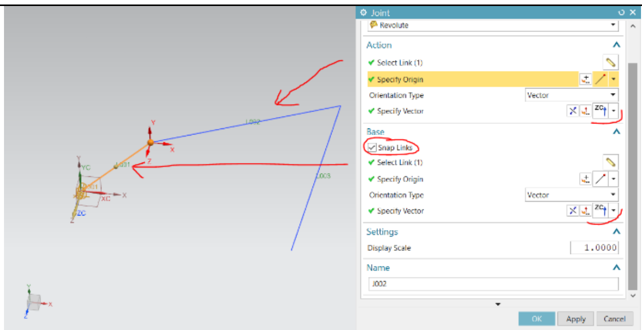
Picture 4



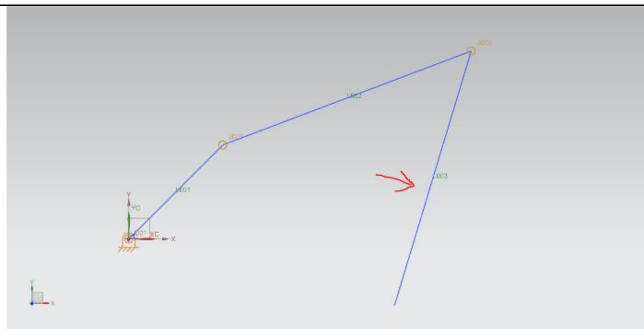
Picture 5



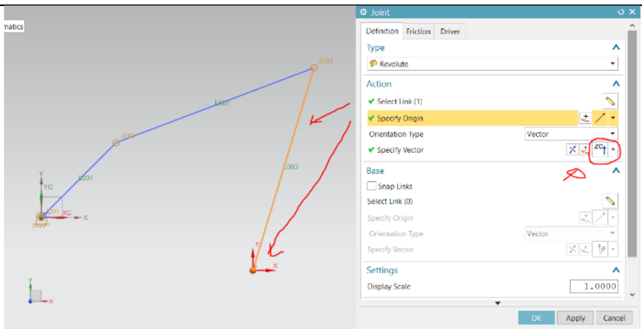
Picture 6



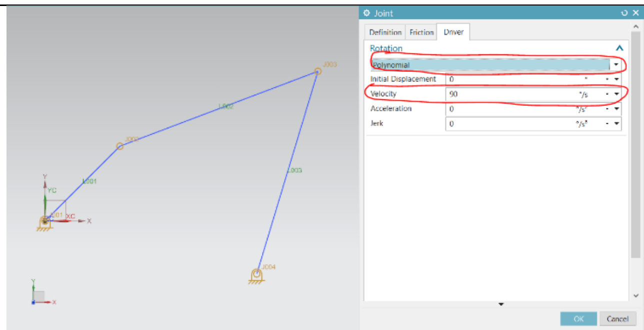
Picture 7



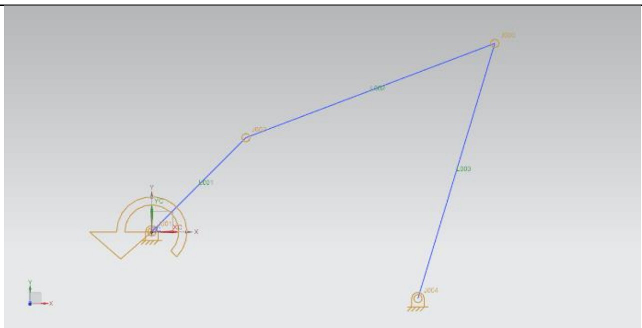
Picture 8



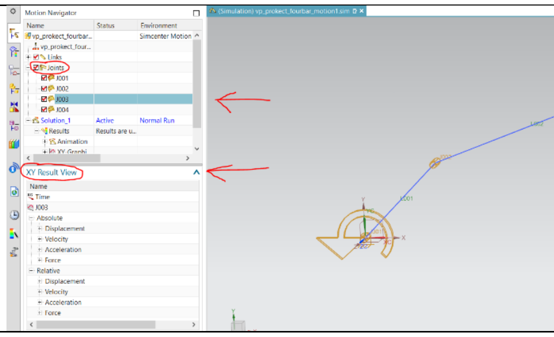
Picture 9



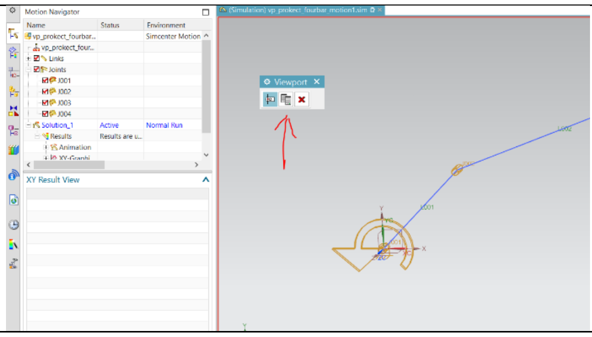
Picture 10



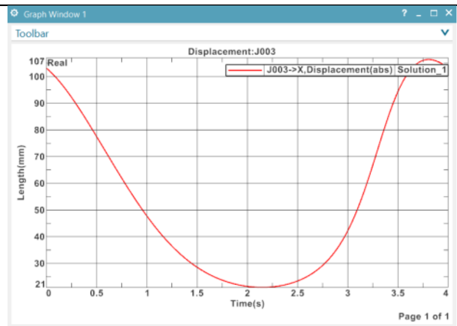
Picture 11



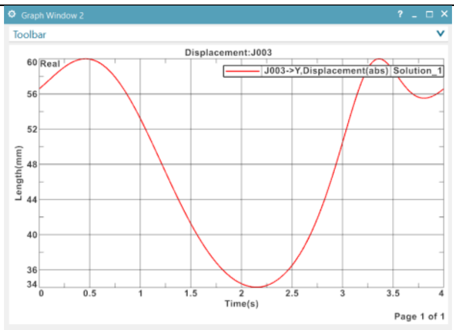
Picture 12



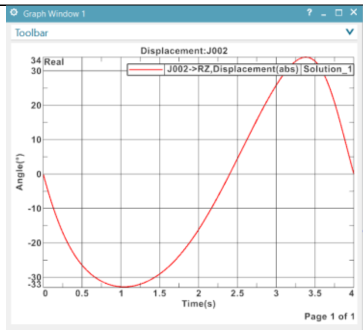
Picture 13



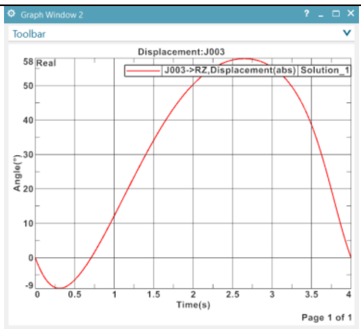
Picture 14



Picture 15



Picture 16



Picture 17

Picture 18

Appendix 1

```
# libraries to import
from scipy.optimize import fsolve
import math
import numpy as np
from copy import deepcopy
import matplotlib.pyplot as plt


# 0000000000000000000000000000000000000000000000000000000000000000
# Task 1: Write the script which calculates current values of
# 'fi2' and 'fi3' (angles) of the mechanism from the picture.
# ................................................................


# ----------------------------------------------------------------
# Mechanism parameters (dimensions and initial angles)
# ----------------------------------------------------------------

# Bases coordinates
A = [0, 200]
B = [800, 0]

# Arms length
AB = 400
BC = 800
CD = 800
BE = 400

# Driving arm starting angle (can be changed to any number)
fi1 = 45

# Driving arm angular velocity (for time based simulations)
omega = 90
```

```python
# ---------------------------------------------------------------
# Equation formulation (this section has to be commented)
# ---------------------------------------------------------------

# general formula
# I1cos(fi1) + I2cos(fi2) + I3cos(fi3) + I4cos(fi4) + I5cos(fi5) = 0
# I1sin(fi1) + I2sin(fi2) + I3sin(fi3) + I4sin(fi4) + I5sin(fi5) = 0

# fi4 and fi5 are always fixed
# cos(fi4) = 0, cos(fi5) = 1, sin(fi4) = 0, cos(fi4) = 1

# general formula after simplification
# I1cos(fi1) + I2cos(fi2) + I3cos(fi3) + I5 = 0
# I1sin(fi1) + I2sin(fi2) + I3sin(fi3) + I4 = 0

# ---------------------------------------------------------------

# Vector lengths calculation (for code clarity)
I1 = AB
I2 = BC
I3 = CD
I4 = np.linalg.norm(B[0] - A[0])    # calculated from bases coordinates
I5 = np.linalg.norm(B[1] - A[1])

# I4, I5 vectors angle definition
fi4 = 180
fi5 = 90

# Angles transformation from arc to radians
fi1 = math.radians(fi1)
fi4 = math.radians(fi4)
fi5 = math.radians(fi5)


# Function defines system of equations
def f(p):
    fi2, fi3 = p    # other way to pass those arguments could be considered
    e1 = I1*math.cos(fi1) + I2*math.cos(fi2) + I3*math.cos(fi3) + I4*math.cos(fi4) +
I5*math.cos(fi5)
    e2 = I1*math.sin(fi1) + I2*math.sin(fi2) + I3*math.sin(fi3) + I4*math.sin(fi4) +
I5*math.sin(fi5)
    return e1, e2


# Solving system of equations
s = fsolve(f, np.array([0, 0]))  # np.array([0, 0]) defines input arguments (predicted
solutions)

# print(type(s))  # All 'print' commands can be 'commented'.
s = getattr(s, "tolist", lambda: s)()   # Convert to native python format (list)

# converting angle from radians to degrees (s[0] = fi2, s[1] = fi3)
s[0] = math.degrees(s[0])
s[1] = math.degrees(s[1])

# Converting to positive-only angles
if s[0] < 0:
    s[0] = 360 - abs(s[0])

if s[1] < 0:
    s[1] = 360 - abs(s[1])

# Final outcome
```

```
fi2 = s[1]
fi3 = s[0]

print('The mechanism has following angles: fi1 = %d, fi2 = %d, fi3 = %d' % (fi1, fi2, fi3))

# At this point we have fully defined all vectors which represents the current state of the
mechanism
# The first task is done


# 0000000000000000000000000000000000000000000000000000000000000000
# Task 2: Write the script which calculates the position of 'C' joint
# for any value of 'fi1' parameter.
# ................................................................

# ----------------------------------------------------------------
# Joint C position definition (vector) (this section has to be commented)
# ----------------------------------------------------------------

# x-axis position
# Cx = AB*sin(fi1) + BC*sin(fi2)
# y-axis position
# Cy = AB*cos(fi1) + BC*cos(fi2)


# ----------------------------------------------------------------
# Input vector definition
# ----------------------------------------------------------------

# Start/stop angle
ss_angle = [0, 359]     # In this situation the full range of motion will be calculated

# Number of steps
# step_no = 90            # The more steps the longer computational time (and other thing
which will be explained later)
step_no = ss_angle[1] - ss_angle[0]

# Input vector (in degrees)
ff1 = [ss_angle[1]/step_no * x for x in range(step_no+1)]


# Converting input vector into radians
ff1 = [math.radians(x) for x in ff1]

# Creating numpy array for parameters
a = np.zeros((step_no+1, 5))    # five columns for parameters 'fi1' to 'fi5'
a[:, 0] = ff1
a[:, 3] = fi4
a[:, 4] = fi5

# creating array for x-y coordinates  of 'C' joint
c = np.zeros((step_no+1, 2))

# solving equations (in loop)
for i in range(len(ff1)):
    fi1 = a[i, 0]
    s = fsolve(f, np.array([0.2, 1]))   # INPUT ARGUMENTS HAS BEEN CHANGED !! Try other
parameters
    # convert to native python format (float)
    s = getattr(s, "tolist", lambda: s)()
    a[i, 1] = s[1]
    a[i, 2] = s[0]

# Array of parameters in arc degrees (deep copy needed to obtain new object in memory)
```

```python
a_arc = deepcopy(a)

# converting from radians to degrees
for i in range(len(a_arc)):
    for n in range(5):
        a_arc[i, n] = math.degrees(a_arc[i, n])

# Converting to positive-only angles
for i in range(len(a_arc)):
    for n in range(5):
        if a_arc[i, n] < 0:
            a_arc[i, n] = 360 - abs(a_arc[i, n])

# Result plot
plt.subplot(121)
plt.plot(a_arc[:, 1])
plt.subplot(122)
plt.plot(a_arc[:, 2])
plt.show()
# Compare obtained results with NX Motion (Figures 16 and 17)


# Function calculates x and y position of C joint
def c_position(d):
    f1, f2 = d
    p_x = I1 * math.cos(f1) + I2 * math.cos(f2)
    p_y = I1 * math.sin(f1) + I2 * math.sin(f2)
    return p_x, p_y


# New array of positive-only radian parameters
a_rad = deepcopy(a_arc)
for i in range(len(ff1)):
    for n in range(5):
        a_rad[i, n] = math.radians(a_rad[i, n])

# Calculating c-coordinates for all parameters
for i in range(len(c)):
    c[i, :] = c_position([a_rad[i, 0], a[i, 1]])


# Result plot
plt.subplot(121)
plt.plot(c[:, 0])
plt.subplot(122)
plt.plot(c[:, 1])
plt.show()
# Compare obtained results with NX Motion (Figures 14 and 15)

# At this point we have calculated relationship between fi1 angle and X and Y position of
'C' joint.
# fsolve numerical function has been used to solve systems of equations
# Used function is sensitive to changes in input arguments. Completely wrong results can be
easily
# obtained if function is used with incompetently

# The second task is done
```