Experiment No. – 1				
Date of Performance:	16/1/25			
Date of Submission:	16/1/25			
Program Execution/ formation/ correction/ ethical practices (06)	Timely Submission (01)	Viva (03)	Experiment Total (10)	Sign with Date

1.1Aim: Study of given robot models and determine the Work Space

Envelope of SCARA robot

1.2 Course Outcome: Explain the major technologies, languages, applications, social, economic and ethical consequences of a robotic manipulator.

1.3 Learning Objectives: Understand the structure, configuration of a SCARA robot.

1.4 Requirement: Robot models

1.5Related Theory:

Work Space Envelope: The locus of all the points traced by the tip of the end effectors in the three-dimensional space is defined as WSE. The area in which robot can do useful work is work envelope area.The geometry of the work envelope is determined by the sequence of joints used for the first three axes.

Robots are classified based on the work envelope geometry as

Cartesian Robot: The three major axes of a Cartesian robot are all prismatic. The prismatic joints correspond to moving the wrist up and down, in and out and back and forth.

Cylindrical Robot:If the first joint of the Cartesian robot is replaced with revolute joint it becomes a cylindrical robot.the configuration of cylindrical robot is RPP.(R-revolute,P-prismatic).The first revolute joint swings the arm back and forth about a vertical base axis.The second prismatic joint then move the wrist up and down along vertical axis and the third prismatic joint moves the wrist in and out along the radial axis.Since there will be some radial position work envelope generated by this configuration is the volume between two vertical concentric cylinders.

Spherical Robot: If the second joint of a cylindrical robot is replaced by a revolute joint we get a spherical robot.The first revolute joint swings the arm back and forth about the vertical axis while the second revolute joint pitches the arm up and down about a horizontal shoulder axis.The prismatic joint moves the wrist radially in and out.The work envelope generated is the volume between two spheres

SCARA Robot: SCARA is Selective Compliance Assembly Robot Arm. robot has two revolute joints and one prismatic joint to position the wrist. for a SCARA robot the axes of all the three joints are vertical The first revolute joint swings the arm back and forth about a base axis that can also be thought of asa vertical shoulder axes .the second revolute joint swings the forearm back and forth about vertical elbow axis. Thus two revolute joints control motion in a horizontal position The vertical component of the motion is provided by third joint which slides wrist up and down.

The work envelope of SCARA robot can be complex depending upon the limits on the ranges of travel for the first two axes.

Articulated Robot: An articulated robot is the dual of Cartesian robot in the sense all three of the major axes are revolute rather than prismatic.

The first revolute joint swings the robot back and forth about a vertical base axis. The second joint pitches arm up and down about a horizontal shoulder axis and the third joint pitches the forearm up and down about horizontal elbow axis.

1.6 Procedure for Work Envelope Calculation of SCARA

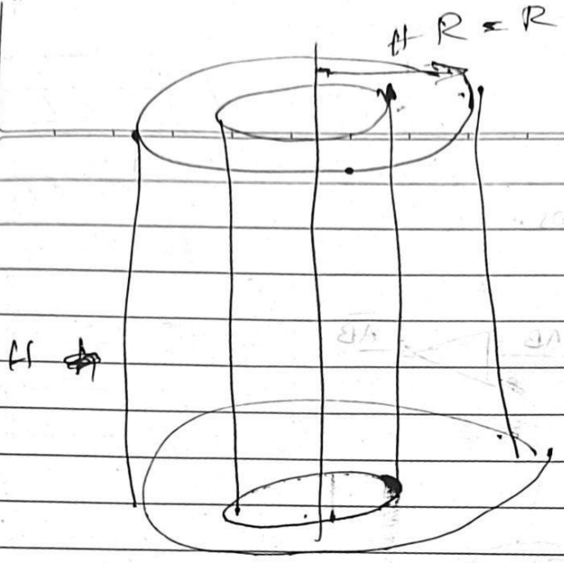
Measure Horizontal Reach and Stroke

Measure Vertical Reach and Stroke

Calculate WSE.

1.7 Program and Output:

PAGE No.	
DATE	/ /

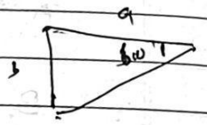


Work volume.

$\pi R^2 h - \pi r^2 h.$

$3.14(60)^2 30 -$
$3.14(42.42)^2 30.$

$= \underline{\underline{169698}}$



~~H~~R = R.

Horizontal Stroke = R − r.

$$\begin{array}{r} 339292 \\ -169594 \\ \hline 169698 \end{array}$$

$r = \sqrt{a^2 + b^2 + 2ab\cos\theta}.$

60 − 42.42

VR = Max VR. = ~~h~~.H

VS = Max VR − Min VR. = H − h.

VR = $\underline{\underline{30}}$

h = 30. a = 45 θ = 60°.
l = 60 b = 15

$\sqrt{a^2 + b^2 + 2ab\cos\theta}.$

$= \sqrt{45^2 + 15^2 + 2(45 \times 15)\cos 60}$

$= \sqrt{~~2025~~\ 2250 + 1350} \times \frac{1}{2}$

$= \sqrt{1800}$

$= \sqrt{1800}$

$r = \underline{\underline{42.42}}$

45 = R ~~R~~
= $\underline{\underline{17.58}}$ R = 60.

1.8 Conclusion:

Hence we successfully understood the basic architectural functional architecture of the robotic arm.

The work space envelope of SCARA robot is obtained as 17.58

Experiment No. – 2				
Date of Performance:	23/1/25			
Date of Submission:	23/1/25			
Program Execution/ formation/ correction/ ethical practices (06)	Timely Submission (01)	Viva (03)	Experiment Total (10)	Sign with Date

2.1. Aim: To understand the Simple Ladder program.

2.2. Course Outcome: Explain the major technologies, languages, applications, social, economic and ethical consequences of a robotic manipulator.

2.3. Learning Objectives:. To Solve the problem using ladder programming.

2.4.Requirement:. Portal link: https://plc-coep.vlabs.ac.in/exp/implementation-logic-gates/

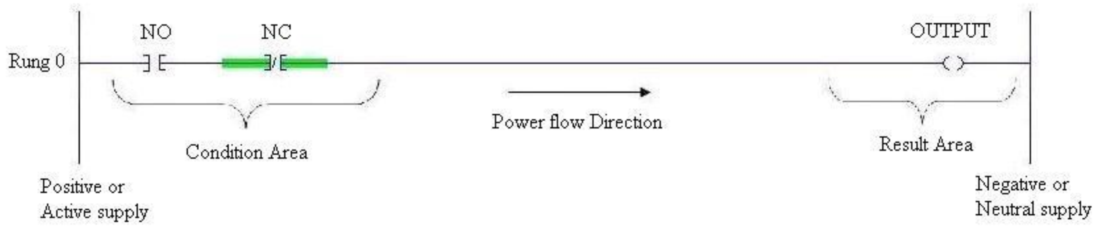
2.5 Related Theory:

Each manufacturer of PLC systems has own style of writing the instructions. Different PLCs has different instruction sets but even some common basic instructions are shared by all the PLCs. All manufacturers give different software packages for programming PLCs. Ladder is most commonly used programming language. Prior to PLCs, relay logic was used in industry. Ladders were developed to mimic or imitate relay logic.

Relay Logic / Instructions A relay is simple magnetic device which acts as a control switch. When the switch is on, current will flow through the coil on iron piece. This iron core acts a electromagnet and due to the magnetic field upper contact gets attracted towards lower one and circuit gets completed, allowing current to flow from load.

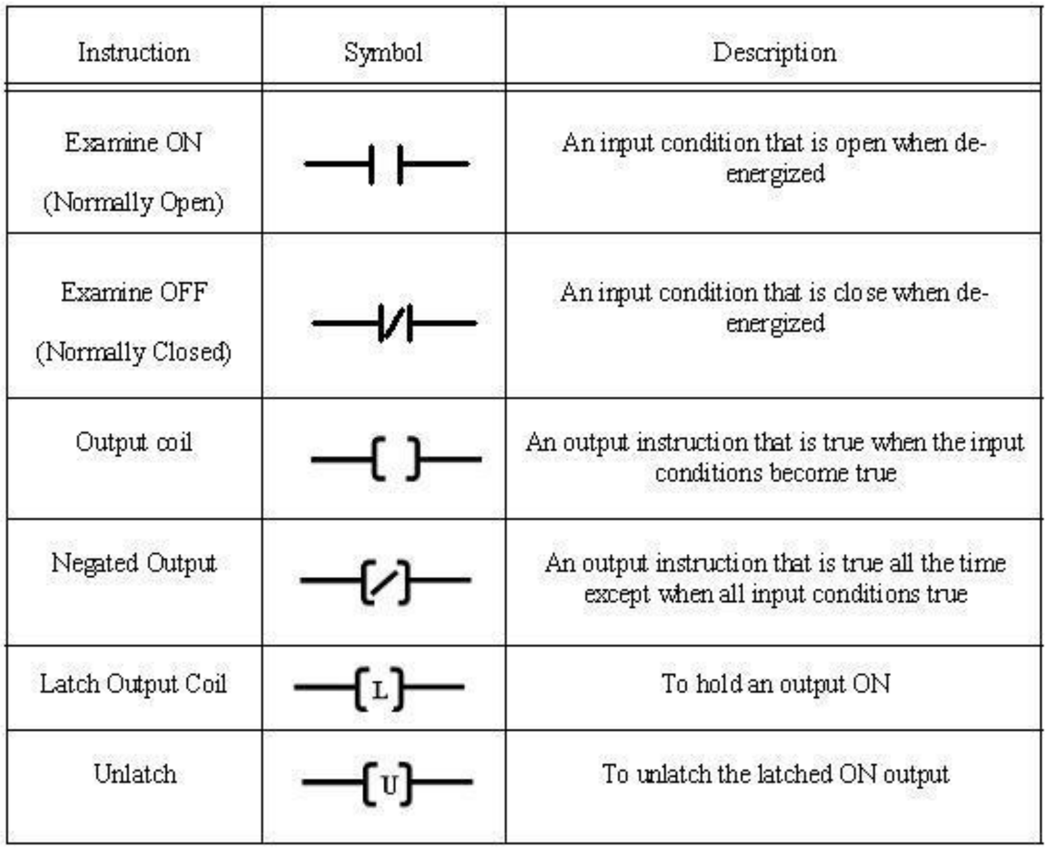
Ladder Programming

Ladder diagram is popular language of programming the PLCs. Ladder diagram shows the sequence of the logic execution which is presented diagrammatically. In ladder diagram, There are two vertical lines generally called as Phase (positive) or neutral. Rungs which show current flow in horizontal direction are the sequence in which the logic executes. The Analogous to relay, ladder has two main symbols which are contacts and output coil. Generally each rung has inputs (contacts) on left hand side and outputs (coil) on the right hand side. These contacts and coils are called as bits of the relays. Each input and output are individual bit in I/O files. An instruction in ladder instructs PLCs how to respond to the bits in I/O files which are stored in the memory. Input contacts are the condition area, the conditions must be fulfilled to change the status of the output coils.



Typical Ladder Diagram

Most commonly used relay instructions used in PLC programming are as shown in the table below.



Instruction	Symbol	Description
Examine ON (Normally Open)	—| |—	An input condition that is open when de-energized
Examine OFF (Normally Closed)	—|/|—	An input condition that is close when de-energized
Output coil	—( )—	An output instruction that is true when the input conditions become true
Negated Output	—(/)—	An output instruction that is true all the time except when all input conditions true
Latch Output Coil	—(L)—	To hold an output ON
Unlatch	—(U)—	To unlatch the latched ON output

2.6 Procedure:

Important steps for developing Ladder using Simulator:

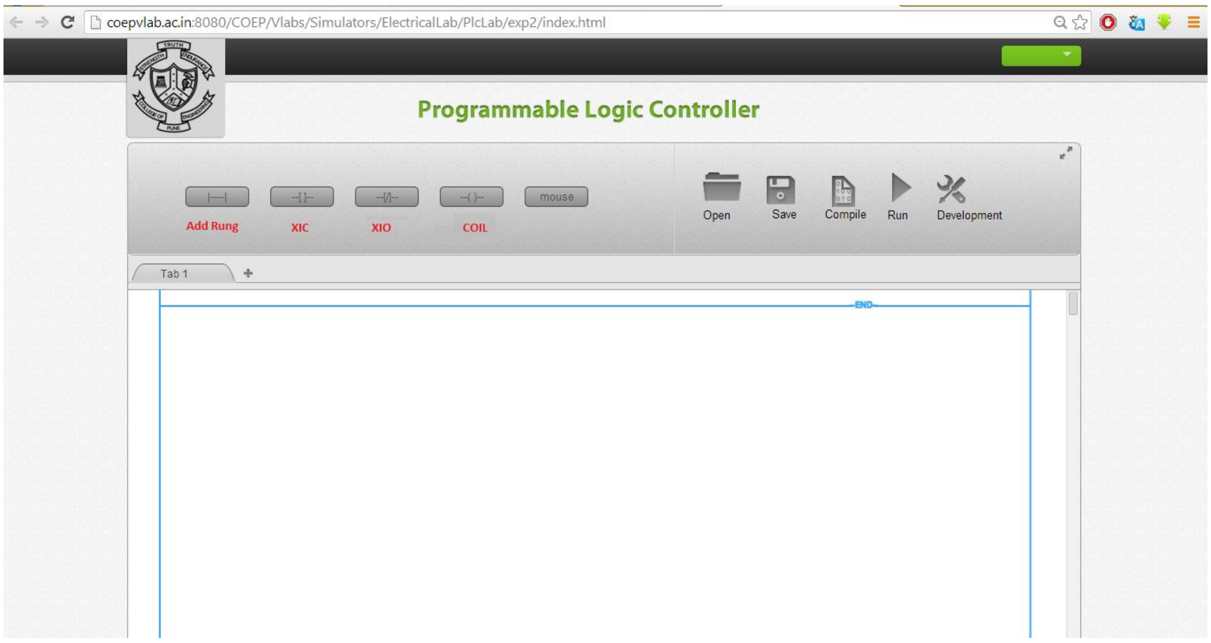
Please follow the steps so as to understand the procedure for developing ladder logic for various logical gates.

1. Prior to starting of the development of ladder diagram following steps needs to be understood:

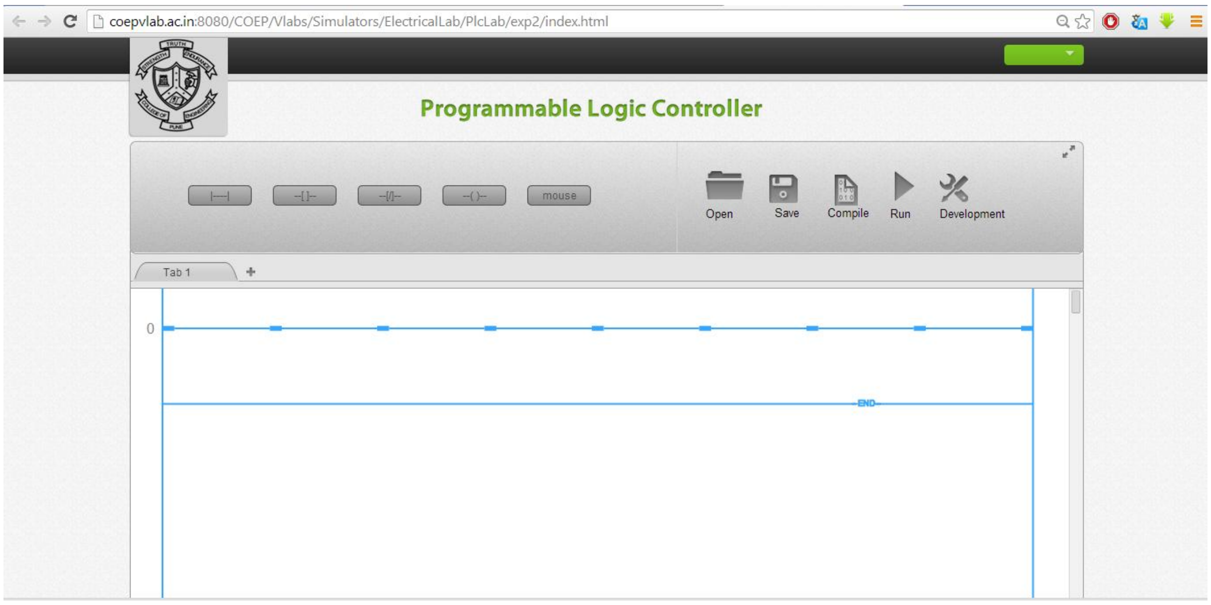
- Understand the problem statement like test the logic for OR gate, AND Gate etc.
- Develop the logic on paper and validate the logic by considering various cases
- Prepare the truth table and test the logic using all valid cases
- Go to simulator icon and click on the “Simulator” button
- The PLC simulator will be opened in new window.

2. The procedure for writing the ladder diagram in the work space is as follows:

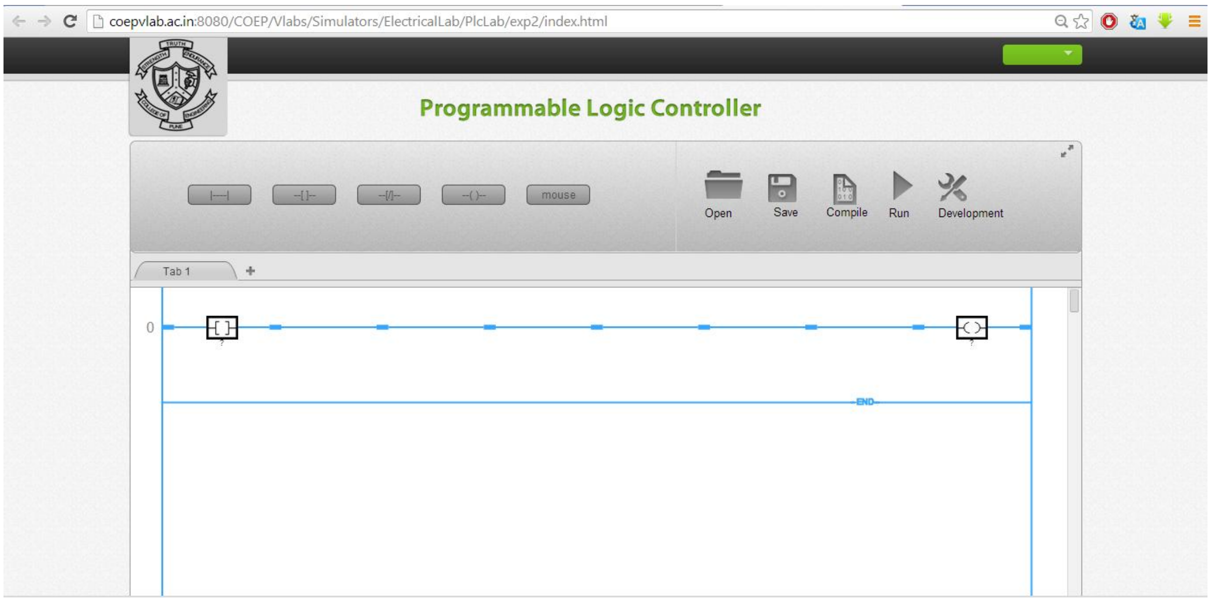
The screen shot for the first window will appear like this.


coepvlab.ac.in:8080/COEP/Vlabs/Simulators/ElectricalLab/PlcLab/exp2/index.html
Programmable Logic Controller
mouse
Add Rung
XIC
XIO
COIL
Open
Save
Compile
Run
Development
Tab 1
END

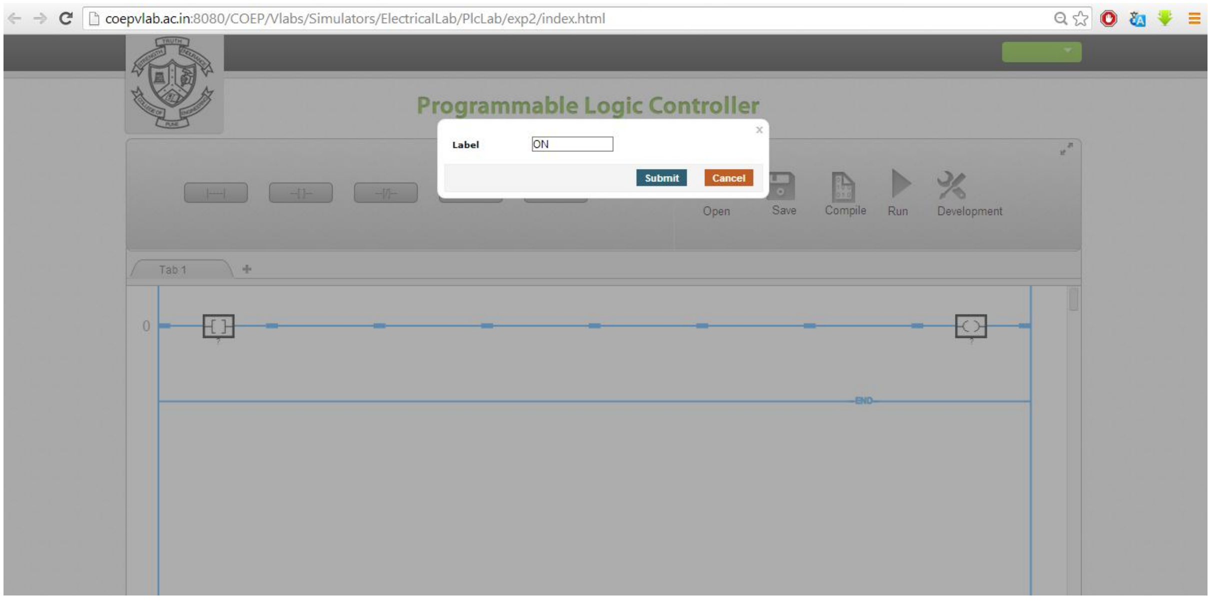
Add a new rung by clicking on the 'Add Rung' icon. The window will appear like this:


coepvlab.ac.in:8080/COEP/Vlabs/Simulators/ElectricalLab/PlcLab/exp2/index.html
Programmable Logic Controller
mouse
Open
Save
Compile
Run
Development
Tab 1
0
END

Place the contacts as per the requirement by left clicking the appropriate contact shown at the top side. In the example demonstrated below one normally contact and one coil is placed as shown the figure.


coepvlab.ac.in:8080/COEP/Vlabs/Simulators/ElectricalLab/PlcLab/exp2/index.html
Programmable Logic Controller
mouse
Open
Save
Compile
Run
Development
Tab 1
0
END

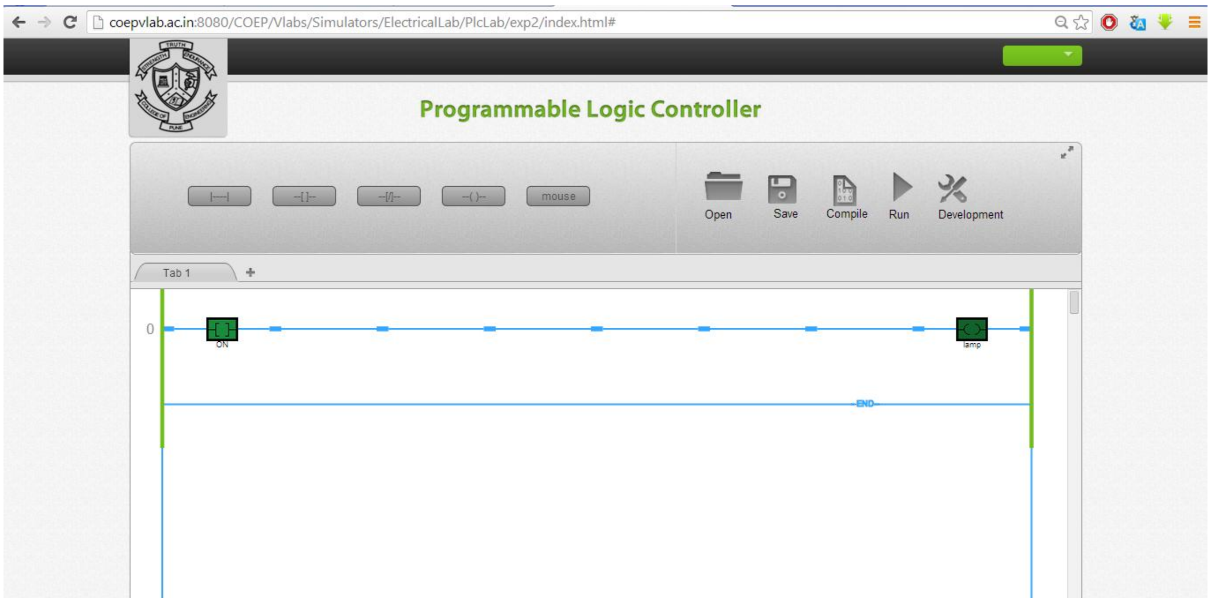
Right click on the contact or coil and you can give tag name like “start”, “stop” etc. Please ensure that the tag numbers are true replica of process connections. Similarly give tag name to coil like “motor”, “Lamp” etc. The final ladder will look like this.


coepvlab.ac.in:8080/COEP/Vlabs/Simulators/ElectricalLab/PlcLab/exp2/index.html
Programmable Logic Controller
Label
ON
Submit
Cancel
Open
Save
Compile
Run
Development
Tab 1
0
END

Please note that the tag names are case sensitive and if you are using them in circuit as bit make sure that the correct tag name appears.

Click the Compile button so the ladder will be ready for running. For testing the logic you need to click on Run. Both sides of the rung will become green and this is the indication of run mode. Please not that in run mode you can't make changes in the ladder. For modifications user has to go to development mode.

By right clicking on the contact toggle the state of contact.Check the output contact status.


coepvlab.ac.in:8080/COEP/Vlabs/Simulators/ElectricalLab/PlcLab/exp2/index.html#
Programmable Logic Controller
mouse
Open
Save
Compile
Run
Development
Tab 1
0
ON
lamp
END

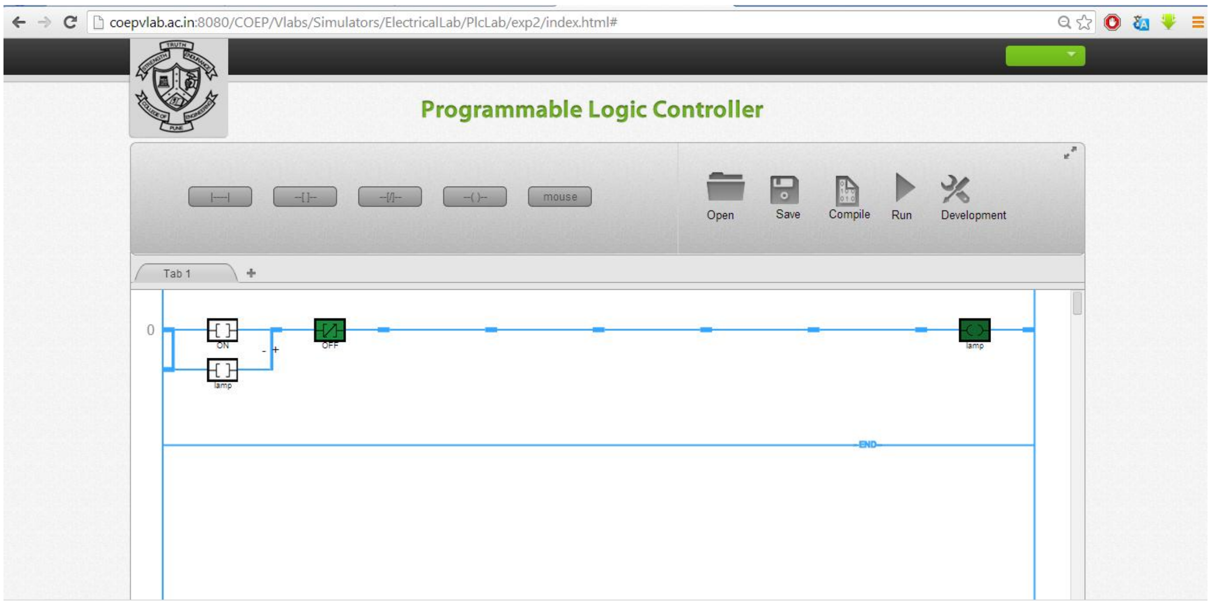
Please remember the ladder contacts or the state of the inputs and outputs are always in de-energised state. The de-energised is that state wherein the contacts are in non-active state.

You can once again toggle the contact and the output state will change. To add any contact you will have to go to development mode. Click on the rung and add contacts.

To delete any contact or output right click on the contact and press “delete”, the contact will be deleted.

You can add seven elements in series and 5 elements in parallel.

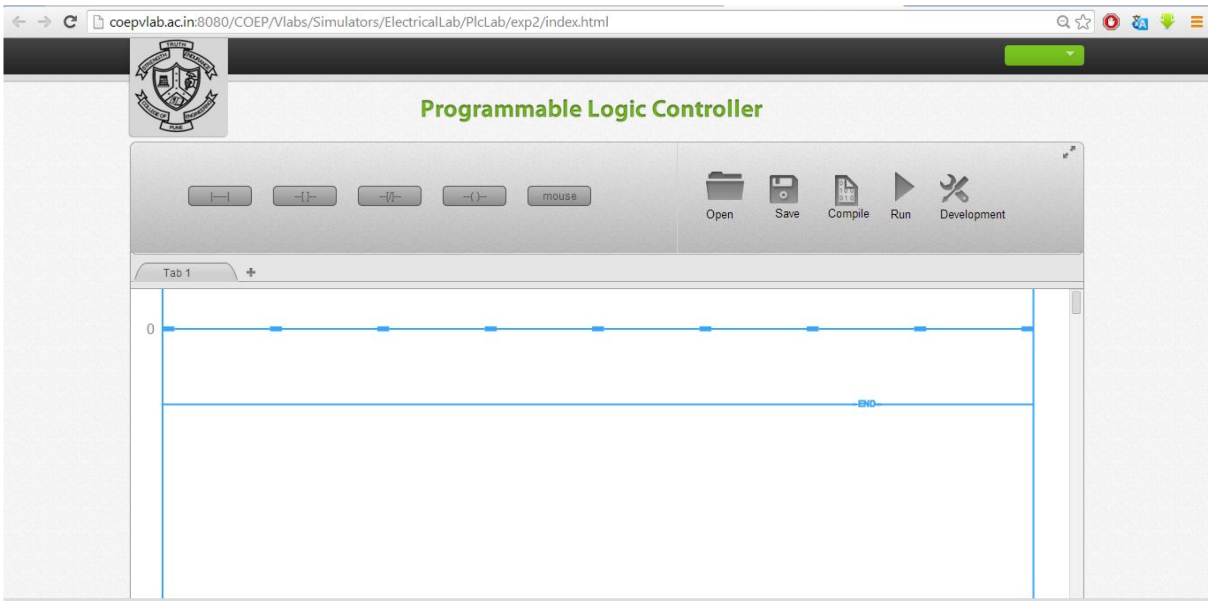
To add element in parallel click on the node near contact where you wish to add parallel branch. Select the branch and click on the '+' sign to complete the loop. The screen will appear as shown below.

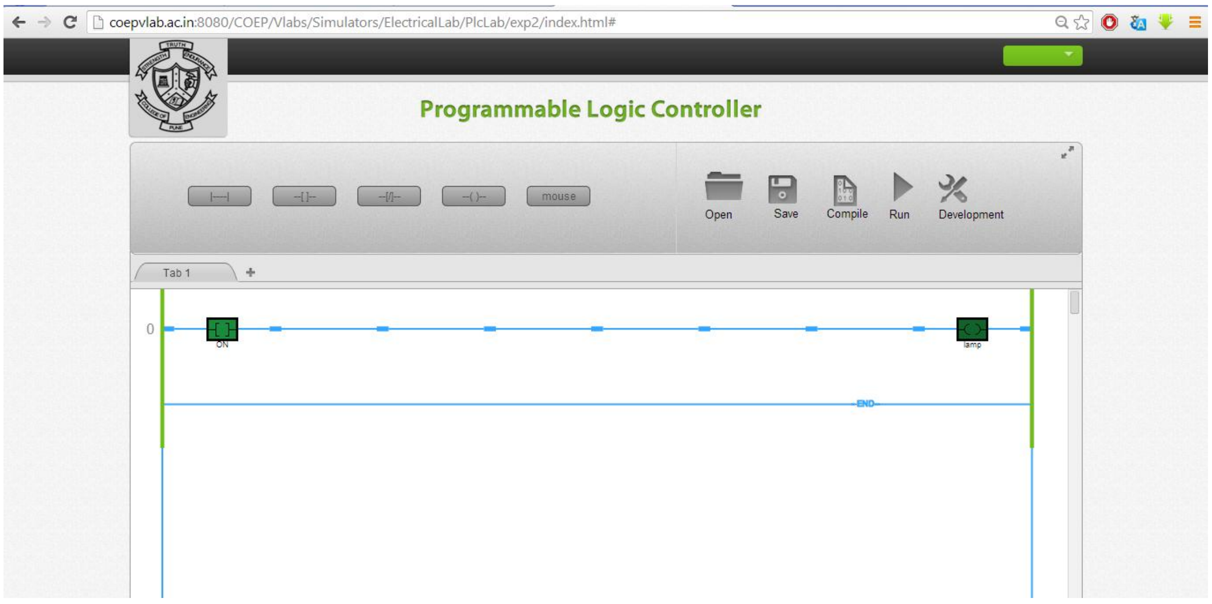

coepvlab.ac.in:8080/COEP/Vlabs/Simulators/ElectricalLab/PlcLab/exp2/index.html#
Programmable Logic Controller
mouse
Open
Save
Compile
Run
Development
Tab 1
0
ON
lamp
OFF
lamp
END

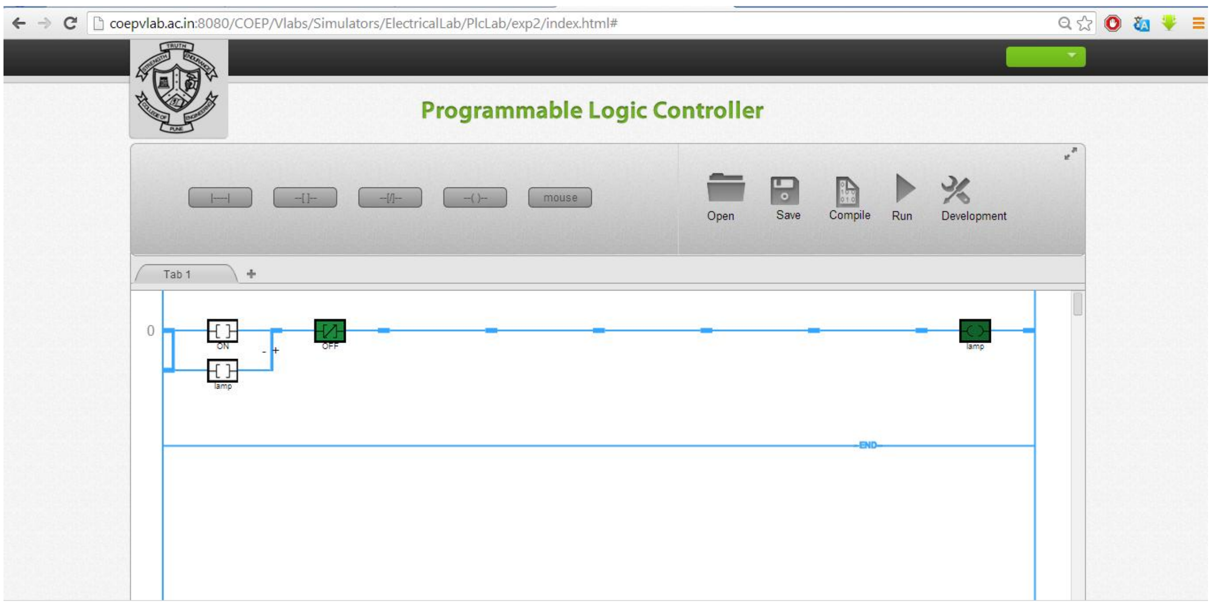
Repeat the procedure and verify your logic.

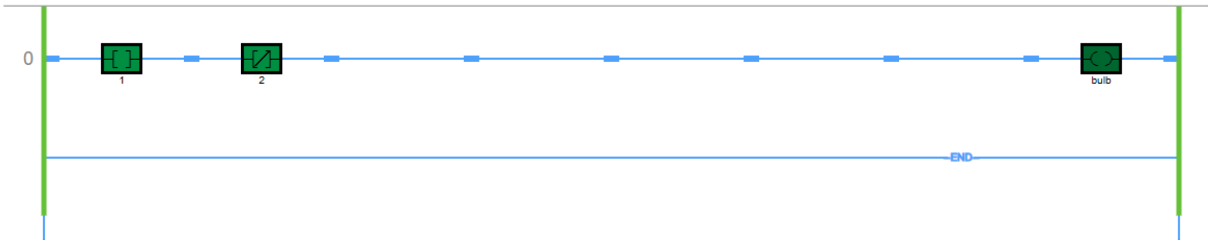
Similarly you can check the logic for OR, NOR, and NAND gates. Validate the truth tables and confirm the results.

2.7 Program and Output:


coepvlab.ac.in:8080/COEP/Vlabs/Simulators/ElectricalLab/PlcLab/exp2/index.html
Programmable Logic Controller
mouse
Open
Save
Compile
Run
Development
Tab 1
0
END


coepvlab.ac.in:8080/COEP/Vlabs/Simulators/ElectricalLab/PlcLab/exp2/index.html#
Programmable Logic Controller
mouse
Open
Save
Compile
Run
Development
Tab 1
0
ON
lamp
END


coepvlab.ac.in:8080/COEP/Vlabs/Simulators/ElectricalLab/PlcLab/exp2/index.html#
Programmable Logic Controller
mouse
Open
Save
Compile
Run
Development
Tab 1
0
ON
lamp
OFF
lamp
END



2.8 Conclusion: Hence we successfully learned about the ladder programming and implemented in creating the logic gates.

2.9 Questions:

1. The scan time of PLC depends upon:

   a: Processor speed and program length

   b: I/O density

   c: None of these

   d: Both a and b

2.The programming device must be connected to the controller

   a: At all times

   b: When entering the program

   c: When program is running

   d: When program is stopped

3.The ----- will account for the most of the total memory of a given PLC program.

a: User program

b: Output image table

c: Input image table

d: All of these

4.While developing the ladder program the contacts are in:

a: Any state (either energised or de-enrgised)

b: Energised state

c: De-energised state

d: All of these

5.Which of the following is not a factor in determining the total scan time of a PLC:

a: Total no. of inputs and outputs

b: Programming method

c: Program Length

d: All of these

6. The logical AND function is similar to:

a: Combination of series- parallel contacts

b: Contacts in parallel

c: Contacts in series

d: None of these

7.The logical OR function is similar to:

a: Contacts in parallel

b: Contacts in series

c: Combination of series- parallel contacts

d: None of these

8.Which of the following logic operators are available in PLC?

a: AND, OR

b: NOT

c: All of the above

d: None of these

9.What is the output of NOR block if any one of the input is high or active or true?

a: 1

b: 0

c: Can’t say

10. What is the output of NAND block if any one of the two inputs is high or active or true?

a: 1

b: 0

c: Can’t say

Experiment No. – 3				
Date of Performance:	30/1/25			
Date of Submission:	30/1/25			
Program Execution/ formation/ correction/ ethical practices (06)	Timely Submission (01)	Viva (03)	Experiment Total (10)	Sign with Date

<u>3.1Aim:</u> Implement Coordinate Transformation problem and verify YPR is equivalent to RPY

<u>3.2 Course Outcome:</u>Solve and explain robot's kinematics, trajectories and task planning.

<u>3.3 Learning Objectives:</u> Implement coordinate transformations between different frames.(between mobile and fixed frames)

<u>3.4 Requirement:</u> MATLAB

<u>3.5Related Theory:</u>

Rotation: It is defined as the movement of rigid body about particular axis and is mathematically represented by rotation matrix.

Rotation matrix: A 3X3 matrix is used to describe the rotation of a body w.r.t a standard frame of reference.

Fundamental Rotation matrices:If M is rotated about F then the resulting rotational matrices are called as fundamental rotational matrices.

First Fundamental Rotation Matrix R1(θ)

Let F= {f1,f2,f3}and M={m1,m2,m3} be two RHOCF which are coincident. The first FRM is obtained by the rotation of M frame about f1 axis of F by an amount Ɵ as shown.

$$R1(\theta) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\theta) & -\sin(\theta) \\ 0 & \sin(\theta) & \cos(\theta) \end{bmatrix}$$

Second Fundamental Rotation Matrix R2(θ)

Let F= {f1,f2,f3}and M={m1,m2,m3} be two RHOCF which are coincident. The second FRM is obtained by the rotation of M frame about f2 axis of F by an amount ϴ as shown.

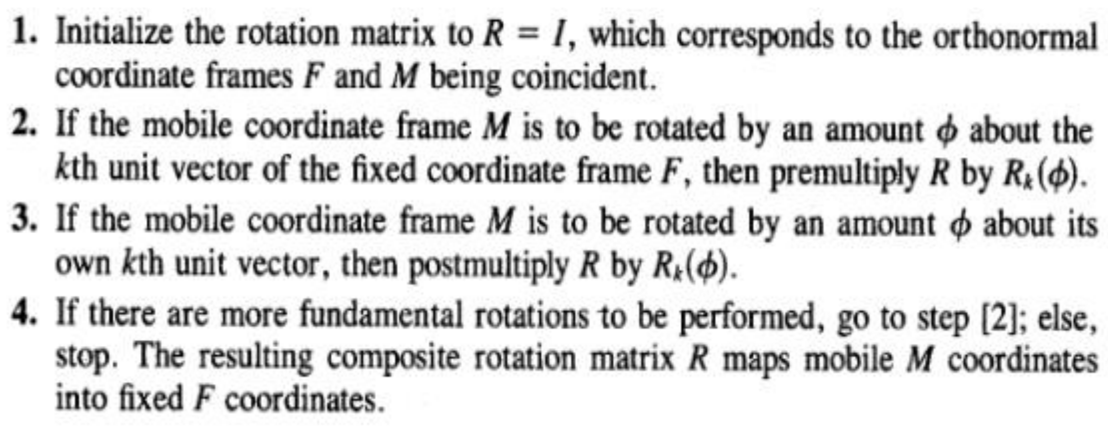
$$R2(\theta) = \begin{bmatrix} \cos(\theta) & 0 & \sin(\theta) \\ 0 & 1 & 0 \\ -\sin(\theta & 0 & \cos(\theta) \end{bmatrix}$$

Third Fundamental Rotation Matrix R3(θ)

Let F= {f1,f2,f3}and M={m1,m2,m3} be two RHOCF which are coincident. The third FRM is obtained by the rotation of M frame about f3 axis of F by an amount ϴ as shown.

$$R3(\theta) = \begin{bmatrix} \cos(\theta) & -\sin(\theta & 0 \\ \sin(\theta & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

3.6 Procedure

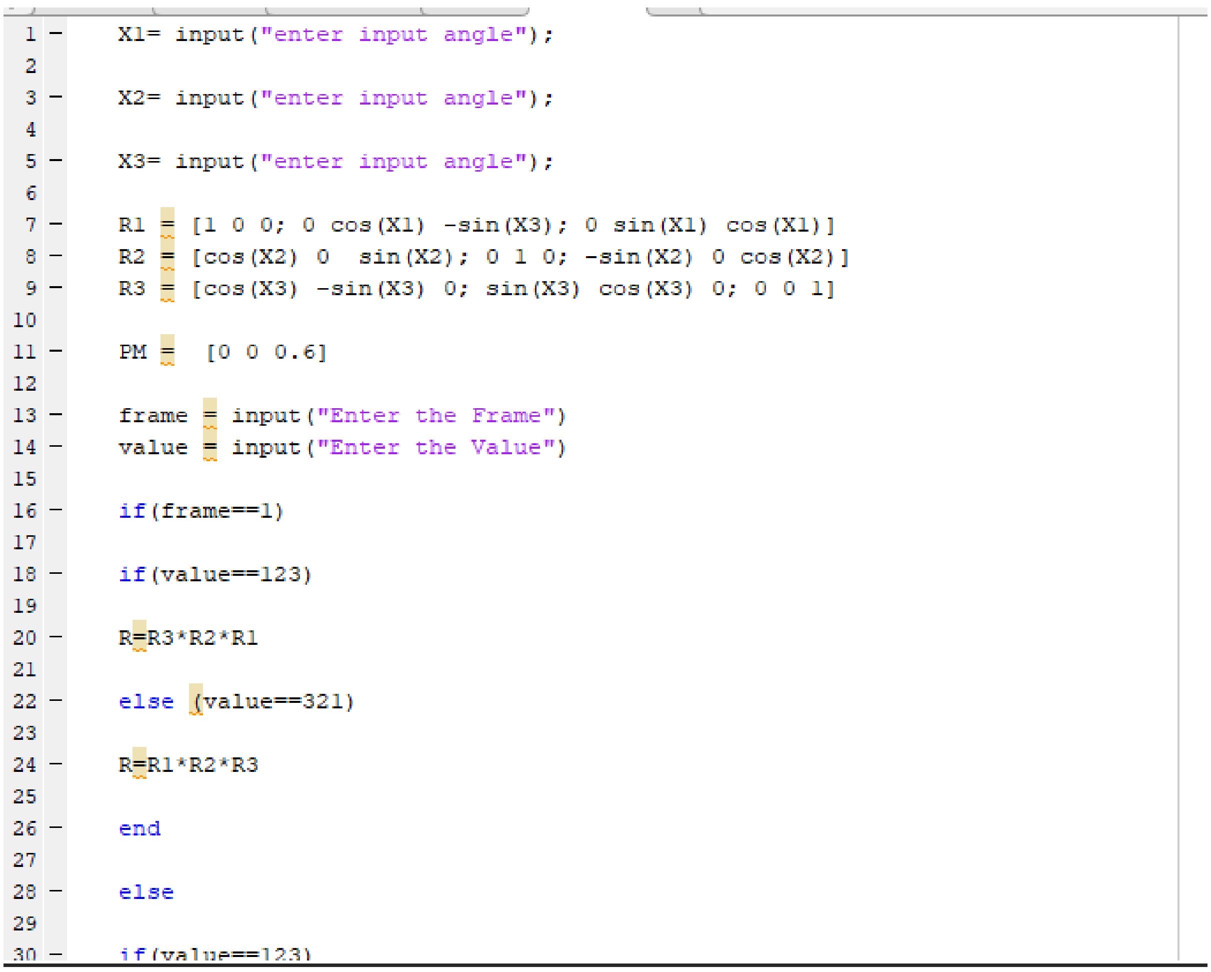


1. Initialize the rotation matrix to $R = I$, which corresponds to the orthonormal coordinate frames F and M being coincident.
2. If the mobile coordinate frame M is to be rotated by an amount ϕ about the kth unit vector of the fixed coordinate frame F, then premultiply R by $R_k(\phi)$.
3. If the mobile coordinate frame M is to be rotated by an amount ϕ about its own kth unit vector, then postmultiply R by $R_k(\phi)$.
4. If there are more fundamental rotations to be performed, go to step [2]; else, stop. The resulting composite rotation matrix R maps mobile M coordinates into fixed F coordinates.

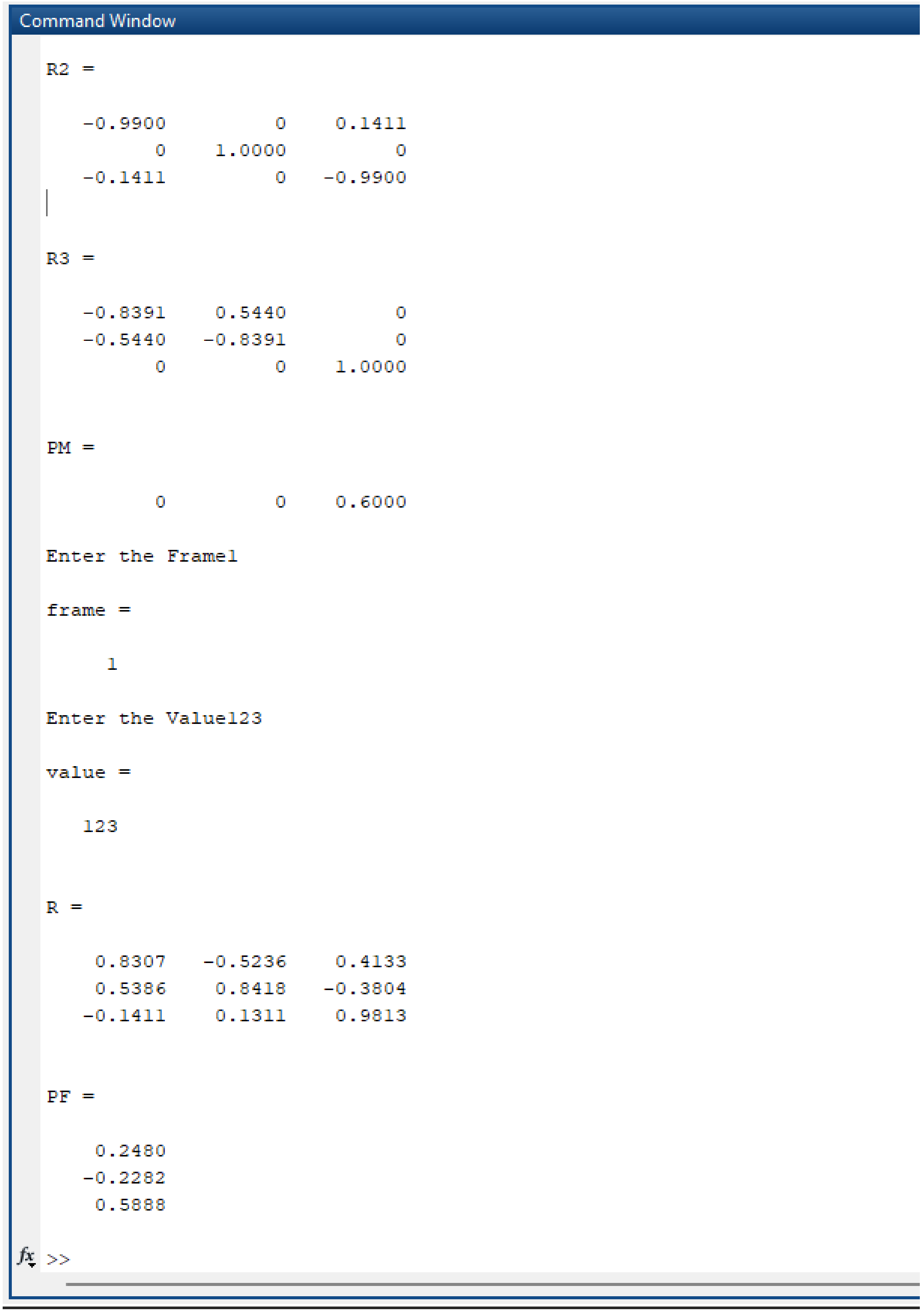
3.7 Program and Output:

```
X1= input("enter input angle");

X2= input("enter input angle");

X3= input("enter input angle");

R1 = [1 0 0; 0 cos(X1) -sin(X3); 0 sin(X1) cos(X1)]

R2 = [cos(X2) 0  sin(X2); 0 1 0; -sin(X2) 0 cos(X2)]
```

```
R3 = [cos(X3) -sin(X3) 0; sin(X3) cos(X3) 0; 0 0 1]
PM =  [0 0 0.6]
frame = input("Enter the Frame")
value = input("Enter the Value")
if(frame==1)
if(value==123)
R=R3*R2*R1
else (value==321)
R=R1*R2*R3
end
else
if(value==123)
R=R1*R2*R3
else(value==231)
R=R2*R3*R1
end
end
PF =R*transpose(PM)
```



```
X1= input("enter input angle");

X2= input("enter input angle");

X3= input("enter input angle");

R1 = [1 0 0; 0 cos(X1) -sin(X3); 0 sin(X1) cos(X1)]
R2 = [cos(X2) 0  sin(X2); 0 1 0; -sin(X2) 0 cos(X2)]
R3 = [cos(X3) -sin(X3) 0; sin(X3) cos(X3) 0; 0 0 1]

PM =  [0 0 0.6]

frame = input("Enter the Frame")
value = input("Enter the Value")

if(frame==1)

if(value==123)

R=R3*R2*R1

else (value==321)

R=R1*R2*R3

end

else

if(value==123)
```


Command Window

```
R2 =

   -0.9900         0    0.1411
         0    1.0000         0
   -0.1411         0   -0.9900


R3 =

   -0.8391    0.5440         0
   -0.5440   -0.8391         0
         0         0    1.0000


PM =

         0         0    0.6000

Enter the Frame1

frame =

     1

Enter the Value123

value =

   123


R =

    0.8307   -0.5236    0.4133
    0.5386    0.8418   -0.3804
   -0.1411    0.1311    0.9813


PF =

    0.2480
   -0.2282
    0.5888

fx >>
```

3.8 Conclusion: Using fundamental rotation matrices Coordinate Transformation is performed and implemented and verified YPR is equivalent to RPY in MATLAB.

Experiment No. – 4				
Date of Performance:	/1/25			
Date of Submission:	30/1/25			
Program Execution/ formation/ correction/ ethical practices (06)	Timely Submission (01)	Viva (03)	Experiment Total (10)	Sign with Date

<u>4.1 Aim:</u> To implement Composite homogeneous coordinate transformations

<u>4.2 Course Outcome</u>: Solve and explain robot's kinematics, trajectories and task planning.

<u>4.3 Learning Objectives:</u>. To derive and construct transformation matrices for basic operations (translation, and rotation) using homogeneous coordinates.

<u>4.4 Requirement:</u>.MATLAB

<u>4.5 Related Theory</u>:

1. Initialize transformation matrix as Identity matrix

2. Represent rotations and translations using separate HCTM matrix

3. Represent Composite rotations in separate HCTM matrix

4. If the mobile coordinate frame M is to be rotated about or translated along a unit vector of the fixed coordinate frame F pre multiply the HCTM by the appropriate fundamental HCTM matrices

5. If the mobile coordinate frame M is to be rotated about or translated along a unit vector of the mobile coordinate frame M post multiply the HCTM by the appropriate fundamental HCTM matrices

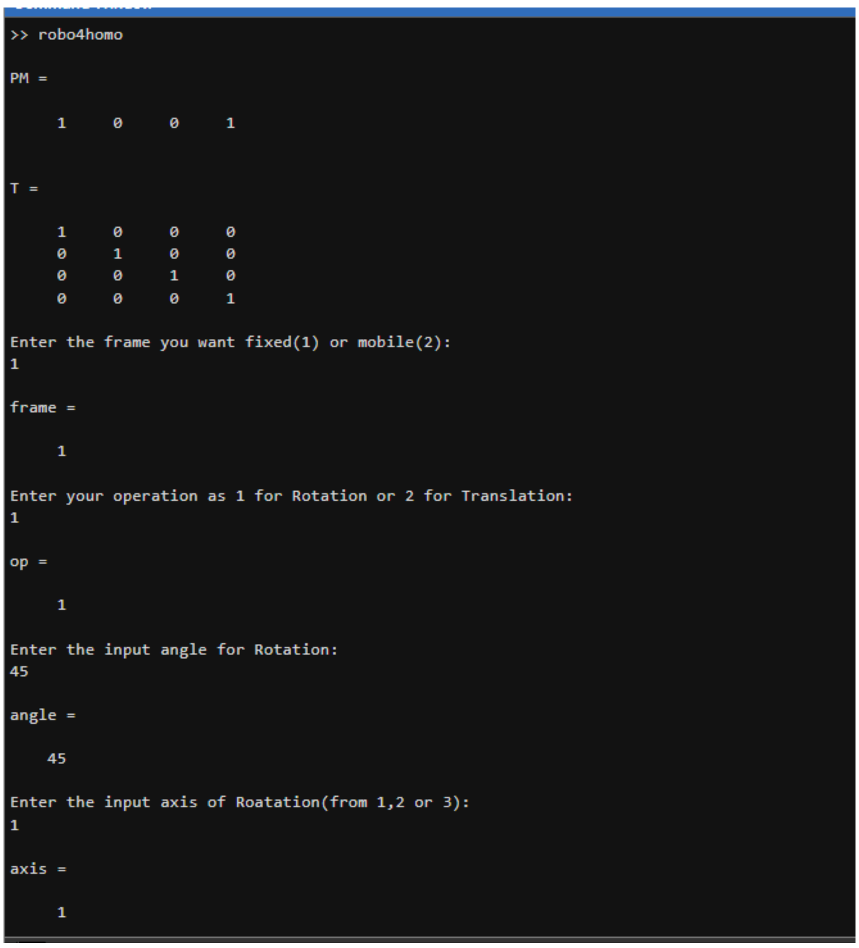
6. If there are more fundamental rotations or translations go to step 4,else stop.The resulting CHCTM matrix T transforms mobile M frame coordinates into F frame coordinates

4.6 Program and Output:

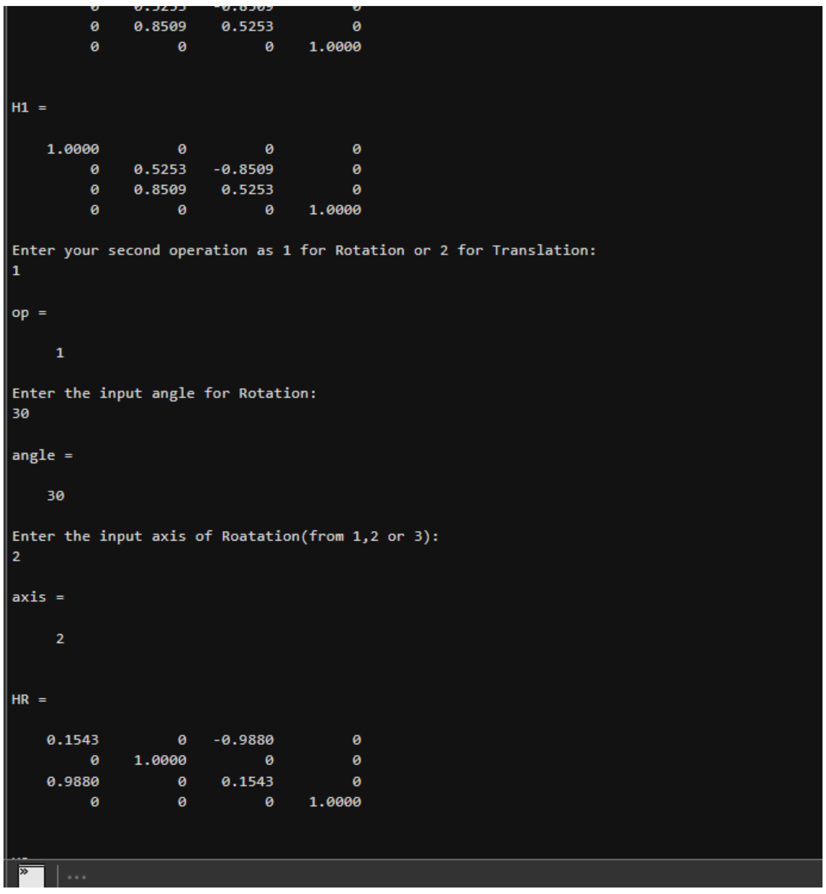
```
PM = [1 0 0 1]

%initialize the transformation matrix as Identity

T = [1 0 0 0; 0 1 0 0; 0 0 1 0; 0 0 0 1]

frame = input("Enter the frame you want fixed(1) or mobile(2): ")

if(frame == 1)

   %First Operation

   op = input("Enter your operation as 1 for Rotation or 2 for Translation: 
")


   if(op == 1)

       angle = input("Enter the input angle for Rotation: ")

       axis = input("Enter the input axis of Roatation(from 1,2 or 3): ")

       if(axis == 1)

           HR = [1 0 0 0 ; 0 cos(angle) -sin(angle) 0; 0 sin(angle) 
cos(angle) 0; 0 0 0 1]

       elseif(axis == 2)

           HR = [cos(angle) 0 sin(angle) 0; 0 1 0 0; -sin(angle) 0 
cos(angle) 0; 0 0 0 1]

       elseif(axis == 3)

           HR = [cos(angle) -sin(angle) 0 0; sin(angle) cos(angle) 0 0; 0 0 
1 1]

       end

       H1 = HR*T

   else(op == 2)

       dist = input("Enter the distance of Translation: ")

       T_axis = input("Enter the Axis on which the previous translation has 
happned(1,2 or 3): ")

       if(T_axis == 1)

           HT = [1 0 0 dist; 0 1 0 0; 0 0 1 0; 0 0 0 1]

       elseif(T_axis == 2)

           HT = [1 0 0 0; 0 1 0 dist; 0 0 1 0; 0 0 0 1]

       elseif(T_axis == 3)

           HT = [1 0 0 0; 0 1 0 0; 0 0 1 dist; 0 0 0 1]

       end
```

```
        H1 = HT*T
    end


    %Second Operation
    op = input("Enter your second operation as 1 for Rotation or 2 for
Translation: ")


    if(op == 1)
        angle = input("Enter the input angle for Rotation: ")
        axis = input("Enter the input axis of Roatation(from 1,2 or 3): ")
        if(axis == 1)
            HR = [1 0 0 0 ; 0 cos(angle) -sin(angle) 0; 0 sin(angle)
cos(angle) 0; 0 0 0 1]
        elseif(axis == 2)
            HR = [cos(angle) 0 sin(angle) 0; 0 1 0 0; -sin(angle) 0
cos(angle) 0; 0 0 0 1]
        elseif(axis == 3)
            HR = [cos(angle) -sin(angle) 0 0; sin(angle) cos(angle) 0 0; 0 0
1 0; 0 0 0 1]
        end
        H2 = HR*T
    else(op == 2)
        dist = input("Enter the distance of Translation: ")
        T_axis = input("Enter the Axis on which the previous translation has
happned(1,2 or 3): ")
        if(T_axis == 1)
            HT = [1 0 0 dist; 0 1 0 0; 0 0 1 0; 0 0 0 1]
        elseif(T_axis == 2)
            HT = [1 0 0 0; 0 1 0 dist; 0 0 1 0; 0 0 0 1]
        elseif(T_axis == 3)
            HT = [1 0 0 0; 0 1 0 0; 0 0 1 dist; 0 0 0 1]
        end
        H2 = HT*T
    end
    H = H2*H1
```

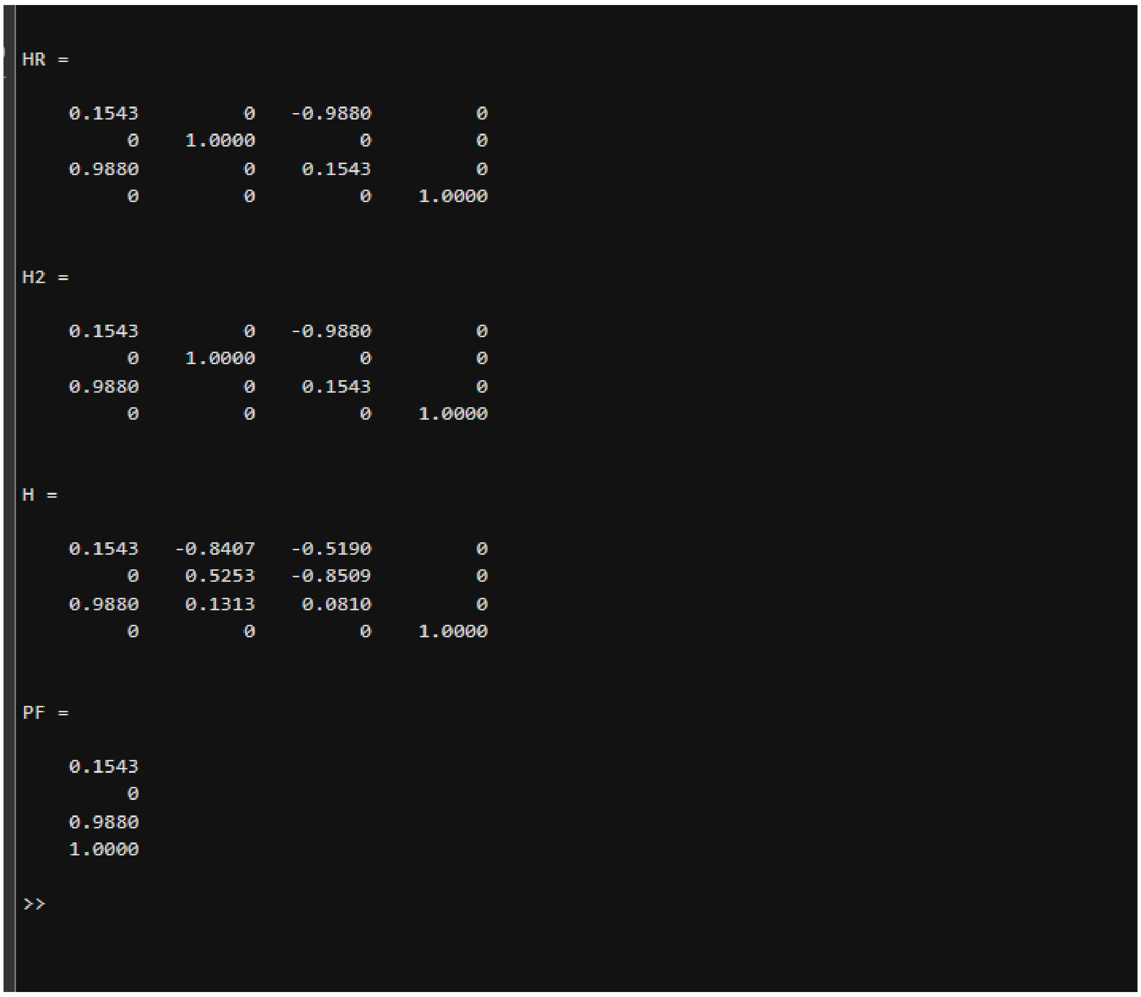
```
   PF = H*transpose(PM)

else(frame == 2)

   %First Operation

   op = input("Enter your operation as 1 for Rotation or 2 for Translation:
")


   if(op == 1)

       angle = input("Enter the input angle for Rotation: ")

       axis = input("Enter the input axis of Roatation(from 1,2 or 3): ")

       if(axis == 1)

           HR = [1 0 0 0 ; 0 cos(angle) -sin(angle) 0; 0 sin(angle)
cos(angle) 0; 0 0 0 1]

       elseif(axis == 2)

           HR = [cos(angle) 0 sin(angle) 0; 0 1 0 0; -sin(angle) 0
cos(angle) 0; 0 0 0 1]

       elseif(axis == 3)

           HR = [cos(angle) -sin(angle) 0 0; sin(angle) cos(angle) 0 0; 0 0
1 1]

       end

       H1 = T*HR

   else(op == 2)

       dist = input("Enter the distance of Translation: ")

       T_axis = input("Enter the Axis on which the previous translation has
happned(1,2 or 3): ")

       if(T_axis == 1)

           HT = [1 0 0 dist; 0 1 0 0; 0 0 1 0; 0 0 0 1]

       elseif(T_axis == 2)

           HT = [1 0 0 0; 0 1 0 dist; 0 0 1 0; 0 0 0 1]

       elseif(T_axis == 3)

           HT = [1 0 0 0; 0 1 0 0; 0 0 1 dist; 0 0 0 1]

       end

       H1 = T*HT

   end


   %Second Operation
   op = input("Enter your second operation as 1 for Rotation or 2 for
Translation: ")


   if(op == 1)

       angle = input("Enter the input angle for Rotation: ")

       axis = input("Enter the input axis of Roatation(from 1,2 or 3): ")

       if(axis == 1)

           HR = [1 0 0 0 ; 0 cos(angle) -sin(angle) 0; 0 sin(angle)
cos(angle) 0; 0 0 0 1]

       elseif(axis == 2)

           HR = [cos(angle) 0 sin(angle) 0; 0 1 0 0; -sin(angle) 0
cos(angle) 0; 0 0 0 1]

       elseif(axis == 3)

           HR = [cos(angle) -sin(angle) 0 0; sin(angle) cos(angle) 0 0; 0 0
1 0; 0 0 0 1]

       end

       H2 = T*HR

   else(op == 2)

       dist = input("Enter the distance of Translation: ")

       T_axis = input("Enter the Axis on which the previous translation has
happned(1,2 or 3): ")

       if(T_axis == 1)

           HT = [1 0 0 dist; 0 1 0 0; 0 0 1 0; 0 0 0 1]

       elseif(T_axis == 2)

           HT = [1 0 0 0; 0 1 0 dist; 0 0 1 0; 0 0 0 1]

       elseif(T_axis == 3)

           HT = [1 0 0 0; 0 1 0 0; 0 0 1 dist; 0 0 0 1]

       end

       H2 = T*HT

   end

   H = H1*H2

   PF = H*transpose(PM)

end
```



```
>> robo4homo

PM =

     1     0     0     1


T =

     1     0     0     0
     0     1     0     0
     0     0     1     0
     0     0     0     1

Enter the frame you want fixed(1) or mobile(2):
1

frame =

     1

Enter your operation as 1 for Rotation or 2 for Translation:
1

op =

     1

Enter the input angle for Rotation:
45

angle =

    45

Enter the input axis of Roatation(from 1,2 or 3):
1

axis =

     1
```



```
         0    0.8509    0.5253         0
         0         0         0    1.0000


H1 =

    1.0000         0         0         0
         0    0.5253   -0.8509         0
         0    0.8509    0.5253         0
         0         0         0    1.0000

Enter your second operation as 1 for Rotation or 2 for Translation:
1

op =

     1

Enter the input angle for Rotation:
30

angle =

    30

Enter the input axis of Roatation(from 1,2 or 3):
2

axis =

     2


HR =

    0.1543         0   -0.9880         0
         0    1.0000         0         0
    0.9880         0    0.1543         0
         0         0         0    1.0000
```



```
HR =

    0.1543         0   -0.9880         0
         0    1.0000         0         0
    0.9880         0    0.1543         0
         0         0         0    1.0000


H2 =

    0.1543         0   -0.9880         0
         0    1.0000         0         0
    0.9880         0    0.1543         0
         0         0         0    1.0000


H =

    0.1543   -0.8407   -0.5190         0
         0    0.5253   -0.8509         0
    0.9880    0.1313    0.0810         0
         0         0         0    1.0000


PF =

    0.1543
         0
    0.9880
    1.0000

>>
```

4.7 Conclusion: Composite Homogeneous Transformations are performed about different axes and in different order.It is observed that the final orientation of the tool depends on the order in which the rotation and translation is performed and the axes on which the operations are performed.

EXPERIMENT NO: 5				
Date of Performance:	**3/4/25**			
Date of Submission:	**3/4/25**			
Program Execution/ formation/ correction/ ethical practices (06)	Timely Submission (01)	Viva (03)	Experiment Total (10)	Sign with Date

Experiment No:5

A program to implementDirect Kinematics

5.1Aim: To solve the direct kinematics of a 3-axis planar robotic arm and 4 axis SCARA robot and study DH algorithm.

5.2 Course Outcome: Solve and explain robot's kinematics, trajectories and task planning.

5.3 Learning Objectives: To Solve the end-effectors position and orientation of the 3-axis planar robot and the 4-axis SCARA robot using the DH parameters.

5.4 Requirement: MATLAB

5.5 Related Theory:

In Direct Kinematics joint variables of a given robot are given and determine the position and orientation of the tool with respect to a coordinate frame attached to the robot base using DH algorithm.

DH algorithms a two pass algorithm in which the first pass assign a set of RHOCF to the distal end of each link. On the second pass the values for the kinematic parameters are determined.

Once the link coordinates are assigned, we can then transform coordinate frame k to coordinate frame k-1 using a homogeneous coordinate transformation matrix. By multiplying several of these coordinate transformation matrices we get a coordinate transformation matrix which transforms or maps tool coordinates into base coordinates. This composite homogeneous coordinate transformation matrix is called the arm matrix.

DH algorithm: Denavit and Hartenberg proposed a systematic notation for assigning right handed orthogonal coordinate frames ,one to each link in an open kinematic chain of links. To assign coordinate frames to the links of the robotic manipulator, let Lk be the frame associated with link k. The coordinate frames are assigned to the links using the following procedure

1. Number the joints from 1 to n starting with the base and ending with the tool yaw, pitch and roll in that order.
2. Assign a right handed orthonormal coordinate frame Lo to the robot base, making sure that zo aligns with the axis of joint 1.set k = 1.
3. Align zk with the axis of joint int k +1
4. Locate the origin of Lk at the intersection of the zk and zk -1 axes .If they do not intersect ,use the intersection of zk with a common normal between zk and *zk*-1
5. *Select* xk to be orthogonal to both zk and zk -1 .If zk and zk -1 are parallelpoint xk away from zk-1
6. Select yk to form a right handed ortonormal coordinate frame Lk

7. Set k = k+1 .If k < n ,go to step 2; else ,continue
8. Set the origin of Ln at the tool tip.Alignzn with the approach vector, yn with the sliding vector , and Xn with the normal vector of the tool
   Set k = 1
9. Locate point bk at the intersection of the xk and Zk-1 axes .If they do not intersect ,use the intersection of xk with a common normal between xk and zk-1
10. Compute qk as the angle of rotation from xk-1 to xk measured about Zk-1
11. Compute dk as the distance from the origin of frame Lk-1 to point bk measured along Zk-1
12. Compute Bk as the distance from poinf bk to the origin of frame Lk measured along xk
13. Compute ax as the angle of rotation from zk- 1 to zk measured about xk
14. Set k = k+1 If k n, go to step 8 , else stop,

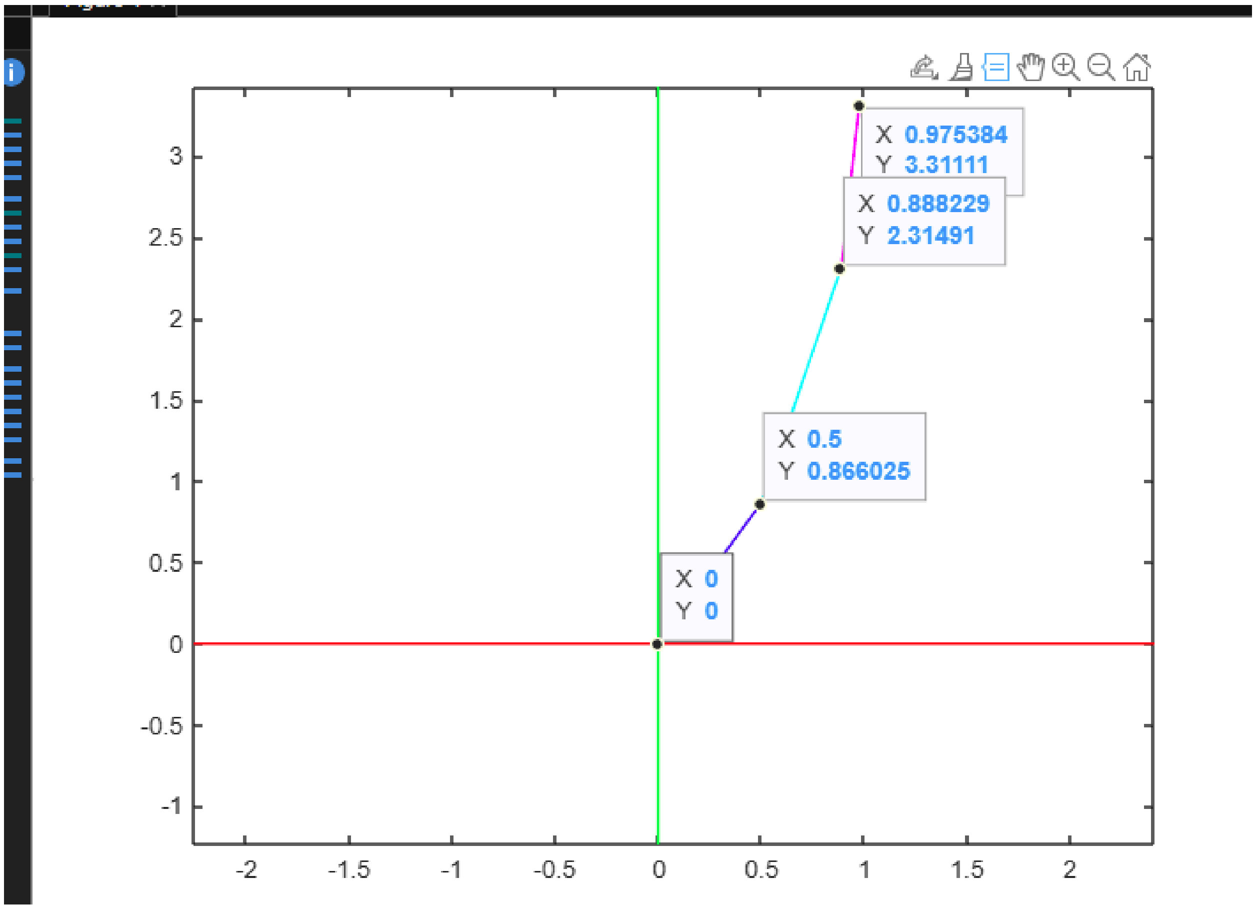
This is called as DH algorithm.

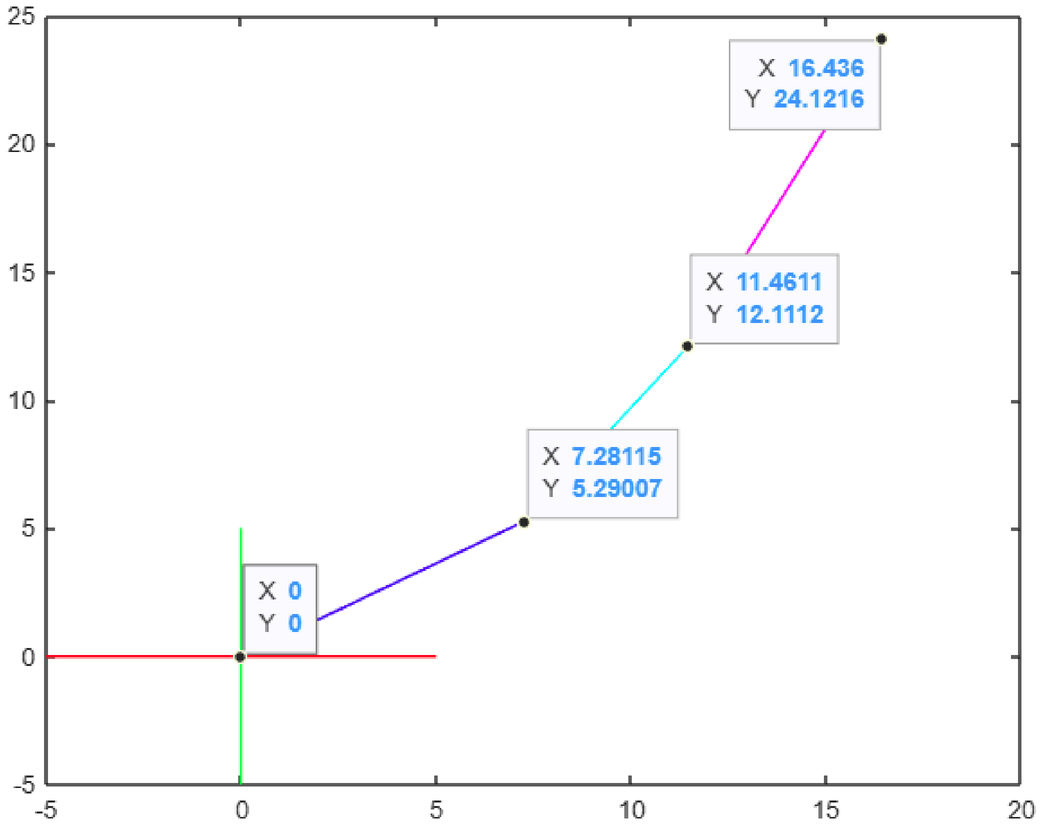
5.6Program and Output:

5a - 3-Axis code

```
% 3-axis code
a1 = 1
a2 = 1.5
a3 = 1
t1 = pi/3
t2 = pi/12
t3 = pi/18
a1x = a1*cos(t1)
a2x = a1x + a2*cos(t1+t2)
```

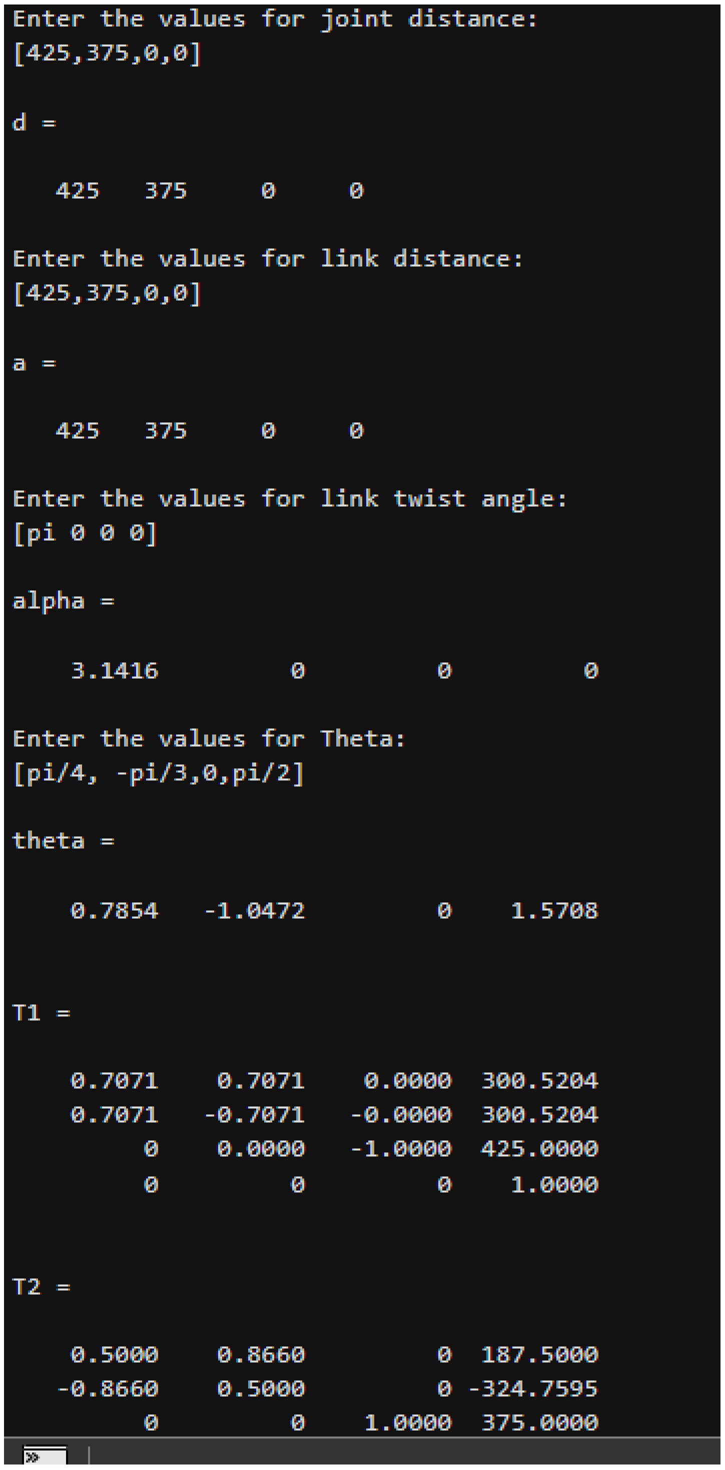
```
a3x = a2x + a3*cos(t1+t2+t3)
a1y = a1*sin(t1)
a2y = a1y + a2*sin(t1+t2)
a3y = a2y + a3*sin(t1+t2+t3)
% Px = a1x + a2x + a3x
% Py = a1y + a2y + a3y
xx = [-5, 5]
xy = [0, 0]
yx = [0, 0]
yy = [-5, 5]
link1x = [0, a1x]
link1y = [0, a1y]
link2x = [a1x, a2x]
link2y = [a1y, a2y]
link3x = [a2x, a3x]
link3y = [a2y, a3y]
plot(xx, xy, 'r', yx, yy, 'g', link1x, link1y, 'b', link2x, link2y, 'c',
link3x, link3y,'m')
```



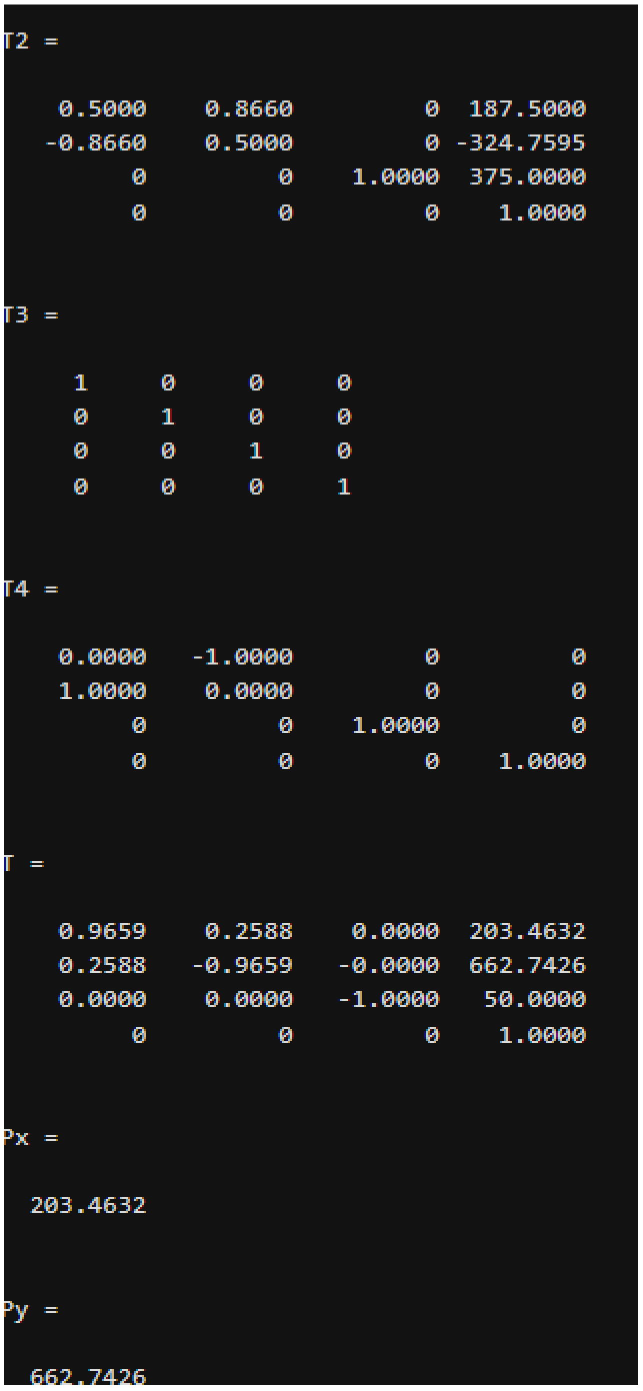

X 16.436
Y 24.1216
X 11.4611
Y 12.1112
X 7.28115
Y 5.29007
X 0
Y 0
25
20
15
10
5
0
-5
-5
0
5
10
15
20

5b - 4-Axis

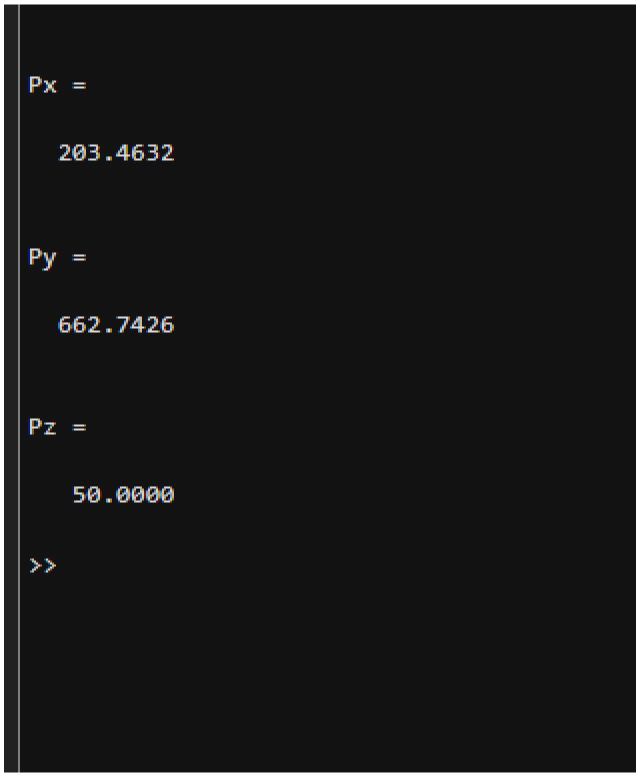
```
%direct kinematics for 3(planar robot) & 4(SCARA) axis robot
%4-Axis code
q = input("Enter the values for joint variables: ")
d = input("Enter the values for joint distance: ")
a = input("Enter the values for link distance: ")
alpha = input("Enter the values for link twist angle: ")
theta = input("Enter the values for Theta: ")
T1 = [cos(theta(1)), -cos(alpha(1))*sin(theta(1)),
sin(alpha(1))*sin(theta(1)), a(1)*cos(theta(1)); sin(theta(1)),
cos(alpha(1))*cos(theta(1)), -sin(alpha(1))*cos(theta(1)),
a(1)*sin(theta(1)); 0, sin(alpha(1)), cos(alpha(1)), d(1); 0, 0, 0, 1]
T2 = [cos(theta(2)), -cos(alpha(2))*sin(theta(2)),
sin(alpha(2))*sin(theta(2)), a(2)*cos(theta(2)); sin(theta(2)),
cos(alpha(2))*cos(theta(2)), -sin(alpha(2))*cos(theta(2)),
a(2)*sin(theta(2)); 0, sin(alpha(2)), cos(alpha(2)), d(2); 0, 0, 0, 1]
T3 = [cos(theta(3)), -cos(alpha(3))*sin(theta(3)),
sin(alpha(3))*sin(theta(3)), a(3)*cos(theta(3)); sin(theta(3)),
cos(alpha(3))*cos(theta(3)), -sin(alpha(3))*cos(theta(3)),
a(3)*sin(theta(3)); 0, sin(alpha(3)), cos(alpha(3)), d(3); 0, 0, 0, 1]
T4 = [cos(theta(4)), -cos(alpha(4))*sin(theta(4)),
sin(alpha(4))*sin(theta(4)), a(4)*cos(theta(4)); sin(theta(4)),
cos(alpha(4))*cos(theta(4)), -sin(alpha(4))*cos(theta(4)),
a(4)*sin(theta(4)); 0, sin(alpha(4)), cos(alpha(4)), d(4); 0, 0, 0, 1]
T = T1*T2*T3*T4
Px = T(1,4)
Py = T(2,4)
Pz = T(3,4)
```



```
Enter the values for joint distance:
[425,375,0,0]

d =

   425   375     0     0

Enter the values for link distance:
[425,375,0,0]

a =

   425   375     0     0

Enter the values for link twist angle:
[pi 0 0 0]

alpha =

    3.1416         0         0         0

Enter the values for Theta:
[pi/4, -pi/3,0,pi/2]

theta =

    0.7854   -1.0472         0    1.5708


T1 =

    0.7071    0.7071    0.0000  300.5204
    0.7071   -0.7071   -0.0000  300.5204
         0    0.0000   -1.0000  425.0000
         0         0         0    1.0000


T2 =

    0.5000    0.8660         0  187.5000
   -0.8660    0.5000         0 -324.7595
         0         0    1.0000  375.0000
```



```
T2 =

    0.5000    0.8660         0  187.5000
   -0.8660    0.5000         0 -324.7595
         0         0    1.0000  375.0000
         0         0         0    1.0000


T3 =

     1     0     0     0
     0     1     0     0
     0     0     1     0
     0     0     0     1


T4 =

    0.0000   -1.0000         0         0
    1.0000    0.0000         0         0
         0         0    1.0000         0
         0         0         0    1.0000


T =

    0.9659    0.2588    0.0000  203.4632
    0.2588   -0.9659   -0.0000  662.7426
    0.0000    0.0000   -1.0000   50.0000
         0         0         0    1.0000


Px =

  203.4632


Py =

  662.7426
```


```
Px =

  203.4632


Py =

  662.7426


Pz =

   50.0000

>>
```

FOR,
Given q=[π/4, -π/4,120,π/2]
d=[877,0,q3, 200]
a= [425,375,0,0]
alpha = [π 0 0 0]
Theta =[π/4, -π/3,0,π/2]

5.7 Conclusion: Direct Kinematics of 3 axis planar articulated robot implemented and studied DH algorithm.

EXPERIMENT NO: 6				
Date of Performance:	**3/4/25**			
Date of Submission:	**3/4/25**			
Program Execution/ formation/ correction/ ethical practices (06)	Timely Submission (01)	Viva (03)	Experiment Total (10)	Sign with Date

Experiment No:6

A program to implementInverse Kinematics

6.1Aim: Write a program to implement the inverse kinematicsSolution of the given SCARA Robot and determine the Joint variable for the given TCV.

6.2 Course Outcome: Solve and explain robot's kinematics, trajectories and task planning.

6.3 Learning Objectives: Implement a program to solve the inverse kinematics of a SCARA robot given a desired TCP (position and orientation).

6.4 Requirement: MATLAB

6.5 Related Theory: Inverse Kinematics Problem ,Given a desired position P and orientation R forthe tooI find the values for the joint variables q which saisfy the arm equation given by

$$T_{base}^{tool}\ (q) = [R\ P\ 0\ 1]$$

The inverse kinematics problem is more difficult than the direct kinematics problem, because no single explicit systematic procedure analogous to the DH algorithm is available .As a result each robot or generally simlilar class of robots has to be treated separately. It makes Robot more versatile..

Solution to an IK problem are not unique. There are multiple solutions

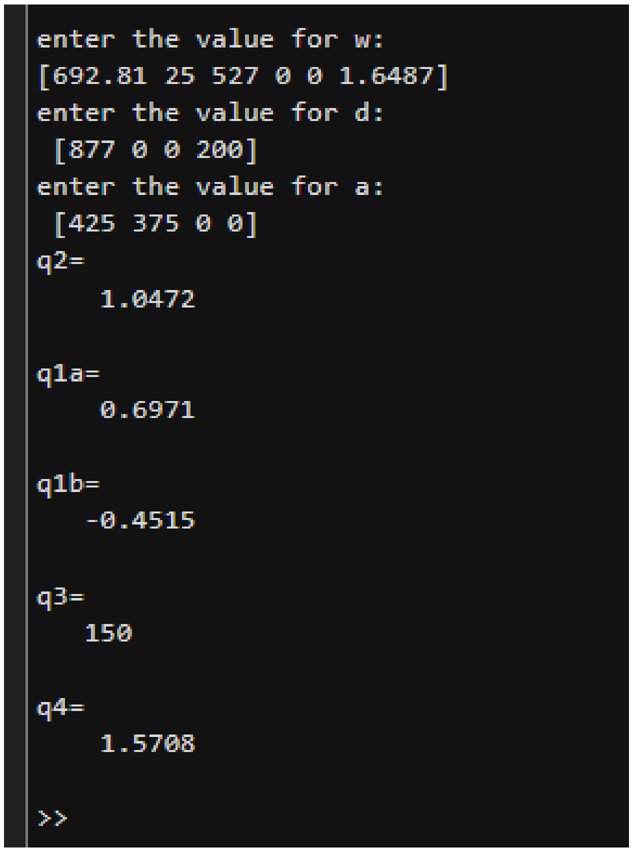
In tool configuration space the two solutions are identical, because they produce the same p and R but in joint space they are clearly distinct.Typically the elbow up solution is preffered because it reduces the chance of collision.

6.6Procedure

1. Start
2. Input tool configuration vector
3. Extract joint variables
4. stop.

6.7Program and Output:

```
w = input("enter the value for w: ");
d = input("enter the value for d: ");
a = input("enter the value for a: ");
q2 = acos(((w(1)^2) + (w(2)^2) - (a(1)^2) - (a(2)^2))/(2*a(1)*a(2)));
Y = (((a(2)*sin(q2)*w(1)) + (a(1) + a(2)*cos(q2))*w(2)));
X = ((a(1) + (2)*cos(q2)) *w(1)) - (a(2)*sin(q2)*(w(2)));
q1a = atan2(Y,X);
q1b = atan2(((a(2)*sin(-q2)*w(1)) + ((a(1) + a(2)*cos(-q2))*w(2))),
(((a(1) + a(2)*cos(-q2))*w(1))-(a(2)*sin(-q2)*w(2))));
q3 = d(1)-d(4)-w(3)';
q4 = pi*log(w(6));
disp("q2= ")
disp(q2)
disp("q1a= ")
disp(q1a)
disp("q1b= ")
disp(q1b)
disp("q3= ")
disp(q3)
disp("q4= ")
disp(q4)
```



```
enter the value for w:
[692.81 25 527 0 0 1.6487]
enter the value for d:
 [877 0 0 200]
enter the value for a:
 [425 375 0 0]
q2=
    1.0472

q1a=
    0.6971

q1b=
   -0.4515

q3=
   150

q4=
    1.5708

>>
```

6.8Conclusion:

Inverse Kinematics of 4 axis SCARA robot implemented using MATLAB tools. It’s observed that Inverse Kinematics is not unique

Mahavir Education Trust's

Shah & Anchor Kutchhi Engineering College

Chembur, Mumbai 400 088

(An Autonomous Institute Affiliated to University of Mumbai)

EXPERIMENT NO:7				
Date of Performance:	28/2/25			
Date of Submission:	15/04/25			
Program Execution/ formation/ correction/ ethical practices (06)	Timely Submission (01)	Viva (03)	Experiment Total (10)	Sign with Date

Experiment No:7

A PROGRAM TO IMPLEMENT TRAJECTORY PLANNING

7.1Aim: To write a program to implement Trajectory Planning.

7.2 Course Outcome: Select suitable image processing, task planning and trajectory planning algorithms for robot applications.

7.3 Learning Objectives: To implement trajectory planning algorithms for robot manipulator.

7.4 Requirement: MATLAB

7.5 Related Theory:

A tool path is defined as the collection of the sequence of configurations a robot makes to go from one place to another without regard to the timing of these configurations. Hence path is a purely spatial representation. If temporal information is added by specifying the times at which the tool must be at various points along the path, then path becomes a trajectory. Trajectory planning can be planned in joint space and cartesian space.

Motion diagram represents graphically described time depending functions of changing position, velocity, acceleration and jerk of manipulator link gripper or tool. Theoretically, these functions may have different forms. The best way to describe motion diagrams using mathematical function is using splines, i.e. curves composed from several polynomials. Manipulator motion may be described in different coordinates, e.g.in joint coordinates, base coordinates or world coordinates. Base and world coordinates are normally cartesian coordinates. In some cases the base coordinates may also be polar(cylindrical or spherical) or angular coordinates. Motion will be planned in the world cartesian coordinates (where also the robot's work is described), but the manipulator will realize the motion in joint coordinates. It is known that velocity is the derivative of position function, acceleration is the derivative of velocity function and jerk is the derivative of the acceleration function. Because of this, to describe motion along the trajectory, the following equations (3.1…3.4) consisting of polynomials are used. Polynomials are the best functions because their derivatives can be easily found.

$$s(t) = c_0 + c_1 t + c_2 t^2 + c_3 t^3$$

$$v(t) = \frac{ds}{dt} = c_1 + 2c_2 t + 3c_3 t^2$$

$$a(t) = \frac{dv}{dt} = 2c_2 + 6c_3 t$$

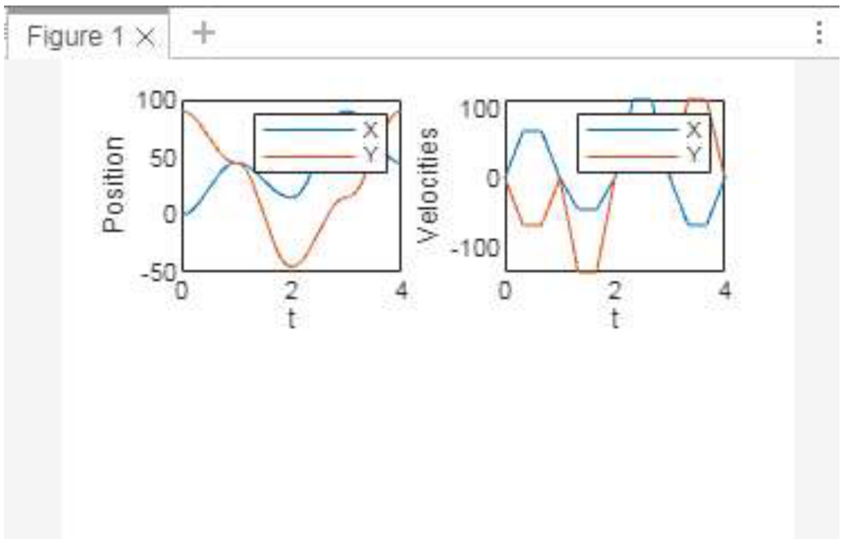
$$j(t) = \frac{da}{dt} = 6c_3$$

where, coefficients $c_{0,} c_1, \ldots c_3$ define the character of motion.

The graphical form of motion (position, velocity, acceleration and jerk) description is named the motion diagram. Motion needed to be planned for the robot after the program instruction is read from program memory and before the drives of the manipulator start motion. Multiple limits are considered on trajectory planning (travel time, maximal values of velocity and acceleration and jerk. The maximum velocity limit can depend on technology or emergency conditions. Higher velocity is more dangerous for people working in the same room with a robot. Higher acceleration and deceleration values need drive motors with higher output power and manipulator construction standing higher forces and torques. Jerk value can be limited by smooth acceleration and deceleration. Motion diagrams can be described by one third polynomial of position, second order of polynomials of velocity and linear acceleration (deceleration) function. In simple cases the motion diagram of the trajectory is described by a simple function. The velocity diagram has the form of triangle or trapezoid.

7.6 Program and Output:

```
% Define waypoints
wpts = [0 45 15 90 45; 90 45 -45 15 90];

% Generate trajectory with trapezoidal velocity profile
[q, qd, qdd, tvec, pp] = trapveltraj(wpts, 501);

% Plot positions
subplot(2,2,1)
plot(tvec, q)
xlabel('t')
ylabel('Position')
legend('X', 'Y')

% Plot velocities
subplot(2,2,2)
plot(tvec, qd)
xlabel('t')
ylabel('Velocities')
legend('X', 'Y')
```



7.7 Conclusion: Planning of trajectory for trapezoidal velocity diagrams is implemented using MATLAB. When the velocity curve has the form of trapezium, after initial acceleration, the manipulator moves with a constant speed. Because the jerk is not limited 86 (has theoretical indefinite value) the position function can be described by the spline consisting of the second order polynomials.

Mahavir Education Trust's

Shah & Anchor Kutchhi Engineering College

Chembur, Mumbai 400 088

(An Autonomous Institute Affiliated to University of Mumbai)

EXPERIMENT NO:8				
Date of Performance:	**21/03/25**			
Date of Submission:	**15/04/25**			
Program Execution/ formation/ correction/ ethical practices (06)	Timely Submission (01)	Viva (03)	Experiment Total (10)	Sign with Date

Experiment No:8
A program to implement template matching

8.1Aim: To write a program to implement Template Matching.

8.2 Course Outcome: Apply Image processing algorithms for robot vision.

8.3 Learning Objectives: Analyze the limitations and challenges associated with Template Matching and explore possible solutions or improvements.

8.4 Requirement: MATLAB

8.5 Related Theory:

Template matching two techniques are there

1. Performance Index Method.
2. Normalized Cross Correlation method.

It is a technique which is used to recognize whether a given part belongs to a particular class of or not. All the parts which belongs to a `particular class of parts is kept in front of the camera one by one at a time, their images are obtained digitized & stored in a library of parts in memory as templates Ti (k,j) of size (mo x no). Now the class of parts (scene) is captured by the camera image is digitized & is given by I(k j) of size (m X n) and stored in the memory.

Now in the memory there are a number of templatesTi (k j) and one image I (k,j). To search whether a given part (template) belongs to the class of parts (image) or not take the template & scan the gray scale image or digital image I (k , j) from left to right & top to bottom using raster scanning method.

At each time of translation obtain & index performance called as performance index Pi(x,y).

Whenever the template matches with the part in the image, the index=0. the match has been folm"d search is successful the location of the part is also found. The performance index is given by

$$Pi(x,y)= \sum_{k=1}^{Mo} \sum_{j=1}^{no} I(k=x,j+y)-Ti(k,j)$$

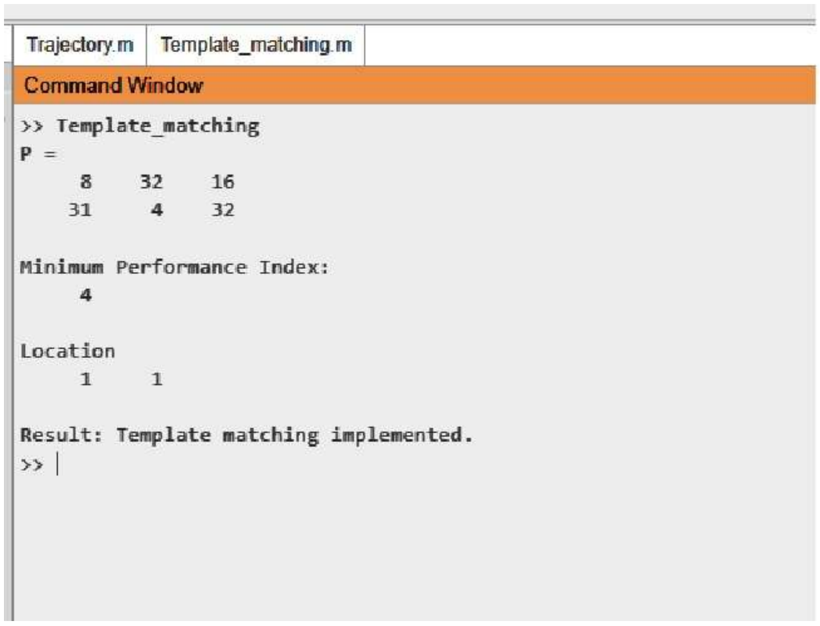
8.6Procedure:

1. set I=1,x=0,y=0,e>0
2. compute Pi(x,y)
3. set y=y+1;if y<=(n-n0) go to step 2
4. set y=0,x=x+1,if x<=(m-m0) go to step2
5. set found =false.

8.7Program and Output:

```
% Template Matching
i = [2 1 0 0 3;
    0 0 5 0 0;
    0 4 0 6 0;
    1 0 5 0 0]; % 4x5 Image Matrix

t = [0 4 0;
    3 0 5;
    0 4 0]; % 3x3 Template Matrix

% Initialize Performance Matrix
P = [0 0 0;
    0 0 0]; % Matrix for storing performance index

m = 5; % Rows of the image
n = 4; % Columns of the image
m0 = 3; % Rows of the template
n0 = 3; % Columns of the template

% Compute Performance Index
for k = 1:n0
   for j = 1:m0
      for x = 1:n-n0+1
         for y = 1:m-m0+1
```

```
            P(x, y) = P(x, y) + abs(i(k + x - 1, j + y - 1) - t(k, j));
        end
    end
  end
end

% Display the Performance Matrix
disp("P = ")
disp(P)

% Find Minimum Performance Index
Y = min(P(:));
disp("Minimum Performance Index: ")
disp(Y)

% Find the location of the best match
[r, c] = size(P);
for i = 1:r
  for j = 1:c
    if P(i, j) == Y
      disp("Location")
      disp([i-1, j-1 ])
    end
  end
end

% Result
disp("Result: Template matching implemented.")
```



```
>> Template_matching
P =
     8    32    16
    31     4    32

Minimum Performance Index:
     4

Location
     1     1

Result: Template matching implemented.
>> 
```

8.8Conclusion:

Template Matching even though it’s a simple technique, one limitation is that the two images being compared should have the same average intensities