IIITD&M KANCHEEPURAM

ME3007 Robotics and Automation Practice

LAB RECORD

BY

RAHUL [me22b2042]

FOR,
Dr. Sreekumar M, PhD (IIT Madras),
Faculty of Mechanical Engineering,
Indian Institute of Information Technology Design
and manufacturing-Chennai-600127
Email: msk@iiitdm.ac.in

July – Nov, 2024

Table of Content

Experiment: 1

Motion analysis of 2D and 3D mechanism:

Aim:
To perform the motion analysis of 2D and 3D mechanisms.

Objective: To draw the kinematic diagram of provided mechanisms, visualizing the desired motions and perform motion analysis successfully.

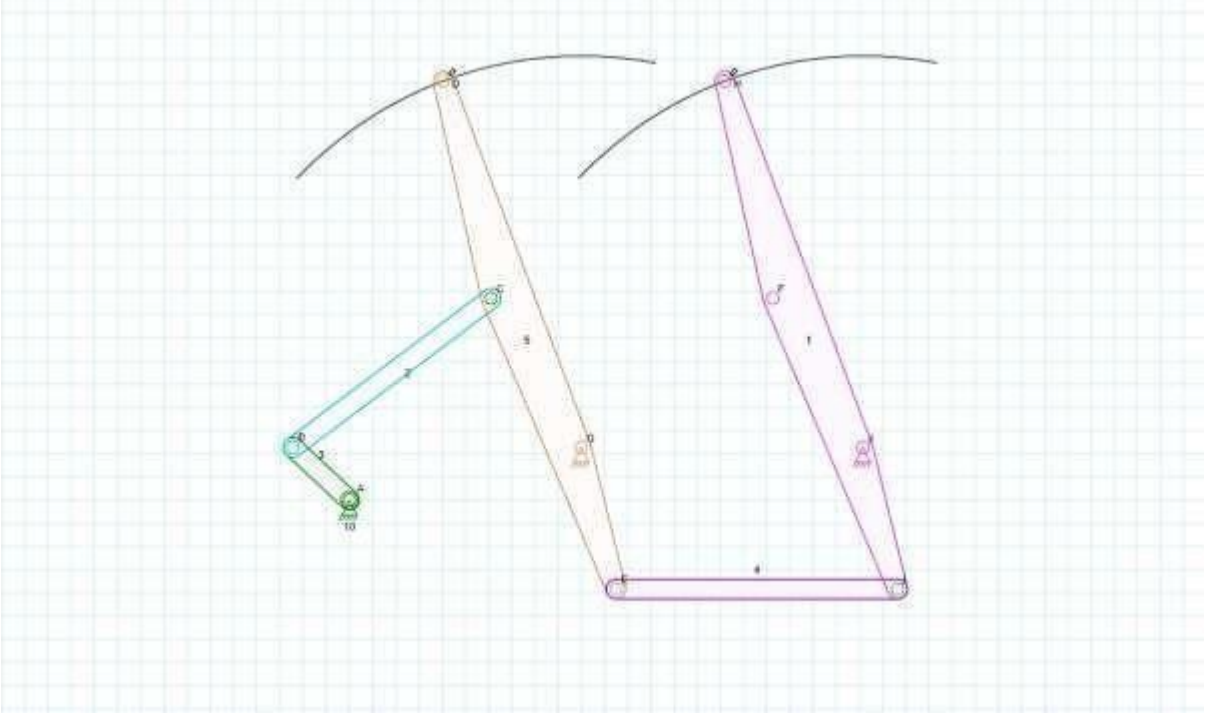
Requirements: Linkage software

Procedure:

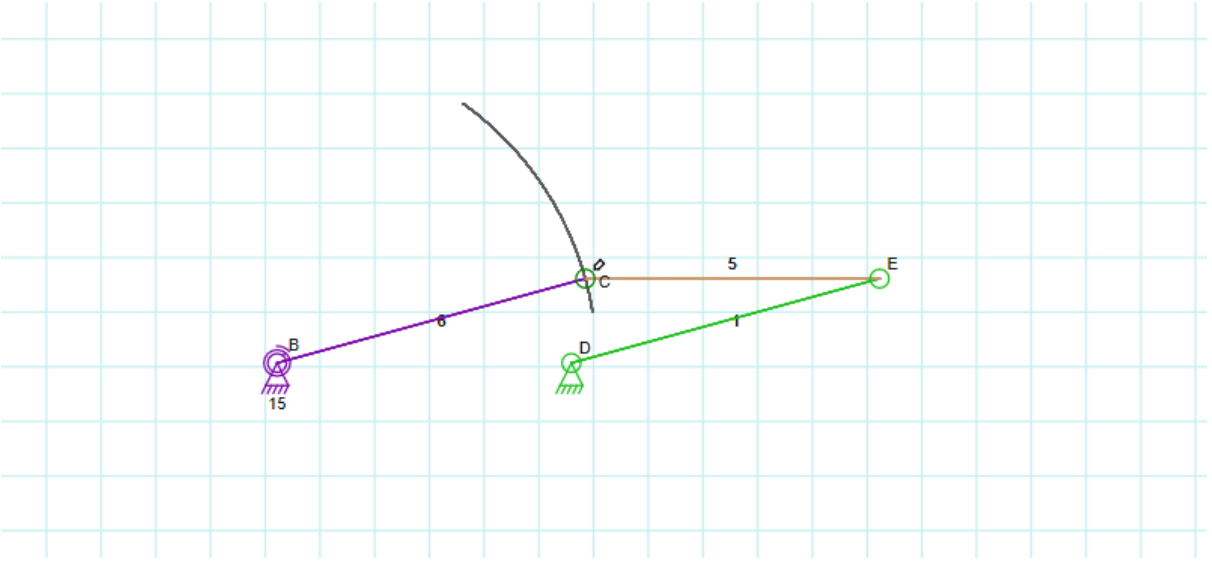
1. Start by opening the LINKAGE software to begin constructing the mechanisms.
2. Carefully review the problem statement, paying particular attention to the number of joints and links needed to create the mechanism. This will guide the design process.
3. Choose the appropriate types of links (such as rods, sliders, etc.) and joints (like pivots or sliders) for each component of the mechanism. The selection should align with the desired motion and functionality outlined in the problem statement.
4. Arrange the selected links in their correct positions within the workspace. Connect them using the specified joints to form the required mechanism. Ensure that the arrangement reflects the intended mechanical design
5. Input the given dimensions from the problem statement; if dimensions are not provided, choose appropriate values.
6. Draw the kinematic diagram of each mechanism, using the required links, joints and anchors. And connect all the links using the appropriate joints.
7. Run a simulation of the mechanism to verify that it functions correctly with the given dimensions. Check for any inconsistencies or errors in the configuration that might cause the mechanism to malfunction.
8. To avoid toggling adjust the dimensions of the links.
9. Apply constraints on angles or rotations if needed. Otherwise, simulate the mechanism and observe the execution path.
10. To visualize the motion path of a specific joint, right-click on the joint, select 'Properties', and enable the 'Draw Motion Path' option. This will help in analyzing the trajectory and behavior of the joint within the mechanism.

Observation:

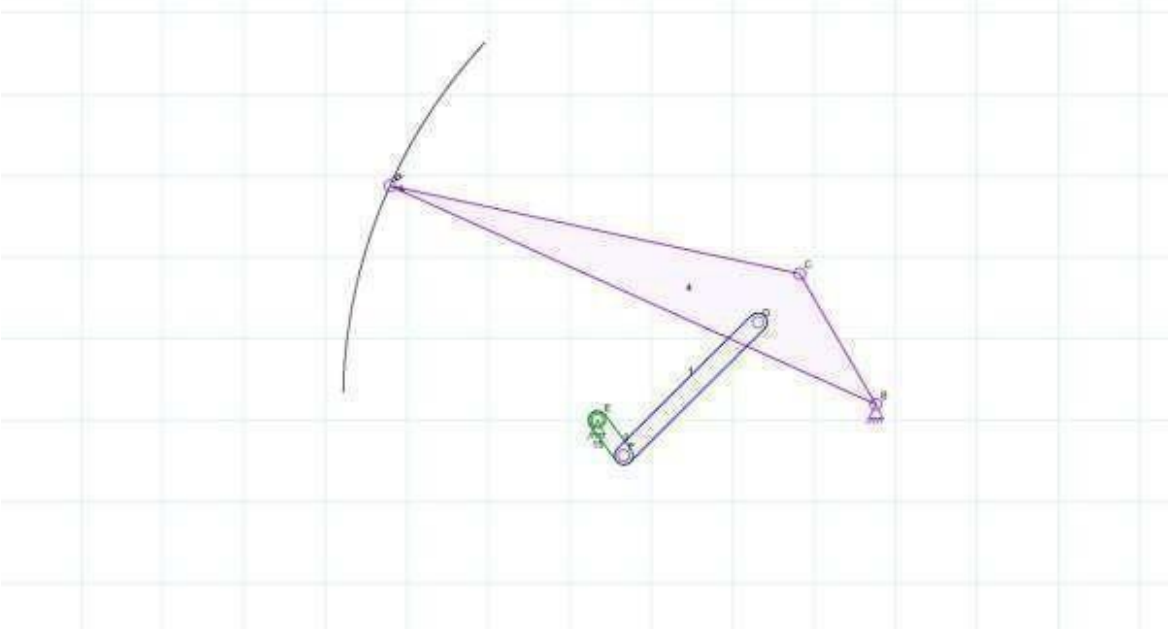
1) **Wiper Mechanism:** This is a four-bar mechanism designed to convert the rotational motion of the wiper motor crank into reciprocating, or back-and-forth, motion. It does this through a system of interconnected bars that collaboratively move the wiper blade across the windshield, ensuring an effective and controlled sweeping action for optimal clearing.



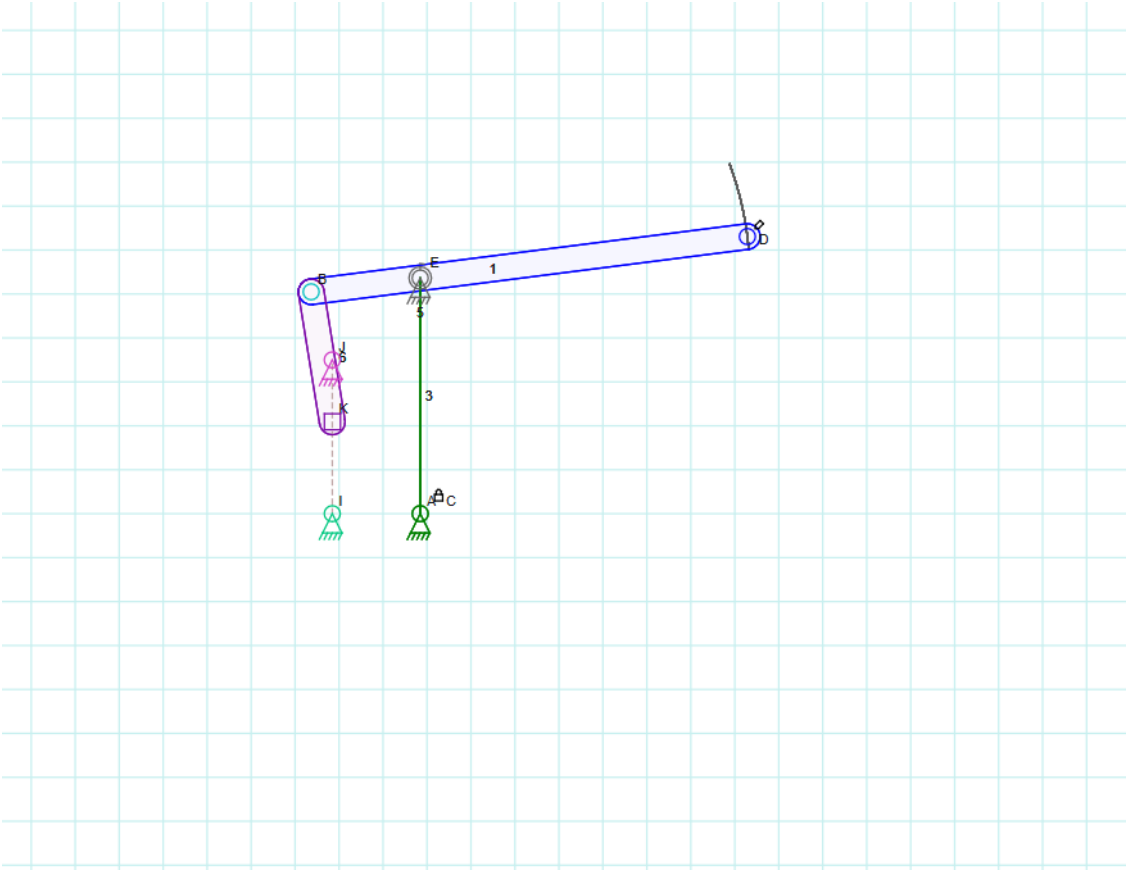
2) **Back and Forth Slider:** The rotating crank drives the motion of the slider, moving it back and forth. This motion is achieved through the precise positioning of the links and the appropriate selection of joints. An additional link is pivoted to provide support and ensure the return of the slider, completing the cycle.



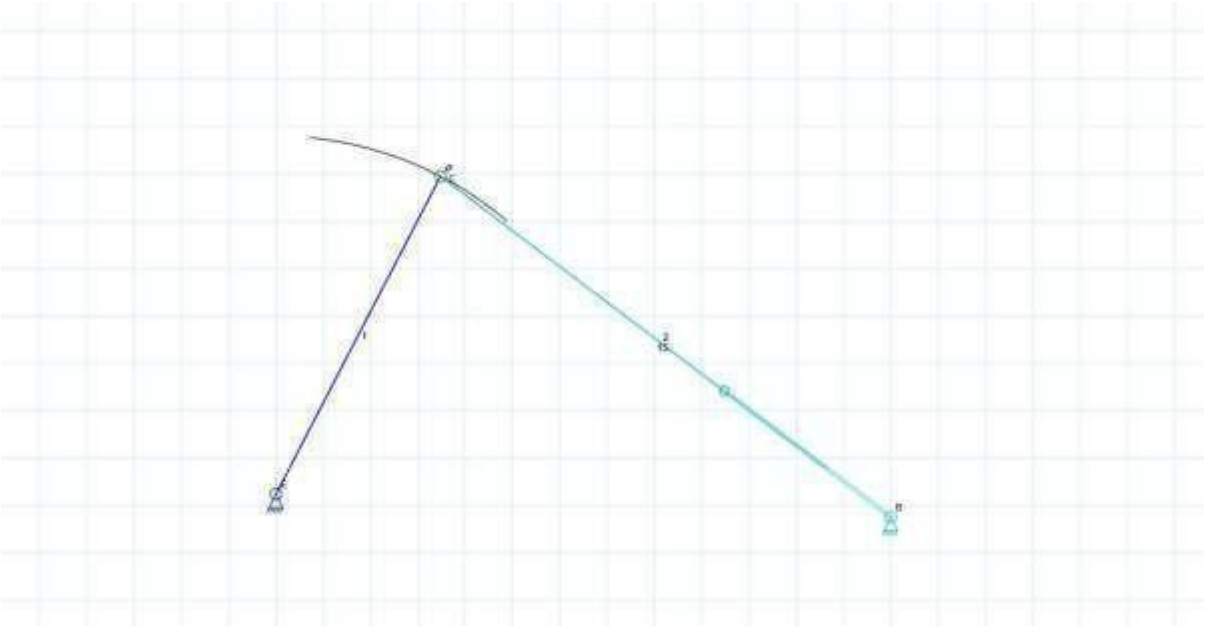
3) PCB Shear Cutting Machine: The PCB shear cutting machine employs a lever arm mechanism linked to a cutting blade. When the lever arm is pressed downward, it pivots around a fixed point, converting this rotational movement into a linear motion for the blade. This linear motion produces a precise shear force required to cut printed circuit boards (PCBs). The design utilizes a straightforward kinematic linkage that effectively transforms the rotational input from the lever arm into a controlled linear cutting action, ensuring both accuracy and efficiency in the PCB cutting process.



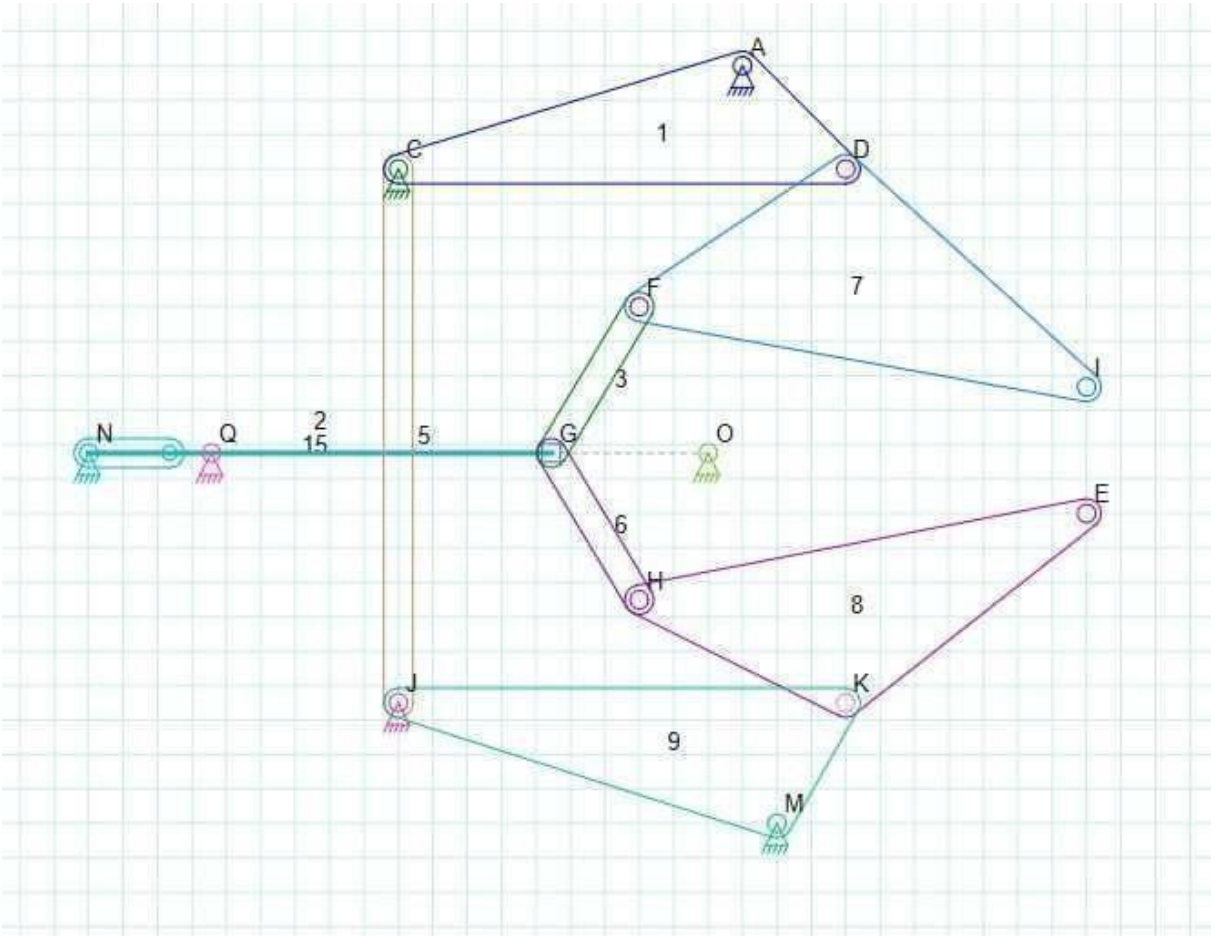
4) Hand Pump Mechanism: In a hand pump mechanism, the handle is attached to a crank, which is connected to a connecting rod. This rod is then linked to a piston. When the handle is moved, it rotates the crank, which in turn moves the connecting rod and drives the piston up and down. As the piston moves up, it creates suction that draws fluid into the pump. As it moves down, it discharges the fluid. This coordinated motion of the connected components converts the manual effort applied to the handle into the reciprocating motion of the piston, allowing for the transfer of fluid.



5) Truck Lifting Mechanism: This system employs a hinge joint to connect the truck bed to the chassis, permitting it to pivot. A hydraulic cylinder is positioned between the bed and the chassis. When hydraulic fluid is pressurized, the cylinder extends, applying force to the bed link and converting hydraulic pressure into linear motion. This motion raises the bed, allowing it to tilt for dumping. The mechanism is based on simple kinematics, utilizing a single lifting link and a hydraulic cylinder to enable controlled vertical movement of the bed.



6) Two-Finger Gripper Mechanism: In this mechanism, a linear actuator is attached to a common joint shared by the two links of each finger. Each link is pivoted at one end, allowing it to rotate around that point. This rotational movement facilitates the opening and closing of the gripper's end effector, enabling it to grasp or release objects effectively.



Result:

Successfully simulated all 6 mechanism’s, and the motion analysis ofall 6 mechanism’s has been done.

Inferences:

This experiment provided a solid understanding of 2D mechanisms by constructing six distinct mechanisms using the LINKAGE software. Through this process, we gained insights into the use of various joints and links, which offered a clear visual comprehension of the relationship between different input motions and the resulting mechanisms. By adjusting the different links, we were able to observe various significant mechanisms and their practical applications.

Experiment: 2

Kinematic Analysis of Robot Manipulators: Frame Transformation & DH Matrix Computation

Aim: To perform Kinematic analysis of Robot Manipulators

Objective: To study the kinematic behaviour of Robot Manipulators by analysing the Frame Transformation and by computing the DH Matrix.

Requirements: RoboAnalyser software.

Procedure:

1. Mapping and Transformation with Respect to Global Frame:

In this experiment, the global frame remains fixed, while the local frame is adjusted through specified translations and rotations. After applying these transformations relative to the global frame, the final position is recorded.

Transformations to apply (relative to Global Frame):

- Translate +100 mm along the Y-axis
- Rotate +90° around the X-axis
- Translate +200 mm along the Z-axis
- Rotate +40° around the Y-axis

2. Mapping and Transformation with Respect to Both Global and Local Frames:

In this experiment, translations and rotations are applied to both the local and global frames. After performing the specified transformations, the final position is recorded.

Transformations to apply:

- Translate +100 mm along the Y-axis (global frame)
- Rotate +90° around the X-axis (local frame)
- Translate +200 mm along the Z-axis (global frame)
- Rotate +40° around the Y-axis (local frame)

3. Determining and Validating the DH Matrix for Serial Manipulators:

Set the degrees of freedom and select the type of robot. Run the simulation to find the robot's final position. Use the "Visualize DH" option to view the DH matrices for each link relative to its predecessor. Then, calculate the DH matrix for the final frame relative to the base frame by multiplying the matrices for each link in sequence. Compare the computed matrix with the final DH matrix to validate it.

Serial Manipulators to Analyze:

- RR (Planar)
- RRR (Planar)
- RRR (Spatial)
- RRPR (Spatial)

Results:

The results of each of the above tasks is given as follows:

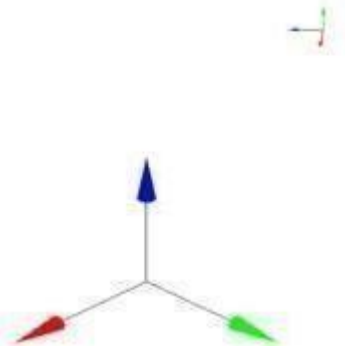
1. **Mapping and transportation w.r.t global frame**:

Current Matrix | Show Position | Visualise

	1	2	3	4
▸	0.766	0.6428	0	19.2836
	0	0	-1	0
	-0.6428	0.766	0	22.9813
	0	0	0	1

(R4*(T3*(R2*(T1*I))))

2. **Mapping and transportation w.r.t both global and local frame**:



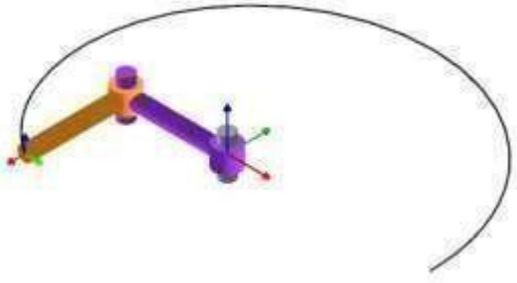
Current Matrix | Show Position | Visualise

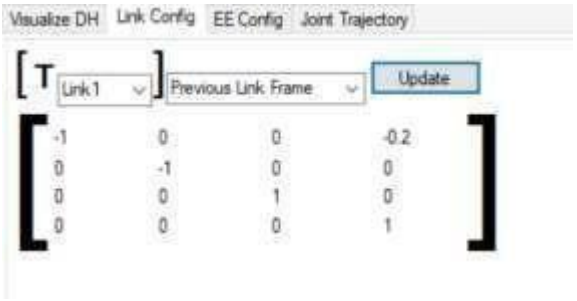
	1	2	3	4
▸	0.766	0	0.6428	0
	0.6428	0	-0.766	10
	0	1	0	20
	0	0	0	1

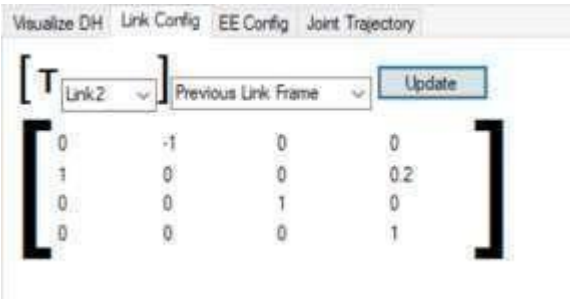
((T3*((T1*I)*R2))*R4)

3. **Finding DH Matrix and Validating it**:

   a) RR manipulator (planar)-

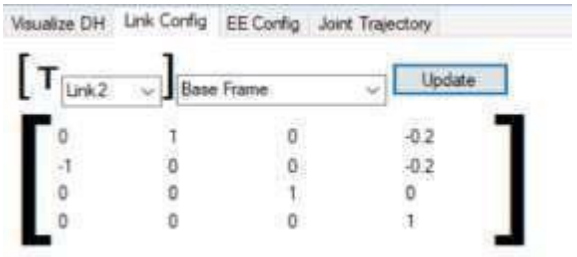






${}^{0}T_{1}$ ${}^{1}T_{2}$

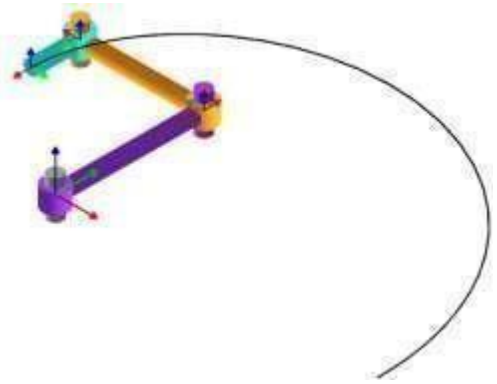
${}^{0}T_{2}$

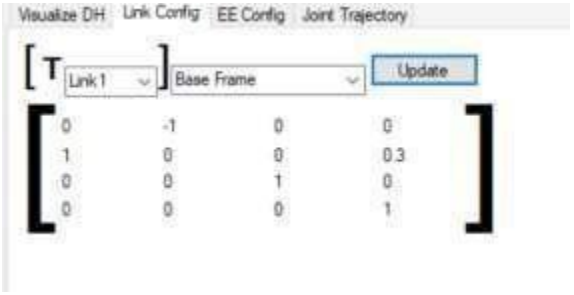


Verification:

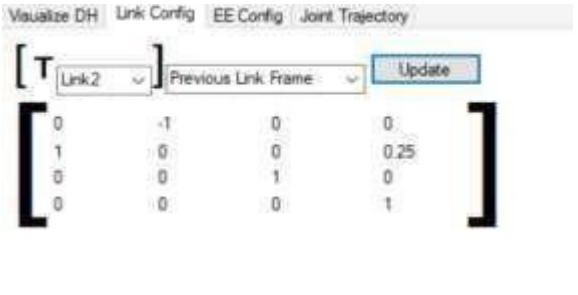
$$\begin{pmatrix} -1 & 0 & 0 & -0.2 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 0 & -1 & 0 & 0 \\ 1 & 0 & 0 & 0.2 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 0 & 1 & 0 & \frac{-1}{5} \\ -1 & 0 & 0 & \frac{-1}{5} \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

b) RRR manipulator (planar)-



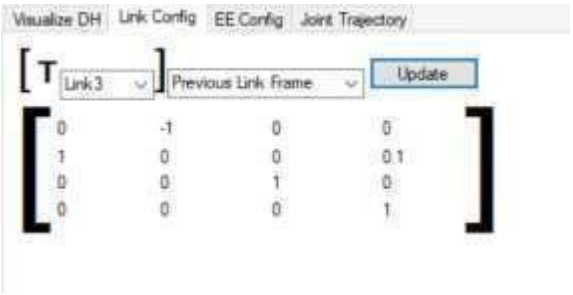


$^0T_1$

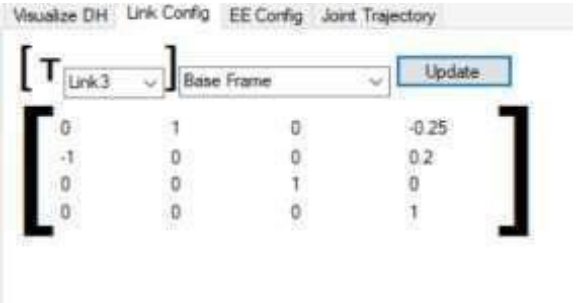


$^1T_2$

$^2T_3$



$^0T_3$

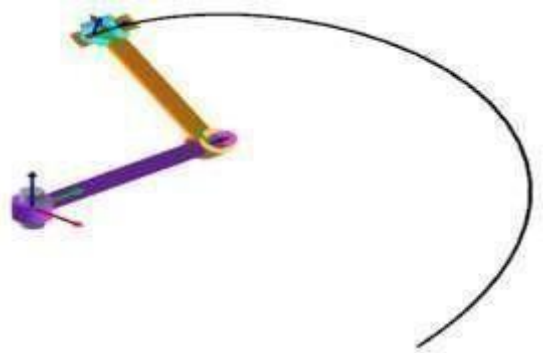


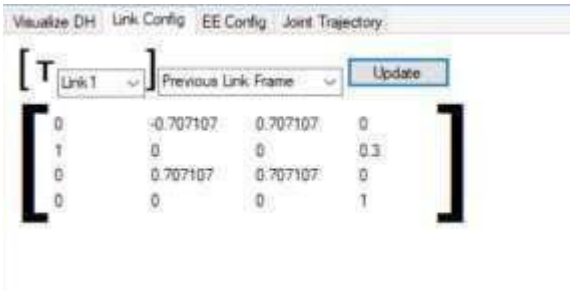
Verification:

$$\begin{pmatrix} 0 & -1 & 0 & 0 \\ 1 & 0 & 0 & 0.3 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 0 & -1 & 0 & 0 \\ 1 & 0 & 0 & 0.25 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} -1 & 0 & 0 & \frac{-1}{4} \\ 0 & -1 & 0 & \frac{3}{10} \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$\begin{pmatrix} -1 & 0 & 0 & \frac{-1}{4} \\ 0 & -1 & 0 & 0.3 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 0 & -1 & 0 & 0 \\ 1 & 0 & 0 & 0.1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 0 & 1 & 0 & \frac{-1}{4} \\ -1 & 0 & 0 & \frac{1}{5} \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

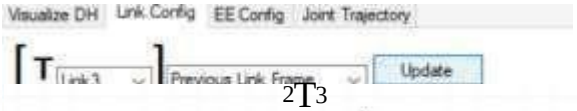
c) RRR manipulator (spatial)-



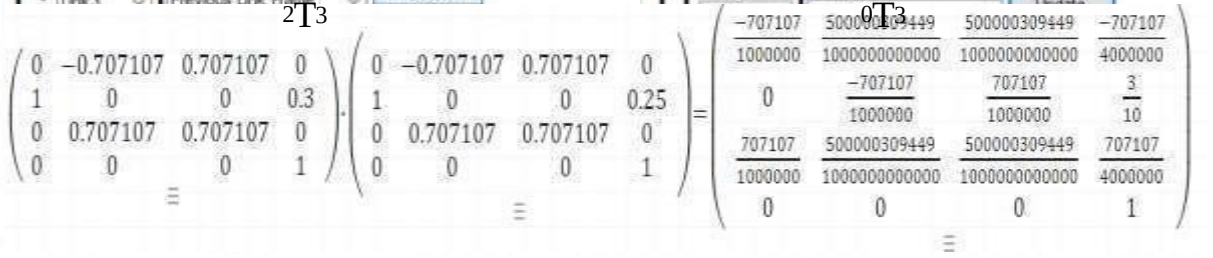


$^0T_1$

$^1T_2$



$^2T_3$

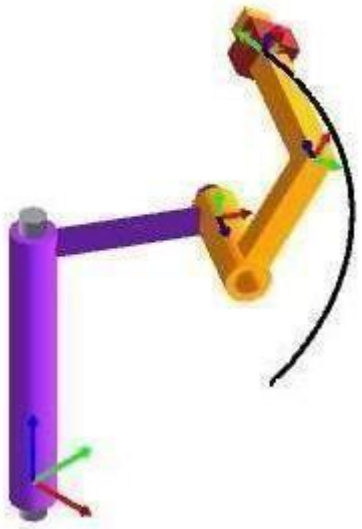


$$\begin{pmatrix} 0 & -0.707107 & 0.707107 & 0 \\ 1 & 0 & 0 & 0.3 \\ 0 & 0.707107 & 0.707107 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 0 & -0.707107 & 0.707107 & 0 \\ 1 & 0 & 0 & 0.25 \\ 0 & 0.707107 & 0.707107 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} \frac{-707107}{1000000} & \frac{500000309449}{1000000000000} & \frac{500000309449}{1000000000000} & \frac{-707107}{4000000} \\ 0 & \frac{-707107}{1000000} & \frac{707107}{1000000} & \frac{3}{10} \\ \frac{707107}{1000000} & \frac{500000309449}{1000000000000} & \frac{500000309449}{1000000000000} & \frac{707107}{4000000} \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$^0T_3$

Verification:

d) RRPR manipulator (spatial)-



T Link1 Previous Link Frame Update

0.5	0	0.866025	0.1
0.866025	0	-0.5	0.173205
0	1	0	0.3
0	0	0	1

$^{0}T_{1}$

T Link2 Previous Link Frame Update

0.5	0.866025	0	0.075
0.866025	-0.5	0	0.129904
0	0	-1	0.1
0	0	0	1

$^{1}T_{2}$

$^{2}T_{3}$

$^{3}T_{4}$

T Link3 Previous Link Frame Update

1	0	0	0
0	0.707107	-0.707107	0
0	0.707107	0.707107	0.2
0	0	0	1

T Link4 Previous Link Frame Update

0.5	0	0.866025	0
0.866025	0	-0.5	0
0	1	0	0
0	0	0	1

T Link4 Base Frame Update

-0.140165	-0.918559	0.369599	0.050897
0.981972	-0.176777	-0.066942	0.288157
0.126826	0.353553	0.926777	0.429904
0	0	0	1

$^{0}T_{4}$

Inferences: The Frame Transformation w.r.t. either global frame or the local frame had been studied, and the DH matrices were found for a few serial manipulators.

Experiment: 3

Kinematic Analysis of Robot Manipulators: Kinematic Analysis of Industrial Robot Manipulator : ABB IRB120

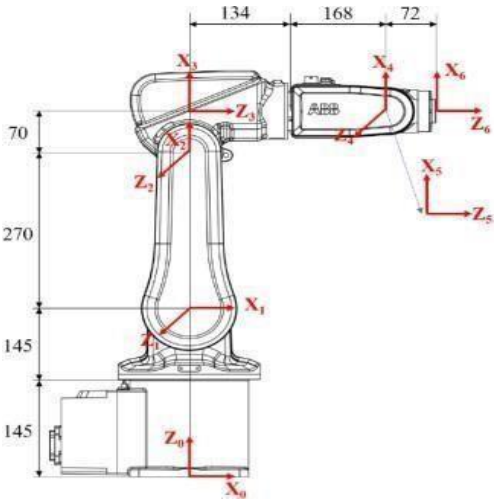
Aim: To perform kinematic analysis of industrial robot manipulator.

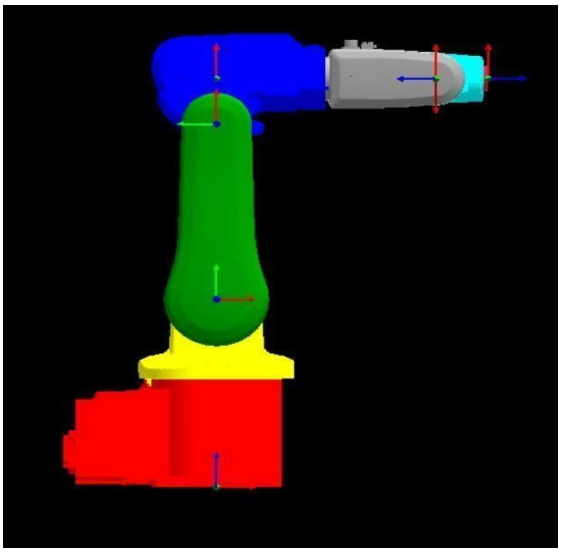
Objective: To validate the DH matrices for the ABB IRB120 industrial robot manipulator at its base frame home postion

Requirements: RoboAnalyser software.

Procedure:

1. Open the RoboAnalyser software and select the Virtual Robot option. The default robotwill be the ABB IRB120.
2. Adjust the values in the Joint Control settings to achieve the desired configuration(refer to the provided images for guidance).





3. Compute the DH matrix for the individual frames relative to their previous frames using the formula: $[^{initial}T_{final}]=$

$$[cos(\theta i) \; -sin(\theta i)cos(\alpha i) \; sin(\theta i)sin(\alpha i) \; (ai)cos(\theta i) \; sin(\theta i) \; cos(\theta i)cos(\alpha i) \; -cos(\theta i)sin(\alpha i) \; (ai)sin(\theta i) \; 0 \; sin(\alpha i) \; cos(\alpha i) \; di \; 0 \; 0 \; 0 \; 1]$$

The DH parameters, which describe the relationship between consecutive links in the robot manipulator, are:

- θi: Angle between two X-axes as viewed from the Z-axis (joint angle for revolute joints)

- di: Distance between two X-axes along the Z-axis (link offset)
- ai: Distance between two Z-axes along the X-axis (link length)
- αi: Angle between two Z-axes as viewed from the X-axis (link twist)
    - By adjusting the joint values in the RoboAnalyser, you can obtain the DH matrices for the ABB IRB120 at its home position. The DH matrices can be computed both manually and using the software's "Visualize DH" option.
    - ai: This parameter signifies the distance between two Z-axes measured along the X-axis. It is known as the link length.
    - αi: This parameter represents the angle between two Z-axes as viewed from the X-axis. It is called the link twist.

4. Repeat the following for all the frames
5. Multiply the Matrices to compute the final DH Transformation and validate them upon formula application

Results:

A. DH Table:

Link	θ_i	d_i	a_i	α_i
1	θ_1	290	0	90
2	θ_2+90	0	270	0
3	θ_3	0	70	90
4	θ_4	302	0	-90
5	θ_5	0	0	90
6	θ_6	72	0	0

B. DH Matrices (in this case, θ_i = 0°):

$^0T_1$ =

$$[1\ 0\ 0\ 0\ 0\ 0\ -1\ 0\ 0\ 1\ 0\ 290\ 0\ 0\ 0\ 1]$$

$$^1T_2 = [0\ -1\ 0\ 0\ 1\ 0\ 0\ 270\ 0\ 0\ 1\ 0\ 0\ 0\ 0\ 1]$$

$$^2T_3 = [1\ 0\ 0\ 70\ 0\ 0\ -1\ 0\ 0\ 1\ 0\ 0\ 0\ 0\ 0\ 1]$$

$$^{3}T_4 = [1\ 0\ 0\ 0\ 0\ 0\ 1\ 0\ 0\ -\ 1\ 0\ 302\ 0\ 0\ 0\ 1\ ]$$

$$^{4}T_5 = [1\ 0\ 0\ 0\ 0\ 0\ -\ 1\ 0\ 0\ 1\ 0\ 0\ 0\ 0\ 0\ 1\ ]$$

$$^{5}T_6 = [1\ 0\ 0\ 0\ 0\ 1\ 0\ 0\ 0\ 0\ 1\ 72\ 0\ 0\ 0\ 1\ ]$$

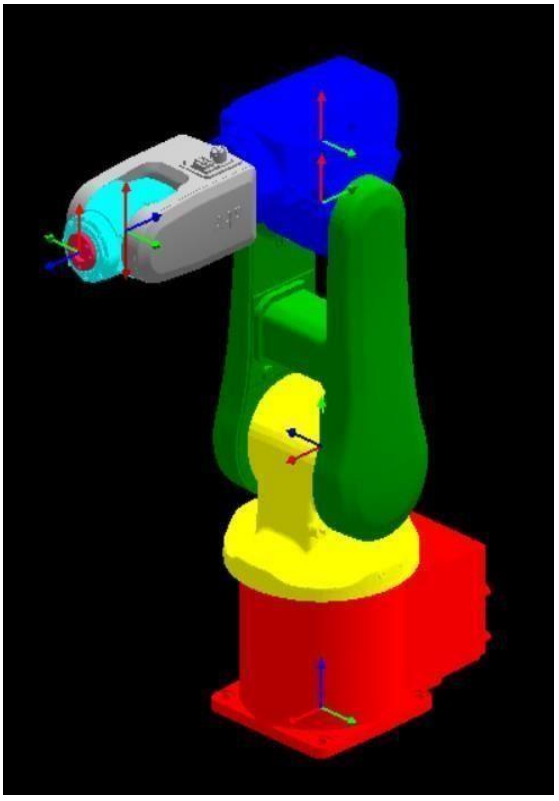
C. Final DH Matrix:

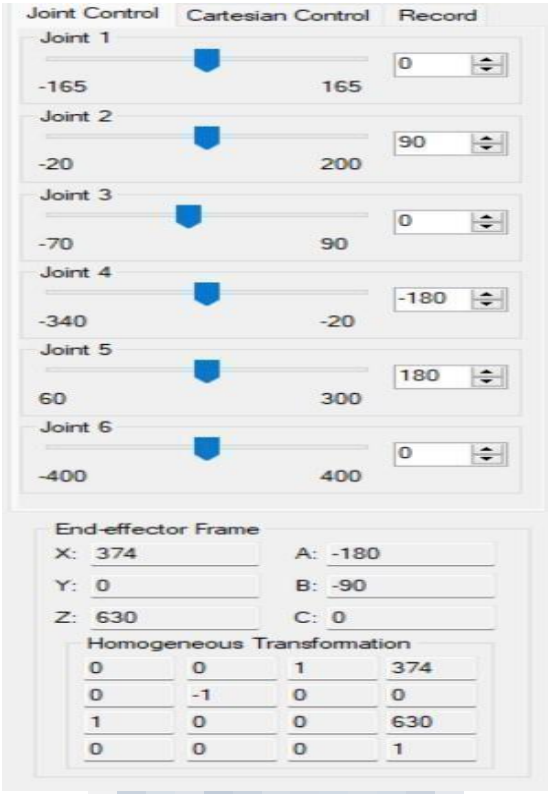
$^{0}T_6 = [^{0}T_1]\times[^{1}T_2]\times[^{2}T_3]\times[^{3}T_4]\times[^{4}T_5]\times[^{5}T_6]$

=

$$[1\ 0\ 0\ 0\ 0\ 0\ -\ 1\ 0\ 0\ 1\ 0\ 290\ 0\ 0\ 0\ 1\ ]\times[0\ -\ 1\ 0\ 0\ 1\ 0\ 0\ 270\ 0\ 0\ 1\ 0\ 0\ 0\ 0\ 1\ ]\times[1\ 0\ 0\ 70$$

= $[0\ 0\ 1\ 374\ 0\ -\ 1\ 0\ 0\ 1\ 0\ 0\ 630\ 0\ 0\ 0\ 1\ ]$ (at its home position)





Inferences: Consequently, the kinematic analysis of the robot manipulator utilizing the DH formulation has been performed both manually and with the RoboAnalyser

software. It is crucial to be mindful of the sign conventions when tabulating the DH parameters to ensure precision.

Share

Experiment: 4
Online training and certification for UR16e collaborative robot manipulator and demonstration

Aim: To grasp the fundamental concepts and commands required for operating a UR Robot.

Objective: To execute a pick-and-place function using the necessary commands on a UR Robot.

Requirements: Universal Robots Website.

Learnings:

Module 1: This module provides an introduction to the website and teach-pendant interfaces, offering an overview of the robot and its input/output functions.

Module 2: Users learn to set up sensors and conveyors for specific tasks, such as pick-and-place operations. It covers connecting the required tools, configuring sensors and conveyors, renaming them as necessary, and introducing the safety board.

Module 3: Focusing on the gripper tool essential for pick-and-place tasks, this module guides users through the setup of the tool and the configuration of the tool center point. It also includes adding the robot's payload capacity.

Module 4: Building on prior setups, this module emphasizes programming the robot's motion by defining waypoints.

Module 5: This module explores the gripper's functionality in pick-and-place operations and its interaction with sensors. It explains how sensors notify the robot when a

payload is on the conveyor, requiring the robot to wait for a signal before proceeding. The programming must incorporate set and wait commands for proper interaction.

Module 6: This module discusses controlling conveyors to enhance pick-and-place operations, utilizing threads for improved execution.

Module 7: This module covers implementing 'reduced mode' and 'safeguard stop' using a safety scanner to manage external disturbances. It includes setting up an emergency button and defining safety boundaries. When a disturbance is detected, the robot automatically switches modes. The safety boundary is established by positioning the tool sphere to ensure any disturbance makes contact with the tool first.

Module 8: This module focuses on optimizing robot performance by reducing the average cycle time for operations. It discusses strategies for decreasing time by increasing the speed and acceleration of the robot's arm and defining the blend radius to minimize acceleration and deceleration times.

Inferences: The pick-and-place operations were successfully completed. Various wait times and motion waypoints were analyzed to assess their effects.

The certificate is attached below:

UNIVERSAL ROBOTS

Certificate of completion

e-Series Core Track

meghavath Rahul

18. September 2024

Martin Woersaa Abildgaard
Universal Robots Academy

Experiment 6

Trajectory analysis of ABB IRB120 manipulator

Objective: To conduct a trajectory analysis of the ABB IRB120 robotic manipulator.

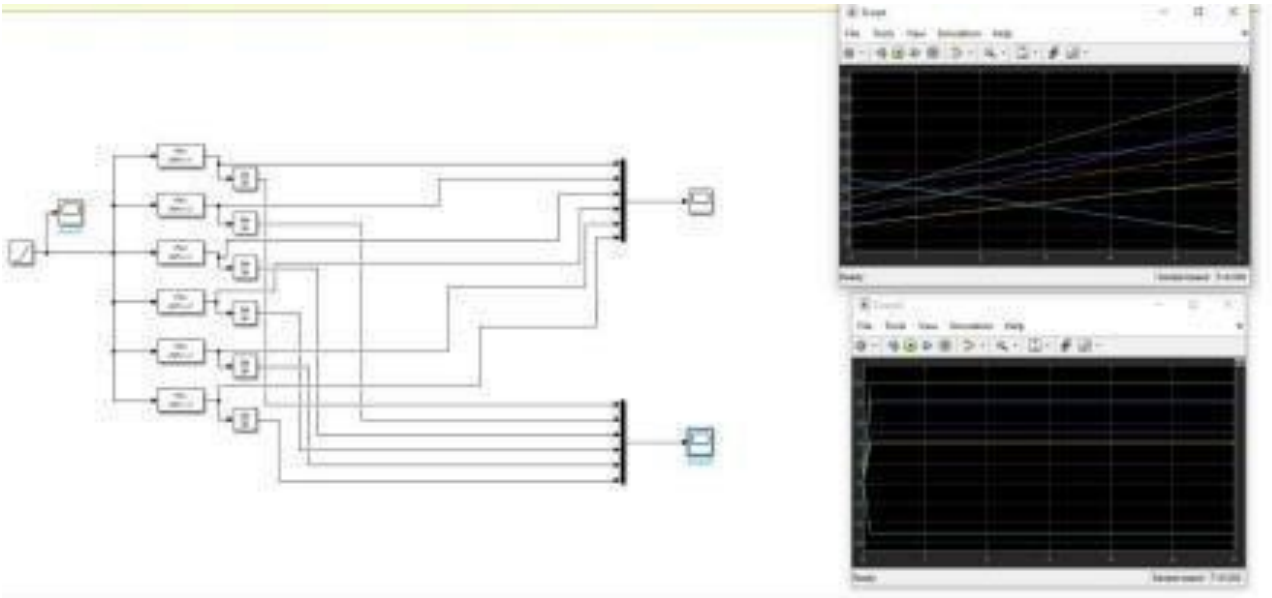
Requirements: MATLAB-Simulink (Software)

Procedure:

1. Identify the coefficients of the cubic polynomial based on the provided question.
2. Utilizing the established linear, cubic, and fifth-order polynomial equations, create a Simulink block diagram to compute position, velocity, and acceleration over the specified time interval.
3. Implement a ramp generator as the input, maintaining a slope of 1 for linear progression over time.
4. Calculate velocity and acceleration using the differentiation module in Simulink.
5. Use a Multiplexer (MUX) module to consolidate the results into a single scope for convenient comparison.

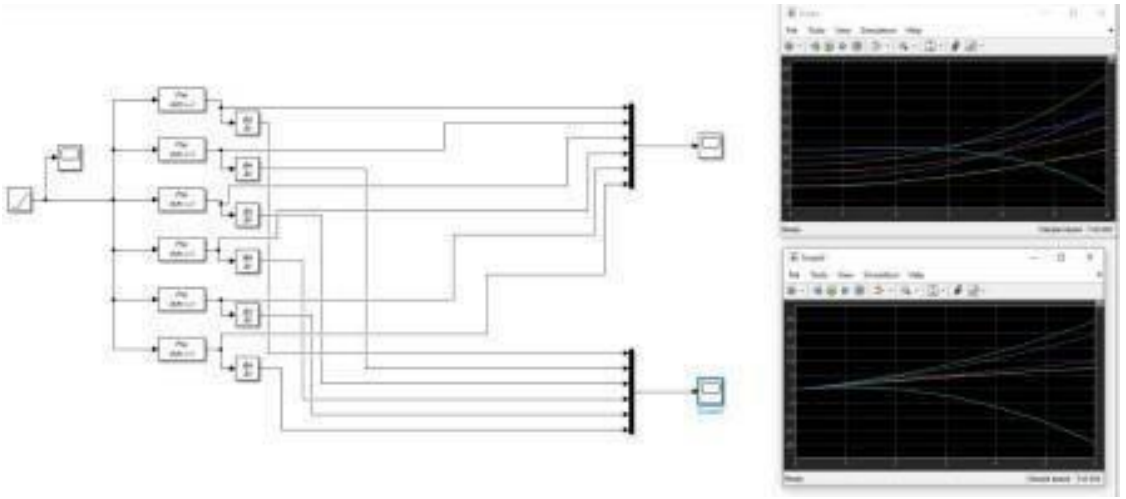
Result: Linear Polynomial: a0 t + a1

a0	a1
4.167	10
8.333	15
5	20
10	25
5	30
-5	35



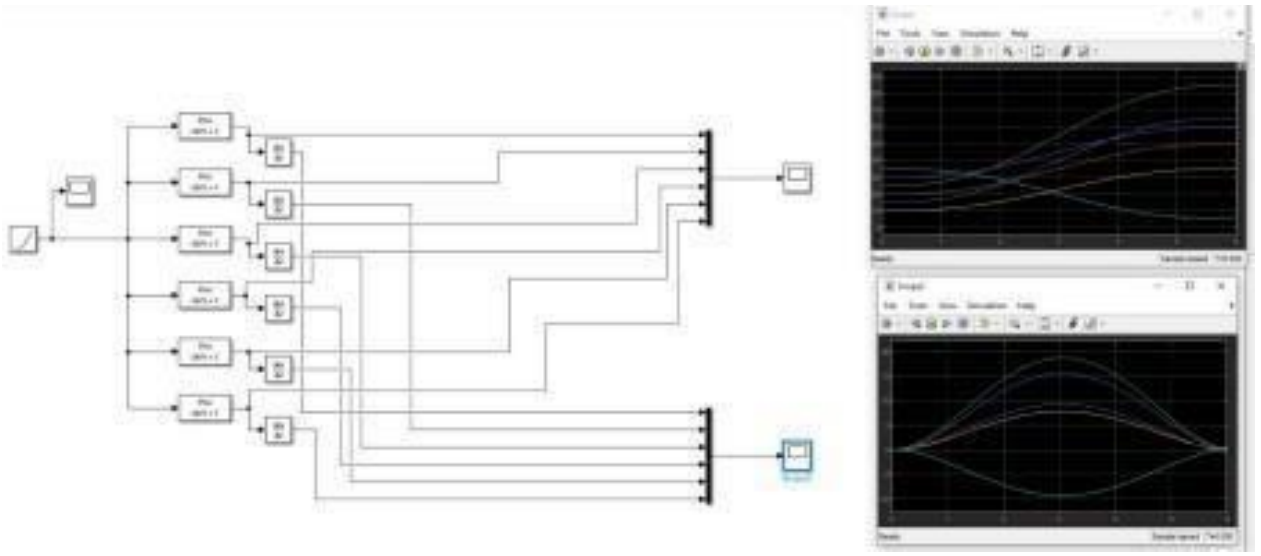
Cubic Polynomial : a0 t 3 + a1 t 2 + a2 t 1 + a3

a0	a1	a2	a3
-0.0225	0.8296	0	10
0.0901	0.8483	0	15
0.0000	0.8333	0	20
0.1351	0.8559	0	25
0.0000	0.8333	0	30
-0.2703	0.7883	0	35



Fifth Order Polynomial: a0 t 5 + a1 t 4+a2 t 3+a3 t 2 + a4 t + a

a0	a1	a2	a3	a4	a5
0.0193	-0.2894	1.1574	0	0	10
0.0386	-0.5787	2.3148	0	0	15
0.0231	-0.3472	1.3889	0	0	20
0.0463	-0.6944	2.7778	0	0	25
0.0231	-0.3472	1.3889	0	0	30
-0.0231	0.3472	-1.3889	0	0	35



Result:

This experiment has clarified the concept of trajectory by visualizing the equations involved.

EXPERIMENT 7

Pneumatic and hydraulic circuit design for cylinder control

AIM: To develop and gain proficiency in designing pneumatic and hydraulic circuits for multi-cylinder systems using FluidSIM.

Objective:

1. Design a pneumatic circuit for single acting cylinder operation : A+A2.
2. Design a hydraulic circuit for two double acting cylinder operation :
3. A+ B+ A- B

Requirements: FluidSIM (Software)

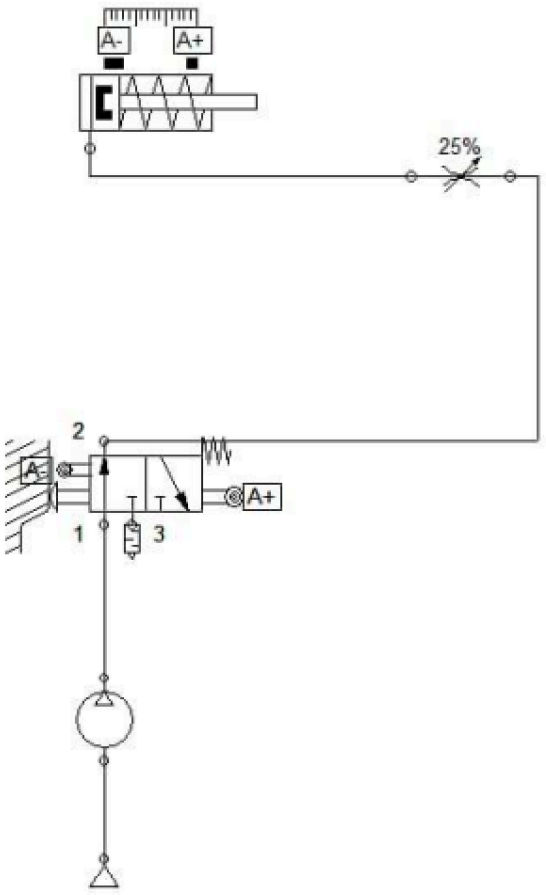
Procedure:

1. Verify that the FluidSIM software is correctly installed on your computer to enable circuit design and simulation.
2. Familiarize yourself with the symbols used for pneumatic and hydraulic components, particularly those related to single-acting and double-acting cylinders.
3. a. **Single-acting Pneumatic Cylinder (A+ A-):**
    - Design a circuit to control a single-acting cylinder, incorporating all essential components for operation, such as valves and pressure regulators.
    - Include features that facilitate both forward and return strokes.

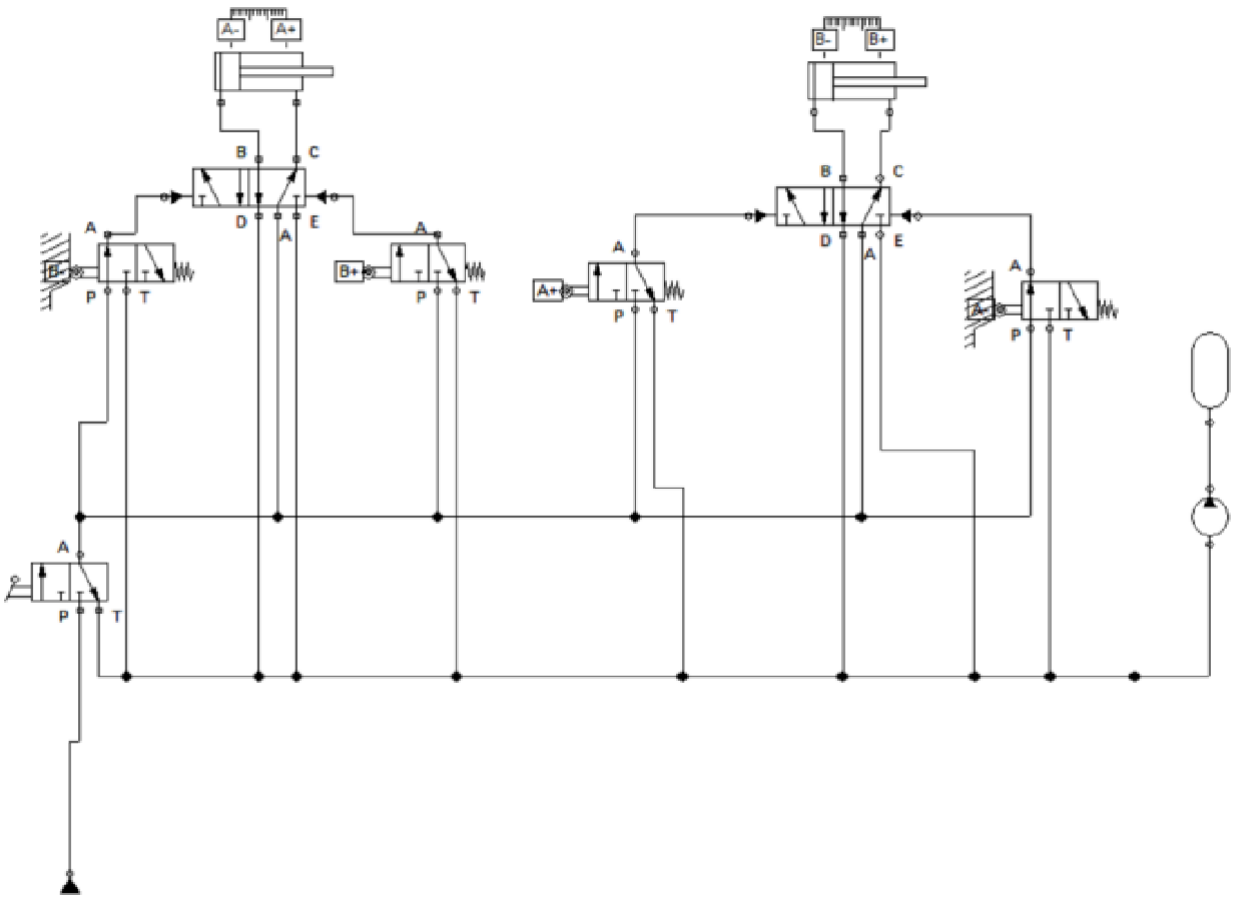
   b. **Double-acting Hydraulic Cylinder (A+ B+ A- B-):**
    - Develop a hydraulic circuit that can operate two double-acting cylinders simultaneously.
    - Ensure the design permits independent control of each cylinder's extension and retraction.
4. Examine both circuits for any open ends, intersecting lines, or components that may hinder functionality. Make necessary adjustments to enhance clarity and operational efficiency.

Observation:

a. Single acting pneumatic cylinder A+ A-:



b. Double acting hydraulic cylinder A+ B+ A- B- :



Result:

By designing the pneumatic and hydraulic circuits, the operation of the multi-cylinder system has been successfully managed in the desired sequence.
4o mini

EXPERIMENT 8:

Design And Implementation Of Pneumatic Circuits Using Vacuum Trainer Kit

Controlled release of workpieces retained by vacuum

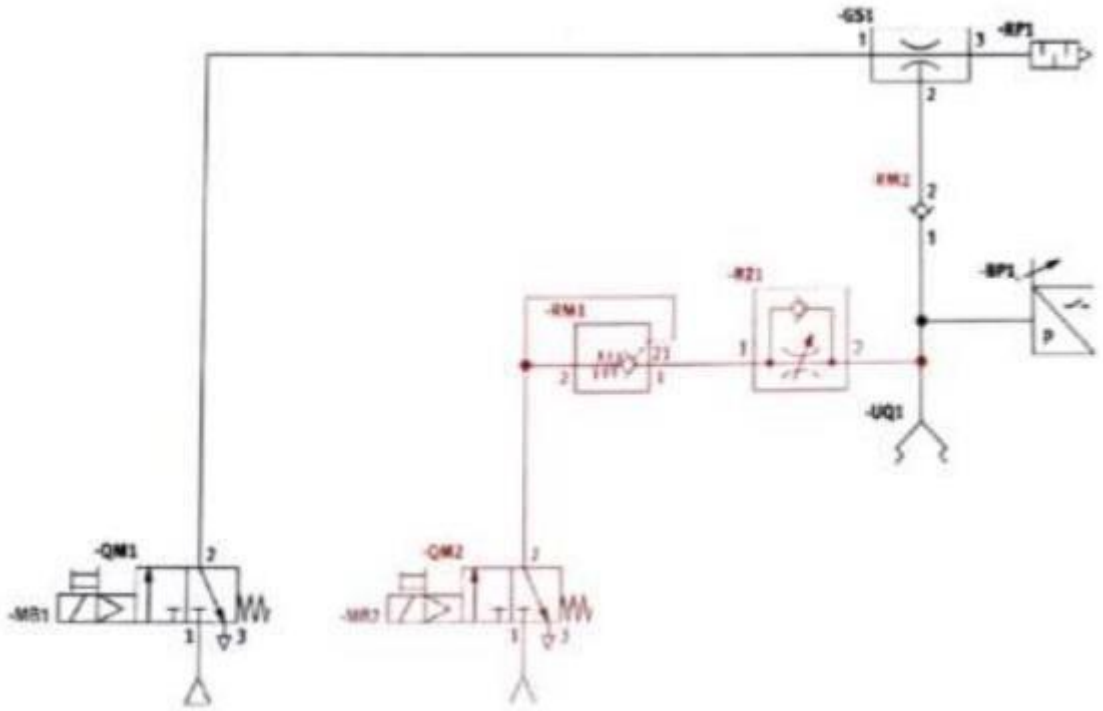
Aim:

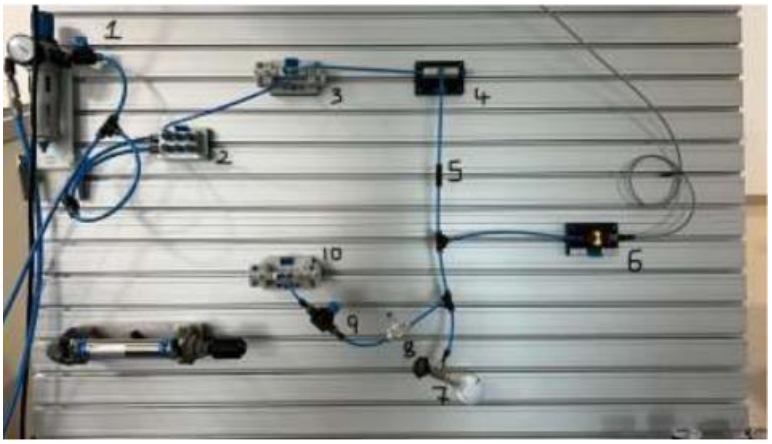
to release workpieces in a controlled way which are retained by vacuum

Components:

1 .Air compressor
2. Vacuum generator
Type H
Type L
3. 3/2 way solenoid valve (pilot operated)
4. Non return valve
5. Vacuum pressure switch
6. Connecting pipes
7. Pressure Regulator (on/off)
8. Suction cup
9. One way flow control valve
10. Shut off valve
11. Connectors

Circuit diagram:





1. Pressure Regulator
2. Distributor
3. 3/2 way solenoid valve
4. Vacuum generator
5. Non return valve
6. Vacuum switch
7. Suction gripper with vacuum security valve
8. One way flow control valve
9. Shut off valve
10. Solenoid valve

Procedure:

- check the air compressor
- connect pressure regulator to air compressor (via connecting pipes)
- connect 3/2 way solenoid valve to pressure regulator (via connecting pipes)
- connect vacuum generator (Type H) to 3/2 way solenoid valve (via connecting pipes)
- connect vacuum generator (Type H) to non-return valve
- connect non return valve to vacuum pressure switch and suction cup using barbed T
connector
- connect one way flow control valve to suction cup using barbed T connector
- connect one end of the shut off valve to one way flow control valve and the other end
to another solenoid valve.

Result:

we can control object, however we want in a given direction which are retained by vaccum

EXPERIMENT: 9

Production Rate Monitoring Of Manufactured Boxes In Conveyor System

Objective:

Create a system that detects red and green boxes moving along a conveyor belt, counting the number of each box color as they pass through designated sensors.

Requirements:

- CoppeliaSim (formerly V-REP) simulation software
- Lua scripting knowledge for sensor programming
- Conveyor belt, red, and green boxes in the scene
- Proximity and vision sensors for box detection
- A user interface (UI) for displaying the box count

Procedure:

1. Create a New Scene
   - Open CoppeliaSim software.
   - Create a new scene by selecting File > New Scene.
2. Add a Conveyor Belt
   - Under the Model Browser, go to Conveyor Belt.
   - Drag and drop a Conveyor Belt model into the scene.
   - Position the conveyor belt as needed in the scene (e.g., on the floor).
3. Add Red and Green Boxes
   - To create boxes, go to Primitive Shapes and select Cuboid.
   - Drag and drop cuboids into the scene and position them along the conveyor belt.
   - Change the color of the boxes:
     - Select a cuboid, then in the Shape Properties, change its Color to Red for one box and Green for the other.
   - Ensure that the boxes are sized appropriately and aligned with the belt movement.
4. Place Proximity and Vision Sensors
   - Proximity Sensor: Add a proximity sensor to the scene by navigating to Sensors > Proximity Sensor in the model browser.
     - Position the sensor in a way that it detects the boxes as they pass by.

- Vision Sensor: Add a vision sensor (camera) to the scene via Sensors > Vision Sensor.
    - Place the vision sensor above or alongside the conveyor belt, making sure it has a clear line of sight to the boxes.

5. Write Lua Scripts for Sensors
- Write a non-threaded Lua script for each sensor to detect red and green boxes.

Code:

Example Lua Script for Proximity Sensor:

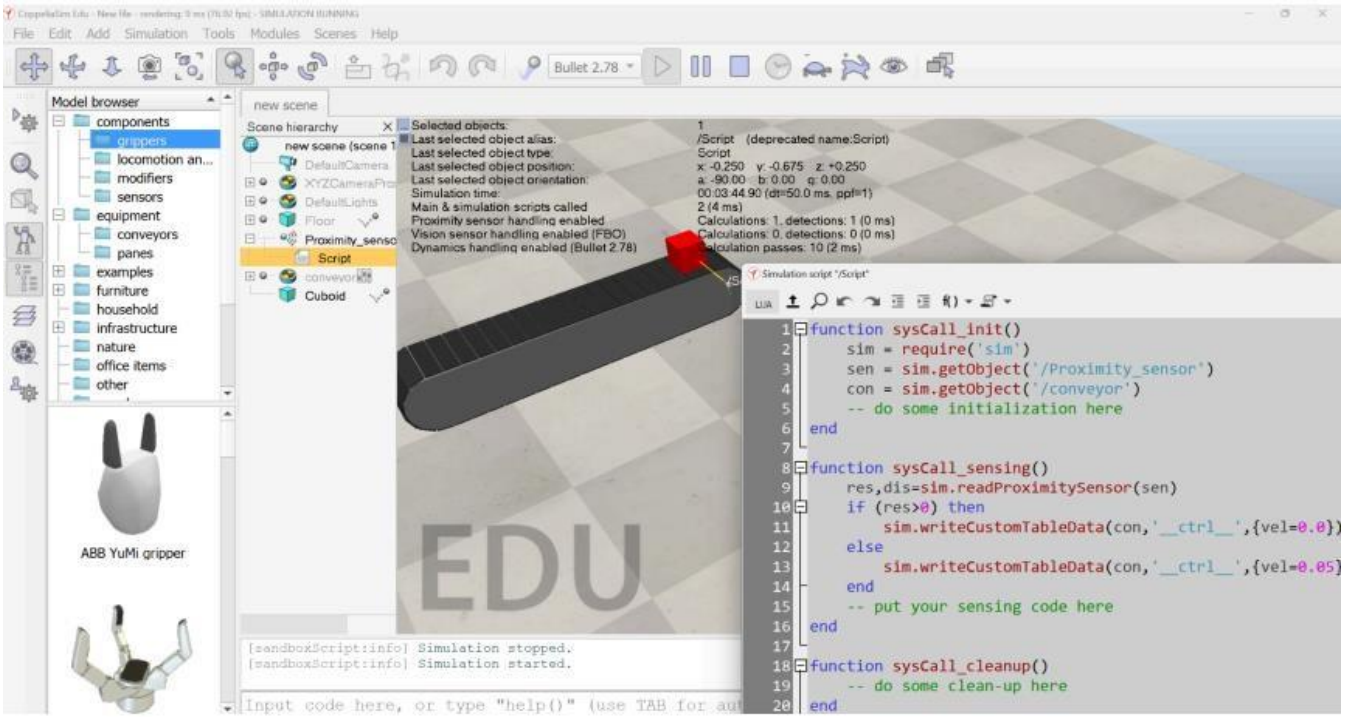
```
-- Initialize sensor handles
proximitySensor = sim.getObjectHandle("Proximity_sensor")
greenBox = sim.getObjectHandle("Green_Box")
redBox = sim.getObjectHandle("Red_Box")

-- Define counters for green and red boxes
greenCount = 0
redCount = 0

-- Non-threaded script to detect objects
function sysCall_nonSimulation()
    -- Check for proximity detection
    local result, detectedObject = sim.readProximitySensor(proximitySensor)
    if result > 0 then  -- Object detected
        if detectedObject == greenBox then
            greenCount = greenCount + 1
        elseif detectedObject == redBox then
            redCount = redCount + 1
        end
    end
end
```

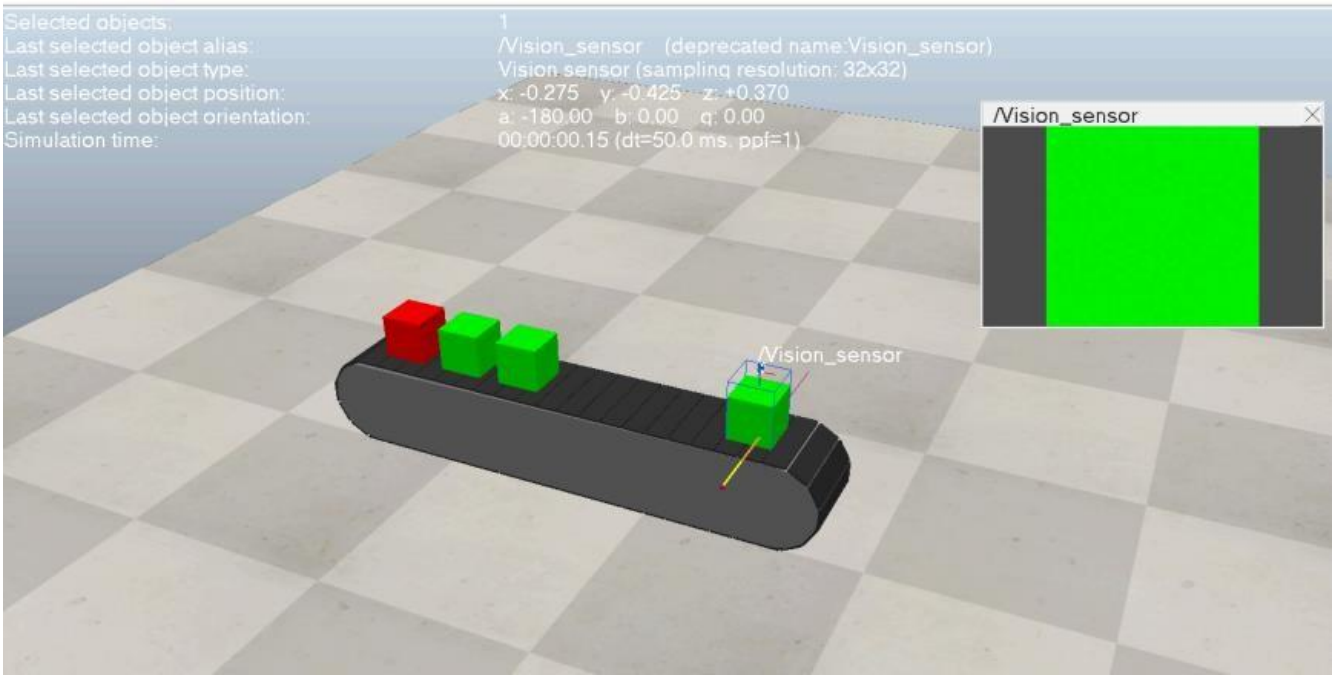
Example Lua Script for Vision Sensor:

```
-- Initialize vision sensor handle
visionSensor = sim.getObjectHandle("Vision_sensor")

-- Define counters for green and red boxes
greenCount = 0
redCount = 0

-- Non-threaded script to process vision sensor data
```

```
function sysCall_nonSimulation()
    -- Capture image from vision sensor
    local resolution, image = sim.getVisionSensorImage(visionSensor)

    -- Analyze the image to find red and green boxes
    if resolution > 0 then
        -- Example: Color detection (you will need to implement a more complex algorithm)
        for x = 1, resolution[1] do
            for y = 1, resolution[2] do
                local color = image[(y-1)*resolution[1]*3 + (x-1)*3 + 1] -- Extract RGB values
                if color == {255, 0, 0} then  -- Red box
                    redCount = redCount + 1
                elseif color == {0, 255, 0} then  -- Green box
                    greenCount = greenCount + 1
                end
            end
        end
    end
end
```

Result:





The simulation successfully transported red and green boxes along the conveyor belt. As the boxes moved, proximity sensors detected them accurately, and the system kept track of the production rates for each type of box. Through this process, we gained hands-on experience with CoppeliaSim, becoming familiar with the software's features, such as sensor integration and UI elements, and learning how to use Lua scripting for automation tasks.

Experiment: 10
PLC ladder logic programming – Fundamentals

Aim:

To understand and construct the PLC Ladder logic of some basic circuits.

Objective:

To create PLC circuit for the following Logics/Operations-

1. P = A'.(B+C).(D.E).(F+G+H')

2. P = (A+B+C).(D+E+F).G.H

3. Conveyor (C) shall be ON when any one of the four "start buttons" is pressed, it shall turn OFF when any one of the four "stop buttons" is pressed.

4. Logic Construction using the truth table:

A	B	C	Start
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

Procedure:

1. **Access PLC Fiddle:**
   - Go to the PLC Fiddle website and start a new project.
   - Add a rung (which is like a line in the ladder diagram).
2. **Symbols for Contacts and Coils:**
   - A **Normally Open (NO) contact** is represented by the symbol: --||--.
   - A **Normally Closed (NC) contact** is represented by the symbol: --|\|--.
   - A **Coil (Output)** is represented by the symbol: --O--.
3. **Understanding Logic in the Circuit:**
   - When two variables (inputs) are connected with a **dot (.)**, it means they are

in **series**. Both must be true for the rung to be true.
- When two variables (inputs) are connected with a **plus (+)**, it means they are in **parallel**. Either one can be true for the rung to be true.

4. **Create the Logic for Questions 1 and 2:**
   - Use the above rules to create the PLC logic for the first two questions.
5. **Create a Circuit for 4 Start and 4 Stop Buttons:**
   - For **Question 3**, connect 4 **Start buttons** (NO type) in **parallel**.
   - Connect 4 **Stop buttons** (NC type) in **series**.
6. **Create a Logic Based on a Truth Table:**
   - For **Question 4**, the truth table shows the logic: A.B + B.C + C.A.
   - Use this logic to create the PLC by combining the inputs (A, B, and C) as per the formula.

Result:

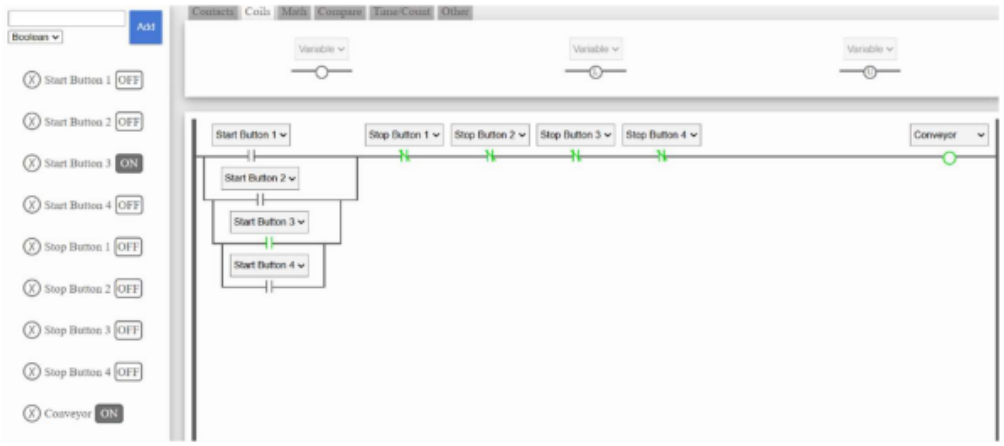
1. P = A'.(B+C).(D.E).(F+G+H') –

---

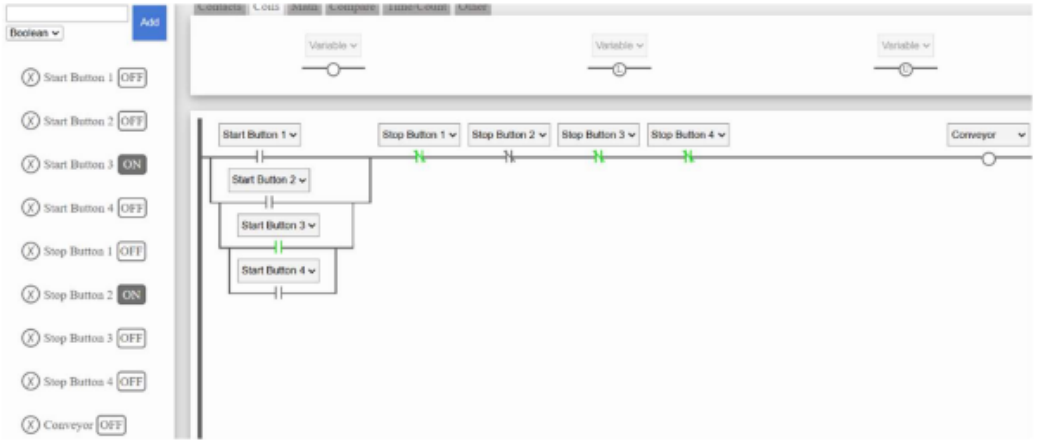
2. P = (A+B+C).(D+E+F).G.H -

---

3. Conveyor-



Start Button 3 Switched ON



Stop Button 2 Switched ON

---

4. Truth Table PLC-

EXPERIMENT 11

AIM :

To solve real world problems using PLC

Software used :

PLC fiddle

Question 1 :

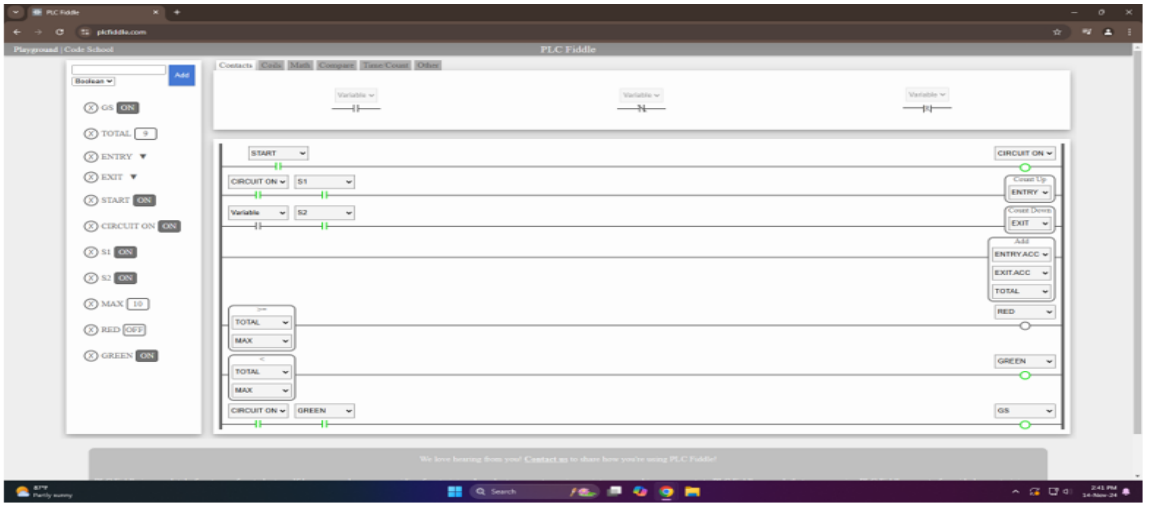
The parking lot which has a capacity of 10 cars is to be controlled by a PLC system. The sensor S1 and S2 are used to count the car at the entrance and exit. If the number of the cars reaches to 10, red light is lit and the gate arm is closed otherwise green light is lit and gate arm stays opened. The arm stays closed until one or more parking space is available in the lot. The gate arm is controlled by activating/deactivating the gate solenoid (GS). Construct LAD for the parking lot controller.

Logic :

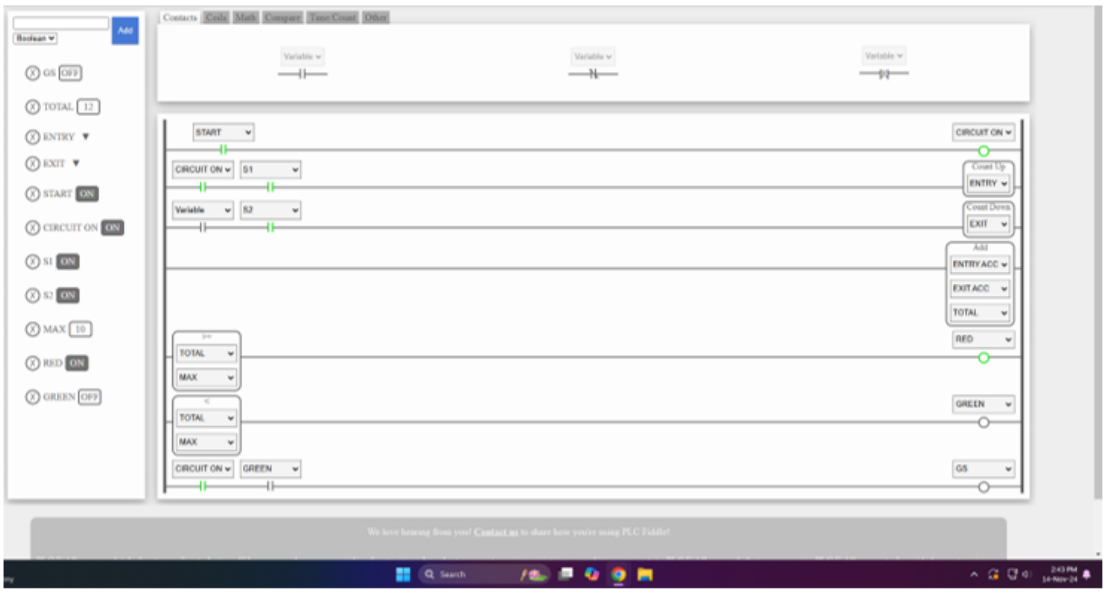
Use a **counter** to keep track of the current number of cars in the lot.

- Increment the counter with **S1**.
- Decrement the counter with **S2**.
- Monitor the counter value:
- If the count = 10, activate the red light, deactivate the green light, and close the gate arm (deactivate GS).
- If the count < 10, activate the green light, deactivate the red light, and open the gate arm (activate GS).

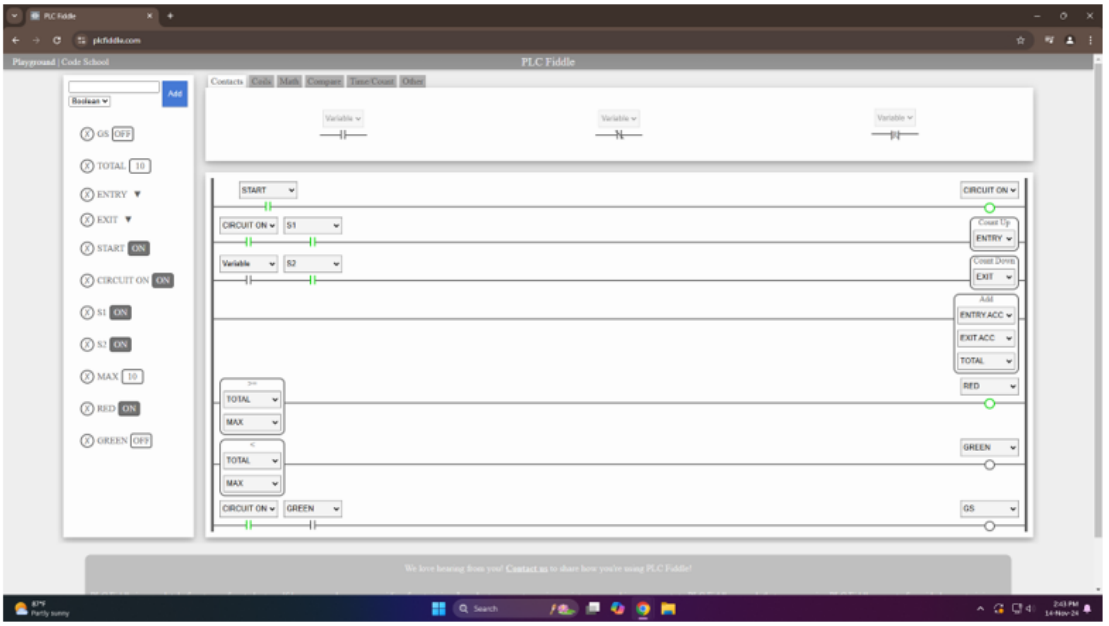
Circuit:



Operation:



When car count reaches 10 , red light gets triggered



Question 2 :

Material A and Material B are collected in a tank. These materials will be mixed for particular time and then mixed product drained out through the outlet valve. Construct Ladder and simulate this application in PLCfiddle.

Logic :

Step 1: Filling Material A and B

- When the process starts, open the valves for Material A and Material B until the sensors detect their presence.

Step 2: Mixing

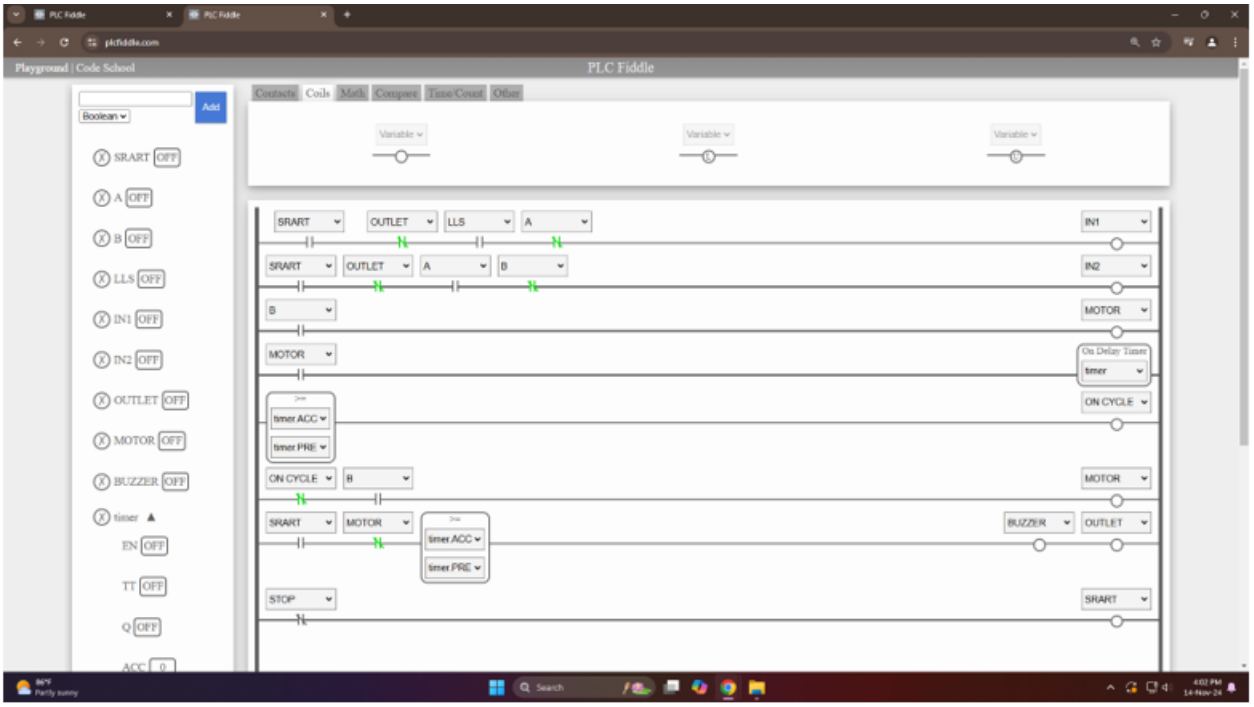
- Once both materials are filled, start the mixer motor for a specific time period (e.g., 10 seconds).

Step 3: Draining

- After mixing, stop the mixer and open the outlet valve to drain the mixed product.

Step 4: Reset

- Close all valves and wait for the next start signal.



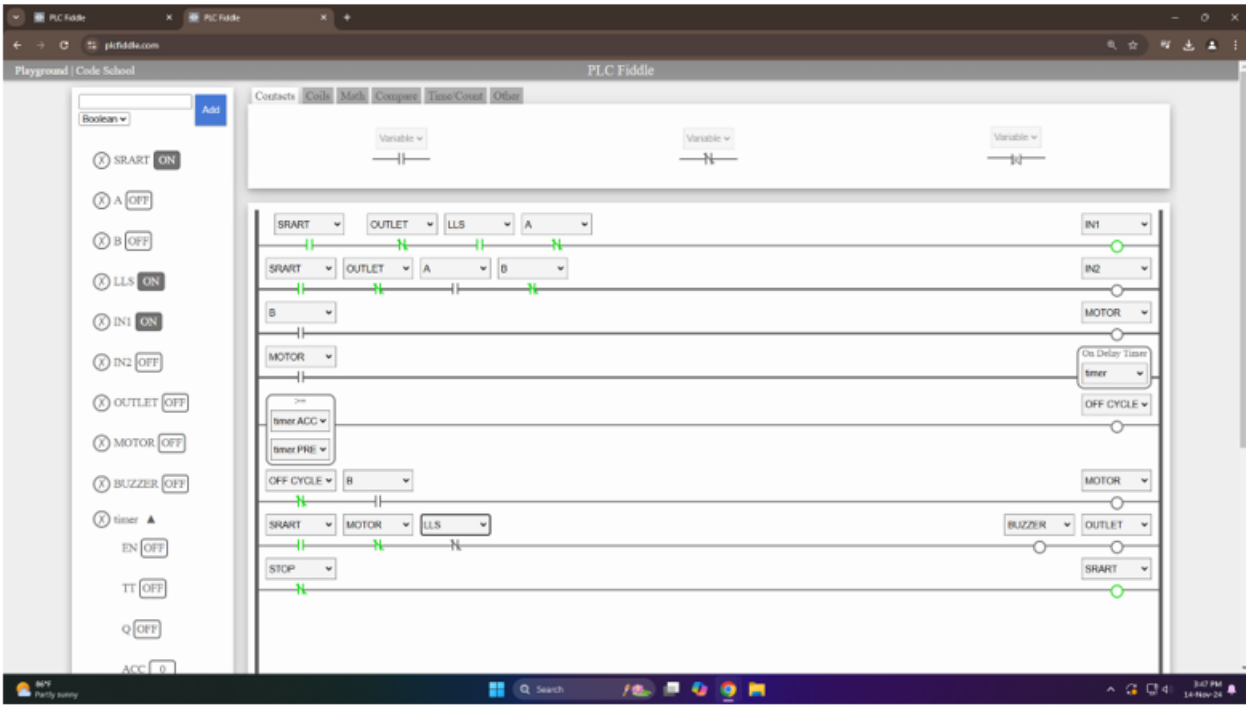
The above is the final circuit

Flow :

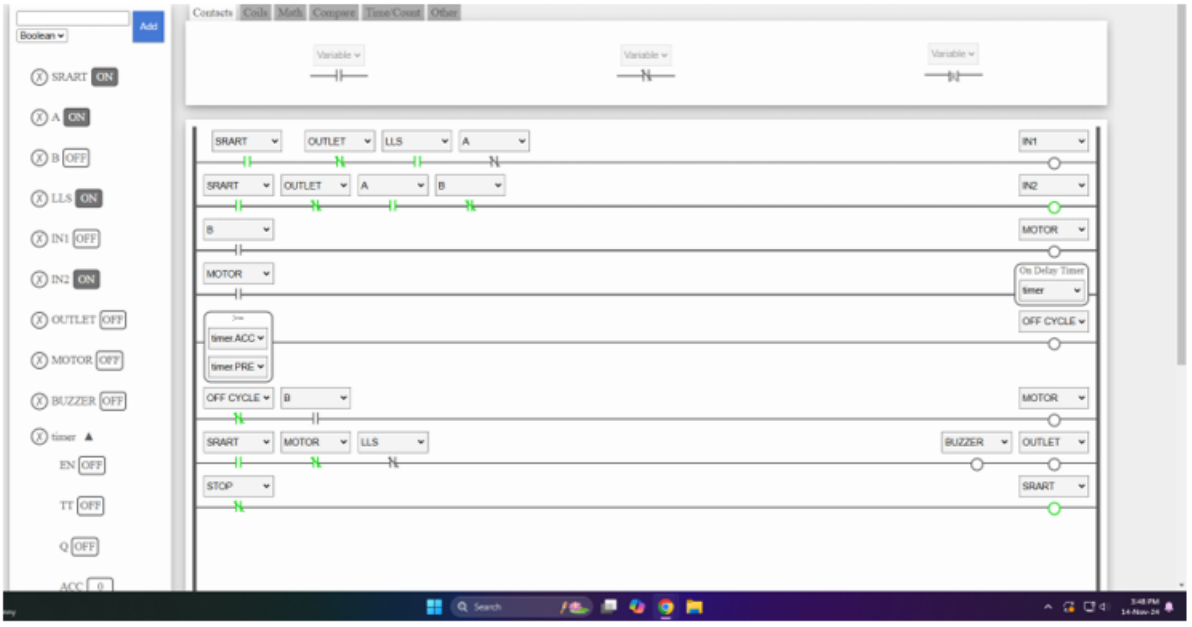
LLS sends 1 --> IN1(input one for material A is opened) -- > A sends 1 -- > IN2(input for material B is opened) -- > B sends 1 -- > Motor starts and timer also starts -- > motor runs for 30 seconds -- > Outlet opens -- > Stop button pressed to reset all

Operation :

1. LLS is ON So IN1 is ON

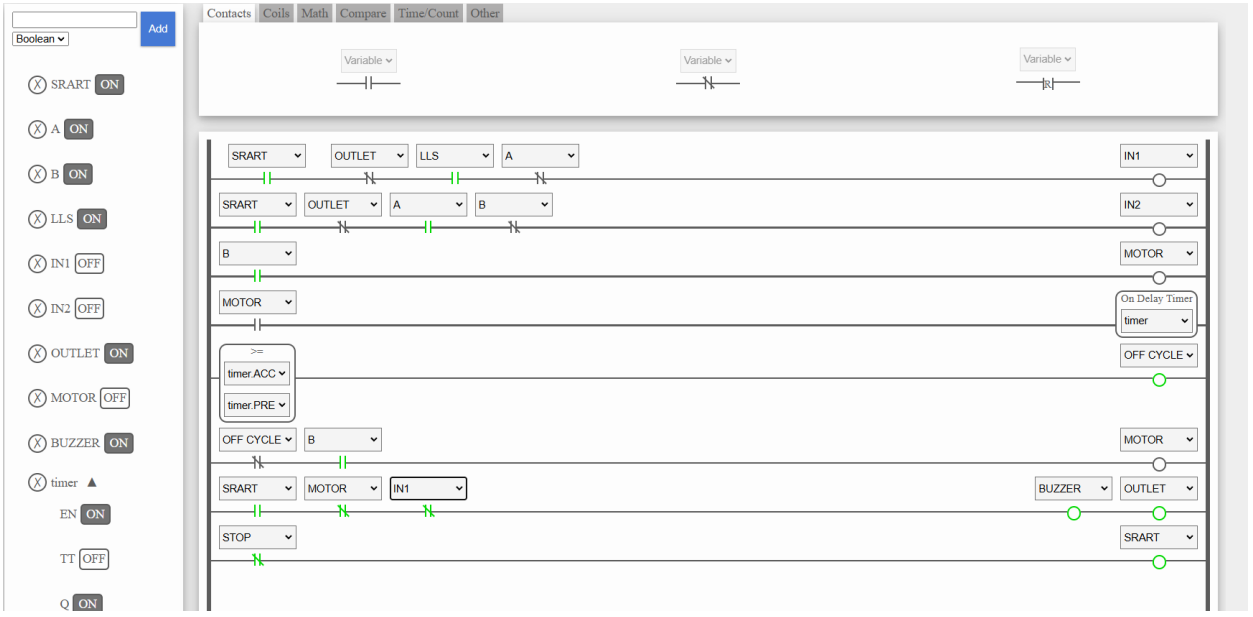


2. LLS and A is ON so IN2 is ON



3. B is ON -- > So materials A and B are filled , so Motor turns ON and runs for particular time

4. Motor turns OFF after 10 secs , and Outlet is open



Question 3 :

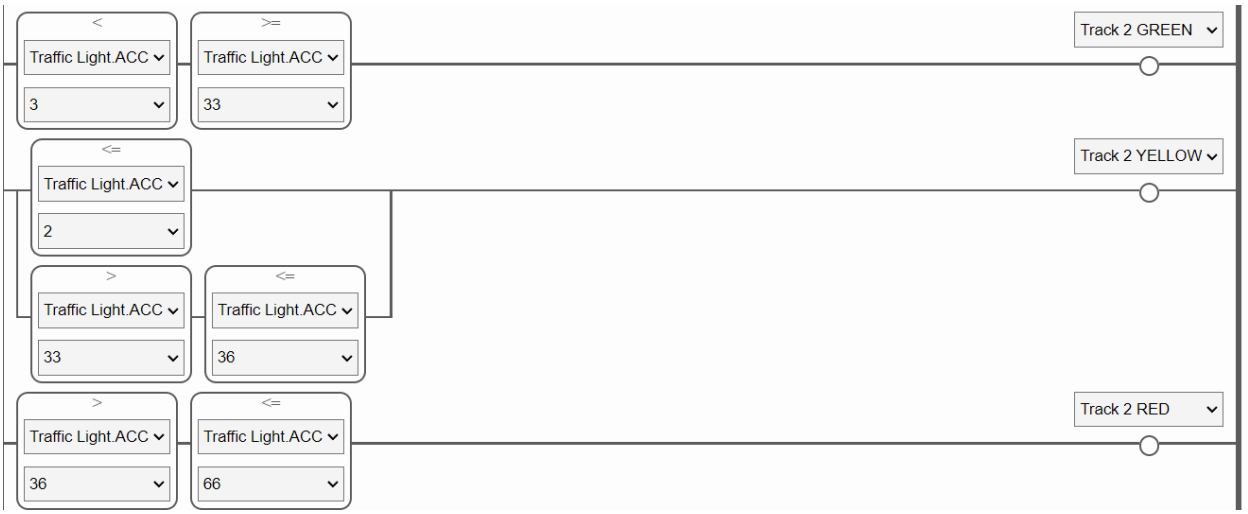
Time	Track A	Track B	Track C	Track D
0-2 secs	Yellow	Yellow	Yellow	Yellow
3-33 secs	Green	Green	Red	Red
34-36 secs	Yellow	Yellow	Yellow	Yellow
36-66	Red	Red	Green	Green

Program and Simulate 4 way Intelligent Traffic Control System using PLC

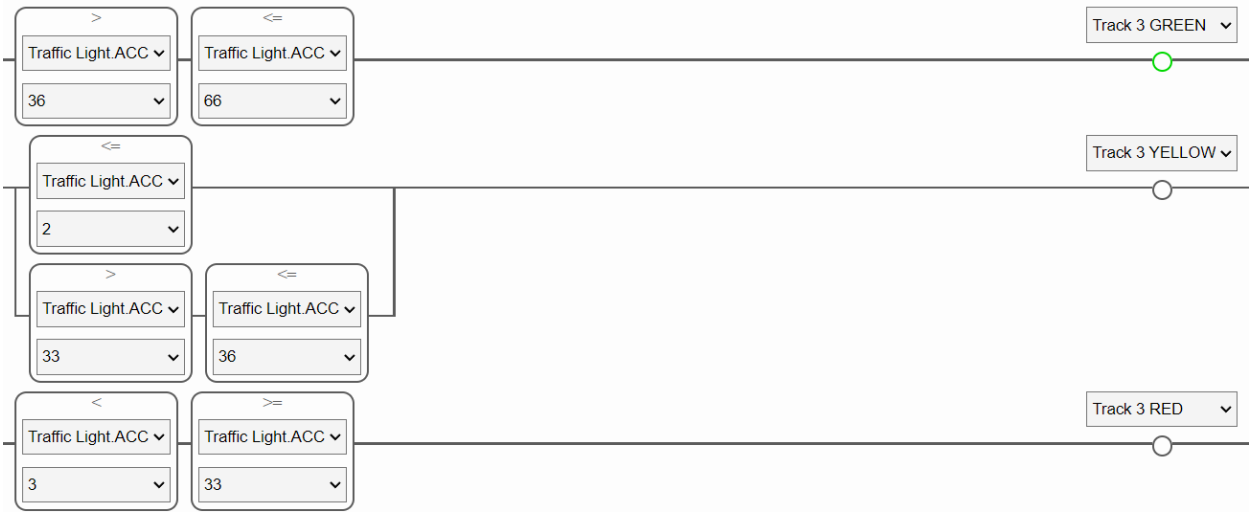
Circuit :

Track 1 :

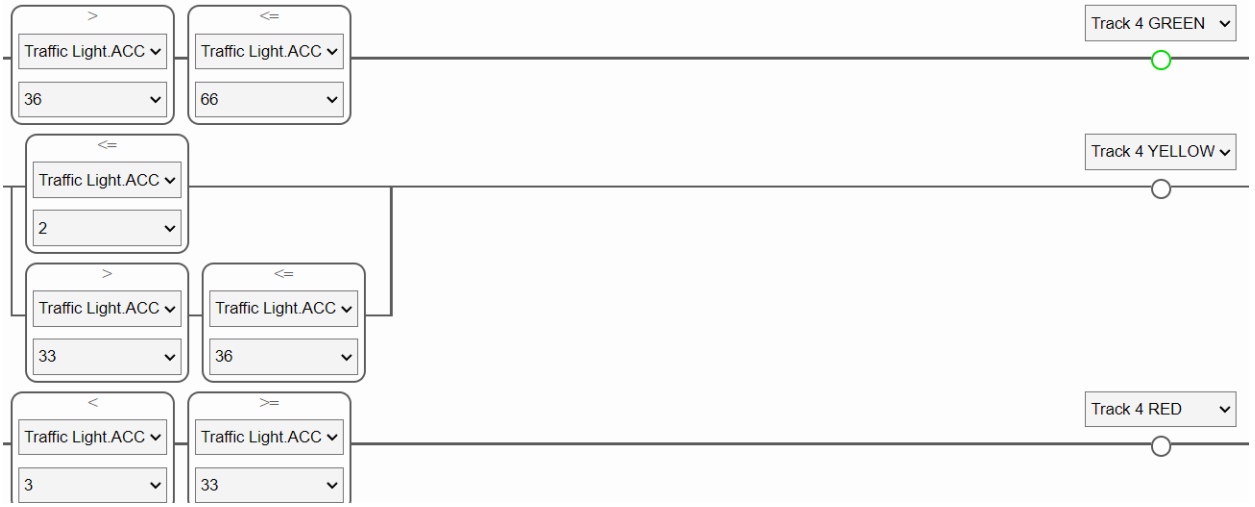
Track 2



Track 3 :



Track 4 :



Traffic Light Timer :

It resets for every 66 seconds

Question 4 :

Program and Simulate Pneumatic Sequential Circuit using plc fiddle for the following sequence

i) A+,B+,A-,B- ii) A+,B+,B-,A-

4.1 : first sequence : A+ , B+ , A- , B-

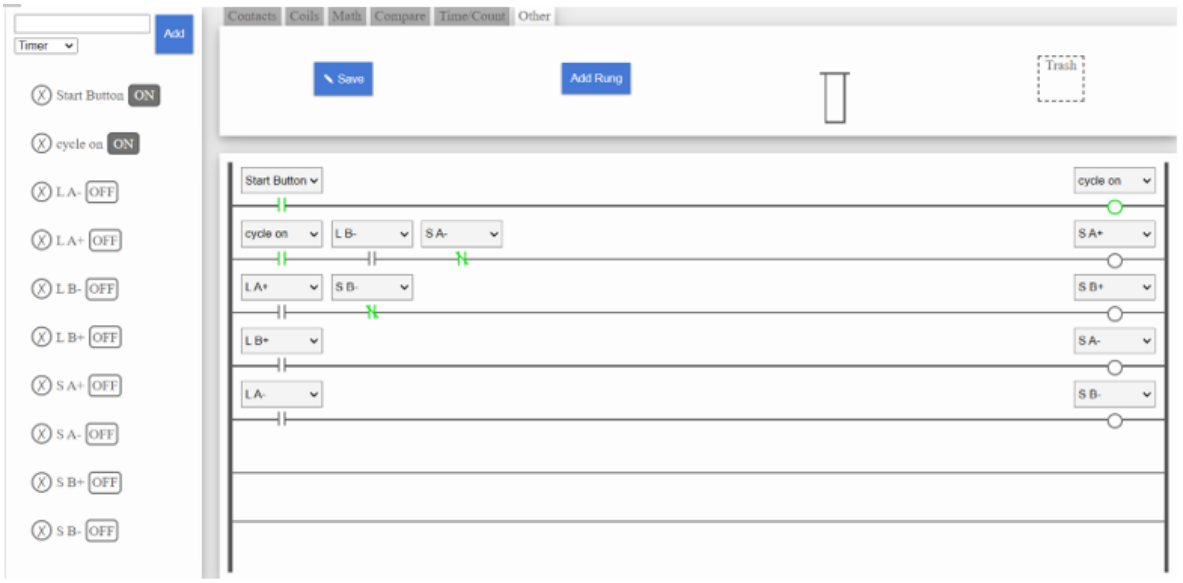
Logic :

The limit switch input is used and solenoid is used to control the cylinder movement

Flow :

(Limit switch B-) -- > Solenoid of A+ -- > Limit Switch of A+ -- > Solenoid of B+ -- > Limit Switch of B+ -- > (Solenoid of A-) -- > (Limit Switch of A-) -- > (Solenoid of B-) -- > Repeat

Circuit:



Limit switch B-) -- > Solenoid of A+ -- > Limit Switch of A+ -- > Solenoid of B+ -- > Limit Switch of B+ -- > (Solenoid of A-) -- > (Limit Switch of A-) -- > (Solenoid of B-) -- > Repeat

(Limit switch B-) -- > Solenoid of A+ -- > Limit Switch of A+ -- > Solenoid of B+ -- > Limit Switch of B+ -- > (Solenoid of A-) -- > (Limit Switch of A-) -- > (Solenoid of B-) -- > Repeat

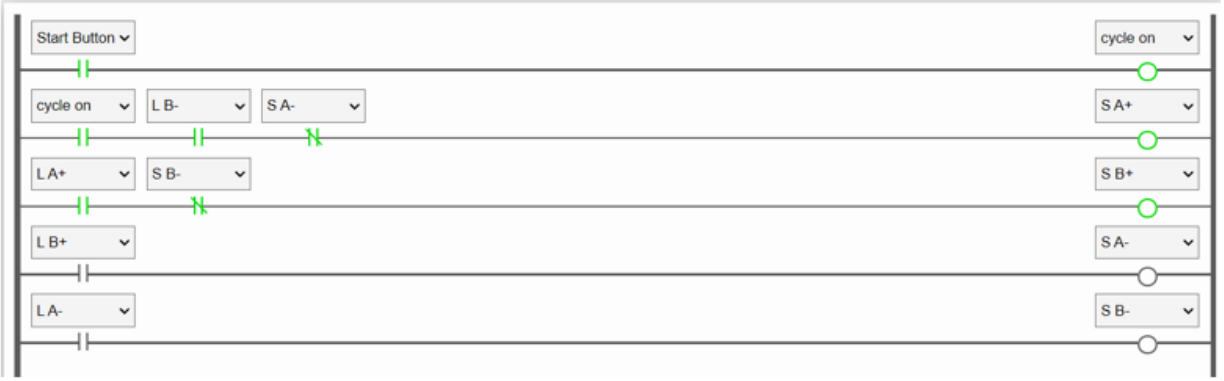
Operation:

Input : Limit Switch of B- Output : solenoid of A+

Operation:

Input : Limit Switch of B- Output : solenoid of A+

2.Input : Limit Switch of A+ Output : solenoid of B+



3.Input : Limit Switch of B+ Output : solenoid of A-

4.Input : Limit Switch of A- Output : solenoid of B-

And the sequence repeats

4.2 : Sequence : A+,B+,B-,A-

Logic :

(Limit switch A-) -- > Solenoid of A+ -- > Limit Switch of A+ -- > Solenoid of B+ -- > Limit Switch of B+ -- > (Solenoid of B-) -- > (Limit Switch of B-) -- > (Solenoid of A-) -- > Repeat

Inference:

The PLC-based simulation successfully automated the control of the mixing tank system. Material A and Material B were accurately filled into the tank, mixed for the specified duration, and then drained through the outlet valve. Sensors effectively detected the presence of materials, and timers ensured precise mixing times. Through this experiment, we gained hands-on experience with PLC programming, ladder logic design, and real-time process control using PLCfiddle.

Construction of various mobile robots using kit

Experiment 5

Aim: To build and program a Smart Car AD118 robot using Arduino IDE.

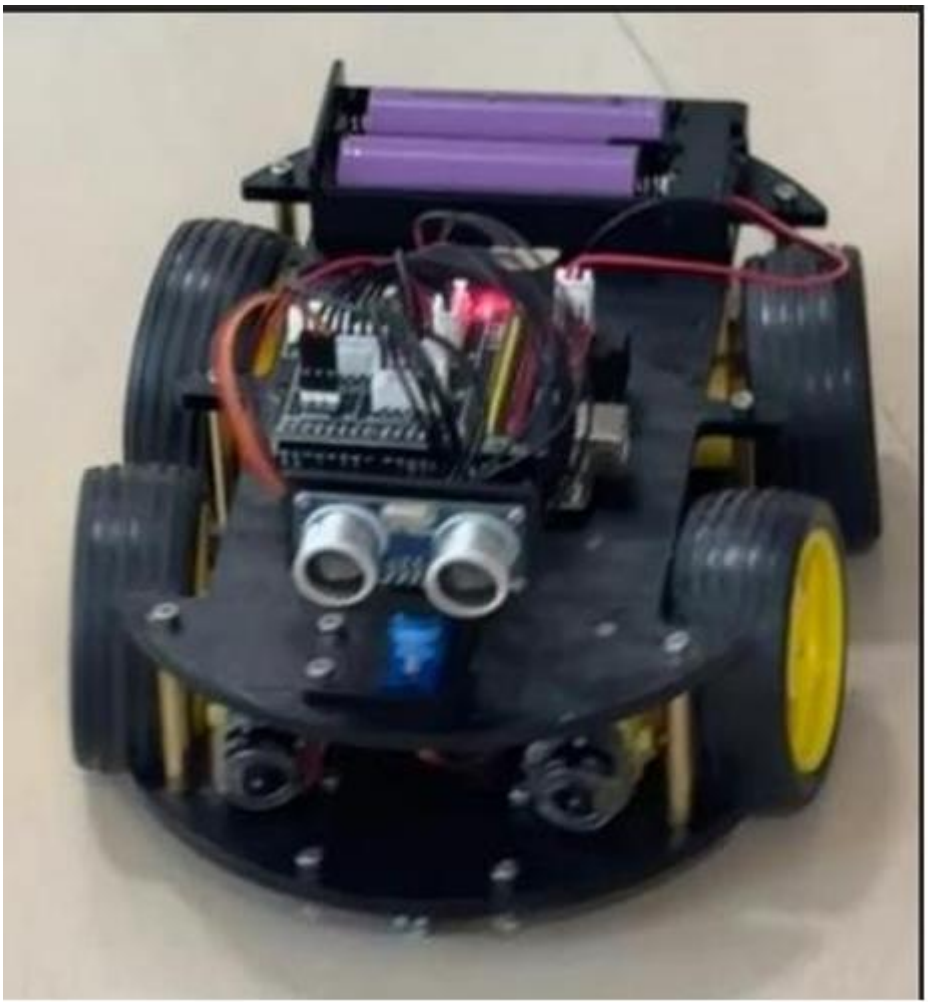
Objective: To assemble the smart car and program it using Arduino IDE to control its movement as needed.

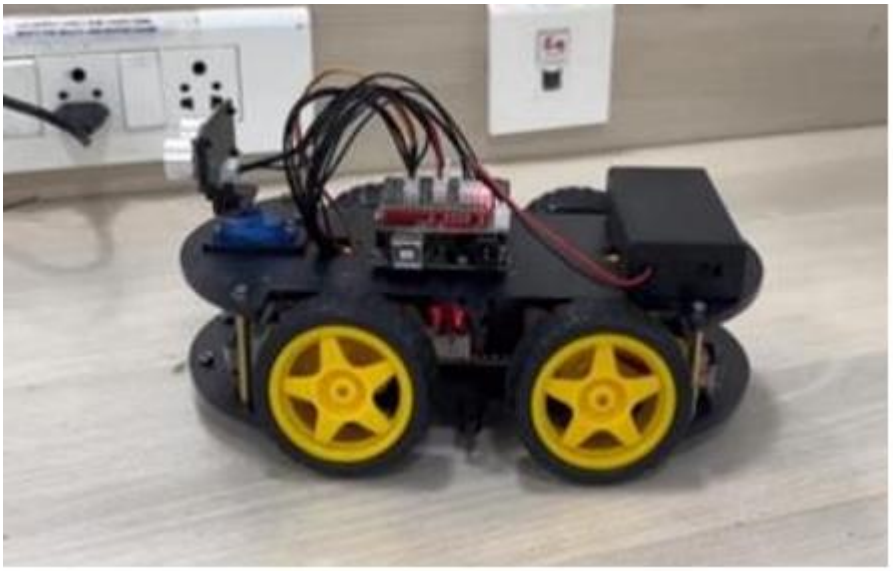
Requirements:

- **Bluetooth module**
- **Battery compartment**
- **Arduino UNO R3**
- **L298N motor driver**
- **DC motors**
- **Ultrasonic sensor with holder**
- **IO expansion board**
- **SG90 micro servo**
- **Acrylic chassis**
- **Line tracking module**
- **Wheels**
- **Screws and fittings**
- **Arduino IDE**

Procedure:

- **Start by assembling the chassis, which forms the base structure of the smart car.**
- **Attach all the components (motors, sensors, etc.) to the chassis as per the instructions provided in the manual.**
- **Carefully connect the wiring between the components, following the wiring diagram or guide.**
- **Open the Arduino IDE on your desktop and upload the pre-downloaded program to the Arduino UNO.**
- **After uploading, reset the Arduino UNO to initialize the program.**
- **Test the car to ensure it moves and behaves as programmed, adjusting the code if needed.**





Inferences:

After uploading the program, the Smart Car AD118 responded to the commands as expected.

The car's movements were observed to align with the programmed instructions.

Proper wiring and component placement were critical for the car's functionality.

Result:

The Smart Car AD118 was successfully built and programmed, functioning correctly as per the given commands and achieving the desired movement