Design, Simulation, and Control of Planar RRR Manipulators using LabVIEW

Youssef Adel Wahba
Department of Mechatronics and Robotics
LabVIEW Robotics Integration Project 2025

December 9, 2025

Abstract

This report details the development and simulation of a 3-Degree-of-Freedom (3-DOF) planar robotic manipulator using the LabVIEW graphical programming environment. The project encompasses four distinct modules: Forward Kinematics (FK), Inverse Kinematics (IK), Trajectory Generation, and Joystick-based Teleoperation. By leveraging LabVIEW's advanced visualization capabilities, Formula Nodes, and parallel loop structures, a complete digital twin of the RRR robot was created. This system allows for real-time validation of kinematic algorithms, workspace analysis using Green's Theorem, and safe path planning before physical deployment. The results demonstrate the efficacy of LabVIEW in bridging the gap between theoretical mechanics and practical control systems.

Keywords: LabVIEW, Robotics, Forward Kinematics, Inverse Kinematics, Trajectory Generation, Teleoperation.

Table of Contents

1 Introduction

Robotic manipulators are ubiquitous in modern industrial automation, requiring precise control systems to manage complex kinematic chains. While traditional programming languages (C++, Python) are widely used, LabVIEW (Laboratory Virtual Instrument Engineering Workbench) offers a unique graphical approach that simplifies the integration of user interfaces (UI) with complex mathematical logic [1].

The purpose of this project suite was to design a virtual testing environment for an RRR (Revolute-Revolute-Revolute) planar arm. This environment serves three primary goals: verification of kinematic equations, visualization of reachable workspaces, and implementation of human-in-the-loop control via standard gaming peripherals.

2 Theoretical Background

2.1 Forward Kinematics (FK)

Forward kinematics resolves the Cartesian position (x, y) and orientation (ϕ) of the end-effector given the joint angles $\theta_1, \theta_2, \theta_3$. For a planar RRR arm with link lengths L_1, L_2, L_3, the transformation is described by summing the link vectors [2]:

$$x_{tip} = L_1 c_1 + L_2 c_{12} + L_3 c_{123} \tag{1}$$

$$y_{tip} = L_1 s_1 + L_2 s_{12} + L_3 s_{123} \tag{2}$$

$$\phi = \theta_1 + \theta_2 + \theta_3 \tag{3}$$

Where c_{12} denotes $\cos(\theta_1 + \theta_2)$ and s_{12} denotes $\sin(\theta_1 + \theta_2)$. These equations form the foundation of the visualization engine.

2.2 Inverse Kinematics (IK)

Inverse kinematics determines the joint angles required to reach a specific target pose (x, y, ϕ). A geometric decoupling approach was utilized, isolating the wrist center (x_w, y_w) first:

$$x_w = x_{target} - L_3 \cos(\phi) \tag{4}$$

$$y_w = y_{target} - L_3 \sin(\phi) \tag{5}$$

The elbow angle θ_2 is derived using the Law of Cosines on the triangle formed by the first two links:

$$\cos(\theta_2) = \frac{x_w^2 + y_w^2 - L_1^2 - L_2^2}{2L_1L_2} \tag{6}$$

This yields two solutions $(\pm\theta_2)$, corresponding to "Elbow Up" and "Elbow Down" configurations. The base angle θ_1 is then solved via trigonometric difference [3]:

$$\theta_1 = \text{atan2}(y_w, x_w) - \text{atan2}(L_2 \sin\theta_2, L_1 + L_2 \cos\theta_2) \tag{7}$$

2.3 Trajectory Generation

To move the end-effector along a straight line in Cartesian space, linear interpolation is applied. The path is discretized into N steps. For a step i where $0 \leq i < N$:

$$P_i = P_{start} + (P_{end} - P_{start}) \cdot \frac{i}{N-1} \tag{8}$$

This interpolation is applied to x, y, and the orientation ϕ simultaneously to ensure smooth, synchronized motion.

3 Methodology and LabVIEW Implementation

The development was divided into four distinct Virtual Instruments (VIs).

3.1 The Geometry Design: Parametric Robot Link SubVI

To achieve a high-fidelity animation for the 3-RRR Planar Robot, a modular visualization approach was adopted. Instead of hard-coding the geometry for all three links directly into the main loop, a dedicated **Robot Arm Section SubVI** was developed from scratch. This design was inspired by a reference 2-Link (2R) robot example but was significantly expanded to support the 3-DOF requirements and modular reuse across four distinct project interfaces (Forward Kinematics, Inverse Kinematics, Trajectory Generation, and Joystick Control).

3.1.1 Geometric Definition and Customization

The arm section is modeled as a 2D parametric object. Unlike static bitmaps, the geometry is defined mathematically using LabVIEW's `Picture Control` functions. This allows for real-time scaling and coloring without pixelation.

The SubVI accepts the following parameters to define a "Section" (Link):

- **Start Position** (x_0, y_0)**:** The Cartesian coordinates of the joint connecting to the previous link.
- **Length** (L)**:** The physical length of the arm segment.
- **Angle** (θ)**:** The absolute angle of the link relative to the global horizontal axis.
- **Visual Style:** Inputs for `Color`, `Width`, and `Joint Radius`.

The visual representation consists of two primary primitives: a **Circle** representing the joint and a **Trapezoid/Rectangle** representing the link body. The vertices of the link body are calculated dynamically:

$$x_{end} = x_0 + L \cdot \cos(\theta) \tag{9}$$

$$y_{end} = y_0 + L \cdot \sin(\theta) \tag{10}$$

This parametric approach ensures that the "skin" of the robot stretches correctly as the link parameters (L_1, L_2, L_3) are adjusted by the user.

3.1.2 LabVIEW Block Diagram Implementation

The internal logic of the SubVI was designed to encapsulate the complexity of the coordinate transformations.

1. Coordinate Transformation Engine: The core logic utilizes a Rotation-Translation algorithm. A defining characteristic of this module is its ability to "chain" geometries. The output of the SubVI provides not only the drawing picture but also the (x_{end}, y_{end}) coordinates. This allows the Main VI to wire the output of *Link 1* directly into the input of *Link 2*, creating a kinematic chain without manual calculation of intermediate points in the upper-level hierarchy.

2. Implementation Challenges: A key challenge during the transition from the 2R reference to the 3-RRR custom build was the handling of the **Picture Control Memory**. In early iterations, redrawing the robot caused flickering or "ghosting" effects where previous frames remained visible. This was resolved by implementing a standard "Erase First" logic within the SubVI or ensuring the Main VI clears the picture buffer before the frame update cycle.

Another challenge was interfacing the SubVI across four different projects. By standardizing the connector pane (Input on Left, Output on Right) and using strict data typing for the coordinates, the module became "Plug-and-Play," significantly reducing the development time for the Trajectory Generation and Joystick Control VIs.

3.1.3 User Interface Integration

The resulting SubVI allows the main Front Panel to remain clean, with complex drawing logic hidden.

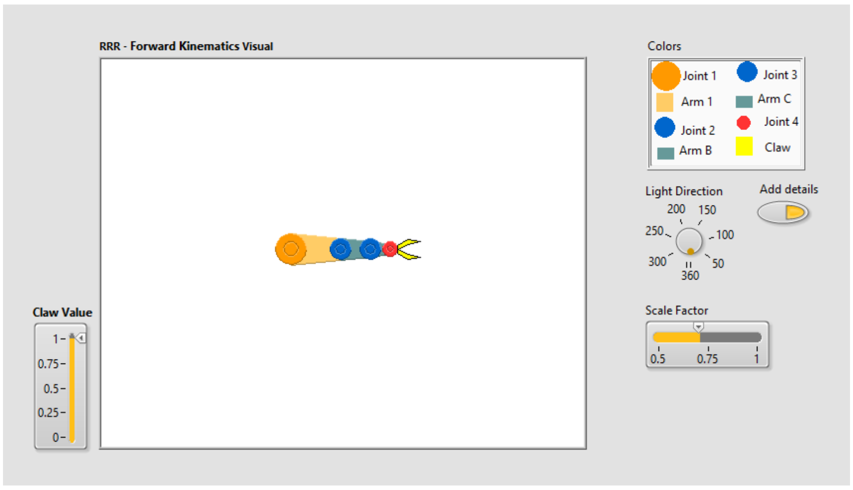


Figure 1: Front Panel of the Parametric Arm Section SubVI. Note the controls for Length, Angle, and Color which allow for individual customization of each robot link.

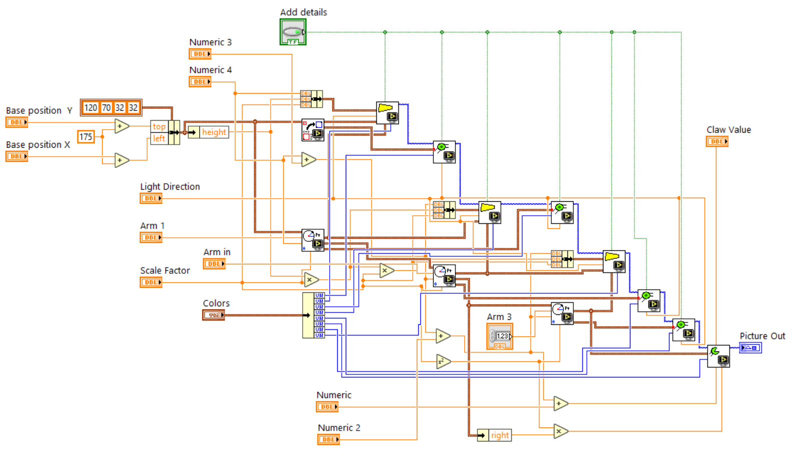


Figure 2: Block Diagram of the Arm Section SubVI. The logic demonstrates the calculation of end-point coordinates based on trigonometric inputs and the bundling of graphics for the Picture Control.

3.2 Forward Kinematics Visualization

The core of the simulation is the visualization engine, designed to provide immediate geometric feedback to the user. Unlike the complex Picture Control rendering used in the final polished interface, the primary verification stage utilizes a lightweight "Stick Figure" model plotted on a standard LabVIEW `XY Graph`.

3.2.1 Geometric Implementation

The transformation logic is encapsulated within a `Formula Node`, which performs the necessary trigonometric calculations to resolve the Cartesian coordinates of each joint. To render the robot links as continuous line segments, the coordinates of each key node—Base, Elbow, Wrist, and Tip—are calculated sequentially.

Using the shorthand notation where $c_i = \cos(\theta_i)$, $s_i = \sin(\theta_i)$, and $c_{ij} = \cos(\theta_i + \theta_j)$, the nodal coordinates are defined as:

- **Joint 1 (Base):** Fixed at the origin.

$$J_1 = (0, 0) \tag{11}$$

- **Joint 2 (Elbow):** Derived from the first link length L_1 and base angle θ_1.

$$J_2 = (L_1 c_1, L_1 s_1) \tag{12}$$

- **Joint 3 (Wrist):** Calculated relative to the elbow position (x_{elbow}, y_{elbow}) using the cumulative angle of the second link.

$$J_3 = (x_{elbow} + L_2 c_{12}, y_{elbow} + L_2 s_{12}) \tag{13}$$

3.2.2 Block Diagram Logic

The LabVIEW implementation follows a linear data flow architecture :

1. **Data Bundling:** The x and y coordinates calculated in the Formula Node are grouped into arrays using the `Bundle` function.

2. **Array Aggregation:** A `Build Array` function aggregates the coordinate clusters from all joints (Base $\to$ Elbow $\to$ Wrist $\to$ Tip) into a single data structure.

3. **Rendering:** This aggregated array is wired directly to an `XY Graph`. The graph is configured to connect points with lines, effectively drawing the rigid links of the manipulator in real-time.

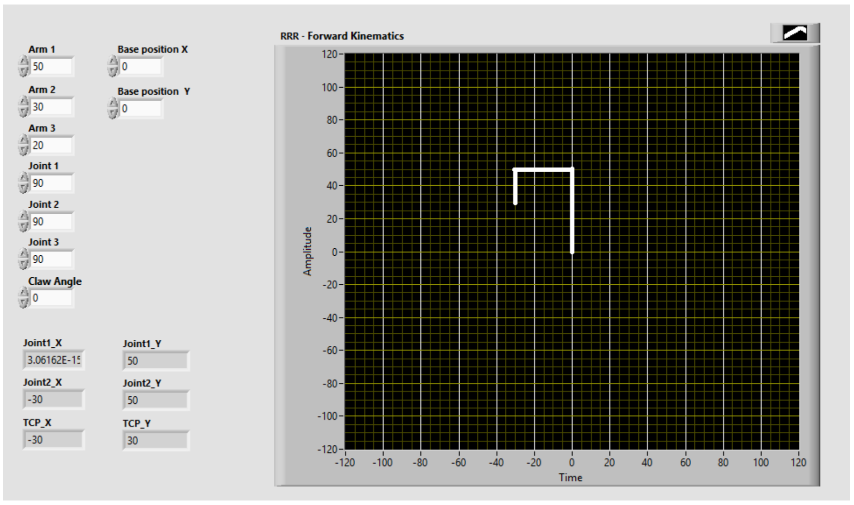

Arm 1: 50
Arm 2: 30
Arm 3: 20
Joint 1: 90
Joint 2: 90
Joint 3: 90
Claw Angle: 0
Base position X: 0
Base position Y: 0
Joint1_X: 3.06162E-15
Joint1_Y: 50
Joint2_X: -30
Joint2_Y: 50
TCP_X: -30
TCP_Y: 30

Figure 3: Front Panel of the Forward Kinematics VI. The XY Graph (right) displays the stick-figure representation, while numeric controls (left) allow manual manipulation of joint angles.

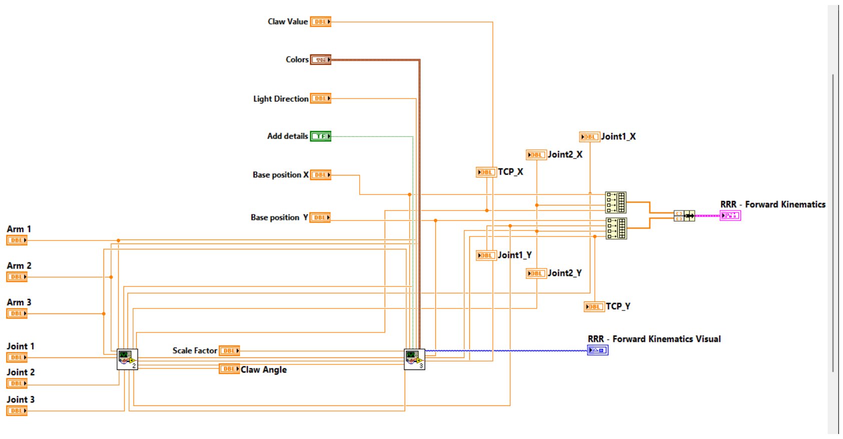


Figure 4: Block Diagram of the Forward Kinematics VI. The logic highlights the flow from the Formula Node (trigonometry) to the Build Array function (visualization).

3.2.3 Challenges and Solutions

A primary challenge during the development of this module was the coordinate system handling. LabVIEW's trigonometric functions accept inputs in **Radians**, whereas standard robotics notation and user interfaces typically use **Degrees**. A conversion stage ($\theta_{rad} = \theta_{deg} \times \frac{\pi}{180}$) was implemented inside the Formula Node to ensure the visual output matched the user input. Additionally, proper array ordering was critical; incorrect bundling orders in the `Build Array` function resulted in "scrambled" links where the wrist connected back to the base, which was resolved by strictly enforcing the kinematic chain order (Base to Tip).

3.3 Inverse Kinematics Solver Implementation

The Inverse Kinematics (IK) Virtual Instrument serves as the computational core for determining the required joint configuration $(\theta_1, \theta_2, \theta_3)$ given a desired end-effector pose (x, y, ϕ). The implementation is grounded in the geometric decoupling method derived in the theoretical background (Section 2.2), utilizing a `Formula Node` to process the non-linear algebraic solutions efficiently.

3.3.1 Algorithm and Multiple Solution Handling

A defining characteristic of the RRR planar manipulator is the existence of two valid geometric solutions for any reachable point within the workspace: the "Elbow Up" and "Elbow Down" configurations.

To provide dynamic control over these postures, a Boolean toggle switch labeled `Elbow Up?` was integrated into the Front Panel. This boolean input modifies the calculation of the intermediate elbow angle θ_2 inside the algorithm. The logic flow is as follows:

1. The magnitude of the elbow angle, $|\theta_2|$, is calculated using the Law of Cosines.
2. A conditional structure checks the state of the Boolean input:

$$\theta_2 = \begin{cases} -|\theta_2| & \text{if Elbow Up is True} \\ +|\theta_2| & \text{if Elbow Up is False} \end{cases} \tag{14}$$

3. The base angle θ_1 is subsequently recalculated based on the signed value of θ_2, ensuring the entire kinematic chain reorients correctly to match the chosen configuration.

3.3.2 Workspace Safety and Stability

In practical robotic simulation, user inputs may inadvertently specify a target coordinate (x, y) that lies outside the robot's reachable workspace (i.e., the target distance $r > L_1 + L_2$). Mathematically, this causes the term inside the arccosine function to exceed the domain $[-1, 1]$, resulting in complex numbers or `NaN` (Not a Number) errors that crash the simulation.

To robustly handle these edge cases, a "Safety Clamp" algorithm was implemented prior to the trigonometric calculation:

$$val = \frac{x_w^2 + y_w^2 - L_1^2 - L_2^2}{2L_1L_2} \tag{15}$$

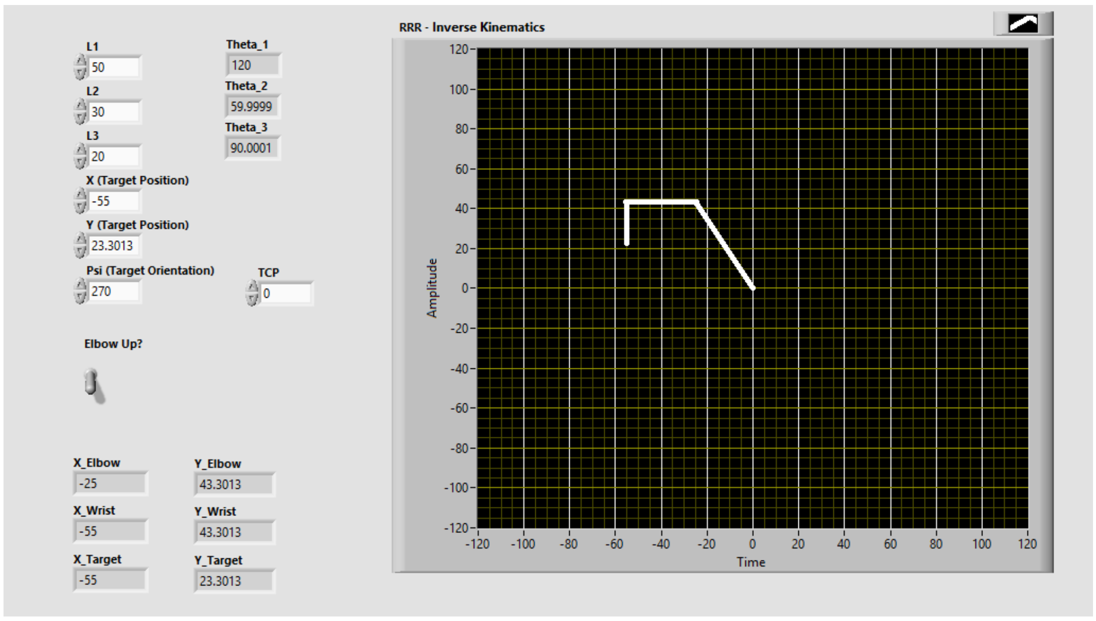
$$val_{clamped} = \max(-1, \min(1, val)) \tag{16}$$

$$\theta_2 = \arccos(val_{clamped}) \tag{17}$$

This mechanism ensures simulation stability. If the user attempts to reach an out-of-bounds point, the robot arm will extend fully towards the target and "lock" at the boundary rather than generating mathematical errors.

3.3.3 LabVIEW Block Diagram Structure

The VI is structured around a central `Formula Node` which accepts the geometric parameters (L_1, L_2, L_3) and target variables $(X_{target}, Y_{target}, \psi_{target})$. The computed joint angles are output in degrees and wired to the visualization SubVI to update the graphical display immediately.



L1 50
L2 30
L3 20
X (Target Position) -55
Y (Target Position) 23.3013
Psi (Target Orientation) 270
TCP 0
Theta_1 120
Theta_2 59.9999
Theta_3 90.0001
Elbow Up?
X_Elbow -25
Y_Elbow 43.3013
X_Wrist -55
Y_Wrist 43.3013
X_Target -55
Y_Target 23.3013

Figure 5: Front Panel of the Inverse Kinematics VI. The interface allows users to input target Cartesian coordinates and toggle the "Elbow Up" switch to observe the reconfiguration of the arm.

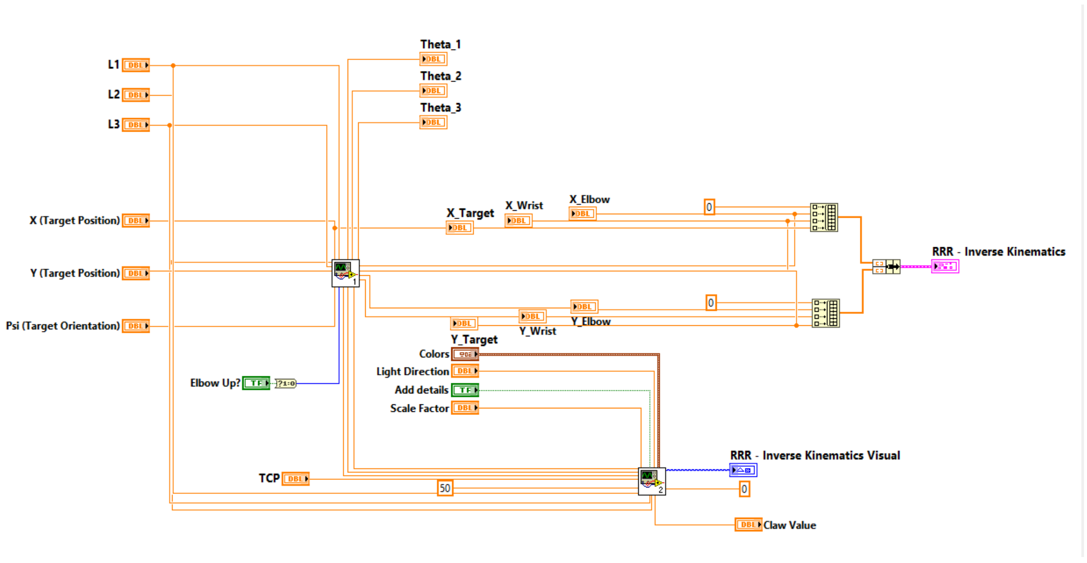


Figure 6: Block Diagram of the Inverse Kinematics VI. The Formula Node (center) encapsulates the geometric equations and safety clamps, isolating the complex math from the data flow wiring.

4 Real-Time Control Module

4.1 Joystick Teleoperation and Input Mapping

To transition from static simulation to dynamic human-in-the-loop control, a teleoperation module was developed. This system interfaces a physical joystick hardware unit with the LabVIEW kinematic engine, allowing for intuitive, real-time manipulation of the robotic arm.

4.1.1 Input Acquisition and Scaling Algorithm

The core control logic relies on the `Input Device Control` palette (or equivalent Analog I/O interface) to read the raw state of the joystick axes. The hardware utilized is an Arduino-based Joystick Arm Shield, which provides dual-axis analog control for the base and shoulder joints, along with digital inputs for the gripper state.

The raw data stream from the hardware typically arrives as a 16-bit integer (ranging from -32768 to 32767) or a normalized voltage signal. To drive the robot model, this raw input V_{raw} must be mapped to the physical joint limits of the manipulator (e.g., $\theta_{min} = -90^\circ$ to $\theta_{max} = 90^\circ$). The linear scaling algorithm implemented is defined as:

$$\theta_{target} = \theta_{min} + \left(\frac{V_{raw} - V_{min}}{V_{max} - V_{min}} \right) \times (\theta_{max} - \theta_{min}) \tag{18}$$

This equation ensures that the full deflection of the joystick corresponds exactly to the maximum reachable angle of the specified joint.

4.1.2 Real-Time Loop Architecture

The control VI operates within a high-priority `While Loop` to ensure low-latency performance. In each iteration, the system:

1. polls the hardware for new axis data,
2. applies the scaling algorithm to convert voltage to degrees,
3. passes the resulting angles to the Forward Kinematics engine, and
4. updates the visual display.

This architecture provides immediate visual feedback, creating a closed-loop perception cycle for the operator.

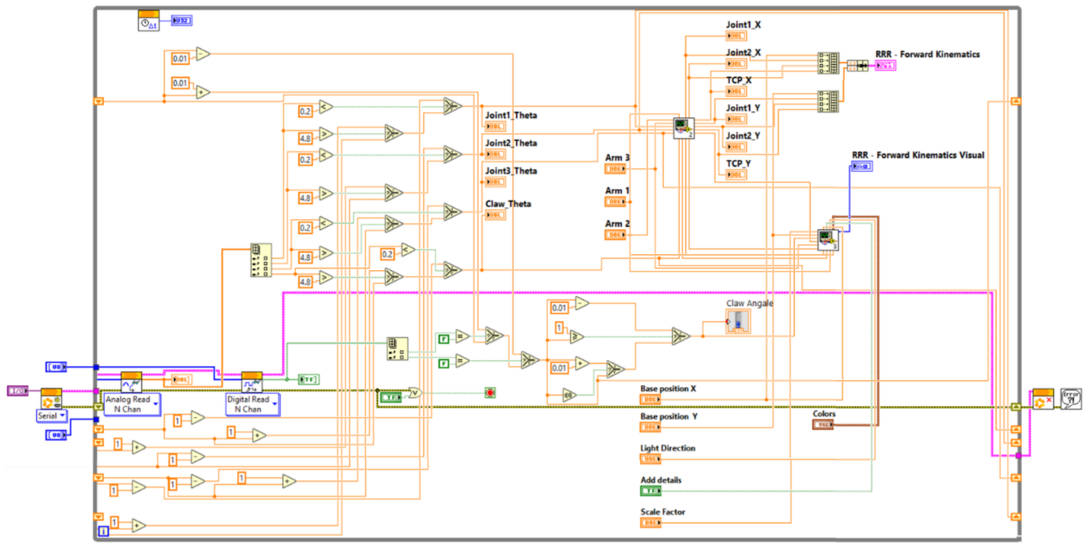


Figure 7: Block Diagram of the Joystick Teleoperation VI. The logic captures analog signals, scales them to joint angles, and feeds them directly into the Forward Kinematics visualization loop.

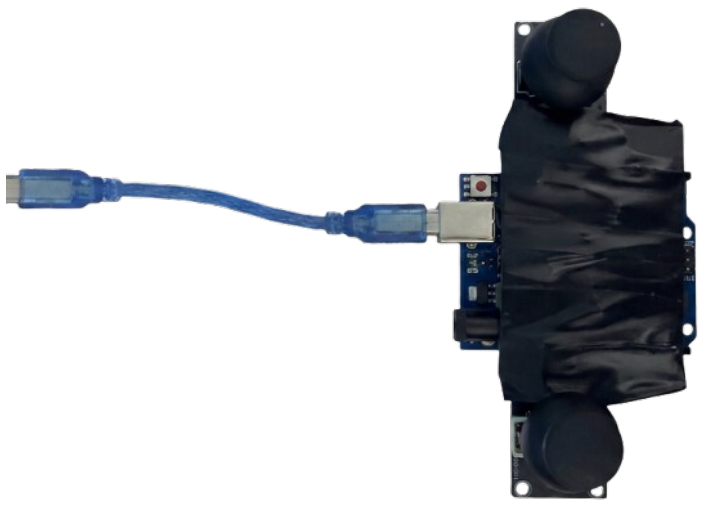


Figure 8: The Hardware Interface: An Arduino-based Joystick Arm Shield used to drive the simulation. The dual thumbsticks provide intuitive X-Y control for the planar robot.

4.2 System Demonstration

A comprehensive video demonstration of the 3-RRR Planar Robot, showcasing the Forward Kinematics, Inverse Kinematics, and Real-Time Joystick Control modules, is available online. The video illustrates the synchronization between the user inputs and the graphical simulation.

Watch the demonstration video here:
`https://youtu.be/pRDtrZHI3r8`

5 Trajectory Generation Module

5.1 Linear Trajectory Generation and Interpolation

The Trajectory Generation module advances the simulation from static pose solving to dynamic path planning. The primary objective is to move the robot's end-effector along a straight line in Cartesian space between a Start Point $P_0(x_0, y_0)$ and an End Point $P_f(x_f, y_f)$, while simultaneously synchronizing the orientation (ϕ), Tool Center Point (TCP) angle, and Claw state.

5.1.1 Interpolation Algorithm

To achieve smooth, synchronized motion, a Linear Interpolation (Lerp) algorithm is implemented within a `Formula Node`. The path is discretized into N steps (controlled by the user input `Steps`). For any given iteration i (where $0 \leq i < N$), the percentage of completion λ is defined as:

$$\lambda = \frac{i \cdot 1.0}{N - 1.0} \tag{19}$$

Using this scalar λ, the instantaneous target values for all kinematic variables are calculated simultaneously. This ensures that the wrist rotation and gripper actuation finish exactly at the moment the robot reaches the target coordinates.

Cartesian Position:

$$x_{target} = x_0 + (x_f - x_0) \cdot \lambda \tag{20}$$

$$y_{target} = y_0 + (y_f - y_0) \cdot \lambda \tag{21}$$

Orientation and End-Effector State:

$$\phi_{target} = \phi_{start} + (\phi_{end} - \phi_{start}) \cdot \lambda \tag{22}$$

$$TCP_{target} = TCP_{start} + (TCP_{end} - TCP_{start}) \cdot \lambda \tag{23}$$

$$Claw_{value} = Claw_{start} + (Claw_{end} - Claw_{start}) \cdot \lambda \tag{24}$$

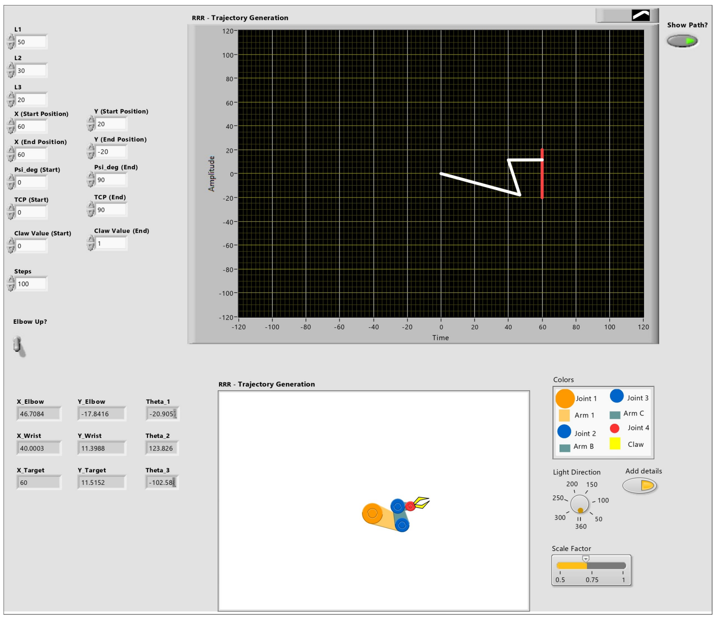
5.1.2 LabVIEW Implementation Architecture

The VI utilizes a specific "Calculation-Animation" split architecture to separate the mathematical solving from the visual rendering.

1. **Calculation Loop (Fast):** The first `For Loop` executes the interpolation equations N times. In each iteration, the calculated (x_t, y_t, ϕ_t) values are passed to the Inverse Kinematics SubVI. The resulting joint angles are auto-indexed at the loop boundary, producing arrays of size N representing the entire motion profile.

2. **Path Visualization Strategy:** A challenge arose in visualizing the trajectory path. The arrays for x_{target} and y_{target} are extracted from the calculation loop without indexing (as a whole array). These are overlayed on the XY Graph as a static red line (Plot 1), providing a visual guide for the path.

3. **Animation Loop (Timed):** A second `For Loop` accepts the auto-indexed arrays of joint angles. A `Wait (ms)` function inside this loop governs the playback speed, allowing the user to observe the motion in real-time. Inside this loop, the Forward Kinematics Visualization SubVI updates the robot's pose (Plot 0) frame-by-frame.



L1	50
L2	30
L3	20
X (Start Position)	60
Y (Start Position)	20
X (End Position)	60
Y (End Position)	-20
Psi_deg (Start)	0
Psi_deg (End)	90
TCP (Start)	0
TCP (End)	90
Claw Value (Start)	0
Claw Value (End)	1
Steps	100

X_Elbow 46.7084	Y_Elbow -17.8416	Theta_1 -20.9057
X_Wrist 40.0003	Y_Wrist 11.3988	Theta_2 123.826
X_Target 60	Y_Target 11.5152	Theta_3 -102.58

Figure 9: Front Panel of the Trajectory Generation VI. The graph displays the static red path overlay, while the robot (blue stick figure) animates along the trajectory from start to finish.

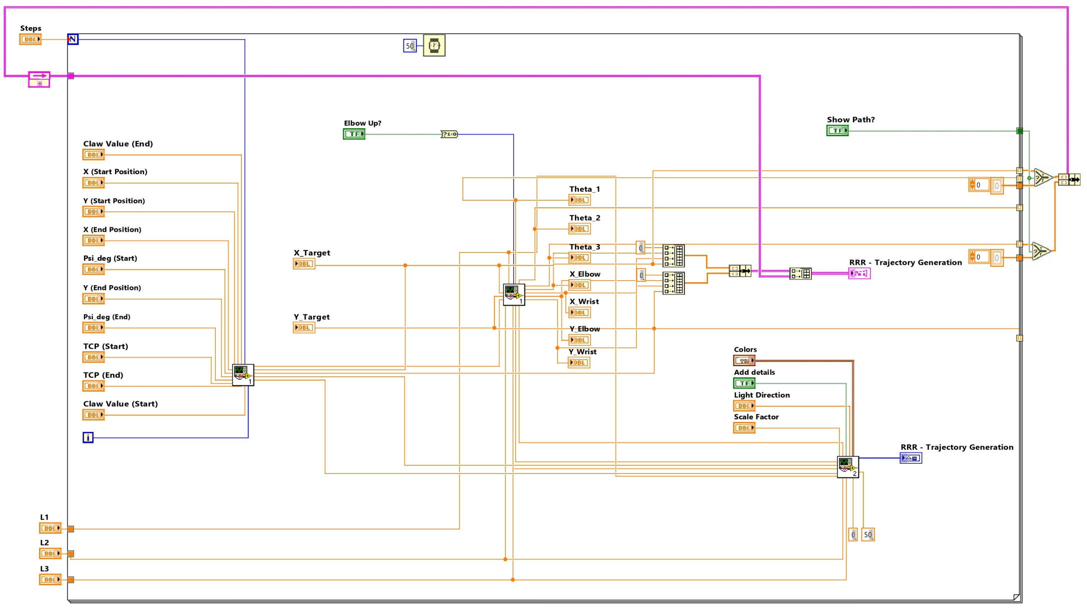


Figure 10: Block Diagram showing the dual-loop architecture. The left loop handles the math and interpolation, while the right loop handles the animation timing and multi-plot graph generation.

6 Results and Discussion

6.1 Simulation Performance

The Forward Kinematics module successfully verified the geometric model. When coupled with the trace buffer (using Shift Registers), the system could draw complex workspace boundaries, confirming the theoretical workspace limits calculated via Green's Theorem.

6.2 Inverse Kinematics Robustness

The IK solver demonstrated accuracy within the reachable workspace. The singularity at full extension ($\theta_2 = 0$) was handled robustly by the safety clamps. The "Elbow Up/Down" switch functioned instantaneously, verifying the duality of the kinematic solutions.

6.3 Challenges Encountered

A significant challenge during trajectory generation was the data type conflict between the static path plot and the dynamic robot plot. The `Build Array` function initially failed because the animation loop was auto-indexing the path array, converting it to a scalar. This was resolved by disabling indexing on the specific input tunnel, preserving the array structure for the static plot overlay.

7 Conclusion and Future Work

This project successfully demonstrated the capability of LabVIEW to model, simulate, and control a planar RRR manipulator. The graphical nature of G-code facilitated rapid debugging of complex vector mathematics and provided immediate visual verification of kinematic theories.

Future improvements will focus on:

- **Dynamics:** Implementing mass and inertia matrices to simulate torque requirements.
- **Hardware-in-the-Loop (HIL):** Interfacing the VIs with physical servo motors (e.g., Arduino or NI myRIO) to drive a real 3D-printed arm.
- **Obstacle Avoidance:** Implementing distance algorithms to prevent collisions with defined workspace obstacles.

8 Project Resources and Source Code

8.1 Video Demonstrations

A complete playlist containing verified test runs for all modules (Forward Kinematics, Inverse Kinematics, Joystick Control and Trajectory Generation) is available online. These videos demonstrate the real-time performance of the VIs described in this report.

- **YouTube Playlist:** `Click here to watch the full demonstration playlist`

8.2 Code Repository

The complete source code, including all Main VIs, SubVIs, and project documentation, has been uploaded to GitHub. This repository allows for version control and open-source collaboration.

- **GitHub Repository:** `youssefadell11/Robtics-LabVIEW-2025`

Note: All VIs were developed and tested using LabVIEW 2020.

References

[1] National Instruments, *Labview core 1 course manual*, Emerson, 2024.

[2] M. W. Spong, S. Hutchinson, and M. Vidyasagar, *Robot Modeling and Control.* John Wiley & Sons, 2005.

[3] J. J. Craig, *Introduction to Robotics: Mechanics and Control*, 3rd ed. Pearson Prentice Hall, 2005.