



OrionKine Lab

USER MANUAL

ROBOTICS DESIGN & SIMULATION PLATFORM

VERSION 1.0.0 / REVISION 1.2

DEVELOPED BY

PRIYANSHU WAGHDHARE

Updated 30 October 2026

DOCUMENT INFORMATION

DOCUMENT TITLE	ORIONKINE LAB USER MANUAL
SOFTWARE	ORIONKINE LAB
VERSION	1.0.0
DOCUMENT VERSION	1.2
AUTHOR	PRIYANSHU WAGHDHARE
RELEASE DATE	JULY 2026
LANGUAGE	ENGLISH

COPYRIGHT © 2026 PRIYANSHU WAGHDHARE.

ALL RIGHTS RESERVED.

THIS DOCUMENT IS PROVIDED AS A GUIDE FOR USING THE ORIONKINE LAB ROBOTICS DESIGN & SIMULATION PLATFORM.

NO PART OF THIS PUBLICATION MAY BE REPRODUCED, DISTRIBUTED, OR TRANSMITTED IN ANY FORM WITHOUT PRIOR WRITTEN PERMISSION FROM THE AUTHOR.

DISCLAIMER

ORIONKINE LAB IS INTENDED FOR EDUCATIONAL, RESEARCH AND DEVELOPMENT PURPOSES. WHEN CONTROLLING PHYSICAL ROBOTS, USERS ARE RESPONSIBLE FOR VERIFYING ROBOT CONFIGURATIONS, MOTION PATHS AND WORKSPACE SAFETY BEFORE EXECUTION.

ALWAYS VERIFY ROBOT MOTION IN SIMULATION BEFORE EXECUTING IT ON PHYSICAL HARDWARE. INCORRECT ROBOT CONFIGURATION OR PROGRAMMING MAY RESULT IN UNEXPECTED ROBOT MOVEMENT.

Revision 1.2 - Updated 30 October 2026

TABLE OF CONTENTS

01	INTRODUCTION	1
02	CREATING YOUR FIRST ROBOT	4
03	ROBOT CONFIGURATION	8
04	FORWARD KINEMATICS	10
05	WAYPOINT PROGRAMMING	12
06	TCP ORIENTATION CONTROL	16
07	WORKSPACE OBJECTS	24
08	OBJECT-BASED WAYPOINT GENERATION	27
09	TOOLS	31
10	AUTOMATION	33
11	APPLICATION MENU	36
12	KINEMATICS LAB	40
13	COLLISION DETECTION	44
14	ORIONKINE LAB ROADMAP	46

CHAPTER 1

INTRODUCTION

1.1 WELCOME TO ORIONKINE LAB

Welcome to OrionKine Lab, a desktop robotics design and simulation platform developed to simplify the design, simulation and programming of robotic manipulators.

OrionKine Lab provides an intuitive environment for creating custom robotic arms, simulating robot motion and validating robot behaviour before deploying programs to physical hardware.

The software combines robot modelling, forward and inverse kinematics, trajectory planning and hardware communication into a single application, making robotics development more accessible for education, research and industrial prototyping.

1.2 PURPOSE OF ORIONKINE LAB

OrionKine Lab was developed to provide a lightweight and easy-to-use robotics platform that bridges the gap between simulation and real-world robot control.

The application enables users to create custom robotic manipulators, configure their mechanical structure, simulate robot movement and communicate with compatible robot controllers through a unified interface.

OrionKine Lab focuses on providing practical robotics tools without requiring complex external robotics frameworks, making it suitable for beginners as well as experienced robotics developers.

1.3 TARGET AUDIENCE

- Students learning robotics and automation
- Robotics enthusiasts and hobbyists
- Robotics researchers
- Automation engineers
- Developers building custom robotic manipulators

1.4 KEY FEATURES

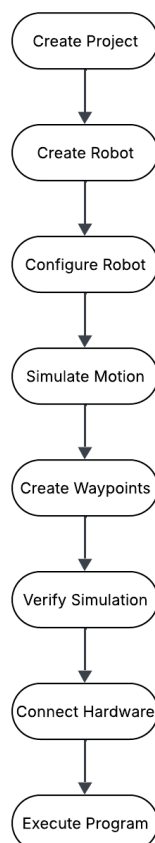
FEATURE DESCRIPTION

ROBOT CREATION

Create custom robotic manipulators with configurable joints and links.

FORWARD KINEMATICS	Simulate robot movement by controlling individual joint angles.
INVERSE KINEMATICS	Calculate joint angles from a desired tool position and orientation.
MOTION PROGRAMMING	Create MoveJ, MoveL and MoveC robot motions.
WAYPOINT PROGRAMMING	Build and execute waypoint-based robot programs.
TOOL CONFIGURATION	Configure Gripper, Suction and Magnetic end effectors.
SERIAL COMMUNICATION	Connect OrionKine Lab to compatible robot controllers.
FIRMWARE SUPPORT	provided a reference firmware for basic hardware.
PROJECT MANAGEMENT	Save, load and manage complete robot projects.
KINEMATICS LAB	Explore robot kinematics through live equations, matrices and synchronized 2D visualization.

1.5 TYPICAL ORIONKINE LAB WORKFLOW



1.6 MAIN INTERFACE

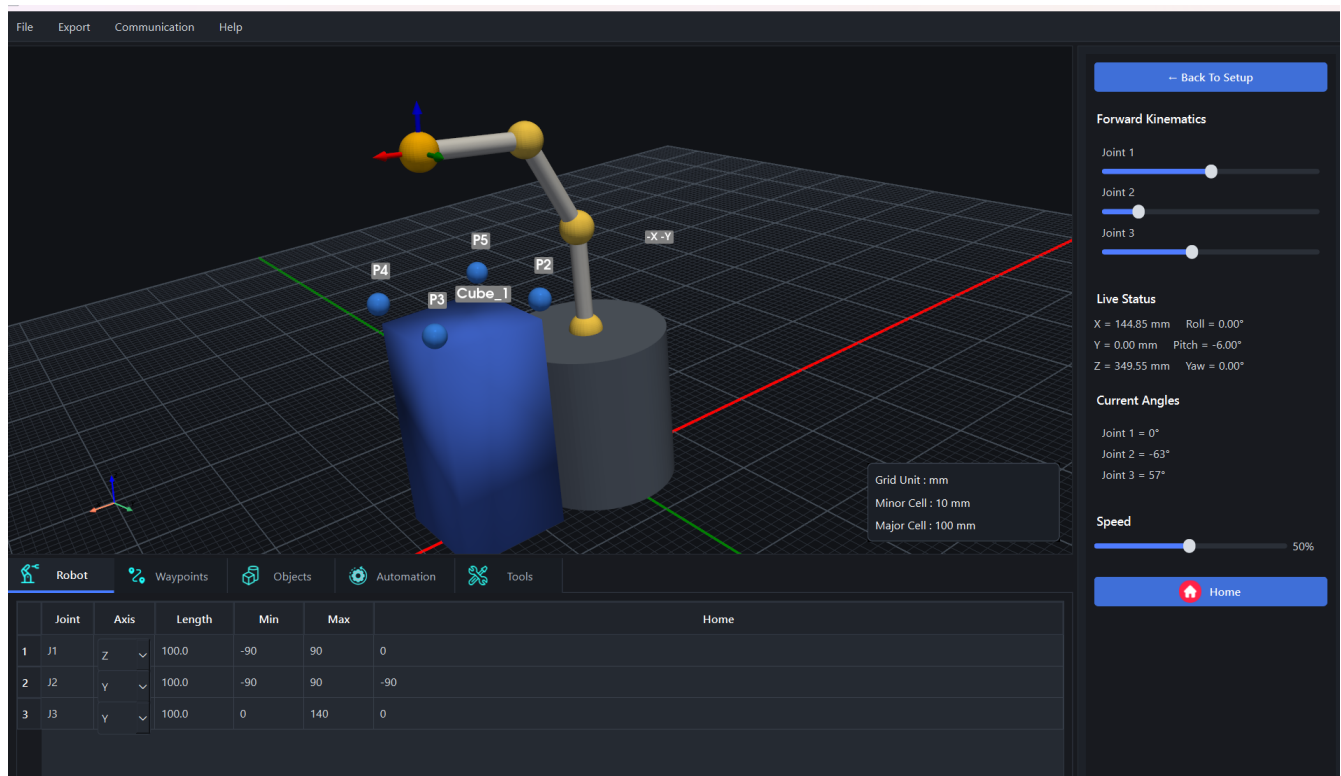


FIGURE 1.1 - ORIONKINE LAB MAIN APPLICATION INTERFACE

FIGURE 1.1 ILLUSTRATES THE MAIN ORIONKINE LAB WORKSPACE. THE APPLICATION CONSISTS OF A 3D SIMULATION ENVIRONMENT, ROBOT CONFIGURATION TOOLS, MOTION CONTROLS, WORKSPACE MANAGEMENT PANELS, AND A MENU BAR PROVIDING ACCESS TO PROJECT, COMMUNICATION, AND EXPORT FUNCTIONS. EACH COMPONENT WILL BE DESCRIBED IN DETAIL IN LATER CHAPTERS.

CHAPTER 2

CREATING YOUR FIRST ROBOT

In this chapter, you will create your first robot using OrionKine Lab. Throughout this guided tutorial, you will become familiar with the application's interface, configure a robotic manipulator, and understand the purpose of each control before moving on to robot simulation and programming.

STEP 1 - LAUNCH ORIONKINE LAB

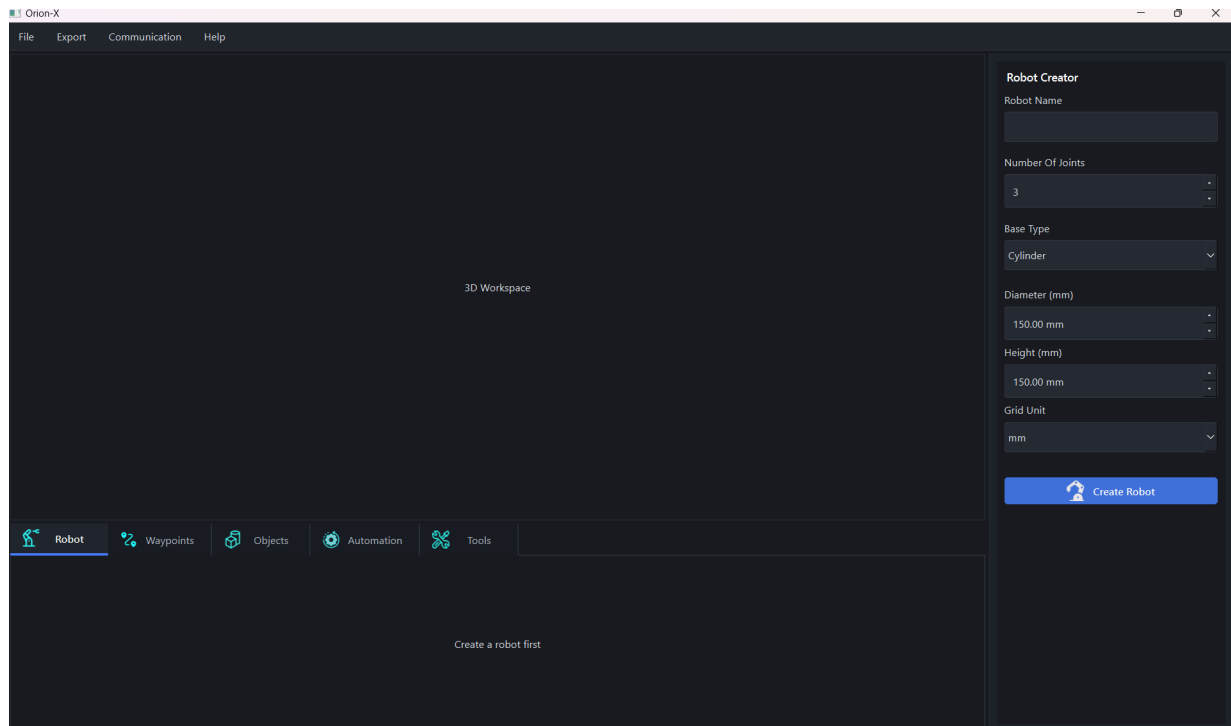
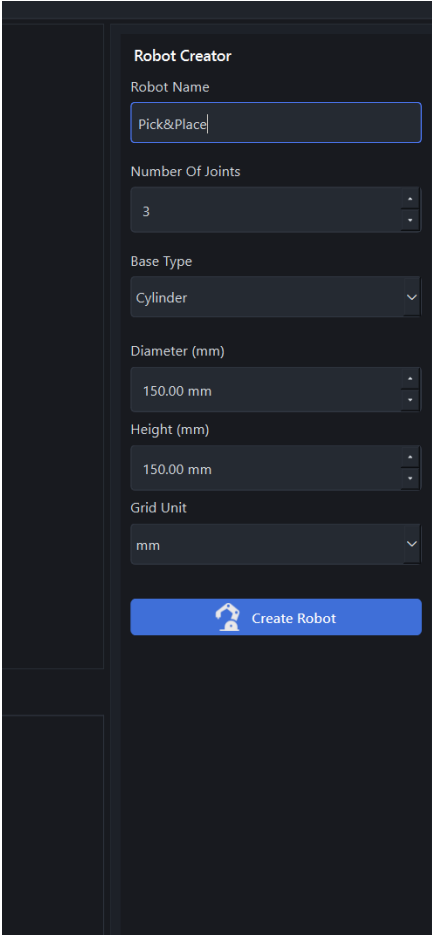


FIGURE 2.1 - ORIONKINE LAB HOME APPLICATION INTERFACE

When OrionKine Lab starts, the main application window is displayed. At this stage, no robot has been created, so the 3D workspace is empty. The Robot Creator panel, located on the right side of the application, is used to configure a new robot. This includes specifying the robot name, selecting the number of Degrees of Freedom (DOF), choosing the base type and its dimensions, and defining the workspace grid unit. Once the required parameters have been configured, the robot can be created using the Create Robot button. The configuration options available in the Robot Creator panel will be explained step by step in the following sections of this chapter.

STEP 2 - CONFIGURE THE ROBOT NAME



The screenshot shows a 'Robot Creator' panel with the following fields and options:

- Robot Name:** A text input field containing 'Pick&Place'.
- Number Of Joints:** A dropdown menu set to '3'.
- Base Type:** A dropdown menu set to 'Cylinder'.
- Diameter (mm):** A text input field set to '150.00 mm'.
- Height (mm):** A text input field set to '150.00 mm'.
- Grid Unit:** A dropdown menu set to 'mm'.
- Create Robot:** A blue button with a robot icon and the text 'Create Robot'.

FIGURE 2.2 - ROBOT CREATOR PANEL

The first step in creating a robot is assigning a Robot Name. The Robot Name uniquely identifies the robot within the current project and is used when saving and managing OrionKine Lab project files. A descriptive name helps distinguish multiple robot configurations and makes project organization easier. It is recommended to use meaningful names that reflect the robot's purpose or configuration.

EXAMPLE ROBOT NAMES:

- PICKANDPLACE_3DOF
- EDUCATIONALROBOT
- WAREHOUSEMANIPULATOR
- SCARA_TEST



TIP: USE MEANINGFUL NAMES THAT REFLECT THE ROBOT'S PURPOSE OR CONFIGURATION.

STEP 3 - SELECT THE NUMBER OF JOINTS

The Number of Joints defines the Degrees of Freedom (DOF) of the robot. Each joint represents one independent rotational axis, allowing the robot to achieve increasingly complex movements.

OrionKine Lab currently supports robots with 3 to 6 Degrees of Freedom, enabling users to design a wide range of robotic manipulators suitable for educational, research, and industrial applications.

For this tutorial, select 3 as the number of joints.

Refer to **Figure 2.2** for the location of the Number of Joints selector.

STEP 4 - CONFIGURE THE ROBOT BASE

The Robot Base represents the fixed foundation of the robotic manipulator and defines its physical appearance within the simulation.

OrionKine Lab currently supports the following base types:

- Cylinder – Suitable for most articulated industrial robots.
- Cube – Useful for educational demonstrations and custom robot designs.

After selecting the desired base type, configure its dimensions by specifying the Diameter and Height. These values determine the size of the robot base displayed in the 3D workspace. For this tutorial, leave the default values unchanged.

Refer to **Figure 2.2** for the Base Type, Diameter, and Height configuration options.

 **Tip: The base dimensions affect only the simulated geometry of the robot and can be adjusted to match the physical dimensions of your robot design.**

STEP 5 - SELECT THE GRID UNIT

The Grid Unit determines the measurement unit displayed throughout the OrionKine Lab workspace. Available units include:

- Millimetres (mm)
- Centimetres (cm)
- Metres (m)

Although different units can be selected for convenience, OrionKine Lab internally performs all calculations using millimetres (mm) to ensure consistent kinematic and motion-planning accuracy.

For this tutorial, leave the Grid Unit set to Millimetres (mm).

Refer to **Figure 2.2** for the location of the Grid Unit selector.

 **Note: Changing the displayed Grid Unit does not affect the robot's calculated motion; it only changes how dimensions are presented in the interface.**

STEP 6 - CREATE THE ROBOT

After completing the Robot Creator configuration, click the Create Robot button to generate the robot model and initialize the OrionKine Lab simulation environment.

Once the robot has been successfully created, the application automatically switches to the Robot Control interface, where the robot can be configured, simulated, and programmed.

Refer to **Figure 2.2** to view the complete Robot Control interface after robot creation.

 **Note:** OrionKine Lab allows you to return to the Robot Creator interface at any time by clicking the Back to Setup button. This makes it easy to modify the robot configuration and regenerate the robot whenever required.

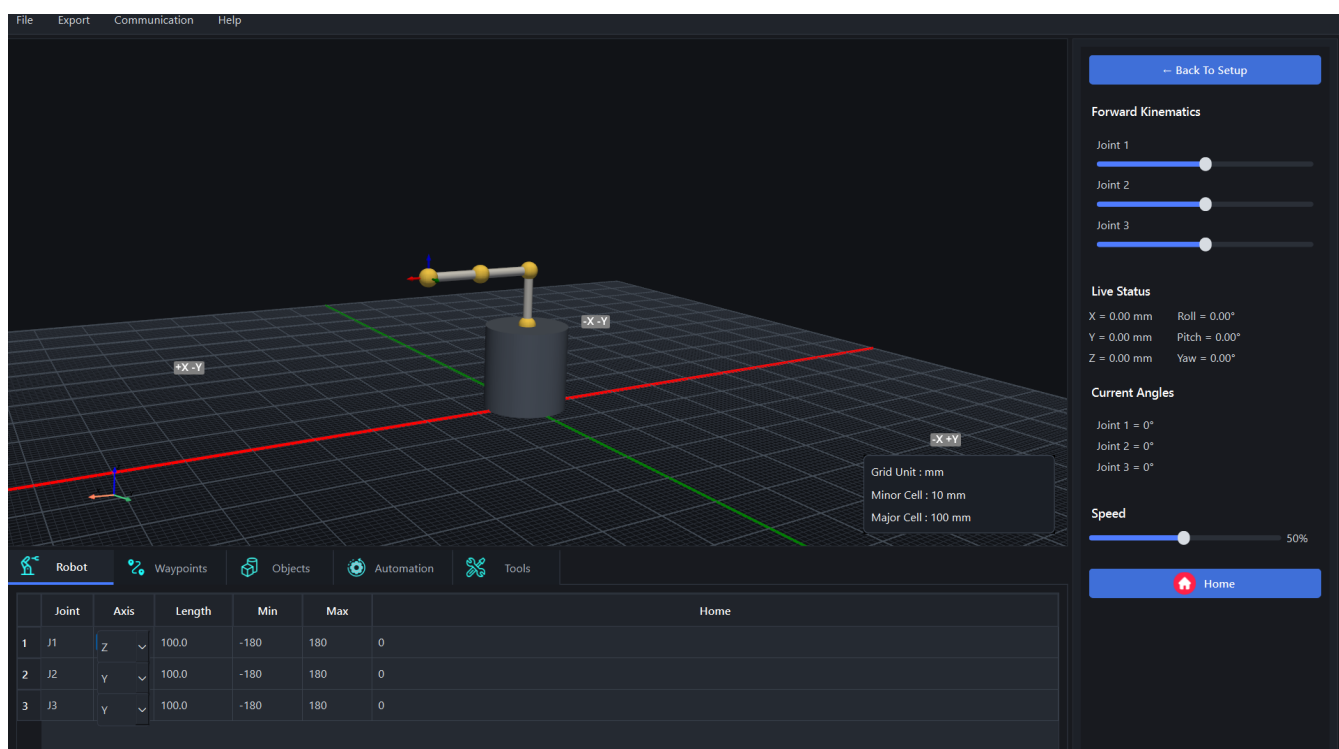


FIGURE 2.3 - ROBOT SUCCESSFULLY CREATED

The robot has now been successfully created and is ready for further configuration. The application automatically switches to the Robot Control interface, where robot parameters can be adjusted and motion can be simulated.

CHAPTER 3

ROBOT CONFIGURATION

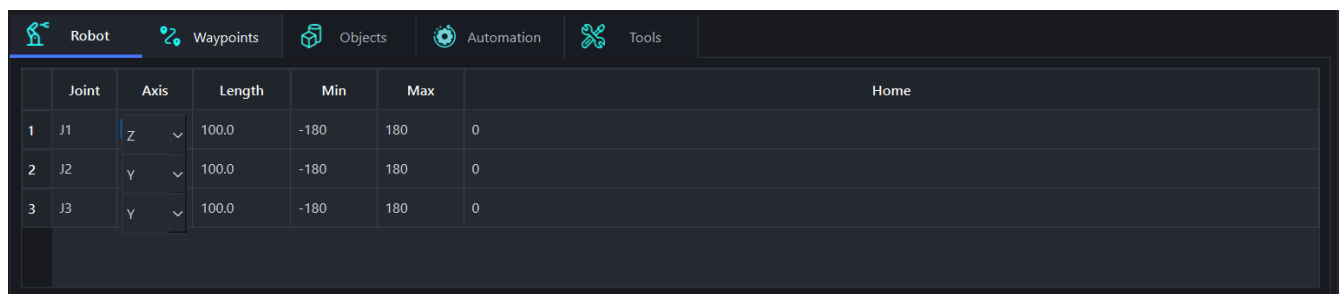
3.1 ROBOT CONFIGURATION OVERVIEW

The Robot Configuration page allows you to define the physical structure and motion limits of the robot. Using the configuration table, you can modify the properties of each joint to match the design of your physical or virtual robot.

Each row in the table represents an individual robot joint, while each column defines a specific parameter such as the joint rotation axis, link length, home position, and allowable rotation limits. Any changes made to these parameters are immediately reflected in the 3D workspace, providing real-time visual feedback during the robot design process.

The Robot Configuration page serves as the foundation for the robot's kinematic model. Properly configuring these parameters ensures accurate robot movement, simulation, and motion planning throughout OrionKine Lab.

Refer to **Figure 3.1** for the default Robot Configuration page displayed immediately after creating a new robot.



	Joint	Axis	Length	Min	Max	Home
1	J1	Z	100.0	-180	180	0
2	J2	Y	100.0	-180	180	0
3	J3	Y	100.0	-180	180	0

 **Note:** OrionKine Lab automatically generates a default robot configuration based on the selected number of joints. These default values can be modified at any time to match the dimensions and movement constraints of your robot.

3.2 CONFIGURING THE ROBOT PARAMETERS

The Robot Configuration table allows you to customize the physical dimensions and motion characteristics of each robot joint. Every row represents an individual joint, while each column defines a specific property of that joint.

The configurable parameters include:

- **Link Length** – Specifies the distance between the current joint and the next joint in the robot structure.
- **Joint Rotation Axis** – Defines the axis (X, Y, or Z) around which the joint rotates.
- **Home Angle** – Specifies the default joint position used when the robot is returned to its home position.
- **Minimum Rotation Angle** – Defines the lowest allowable rotation for the joint.
- **Maximum Rotation Angle** – Defines the highest allowable rotation for the joint.

To modify a parameter, simply select the desired cell in the table and enter the new value. OrionKine Lab immediately applies the changes and updates the robot model within the 3D workspace, allowing you to verify the configuration in real time.

Refer to Figure 3.2 for an example of an updated Robot Configuration table.

Figure 3.2 - Updated Robot Configuration Table

Robot						
Waypoints						
Objects						
Automation						
Tools						
	Joint	Axis	Length	Min	Max	Home
1	J1	Z	100.0	-90	90	0
2	J2	Y	100.0	-90	90	-90
3	J3	Y	100.0	0	140	0

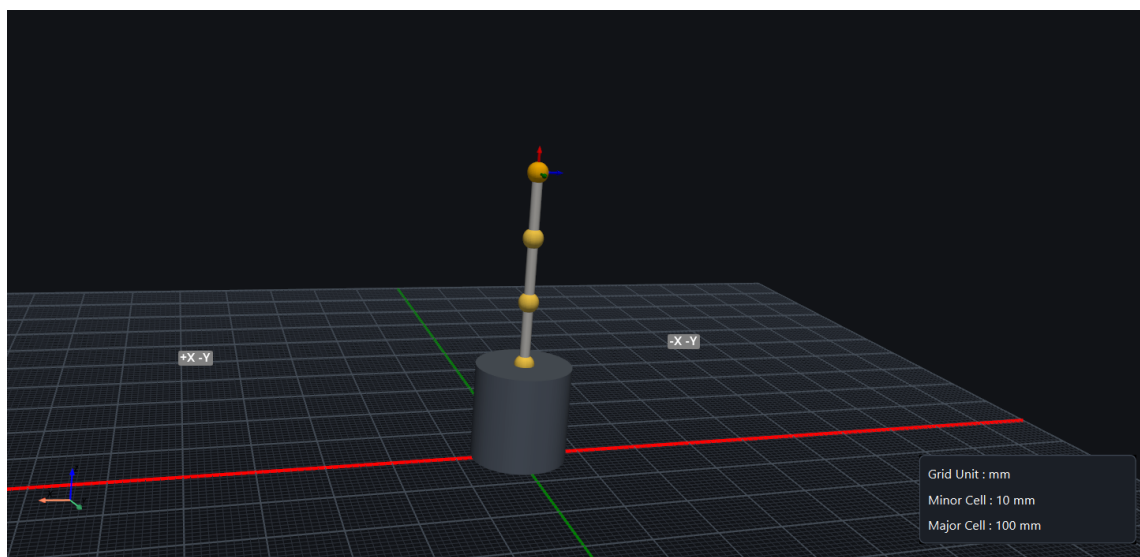
Note: Any modifications made to the Robot Configuration table are reflected immediately in the 3D workspace. This real-time synchronization enables users to verify robot geometry and joint constraints without restarting the simulation.

3.2 CONFIGURING THE ROBOT PARAMETERS

After modifying the Robot Configuration parameters, the robot model is automatically regenerated using the updated dimensions and joint settings. The 3D workspace immediately reflects these changes, allowing users to verify the robot's physical structure before proceeding to robot control and motion planning.

Refer to Figure 3.3 to view the robot after applying the updated configuration.

Figure 3.3 - Robot Model After Configuration Update



Tip: Review the robot model after making configuration changes to ensure that the link lengths, joint orientations, and motion limits accurately represent the intended robot design before continuing to the next chapter.

CHAPTER 4

FORWARD KINEMATICS

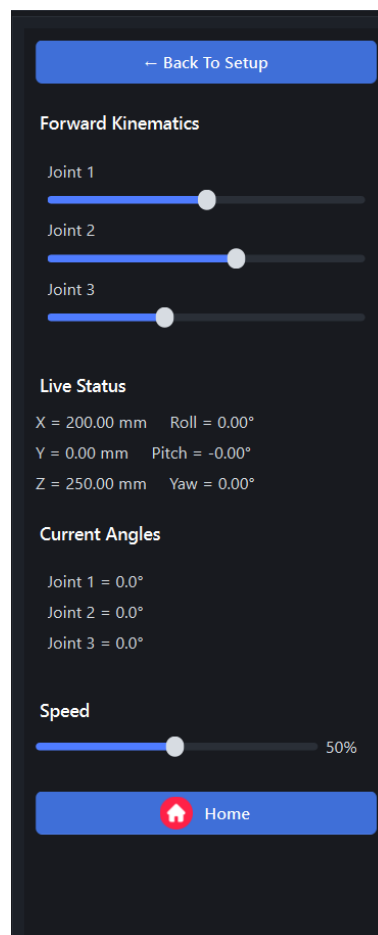
4.1 FORWARD KINEMATICS

The Forward Kinematics page allows you to manually control the robot by adjusting the position of each joint using the available joint sliders. As the joint angles are modified, OrionKine Lab continuously calculates the robot's end-effector position and orientation, providing an accurate real-time representation of the robot's movement within the 3D workspace.

In addition to manual joint control, the Forward Kinematics page provides several tools for monitoring and managing the robot during simulation.

Refer to Figure 4.1 for the layout of the Forward Kinematics page.

Figure 4.1 - Forward Kinematics Interface



The primary components of the Forward Kinematics page are described below:

Forward Kinematics Sliders

The joint sliders allow each robot joint to be rotated independently within its configured rotation limits. Adjusting a slider immediately updates the robot model in the 3D workspace, allowing users to observe the resulting motion in real time.

Live Status Panel

The Live Status panel continuously displays the current position and orientation of the robot's end effector. The displayed values include:

- X Position
- Y Position
- Z Position
- Roll
- Pitch
- Yaw

These values are updated automatically whenever the robot moves, providing continuous feedback during simulation.

Live Joint Angles Monitor

The Live Joint Angles section displays the current rotation angle of every robot joint. The values shown in this panel always correspond to the robot's actual simulated position and are updated in real time as the robot moves.

Move Home Button

The Move Home button returns every joint to its configured Home Angle defined in the Robot Configuration page. This provides a quick and reliable method for restoring the robot to its default starting position before beginning another simulation.

Speed Slider

The Speed Slider controls the movement speed of the simulated robot. Increasing the speed results in faster robot motion, while reducing the speed allows for slower and more precise movements during testing and demonstration.

Back to Setup Button

The Back to Setup button returns the user to the Robot Creator interface. This allows the robot configuration to be modified at any time without creating a new project. After making the required changes, simply create the robot again to continue working with the updated configuration.

 **Note:** All robot movements performed using the Forward Kinematics controls are simulated in real time. Any adjustments made using the joint sliders immediately update the robot model, the Live Status panel, and the Live Joint Angles monitor, ensuring that all displayed information remains synchronized throughout the simulation.

CHAPTER 5

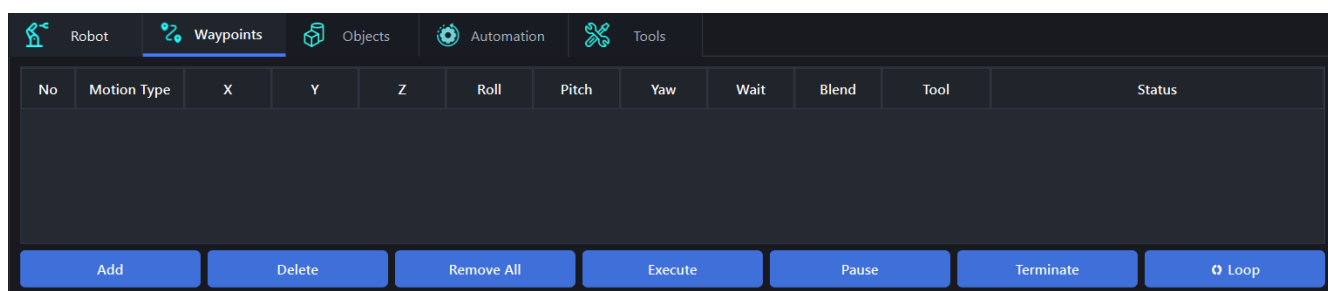
WAYPOINT PROGRAMMING

5.1 WAYPOINT OVERVIEW

A waypoint defines a target in a programmed sequence. Its motion type determines how the robot travels there. Recorded Roll/Pitch/Yaw values and explicit orientation constraints are different; see **Chapter 6 - TCP Orientation Control**.

The table is empty after creating a robot. Use the FK sliders to teach a pose, then click **Add**. An FK-created waypoint stores XYZ, TCP orientation and the joint configuration.

Figure 5.1 - Empty waypoint table before teaching a sequence

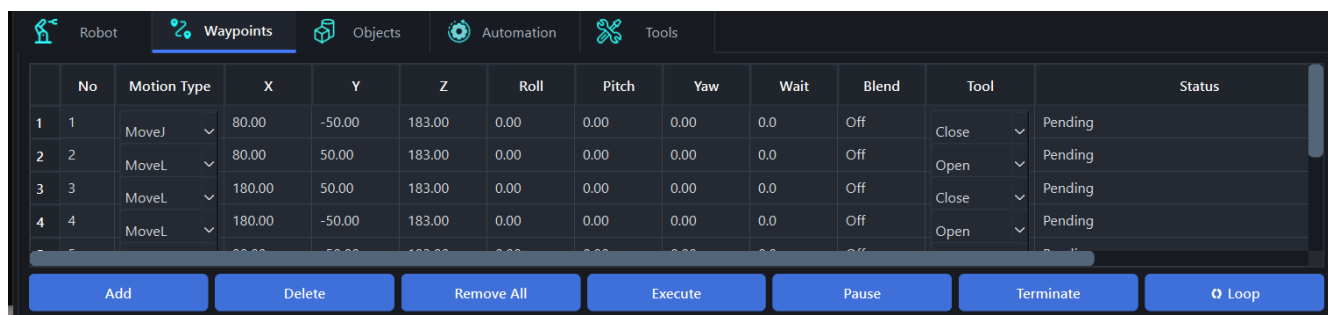


No	Motion Type	X	Y	Z	Roll	Pitch	Yaw	Wait	Blend	Tool	Status
----	-------------	---	---	---	------	-------	-----	------	-------	------	--------

Buttons: Add, Delete, Remove All, Execute, Pause, Terminate, Loop

Each Add appends a row. The example below shows a populated sequence with different motion types and waypoint tool commands.

Figure 5.2 - Waypoint table after adding a movement sequence



No	Motion Type	X	Y	Z	Roll	Pitch	Yaw	Wait	Blend	Tool	Status
1	MoveJ	80.00	-50.00	183.00	0.00	0.00	0.00	0.0	Off	Close	Pending
2	MoveL	80.00	50.00	183.00	0.00	0.00	0.00	0.0	Off	Open	Pending
3	MoveL	180.00	50.00	183.00	0.00	0.00	0.00	0.0	Off	Close	Pending
4	MoveL	180.00	-50.00	183.00	0.00	0.00	0.00	0.0	Off	Open	Pending
5	MoveL	80.00	50.00	183.00	0.00	0.00	0.00	0.0	Off	Open	Pending

Buttons: Add, Delete, Remove All, Execute, Pause, Terminate, Loop

A taught **MoveJ** without explicit orientation constraints restores its saved joint configuration. Position-only **MoveL** / **MoveC** do not promise that taught orientation. Object/XYZ-created waypoints may use position IK instead of a taught configuration.

5.2 WAYPOINT TABLE

Each row is a waypoint. Review its motion, coordinates, wait time and tool command before execution.

COLUMN	DESCRIPTION
No.	Sequence number. The program normally proceeds in table order.
Motion Type	MoveJ: joint-space motion. MoveL: straight Cartesian path. MoveC: circular arc; its following row supplies the consumed endpoint.
X / Y / Z	Target TCP position in millimetres.
Roll / Pitch / Yaw	Recorded workspace orientation in degrees, or the requested values when explicit constraints are configured. Only checked orientation angles are enforced. Edit through O , not by typing into these cells.
Wait	Pause time at the waypoint, in seconds. Zero continues without an added wait.
Blend	Reserved for a future release; blending is currently unavailable.
Tool	A supported tool action, such as gripper Open / Close or suction / magnet ON / OFF.
Status	Execution state plus orientation/path feedback where applicable. For example: Orientation set, checking path, MoveL ready, or MoveL not validated.

Teal rows indicate explicit orientation settings. They do not mean the project has automatically been saved to disk. Press **Ctrl+S** to save. Spatial waypoint-marker colours are a separate indication of motion type.

5.3 WAYPOINT CONTROLS

CONTROL	ACTION
Add	Teach the current FK pose and append a waypoint.
Delete	Remove the selected waypoint without deleting the remaining rows.
Remove All	Remove all waypoints from the current project.
Execute	Start the sequence from its first waypoint. Motion/path validation can reject a segment before playback.
Pause / Resume	Suspend or continue execution. Oriented Cartesian planning also honours the paused state.
Terminate	Stop the current program/planning request. Review the robot state before starting again.
Loop	Repeat the sequence until looping is disabled or execution is terminated.
O key	With one row selected and the table focused, open its orientation settings if the robot and motion type support editing.

TEACHING A SIMPLE SEQUENCE

1. Set the robot geometry, limits and home angles.
2. Use FK to choose a valid pose, then click Add.
3. Repeat for additional waypoints and select their motion types.
4. Configure supported orientation targets with O if required.
5. Set tool commands and waits, save the project and inspect the sequence before Execute.

Guide only: **Help > User Guide** explains these controls in short focus pages. Desktop playback controls are not a physical-controller emergency-stop system.

5.4 WAYPOINT VISUALIZATION

TRAJECTORY PREVIEW

The preview displays the planned TCP path so that you can inspect a segment before playback. A path preview is separate from the orientation editor's endpoint shadow.

TRAJECTORY TRAIL

The trail records the displayed TCP movement during execution. Compare it with the planned path when reviewing a sequence.

TOOL VISUALIZATION

The final sphere colour responds to the selected tool and its state. Tool commands assigned to rows update during execution.

WAYPOINT MARKERS

Spatial markers identify the motion type, using the colours below. The marker colour is separate from the teal table-row indication for explicit orientation settings.

MOTION TYPE	WORKSPACE MARKER
MoveJ	Green
MoveL	Blue
MoveC	Orange

A cyan/red robot shadow previews a candidate endpoint configuration. An amber MoveL failure marker identifies an approximate attempted path location, not a point the robot has travelled to. Chapter 6 explains both.

CHAPTER 6

TCP ORIENTATION CONTROL

6.1 POSITION, ORIENTATION AND THE TCP

Position is the TCP's X/Y/Z location. **Orientation** describes its rotation using workspace Roll, Pitch and Yaw in degrees. In OrionKine Lab, the TCP is the **centre of the final joint sphere**, at the displayed TCP coordinate frame; this feature adds no tool offset.

An FK-taught waypoint stores the robot's joint configuration as well as its TCP pose. An explicit orientation target is a separate request: selected angles must be satisfied while unselected angles remain free.

6.2 SUPPORTED ROBOTS AND MOTION TYPES

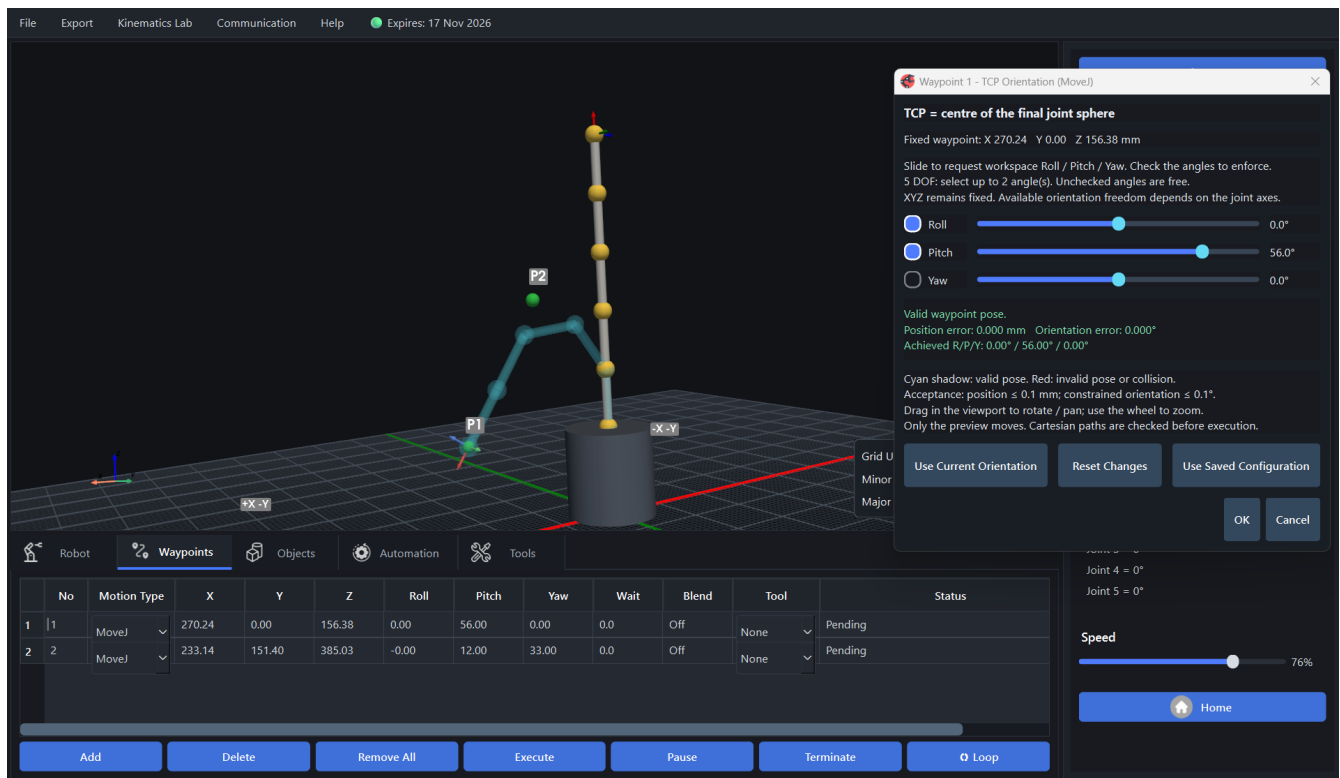
ROBOT	ORIENTATION SETTINGS	SUPPORTED MOTION
3 DOF	No orientation editor. O shows an explanation. A taught MoveJ can restore its saved configuration.	MoveJ: saved configuration. MoveL / MoveC: position-only.
4 DOF	Select one of Roll, Pitch or Yaw. Selecting another deselects the old angle.	Explicit orientation: MoveJ only .
5 DOF	Select up to two angles. One is also allowed. Deselect one before selecting a third.	Explicit orientation: MoveJ and MoveL .
6 DOF	Select one, two or all three orientation angles.	Explicit orientation: MoveJ and MoveL .

DOF is not a reachability guarantee. Joint axes, link lengths, limits, starting pose and requested constraints determine what can be achieved. MoveC and its consumed endpoint are position-only for every model.

6.3 OPENING THE ORIENTATION EDITOR

1. Teach/add a waypoint and choose its motion type.
2. Stop motion and finish pending operations.
3. Select **one row** in the waypoint table.
4. With the table focused, press the letter **O**.
5. Review the fixed XYZ and available orientation controls.

Figure 6.1 - Selecting a row and pressing O opens the editor and shadow



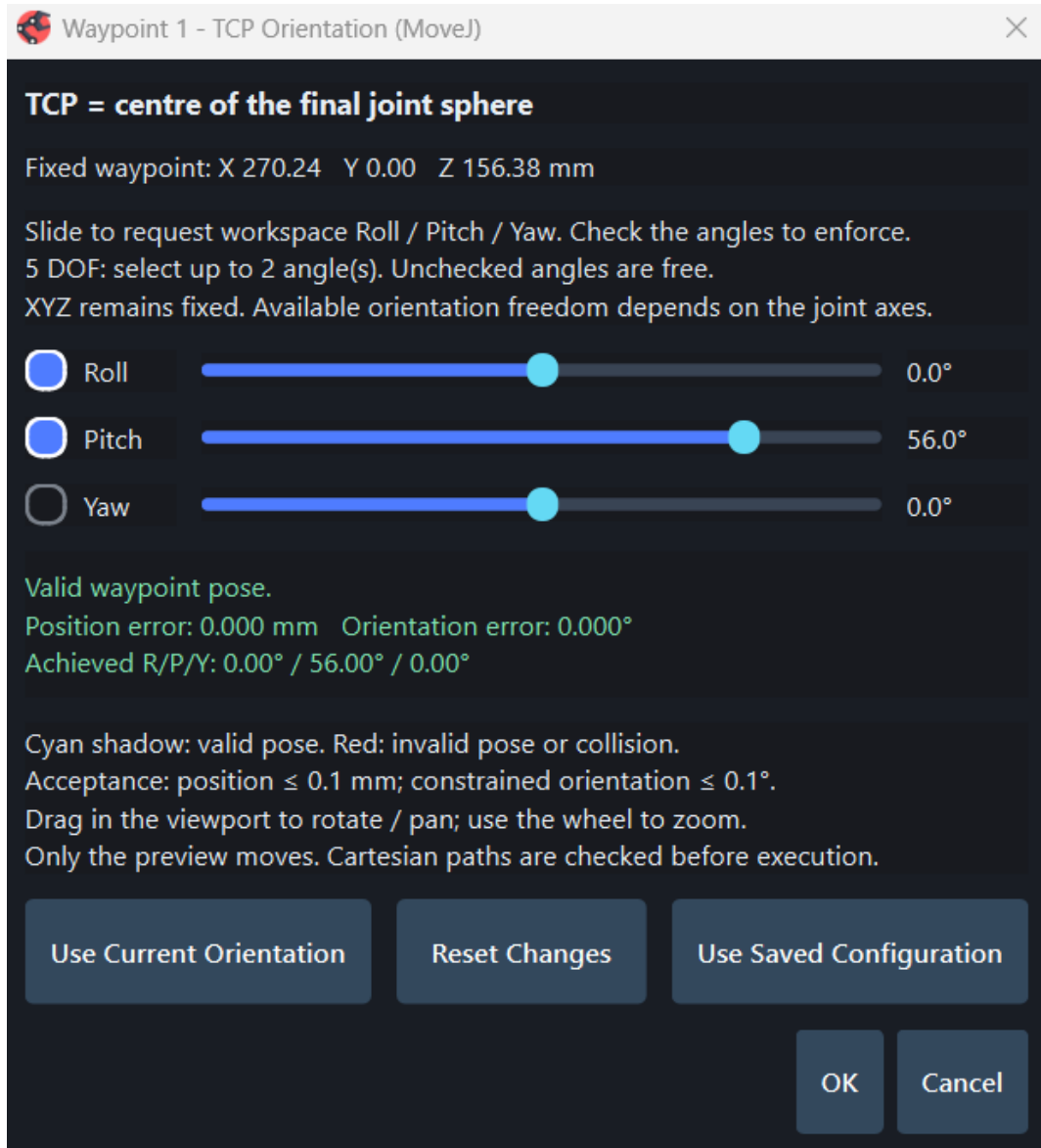
The example uses a **5-DOF MoveJ** waypoint. The dialog identifies the waypoint number and motion type. XYZ stays fixed while the solver searches for a configuration satisfying the selected orientation angles.

There is no button for opening this editor. On unsupported rows, O shows an explanation instead. For MoveC, this applies to both its via row and the following consumed endpoint.

6.4 SETTING AND ACCEPTING ORIENTATION

Check the angles to enforce, then drag their sliders. Unchecked angles are free and their sliders are disabled. Keep at least one angle checked; use the removal action to clear explicit constraints.

Figure 6.2 - Orientation sliders, achieved pose, errors and editor actions



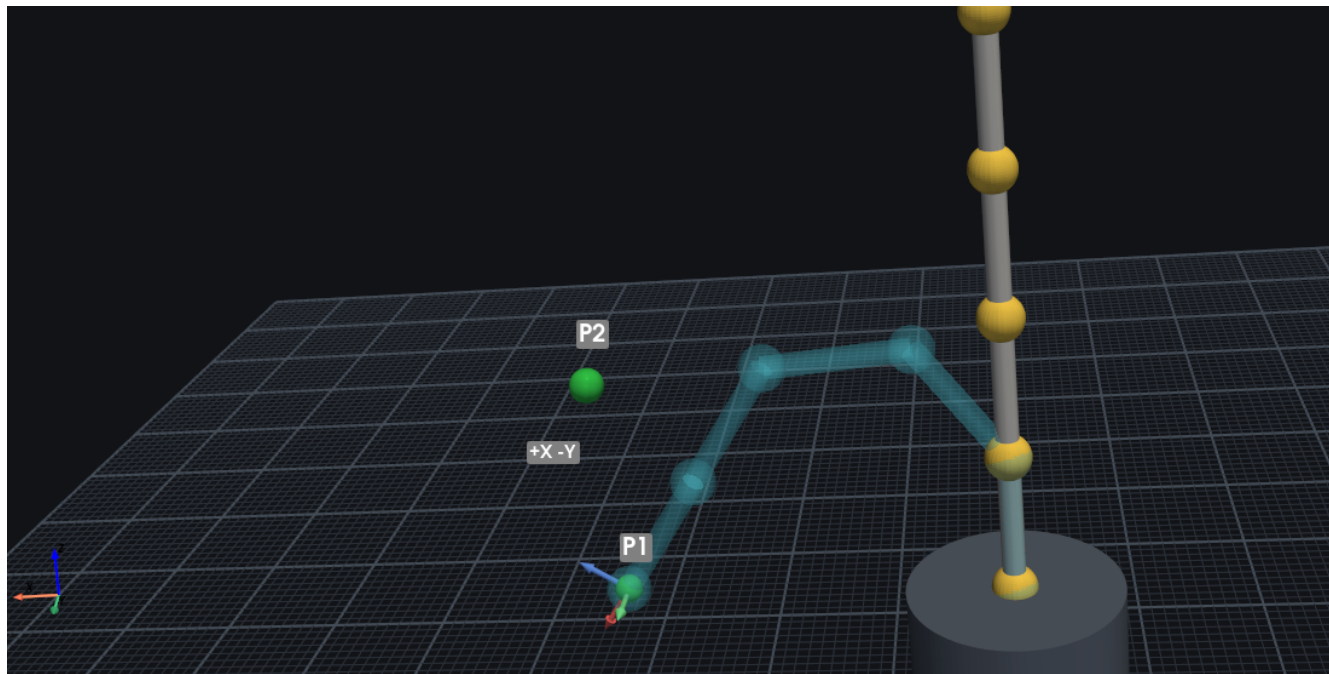
Use Current Orientation copies the current robot orientation into the requested values. **Reset Changes** restores this session's initial values. **OK** accepts a valid endpoint; **Cancel** discards the session.

Read the **achieved R/P/Y** and errors. Simulator acceptance is position ≤ 0.1 mm and selected orientation $\leq 0.1^\circ$. This does not establish physical hardware accuracy. Partial RPY is ambiguous near $\pm 90^\circ$ pitch; choose another pose, or full orientation on a compatible 6-DOF model.

6.5 INSPECTING THE ROBOT SHADOW

The translucent shadow shows the candidate robot configuration at the selected waypoint. Slider changes update this preview without moving the live robot or sending robot commands.

Figure 6.3 - Cyan endpoint shadow for visual inspection of the tool approach



PREVIEW / CONTROL	MEANING
Cyan shadow	The candidate endpoint meets the pose checks and current preview collision check.
Red shadow	The candidate pose is invalid or a checked collision is present. Read the editor feedback; colour alone does not identify the cause.
Camera navigation	Rotate, pan and zoom the viewport while the editor remains open. Inspect the links and TCP frame from multiple angles.

Endpoint validity is not path validity. A cyan shadow checks one pose. MoveL must also achieve intermediate poses while following a straight Cartesian path and the selected orientation transition.

6.6 TABLE INDICATIONS, SAVING AND REOPENING

Figure 6.4 - Teal rows show configured orientation settings

Waypoints												
	No	Motion Type	X	Y	Z	Roll	Pitch	Yaw	Wait	Blend	Tool	Status
1	1	MoveJ	270.24	0.00	156.38	0.00	56.00	0.00	0.0	Off	None	Orientation set
2	2	MoveJ	233.14	151.40	385.03	0.00	12.00	33.00	0.0	Off	None	Orientation set

After OK, the waypoint contains the accepted orientation settings for the current session. Its row is teal and Status includes **Orientation set**. Selecting a cell can temporarily cover its teal background.

OK is not a project disk save. Press Ctrl+S or use File > Save / Save As. Reopening a saved project restores targets, selected angle checkboxes and taught configurations. MoveL readiness is checked again for the restored start/settings.

6.7 EDITING OR REMOVING EXISTING SETTINGS

Edit: select the configured row, press O, adjust the supported angles, inspect the shadow and click OK. Save the project again.

Remove: choose **Use Saved Configuration** when available. This removes explicit constraints so a later MoveJ uses the original taught joint configuration. It does not move the robot while editing.

Without a taught configuration, the action is labelled **Position Only**. Removing explicit orientation clears the teal highlight; it does not delete the waypoint.

Geometry or joint-limit changes can invalidate a taught configuration. Re-teach the waypoint from FK when requested. A stored FK configuration without explicit orientation constraints does not receive the teal-row highlight.

6.8 MOVEJ VERSUS MOVEL ORIENTATION

MOTION / SETTING	WHAT TO EXPECT
MoveJ + taught configuration	Without explicit constraints, restore the saved joint angles and their TCP pose. The TCP travel path need not be straight.
MoveJ + explicit orientation	Solve XYZ and the selected orientation constraints, then move in joint space. Orientation during travel follows the joint motion.
MoveL + explicit orientation	For 5/6 DOF, follow a straight XYZ path while selected orientation constraints transition from the actual starting orientation to the requested target.
Position-only MoveL / MoveC	Control the geometric path without promising the taught orientation or final taught joint configuration.

Figure 6.5 - Execution and orientation feedback, including MoveL ready

No	Motion Type	X	Y	Z	Roll	Pitch	Yaw	Wait	Blend	Tool	Status
1	MoveJ	270.24	0.00	156.38	0.00	56.00	0.00	0.0	Off	None	Done · Orientation set
2	MoveL	233.14	151.40	385.03	-84.30	-64.40	33.00	0.0	Off	None	Done · Orientation set · MoveL ready

Holding orientation: request the segment's starting orientation as its target. The whole line must still be achievable. Full orientation uses shortest-rotation interpolation; partial control interpolates the selected wrapped RPY values.

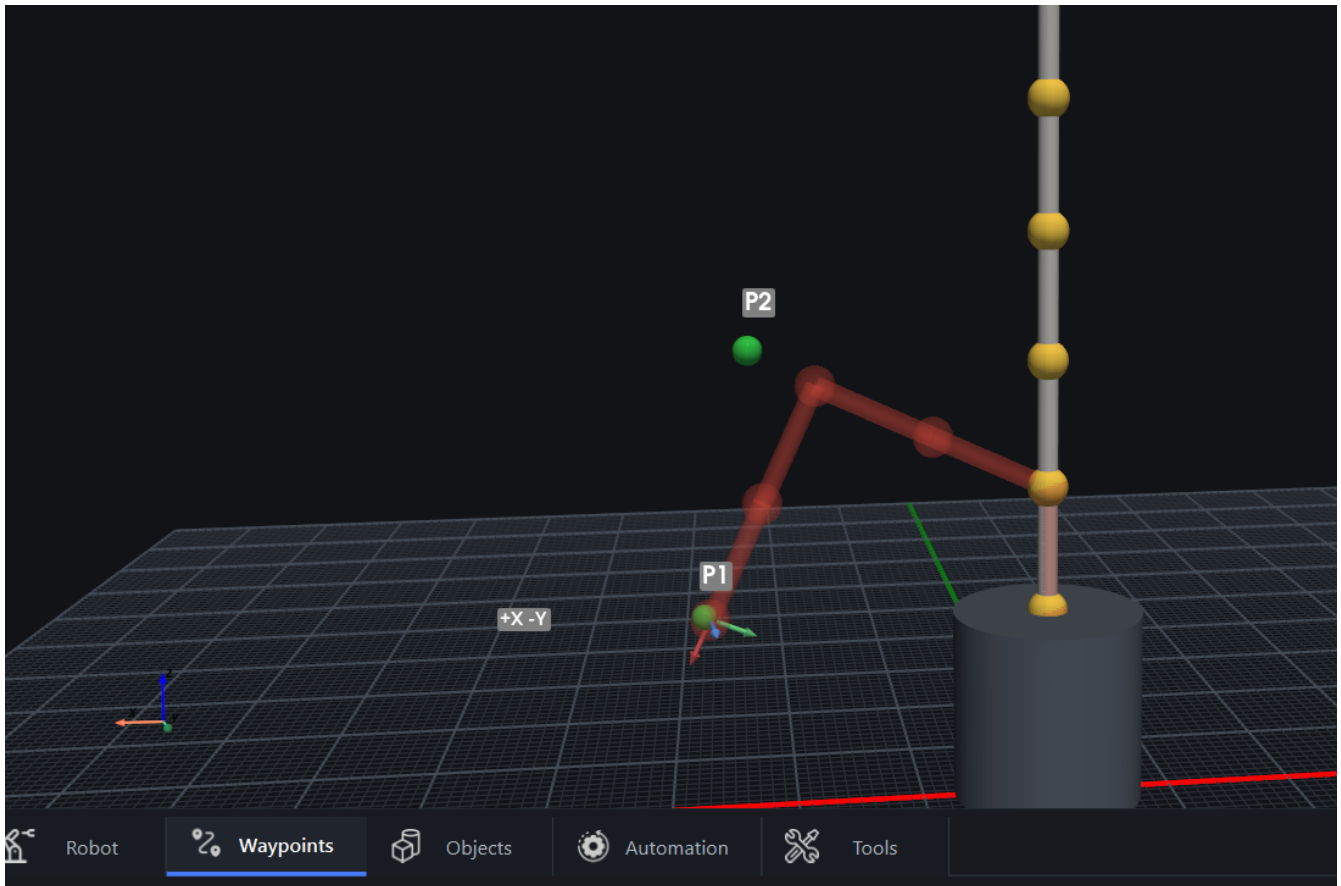
Practical gripping approach: use MoveJ to reach a nearby approach pose with the desired tool orientation, then a short validated MoveL toward the object. This can reduce the orientation change required along the final straight segment.

Choose MoveJ only when its TCP travel path is acceptable. OrionKine Lab does not silently replace a rejected MoveL with MoveJ.

6.9 INVALID PREVIEWS AND REJECTED PATHS

A position and selected orientation may be incompatible with the current geometry or joint limits. A preview can also fail a robot-versus-base/object collision check. Read the editor message and error values before accepting any pose.

Figure 6.6 - Red shadow: an invalid candidate pose or checked collision



The red preview in this screenshot indicates a failed preview check. The image alone does not establish whether the cause is pose error or collision. The editor keeps OK unavailable while its current result is invalid.

WHEN A VALID ENDPOINT STILL FAILS MOVEJ

MoveJ may reach a target that MoveL cannot validate from the current starting configuration. MoveL also requires a continuous straight TCP path, a compatible orientation transition, joint limits and acceptable interpolation accuracy. A local solver may fail to find an available solution within its budget.

A rejection is **not proof that every possible solution is impossible**. No motion starts for an unvalidated Cartesian pose path. Try fewer selected angles, another starting configuration, a smaller change or a shorter approach segment.

6.9 PATH FEEDBACK (CONTINUED)

STATUS / FEEDBACK	INTERPRETATION AND NEXT STEP
path pending / checking path	Background preparation has not finished. The UI remains responsive. Complex segments can take time.
MoveL ready	A matching TCP pose path is prepared. Current base/object collision checks still run before playback.
check at execution	The starting joint configuration is not yet known, for example after position-only IK or a circular move. Validate at execution.
MoveL not validated	Hover the row for details. Review constraints, the starting pose and the reported failure location/errors.
collision rejected	The last execution collision check rejected this path. Change the scene or motion, then execute again to recheck.

Warnings distinguish candidate pose error, joint-continuity/interpolation difficulty, and planning time or sampling limits. Where available, they report an approximate path percentage, position/orientation errors, nearby joint limits and evidence of weak local pose freedom.

An **amber sphere and label** may mark the approximate attempted TCP path location. The robot has **not travelled there**: failure occurs during planning. The marker is not an obstacle, does not alter the camera and clears when editing or relevant changes invalidate it.

Rechecking: changes to start angles, geometry, limits, XYZ, orientation constraints or speeds invalidate matching readiness. Prepared paths are cached only when settings match; collision approval is never cached. Pause/Resume and Terminate remain available during planning.

OrionKine Lab checks base and workspace objects; it does not add link-link self-collision or automatic avoidance. Validate simulation and real-controller behaviour separately before physical operation. Desktop playback is not a hardware safety interlock.

CHAPTER 7

WORKSPACE OBJECTS

7.1 WORKSPACE OBJECTS

The Workspace Objects page allows users to add and manage virtual objects within the 3D simulation environment. These objects can be used to represent workpieces, obstacles, fixtures, pallets, tables, or other components that interact with the robot during simulation.

Adding workspace objects provides a more realistic simulation environment and helps users visualize robot reachability, verify object placement, and prepare automation tasks before deploying them to physical hardware.

Refer to **Figure 7.1** for the default Workspace Objects interface.

Figure 7.1 - Workspace Objects Interface

The screenshot shows the 'Objects' tab in the software interface. The 'Properties' panel is active, displaying the following fields:

- Type:** A dropdown menu currently set to 'Cube'.
- Position (mm):** Three input fields for X, Y, and Z coordinates, all set to '0'.
- Dimensions (mm):** Three input fields for Length, Width, and Height, all set to '50'.
- Selected:** A dropdown menu currently set to 'None'.

At the bottom of the panel are two buttons: 'Create' and 'Remove'.

The Workspace Objects page allows users to define both the physical dimensions and the position of each object placed within the simulation.

OBJECT DIMENSIONS

Each workspace object can be resized by specifying its physical dimensions. All object dimensions are entered in millimetres (mm) to maintain consistency with OrionKine Lab's internal coordinate system.

The available dimension parameters are:

PARAMETER	DESCRIPTION
WIDTH (MM)	Specifies the width of the object.
LENGTH (MM)	Specifies the length of the object.
HEIGHT (MM)	Specifies the height of the object.
DIAMETER (MM)	Specifies the diameter of the cylinder object.

Adjusting these values immediately updates the size of the selected object in the 3D workspace.

OBJECT POSITION

Workspace objects can be positioned anywhere within the simulation by specifying their Cartesian coordinates

The position inputs define the location of the object relative to the workspace origin.

PARAMETER	DESCRIPTION
X	Moves the object along the X-axis of the workspace.
Y	Moves the object along the Y-axis of the workspace.
Z	Moves the object along the Z-axis of the workspace.

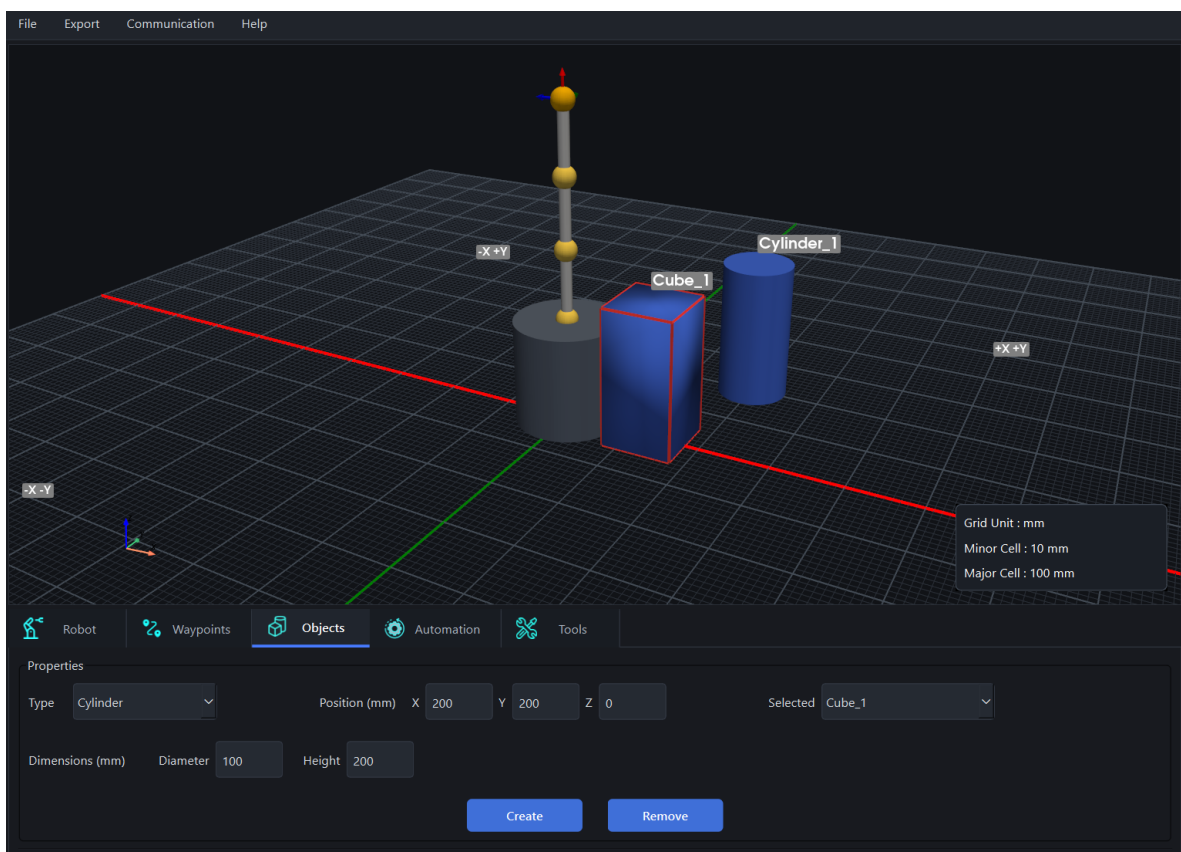
Changing any of these values immediately updates the object's position in the 3D workspace, allowing users to precisely place objects within the robot's operating area.

7.2 WORKSPACE WITH OBJECTS

After adding and positioning workspace objects, the robot and all objects are displayed together within the 3D workspace.

Refer to **Figure 7.2** for an example of a configured workspace containing the robot and multiple workspace objects.

Figure 7.2 - Robot with Workspace Objects



The combined visualization enables users to evaluate the robot's working environment before creating waypoint sequences or automation routines. By placing objects in realistic locations, users can verify robot reach, optimize object placement, and prepare simulation scenarios that closely resemble real- world applications.

 **Tip: Arrange workspace objects to match the physical layout of your robot cell whenever possible. A realistic workspace configuration improves simulation accuracy and simplifies the transition from simulation to physical robot deployment.**

CHAPTER 8

OBJECT-BASED WAYPOINT GENERATION

8.1 OBJECT-BASED WAYPOINT GENERATION

OrionKine Lab allows users to automatically generate waypoints directly from workspace objects. Instead of manually entering Cartesian coordinates, users can select an object within the 3D workspace and generate waypoints based on predefined object locations.

This feature simplifies the creation of robot programs for applications such as pick-and-place, material handling, palletizing, inspection, and repetitive automation tasks.

To begin generating waypoints, first select a workspace object within the 3D workspace.

Refer to **Figure 7.2**, where the workspace object is shown in the selected state.

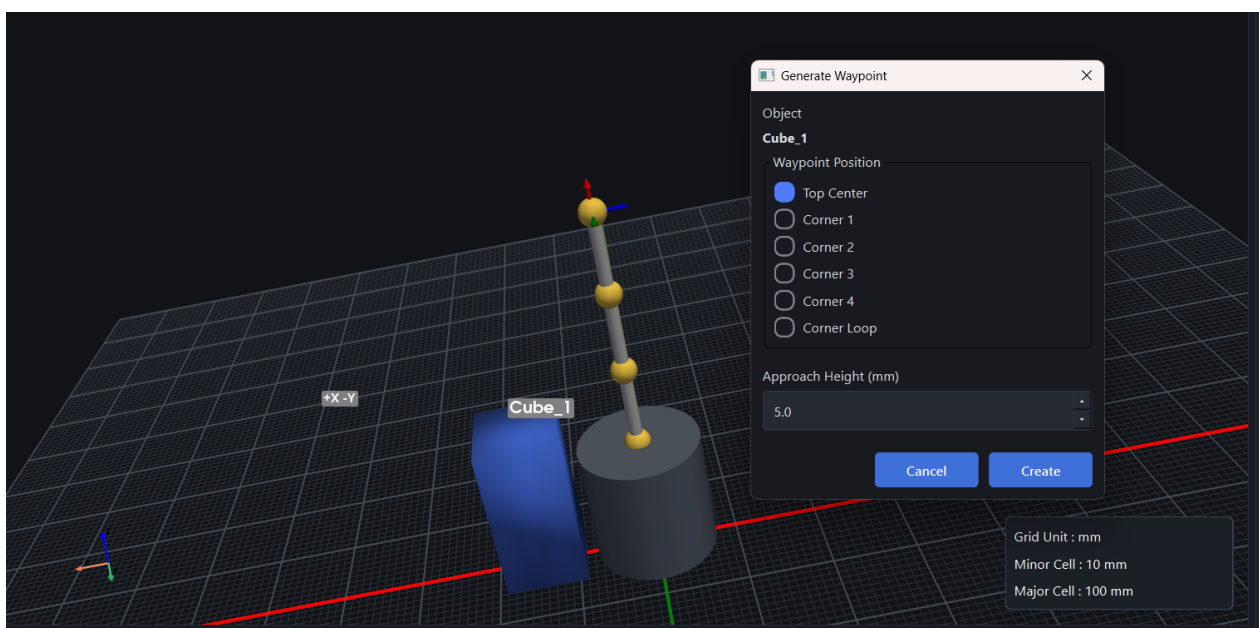
When an object is selected, OrionKine Lab highlights it in the 3D workspace, allowing users to clearly identify the active object before creating waypoints.

8.2 OPENING THE WAYPOINT GENERATION DIALOG

After selecting a workspace object, press the **W** key on the keyboard.

OrionKine Lab opens the Object Waypoint Generation dialog, which provides several predefined waypoint generation options.

Refer to Figure 8.1 - Object Waypoint Generation Dialog.



The dialog allows users to automatically generate waypoints at predefined locations on the selected object.

WAYPOINT OFFSET (APPROCH HEIGHT)

The Waypoint Offset defines the distance between the generated waypoint and the surface of the selected object.

The offset is measured in millimetres (mm).

Instead of placing the robot exactly on the object surface, OrionKine Lab positions the generated waypoint at the specified offset distance above the object.

Using an appropriate offset provides several advantages:

- Prevents the robot from colliding with the object during approach.
- Creates a safe approach position before performing a task.
- Reduces the risk of collisions while testing robot programs.
- Improves the reliability of pick-and-place and inspection operations.

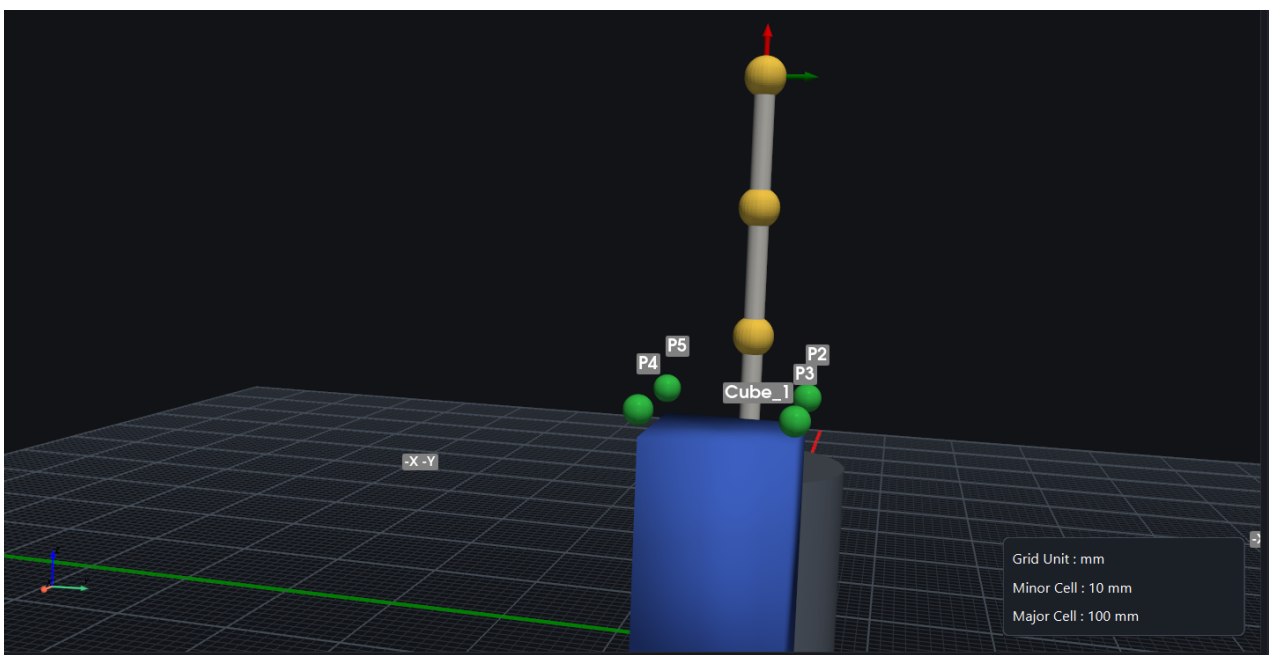
The waypoint offset is particularly useful when working with tools such as grippers, suction cups, or magnetic end effectors, where the robot must approach an object safely before interacting with it.

 **Note: The Waypoint Offset is intended to create a safe approach position above the selected object.** Collision checks and their limitations are discussed in a later chapter of this manual.

8.3 GENERATED WAYPOINTS

After selecting the required waypoint generation option, OrionKine Lab automatically creates the corresponding waypoint sequence.

Refer to Figure 8.2 - Generated Object Waypoints.



The generated waypoints are displayed in both the 3D workspace and the Waypoint table.

Each generated waypoint is automatically assigned its position based on the selected object and becomes part of the current waypoint sequence.

Once generated, the waypoints behave exactly like manually created waypoints and can be edited at any time.

Users can modify properties including:

- Motion Type (**MoveJ**, **MoveL**, or **MoveC**)
- Tool action
- Wait time
- Waypoint order
- Additional waypoint properties available within the Waypoint table

This allows automatically generated waypoints to be fully customized for the desired robot application.

WORKFLOW SUMMARY

The complete object-based waypoint generation process consists of the following steps:

- Add a workspace object.
- Select the object in the 3D workspace.
- Press the **W** key to open the Object Waypoint Generation dialog.
- Choose the desired waypoint generation option.
- Specify the required waypoint offset.
- Generate the waypoints.
- Review the generated waypoints in the 3D workspace.
- Modify the waypoint properties from the Waypoint table if required.
- Execute the waypoint sequence.

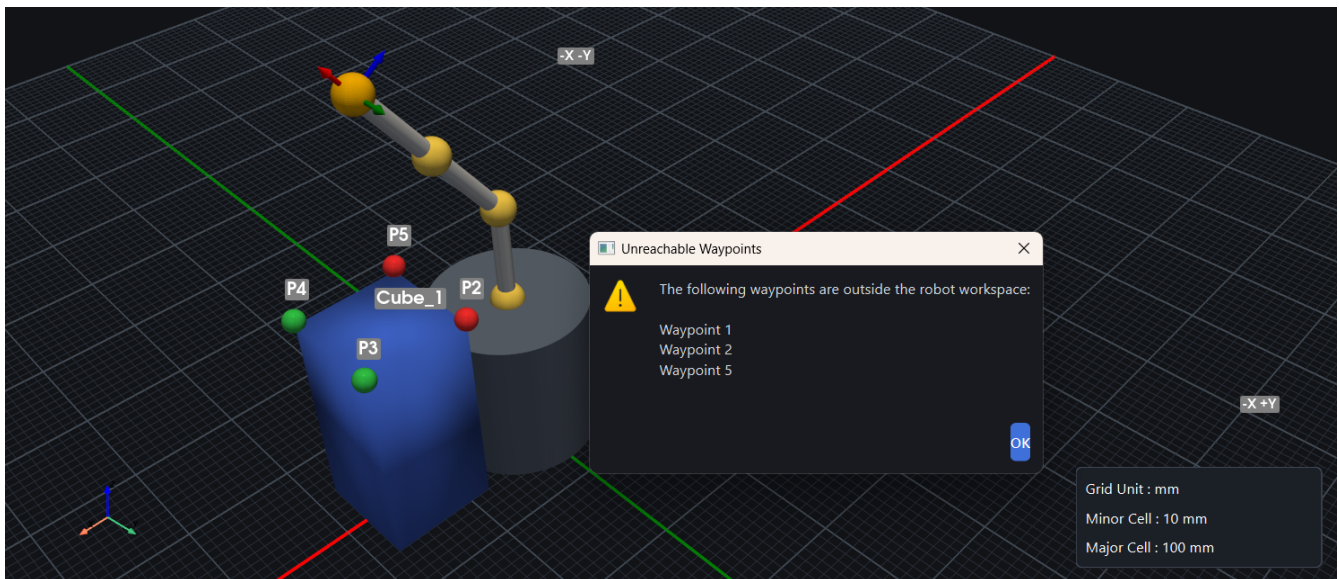
 **Tip: Object-based waypoint generation significantly reduces programming time by automatically creating accurate waypoint positions relative to workspace objects. After generation, the waypoints can be freely edited, allowing users to combine automated waypoint creation with manual robot programming for maximum flexibility.**

8.4 UNREACHABLE OBJECT WAYPOINTS

After generating object-based waypoints, OrionKine Lab automatically verifies whether each waypoint is reachable by the current robot configuration.

If one or more generated waypoints are outside the robot's reachable workspace, the corresponding waypoint markers are displayed in red within the 3D workspace.

Refer to **Figure 8.3 - Unreachable Object Waypoints**.



The red waypoint markers indicate that the robot cannot reach those target positions using its current configuration. These waypoints cannot be executed successfully until they are regenerated or the robot configuration is modified.

This situation commonly occurs when:

- The workspace object is positioned outside the robot's working envelope.
- The robot is too far from the selected object.
- The current robot pose is unsuitable for generating reachable waypoints.

REGENERATING REACHABLE WAYPOINTS

If generated waypoints appear in red, OrionKine Lab recommends the following workflow:

- Remove the generated waypoints from the Waypoint table.
- Use the Forward Kinematics sliders to move the robot closer to the workspace object.
- Select the object again.
- Press the W key to open the Object Waypoint Generation dialog.
- Generate the waypoints again.

By changing the robot's current position before waypoint generation, OrionKine Lab recalculates the waypoint locations relative to the robot, increasing the likelihood that all generated waypoints are reachable.

 **Note:** Red waypoint markers indicate that the generated waypoint is currently outside the robot's reachable workspace. They do not indicate a collision or system error. Regenerating the waypoints after repositioning the robot or moving the workspace object usually resolves the issue.

CHAPTER 9

TOOLS

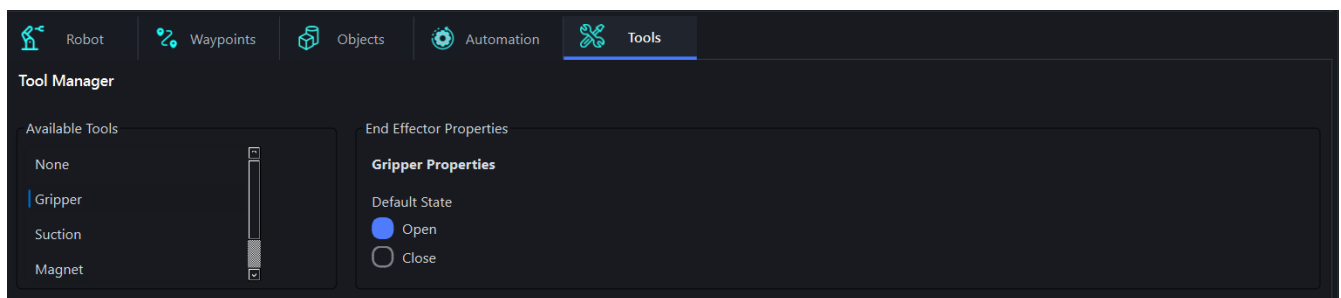
9.1 TOOL CONFIGURATION

The Tools page allows users to configure the end-effector attached to the robot. An end-effector is the device mounted at the robot's wrist that performs the required task, such as gripping, picking, placing, or manipulating objects.

OrionKine Lab allows users to assign a tool to the robot before creating waypoint sequences. The selected tool is used throughout the simulation and can be controlled during waypoint execution.

Refer to **Figure 9.1** for the Tools page.

Figure 9.1 - Tools Configuration



The Tools page contains two configuration options:

- Tool Selection
- Default Tool State

TOOL SELECTION

The Tool Selection option specifies the type of end-effector attached to the robot.

Depending on the selected robot configuration, OrionKine Lab supports the following tool types:

- None
- Gripper
- Suction Cup
- Magnet

Selecting a tool updates the robot configuration and enables tool control during waypoint execution.

If **None** is selected, the robot operates without an end-effector.

DEFAULT TOOL STATE

The Default Tool State defines the initial operating state of the selected tool whenever the robot is created or a simulation begins.

The available default states depend on the selected tool.

Examples include:

TOOL	AVAILABLE STATES
NONE	None
GRIPPER	Open, Close
SUCTION CUP	ON, OFF
MAGNET	ON, OFF

The selected default state is applied automatically before any waypoint sequence is executed.

TOOL VISUALIZATION

OrionKine Lab provides real-time visual feedback for the selected end-effector.

Whenever a tool or its current state changes, the color of the robot's end-effector sphere is automatically updated within the 3D workspace. This visual indication allows users to quickly identify the active tool and verify its operating state without referring to additional controls.

The end-effector visualization remains synchronized throughout the simulation, ensuring that the displayed tool state always matches the robot's current operating condition.

TOOL CONTROL DURING WAYPOINT EXECUTION

Although the default tool state is configured from the Tools page, tool operations can also be programmed as part of a waypoint sequence.

As discussed in **Chapter 5 - Waypoint Programming**, each waypoint contains a **Tool** column that allows users to specify the desired tool action for that particular waypoint.

For example, a robot performing a pick-and-place operation may:

- Approach the object with the gripper open.
- Close the gripper after reaching the pickup location.
- Move the object to the destination.
- Open the gripper to release the object.

By assigning different tool states to individual waypoints, OrionKine Lab enables realistic simulation of automated robotic tasks without requiring manual tool control during execution.



Note: The Default Tool State defines the tool's initial condition when the simulation begins. During waypoint execution, the tool state can be changed at any waypoint using the Tool column in the Waypoint table, allowing the robot to perform task-specific operations automatically.

CHAPTER 10

AUTOMATION

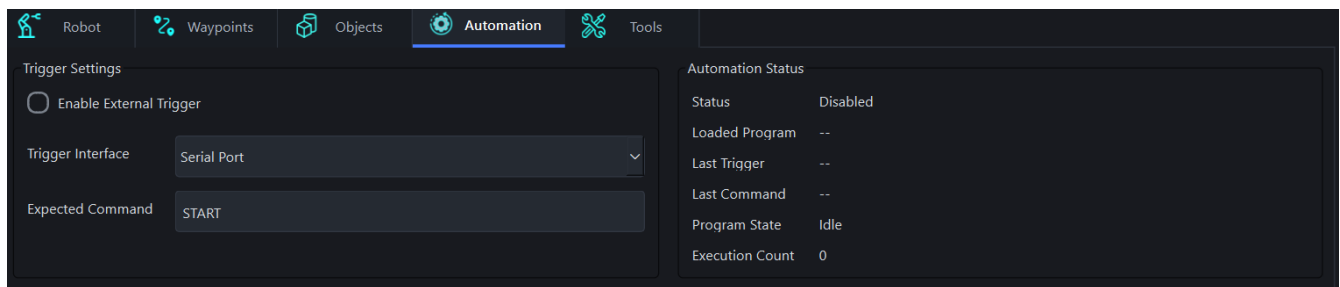
10.1 AUTOMATION OVERVIEW

The Automation page allows OrionKine Lab to automatically execute a programmed waypoint sequence when a predefined command is received from an external device through the serial communication interface.

This feature enables the robot simulation to interact with external controllers, sensors, PLCs, microcontrollers, or other automation systems, making OrionKine Lab suitable for simulating real-world industrial automation applications.

Refer to **Figure 10.1** for the Automation page.

Figure 10.1 - Automation Configuration



The Automation page contains two configuration options:

Enable External Trigger

Expected Trigger Command

ENABLE EXTERNAL TRIGGER

The Enable External Trigger checkbox determines whether OrionKine Lab should accept automation trigger commands from external devices.

When the checkbox is enabled, OrionKine Lab continuously monitors incoming serial data for the configured trigger command.

When the checkbox is disabled, all incoming trigger commands are ignored and waypoint execution must be started manually using the Execute button on the Waypoint page.

This provides users with complete control over whether external devices are allowed to start robot programs.

EXPECTED TRIGGER COMMAND

The Expected Trigger Command specifies the text command that OrionKine Lab waits for before automatically executing the programmed waypoint sequence.

Users are free to define any command that matches their external controller or application.

Examples include:

- START
- GO
- TRUE
- RUN
- PICK
- BEGIN

When the received serial data exactly matches the configured command, OrionKine Lab immediately begins executing the current waypoint sequence.

This flexibility allows OrionKine Lab to integrate with virtually any external device capable of transmitting serial data.

EXTERNAL AUTOMATION WORKFLOW

The automation process consists of the following steps:

- Create the robot and configure the waypoint sequence.
- Enable External Trigger.
- Enter the expected trigger command.
- Connect the external controller through the Serial Communication interface.
- Wait for the configured command to be received.
- When the received command matches the configured trigger command, OrionKine Lab automatically starts executing the programmed waypoint sequence.

EXAMPLE ARDUINO TRIGGER

The following Arduino example demonstrates a simple external trigger using an infrared (IR) sensor.

When the IR sensor detects an object, the sensor output becomes HIGH, and the Arduino sends the command START through the serial port.

```
const int irSensorPin = 2;

void setup()
{
    pinMode(irSensorPin, INPUT);
    Serial.begin(115200);
}

void loop()
{
    if (digitalRead(irSensorPin) == HIGH)
    {
        Serial.println("START");
        delay(500);
    }
}
```

If the **Expected Trigger Command** in OrionKine Lab is configured as **START**, receiving this command automatically begins executing the programmed waypoint sequence.

PRACTICAL APPLICATIONS

The Automation feature can be used in a wide variety of robotics and industrial automation applications, including:

- Conveyor belt object detection
- Infrared sensor triggered pick-and-place systems
- Photoelectric sensor based material handling
- PLC controlled robot automation
- Production line synchronization
- Machine tending
- External push-button operation
- Smart sensor integration

For example, a conveyor belt equipped with a proximity sensor can detect an incoming object. When the object reaches the pickup position, the controller sends the configured trigger command through the serial interface. OrionKine Lab receives the command and automatically executes the programmed robot motion, allowing the robot to pick the object without manual intervention.



Note: OrionKine Lab only executes an external automation command when Enable External Trigger is enabled and the received serial command exactly matches the configured Expected Trigger Command. Any other received commands are ignored to prevent unintended robot motion.

CHAPTER 11

APPLICATION MENU

11.1 APPLICATION MENU OVERVIEW

The Application Menu provides quick access to project management, Kinematics Lab, communication settings, software information, and future export features. The menu bar is located at the top of the OrionKine Lab window and contains File, Export, Kinematics Lab, Communication, and Help. The license expiration date is also displayed in the menu bar.

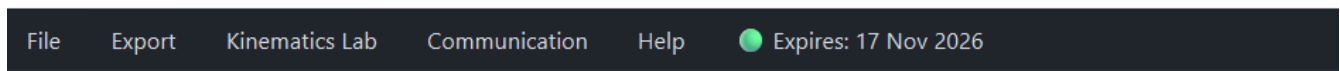


Figure 11.1 - OrionKine Lab Application Menu

11.2 FILE MENU

The File menu contains options for creating and managing OrionKine Lab projects. Figures 11.2 and 11.3 show the supplied File menu views.

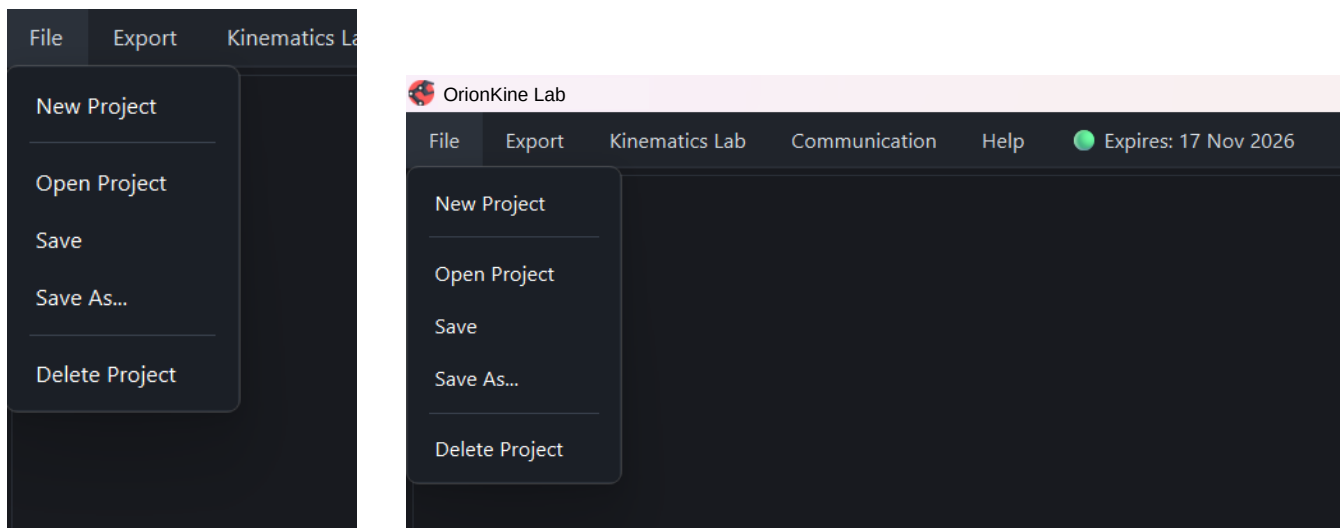


Figure 11.2 - File Menu (left). Figure 11.3 - File Menu within the application window (right).

Use New Project to begin a robot project, Open Project to load an existing project, and Save or Save As... to store your work. Delete Project permanently deletes the selected project. The options are described on the following page.

Project files: New saves use **.orionkine**. Existing **.orion** projects can still be opened; saving one offers Save As to create a new **.orionkine** copy while preserving the original. License files (**.orion**) and activation requests (**.orionreq**) keep their existing formats.

11.2 FILE MENU (CONTINUED)

OPTION	DESCRIPTION
NEW PROJECT	Creates a new OrionKine Lab project and returns to the Robot Creator page.
OPEN PROJECT	Opens an existing OrionKine Lab project.
SAVE	Saves the currently opened project.
SAVE AS...	Saves the current project using a different name or location.
DELETE PROJECT	Permanently deletes the selected OrionKine Lab project.

11.3 EXPORT MENU

The Export menu is reserved for future OrionKine Lab releases. Future versions will provide additional export options for robot programs and project data.

Note: The Export functionality is reserved for future improvements and is not available in Version 1.0.

11.4 KINEMATICS LAB MENU

Create or load a robot, then select **Kinematics Lab > Open Kinematics Lab**. The Lab opens in a separate window for interactive robot kinematics learning and analysis. It uses the current robot to explain geometry, transformations, forward kinematics, inverse kinematics, and motion trajectories. See Chapter 12 for the complete learning guide.

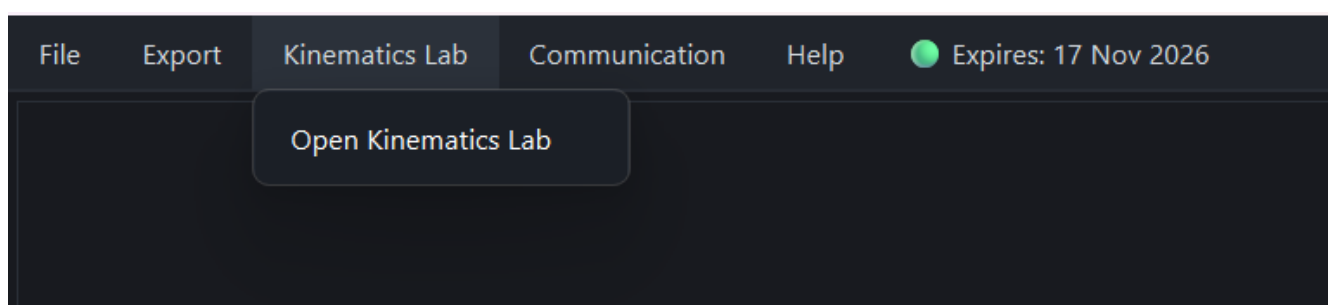


Figure 11.4 - Kinematics Lab Menu

The Lab is a read-only learning environment. Its educational targets and path previews do not send motor commands or change the robot's configured joint angles.

11.5 COMMUNICATION MENU

The Communication menu provides all serial communication settings required to connect OrionKine Lab with external hardware such as Arduino, ESP32, or other compatible controllers.

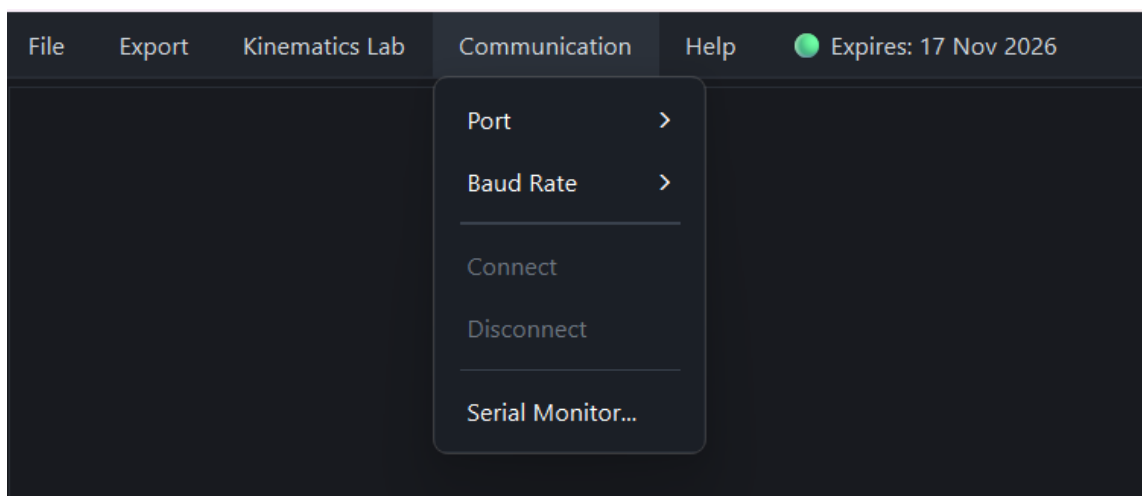


Figure 11.5 - Communication Menu

OPTION	DESCRIPTION
PORT	Selects the serial communication port connected to the external device.
BAUD RATE	Selects the communication speed for the serial connection.
CONNECT	Establishes a serial connection with the selected device.
DISCONNECT	Closes the active serial connection.
SERIAL MONITOR...	Opens the built-in Serial Monitor for viewing transmitted and received serial data.

11.6 LICENSE MANAGEMENT

OrionKine Lab uses a secure license validation system to ensure that only valid licenses can access the software. The current license expiration date is displayed in the application menu bar.

During operation, OrionKine Lab periodically validates the installed license. If the license expires while the application is running, a License Expired dialog is displayed. Users are given the opportunity to save their current work using Save Project As... After saving, OrionKine Lab can be closed safely. A new valid license must be activated before OrionKine Lab can be used again.

Note: OrionKine Lab is designed to help prevent accidental loss of project data by allowing users to save their work before exiting.

11.7 HELP MENU

The Help menu provides quick access to OrionKine Lab documentation, firmware references, contact information, and application details.

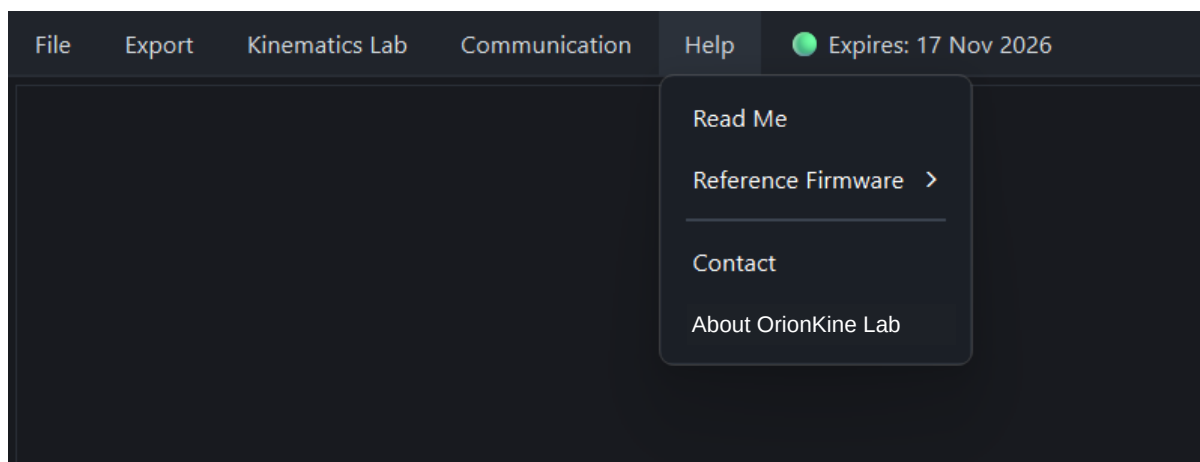


Figure 11.6 - Help Menu

MENU ITEM	DESCRIPTION
READ ME	Opens the OrionKine Lab User Manual, which explains the software interface, robot creation, motion planning, project management, communication, and other features.
REFERENCE FIRMWARE > ARDUINO SERVO REFERENCE CODE	Opens the official Arduino Servo Reference Firmware. This universal firmware supports configurable 3 to 6 DOF servo robots, configurable joint limits, servo mapping, reverse direction, optional end-effectors (Gripper, Suction, Magnet), and optional external trigger support. It serves as the recommended firmware template for OrionKine Lab generated projects.
CONTACT	Displays contact information for technical support, business inquiries, and feedback.
ABOUT ORIONKINE LAB	Shows the OrionKine Lab version, application information, copyright, and developer details.

Note: The Reference Firmware examples are intended as educational reference implementations. Users may modify the firmware as required to suit their specific robot hardware and application.

Note: The Arduino Servo Reference Code is provided as a learning and reference resource. It demonstrates the recommended firmware structure used by OrionKine Lab and can be customized to match different servo-based robotic arm configurations.

CHAPTER 12

KINEMATICS LAB

Interactive Robot Kinematics Learning & Analysis

12.1 WHAT IS KINEMATICS LAB?

Kinematics Lab is a lightweight educational Robot Analyzer inside OrionKine Lab. It teaches beginners how a robotic arm is described mathematically and how joint rotation, forward kinematics, inverse kinematics and motion planning relate to visible robot movement.

Instead of learning formulas in isolation, users explore concepts using a **3-6 DOF revolute robot** created or opened in OrionKine Lab. The Lab reads the robot's current joint axes, angles, link lengths and base height, and displays matching equations, matrices and coordinates.

12.2 OPENING THE ANALYZER

Create or load a robot, then choose **Kinematics Lab → Open Kinematics Lab** from the application menu. The Lab opens in its own window so the existing 3D workspace can remain visible. Figure 12.1 shows the two-panel layout.

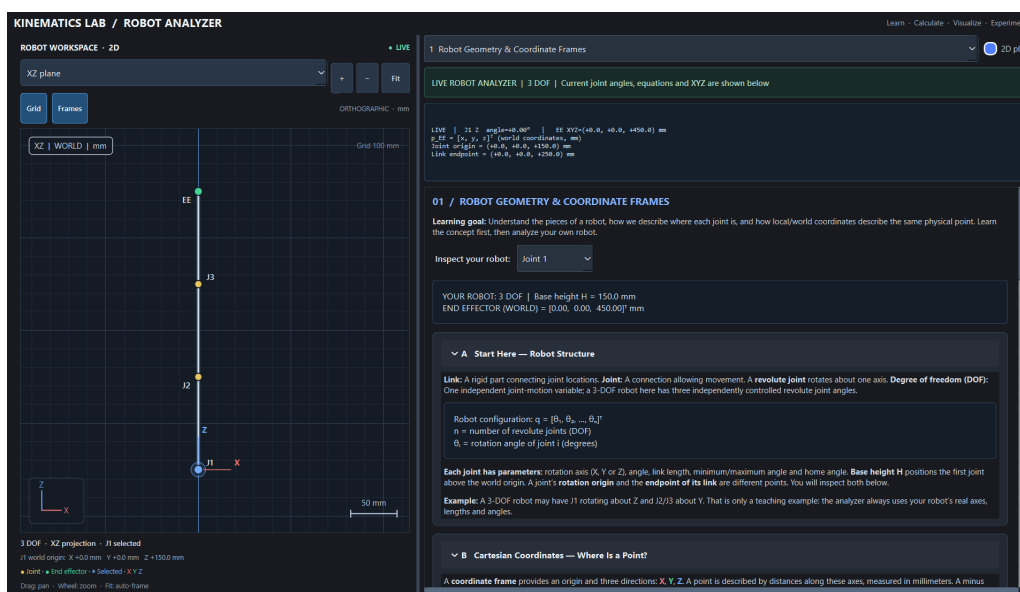


Figure 12.1 - Kinematics Lab with the synchronized 2D robot workspace on the left and live learning content on the right.

12.3 HOW THE LIVE LEARNING WORKFLOW OPERATES

Learn → Calculate → Visualize → Experiment. Select a module to understand a principle, inspect the essential equation and its symbols, and then observe the numerical result as the simulated robot changes. Advanced derivations can be expanded when needed.

Read-only learning: Kinematics Lab does not change the robot's configured joint angles or send motor commands. IK targets and motion paths inside the Lab are educational previews, not instructions to execute a real robot.

12.4 2D PLOTTER AND MODULE SELECTION

The module selector gives access to the five lessons. The **2D plot** checkbox shows or hides the live plotter; hiding it provides more space for equations. The robot is still displayed in the separate OrionKine Lab 3D workspace.

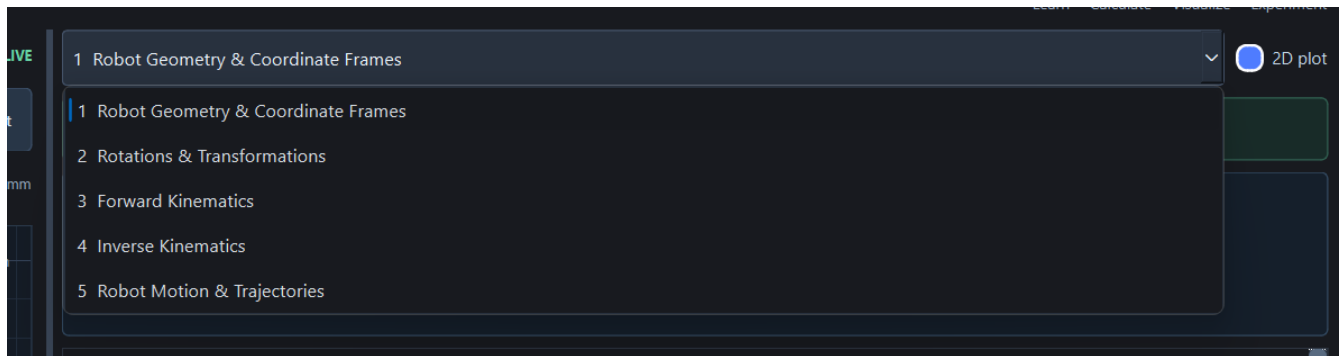


Figure 12.2 - The five-module selector and the 2D plot visibility control.

The 2D view offers **XY** (top), **XZ** (front) and **YZ** (side) projections of the actual 3D geometry. Use **Grid** for measurement references, **Frames** to inspect axes, the mouse wheel to zoom, drag to pan, and **Fit** to reframe the robot. All positions use millimeters.

12.5 MODULE 1 — ROBOT GEOMETRY & COORDINATE FRAMES

Purpose: Learn what links, joints and degrees of freedom (DOF) mean, and how we describe points relative to world and local coordinate frames. Select J1, J2 or another joint to inspect its rotation axis, angle, origin and link endpoint.

$$\mathbf{p} = [\mathbf{x}, \mathbf{y}, \mathbf{z}]^T$$

$\mathbf{p}$ = position vector; $\mathbf{x}, \mathbf{y}, \mathbf{z}$ = distances along the coordinate axes; ^{T} means transpose (written as a column vector).

Example: Figure 12.1 shows a 3-DOF robot with an end-effector position of $X = 0, Y = 0, Z = 450$ mm. Rotate a joint and observe which values change; the result depends on the robot's actual link geometry and joint axes.

12.6 MODULE 2 — ROTATIONS & TRANSFORMATIONS

Purpose: Understand how a revolute joint changes a link's orientation. The Lab shows its axis, angle, sine/cosine values, 3×3 rotation matrix and 4×4 homogeneous transformation.

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

$\mathbf{R}_z$ = rotation about Z; θ = joint angle; **sin** and **cos** describe how the coordinate axes rotate.

Example: At 30° , $\cos 30^\circ \approx 0.866$ and $\sin 30^\circ = 0.500$. The complete homogeneous transform also includes translation so a link's position and orientation can be described together.

$$\mathbf{T} = [\mathbf{R} \ \mathbf{p}; \ 0 \ 1]$$

$\mathbf{T}$ = full 4×4 transform; $\mathbf{R}$ = 3×3 rotation; $\mathbf{p}$ = translation vector; bottom row $[0, 0, 0, 1]$ makes rotation and translation work together.

Click a matrix entry in the Lab to learn how its numerical value is built from the corresponding row and column. Results follow the joint axis and pose selected in the real robot.

12.7 MODULE 3 — FORWARD KINEMATICS (FK)

What it solves: “If I know all joint angles, where is my robot’s end effector?” Forward Kinematics combines the effect of every joint rotation and link displacement to calculate a final Cartesian position and orientation.

The Lab displays the actual joint values, each local 4×4 transformation, intermediate products and final XYZ coordinates. Selecting a matrix cell reveals its row-by-column multiplication; the detailed DH convention introduction is available as supplementary learning.

$$\mathbf{T}_{W,EE} = \mathbf{T}_{WB} \mathbf{A}_1 \mathbf{A}_2 \cdots \mathbf{A}_n$$

$\mathbf{T}_{W,EE}$ = world-to-end-effector pose; $\mathbf{T}_{WB}$ = base placement; $\mathbf{A}_i$ = local transform of joint/link i ; n = total joints. The last column contains X, Y and Z (mm).

Example: A robot with three 100 mm links and a 150 mm base does not always have $Z = 450$ mm. FK accounts for the current joint angles and axis directions; a different pose can move the tool sideways and lower its height.

12.8 MODULE 4 — INVERSE KINEMATICS (IK)

What it solves: “If I want the tool at a chosen XYZ point, which joint angles could reach it?” IK works in the reverse direction from FK. Unlike FK, several valid angle combinations may exist, or the numerical solver may fail to find one.

Enter a target XYZ in the Lab and choose **Solve Preview**. A separate numerical solver searches within the robot’s joint-angle limits and reports candidate angles, achieved coordinates, final position error and solver status. The 2D plot can show the target and candidate endpoint.

$$\mathbf{e} = \|\mathbf{p}_{\text{target}} - \mathbf{p}_{\text{FK}}(\boldsymbol{\theta})\|$$

$\mathbf{e}$ = distance error (mm); $\mathbf{p}_{\text{target}}$ = requested XYZ; $\mathbf{p}_{\text{FK}}(\boldsymbol{\theta})$ = XYZ predicted from candidate angles $\boldsymbol{\theta}$; $\|\cdot\|$ = vector length.

Example: Enter $X = 200$, $Y = 100$ and $Z = 150$ mm. IK tests candidate joint angles, evaluates their FK position and tries to reduce the difference. The final result depends on reachability, joint limits and solver convergence.

12.9 WHY JACOBIANS ARE INTRODUCED

The optional Jacobian section describes how **small joint-angle changes** affect end-effector movement. A positional Jacobian has 3 rows and one column per joint (3×N). It helps learners connect angular and Cartesian motion; it does not imply that every OrionKine Lab IK calculation directly uses the same Jacobian-based method.

Important: A failed IK preview does not prove a target is geometrically unreachable. Preview solving never commands a simulated or physical robot to move.

Learning connection: FK predicts the XYZ for chosen angles; IK uses repeated FK predictions to search for angles that match a chosen XYZ. Both depend on the robot’s geometry, axes and joint limits.

12.10 MODULE 5 — ROBOT MOTION & TRAJECTORIES

Purpose: Learn why robots can reach the same destination using different movement types. The module compares **MoveJ**, **MoveL** and **MoveC** using actual robot parameters and a read-only 2D path preview. A progress slider lets you inspect intermediate positions.

MoveJ — joint-space motion. The joints transition from their starting angles to destination angles. The end-effector route is generally curved, because each joint follows an angular path rather than a prescribed straight Cartesian line.

$$\mathbf{q}(\mathbf{s}) = \mathbf{q}_0 + \mathbf{h}(\mathbf{s}) (\mathbf{q}_1 - \mathbf{q}_0)$$

$\mathbf{q}_0, \mathbf{q}_1$ = initial and final joint-angle vectors; $\mathbf{s}$ = normalized progress (0 to 1); $\mathbf{h}(\mathbf{s})$ = smooth interpolation factor. For OrionKine Lab MoveJ, $\mathbf{h}(\mathbf{s})$ is smoothstep.

MoveL — straight Cartesian path. Points between the start and goal are interpolated along a line. Each sampled XYZ position requires joint-angle calculation to reproduce the straight path.

$$\mathbf{p}(\mathbf{s}) = \mathbf{p}_0 + \mathbf{s} (\mathbf{p}_1 - \mathbf{p}_0)$$

$\mathbf{p}_0$ = start XYZ; $\mathbf{p}_1$ = goal XYZ; $\mathbf{s}$ = path progress from 0 to 1.

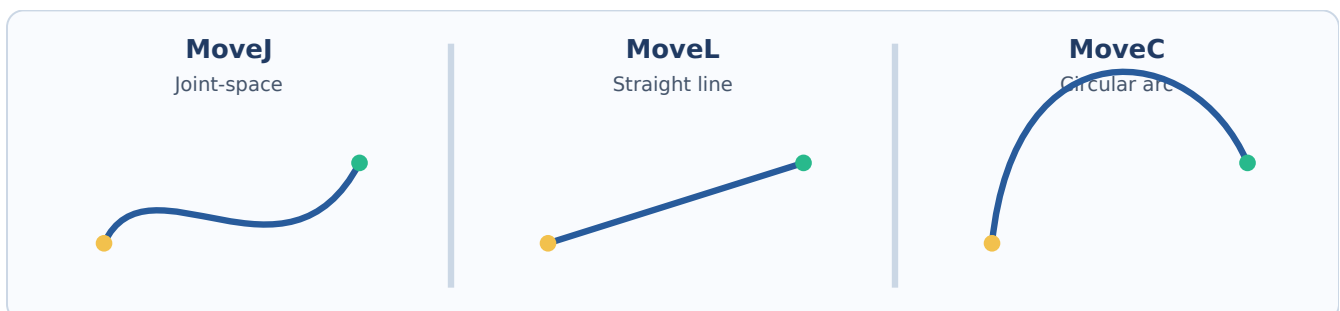
MoveC — circular motion. Start, via and goal points define the intended circular arc. The analyzer illustrates circle geometry, radius, sampled points and estimated travel distance. The selected arc should pass through its via point; the preview reports when the geometry does not satisfy this condition.

12.11 TRY A FIVE-MINUTE BEGINNER EXPERIMENT

1. Create or open a 3-DOF robot; keep the main OrionKine Lab 3D view visible.
2. In Module 1, select Joint 2 and observe its local and world coordinates.
3. In Module 2, rotate Joint 2 slowly and observe changes to the rotation matrix.
4. In Module 3, inspect how the final XYZ values change as the FK chain updates.
5. In Module 4, use the current end-effector XYZ as a target and run Solve Preview.
6. In Module 5, compare a MoveJ preview with MoveL and MoveC paths.

How to interpret a preview: The Lab teaches mathematical and geometric behavior. Previewed IK solutions and MoveL/MoveC paths are not guaranteed executable; joint limits, IK convergence, collisions and other safety checks must still be verified by the main OrionKine Lab system.

PATH SHAPES AT A GLANCE



Illustrative path shapes only; the live analyzer displays the current robot's sampled preview, not a guaranteed executable trajectory.

CHAPTER 13

COLLISION DETECTION

13.1 COLLISION DETECTION

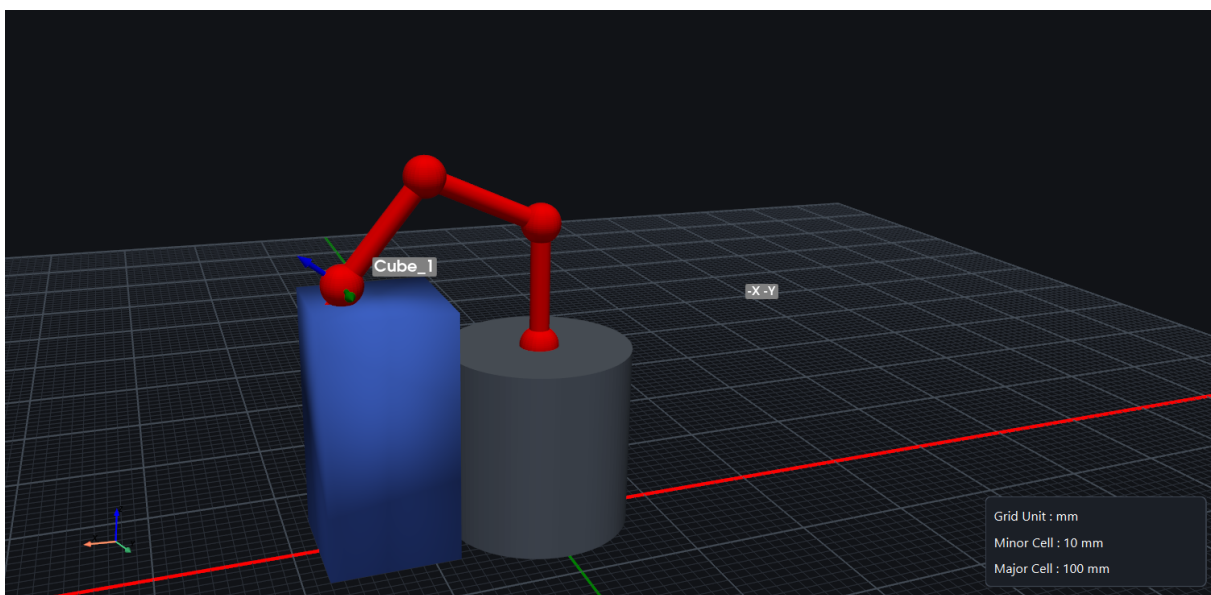
OrionKine Lab checks the robot against its base and workspace objects during FK editing, trajectory validation and waypoint playback. The current collision policy does not check links or joint spheres against one another.

The following collision checks remain active:

- Robot vs. workspace objects.
- Robot vs. base, with mounting exclusions described on the next page.

Refer to **Figure 13.1** for an example of collision feedback.

Figure 13.1 - Collision Feedback in the Robot Workspace



COLLISION RESPONSE

When a checked robot part intersects the base or a workspace object:

- The robot changes to red to indicate the detected collision.
- Active simulation playback stops and a collision notification is shown.
- Trajectory validation may reject a motion before playback begins.

During manual FK editing, a colliding pose can remain visible in red. Adjust the pose or object placement, inspect the result and restart the sequence manually.

Detection only. OrionKine Lab does not automatically steer around a collision or calculate a replacement path. Automatic avoidance and replanning are not available in this release.

13.2 SUPPORTED COLLISION TYPES

COLLISION TYPE	CURRENT BEHAVIOUR
ROBOT VS. WORKSPACE OBJECT	Enabled for robot links and joint spheres.
ROBOT VS. BASE	Enabled, with mounting exclusions below.
LINK VS. LINK	Not checked.
JOINT VS. LINK	Not checked between robot parts.
JOINT VS. JOINT	Not checked.
AUTOMATIC AVOIDANCE / REPLANNING	Not available.

13.3 SCOPE AND LIMITATIONS

Base mounting. The first rendered link and the first joint sphere are excluded from base checks to allow the mounted connection. Later links and joint spheres are checked against the base.

Joint limits. Min and Max restrict each joint angle. They do not guarantee that different robot parts cannot intersect. Link-link, joint-link and joint-joint intersections do not trigger collision warnings or stops under the current policy.

Bounding-box approximation. Base and object checks compare axis-aligned bounding boxes around the shapes. These boxes can overlap even when the visible surfaces do not; warnings can therefore be conservative, especially around rotated links.

Sampled checks. Trajectory validation checks sampled poses and playback checks the displayed poses. This is not continuous collision detection between every pair of samples.

13.4 WORKING WITH COLLISION FEEDBACK

Review a red pose before continuing. Adjust the robot configuration, waypoint coordinates or object placement, then validate and run the sequence again. OrionKine Lab will not generate an avoidance path for you.

Physical hardware: simulation feedback is not a hardware safety interlock. Verify the real robot dimensions, joint limits and motion before hardware operation. Stopping desktop playback does not guarantee an emergency stop of a physical controller.

CHAPTER 14

ORIONKINE LAB ROADMAP

As OrionKine Lab continues to evolve, new features and improvements will be introduced to enhance the user experience, improve robot simulation accuracy, and expand automation capabilities.

The following features are planned for future OrionKine Lab releases.

PLANNED FEATURES

ENHANCED 3D VISUALIZATION

Future versions of OrionKine Lab will include improved workspace visualization, enhanced robot rendering, and additional graphical improvements to provide a more realistic simulation experience.

FURTHER ORIENTATION IMPROVEMENTS

TCP orientation control is available as described in Chapter 6. Future work may extend its supported workflows and improve planning reliability and feedback.

COLLISION AVOIDANCE AND PATH PLANNING

Future releases will introduce automatic collision avoidance and intelligent path planning, allowing the robot to calculate alternative paths when obstacles or self-collisions are detected.

ROBOT CODE EXPORT

Users will be able to export complete robot programs into firmware source code for supported microcontroller platforms, simplifying the transition from simulation to physical robot deployment.

MOTION BLENDING

The current Blend parameter will be fully implemented to provide smooth transitions between consecutive waypoints, reducing unnecessary robot stops and creating more natural robot motion.

ADDITIONAL ROBOT TOOLS

Future versions will support additional end-effectors and expanded tool control options for a wider range of industrial and educational robotics applications.

IMPROVED AUTOMATION FEATURES

The Automation system will continue to expand with additional trigger options, improved workflow integration, and enhanced support for external automation devices.

PERFORMANCE IMPROVEMENTS

Continuous optimization will improve simulation performance, motion planning speed, and overall application responsiveness

 **Note: The features listed in this chapter represent the current OrionKine Lab development roadmap. Future releases may include additional improvements based on user feedback and ongoing software development.**

SUPPORT

If you experience any issues while using OrionKine Lab, such as software errors, unexpected behavior, or bugs, please report them so they can be investigated and resolved in future updates.

User feedback plays an important role in improving OrionKine Lab and helps ensure a better experience for all users.

For technical support, product inquiries, or additional information, please refer to:

Help → Contact

within the OrionKine Lab application.

THANK YOU

Thank you for choosing OrionKine Lab.

We hope this software helps you learn, design, simulate, and develop robotic systems with confidence. We appreciate your support and look forward to continuing to improve OrionKine Lab through future updates and community feedback.

END OF MANUAL

ORIONKINE LAB USER MANUAL

VERSION 1.0