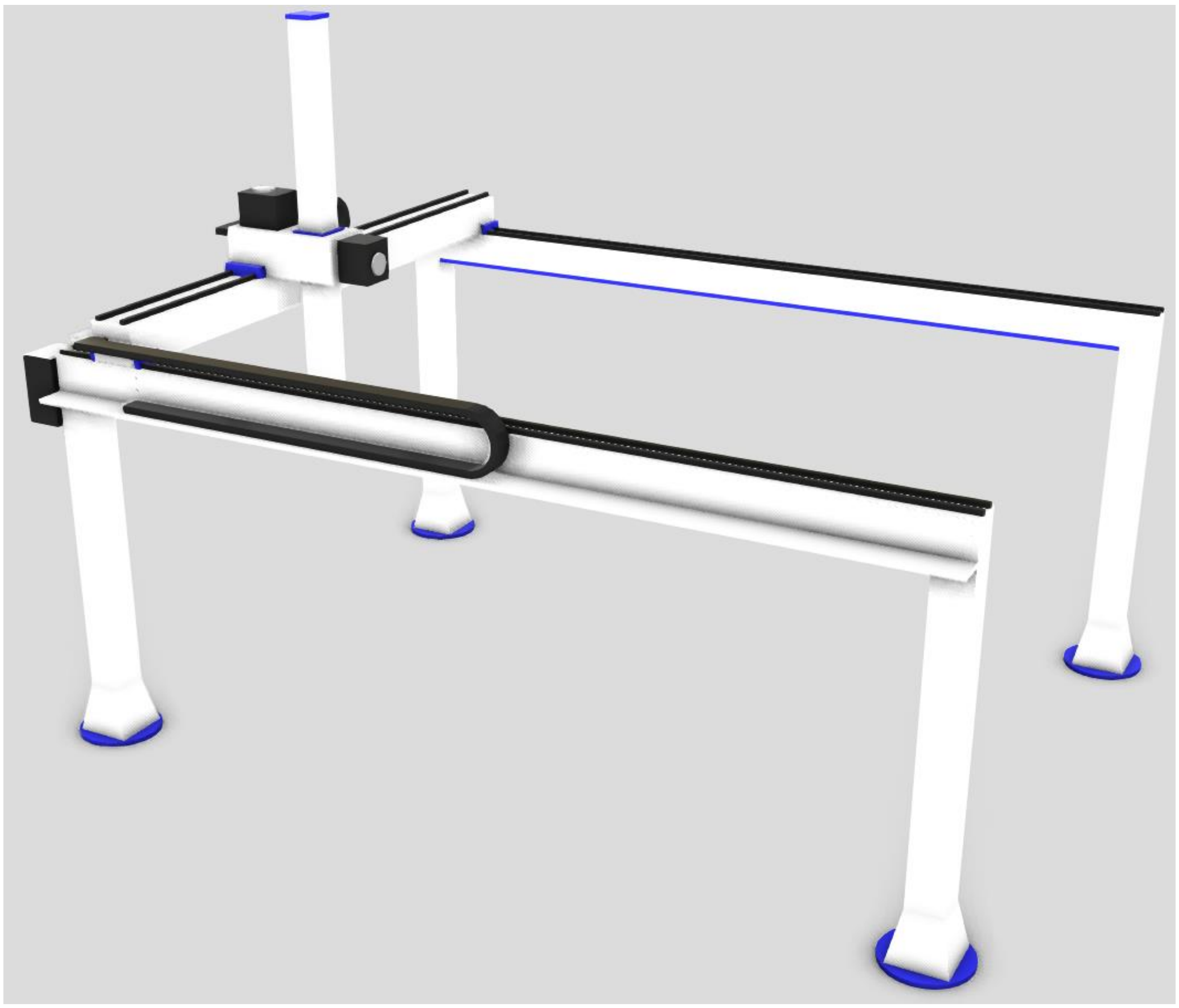
# Model a Cartesian Robot

Visual Components 4.2 Professional | Version: October 21, 2020



This tutorial will teach you how to model a cartesian robot quickly in Visual Components Professional and Premium. The tutorial uses a template robot from the Visual Components eCatalog. The template robot has the required properties and behaviors for simulation. All you need to do is to add the geometries of your robot onto the template.

**Support**
support@visualcomponents.com

**Visual Components Forum**
forum.visualcomponents.com

## Contents

## Introduction

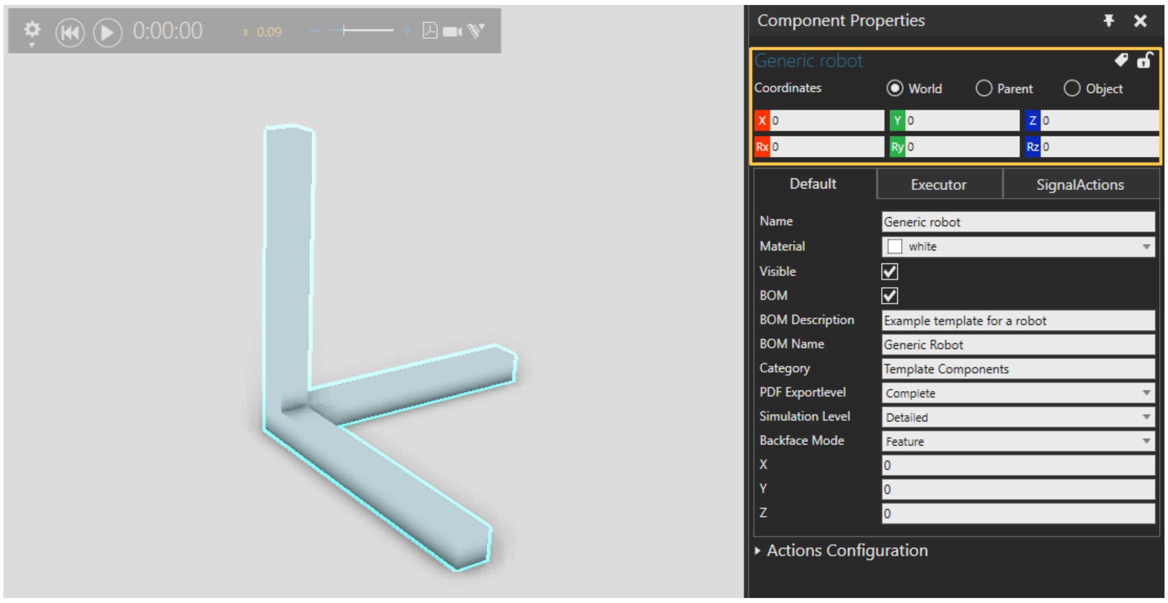
Cartesian robots also called linear robots or gantry robots, are common in many modern industrial plants. Cartesian robots with three linear axes are especially common and come in many sizes and forms, thanks to the simplicity and versatility. After completing this tutorial, you will know how to model such a robot in Visual Components, and you can teach the robot motion commands to use it in your simulations.

This tutorial utilizes a template robot from the eCatalog to simplify the modeling process. The template robot includes the necessary properties and behaviors that are required for a robot. These include the robot controller and executor, different interfaces, I/O ports for signaling, and kinematics solver.

The kinematics are handled by python kinematics behavior, which uses the Jacobian method to solve inverse kinematics. The Jacobian method is a general-purpose solver, but it is slower than calculating a symbolic solution. Thus, it is recommended for cartesian robots with six degrees of freedom to use the cartesian kinematics behavior instead of the python kinematics behavior.

## Getting started

Begin in the **HOME** tab by searching the eCatalog for the template robot. Select the **Models by Type** smart collection and navigate to **Component Templates**. Get the **Template 3-Axis Cartesian** by double-clicking or dragging and dropping it to the 3D world. Make sure that the robot is in the world's origin. You can set the position and orientation values to zero by clicking on each value's property label.



Generic robot

Coordinates: [✓ World] Parent Object

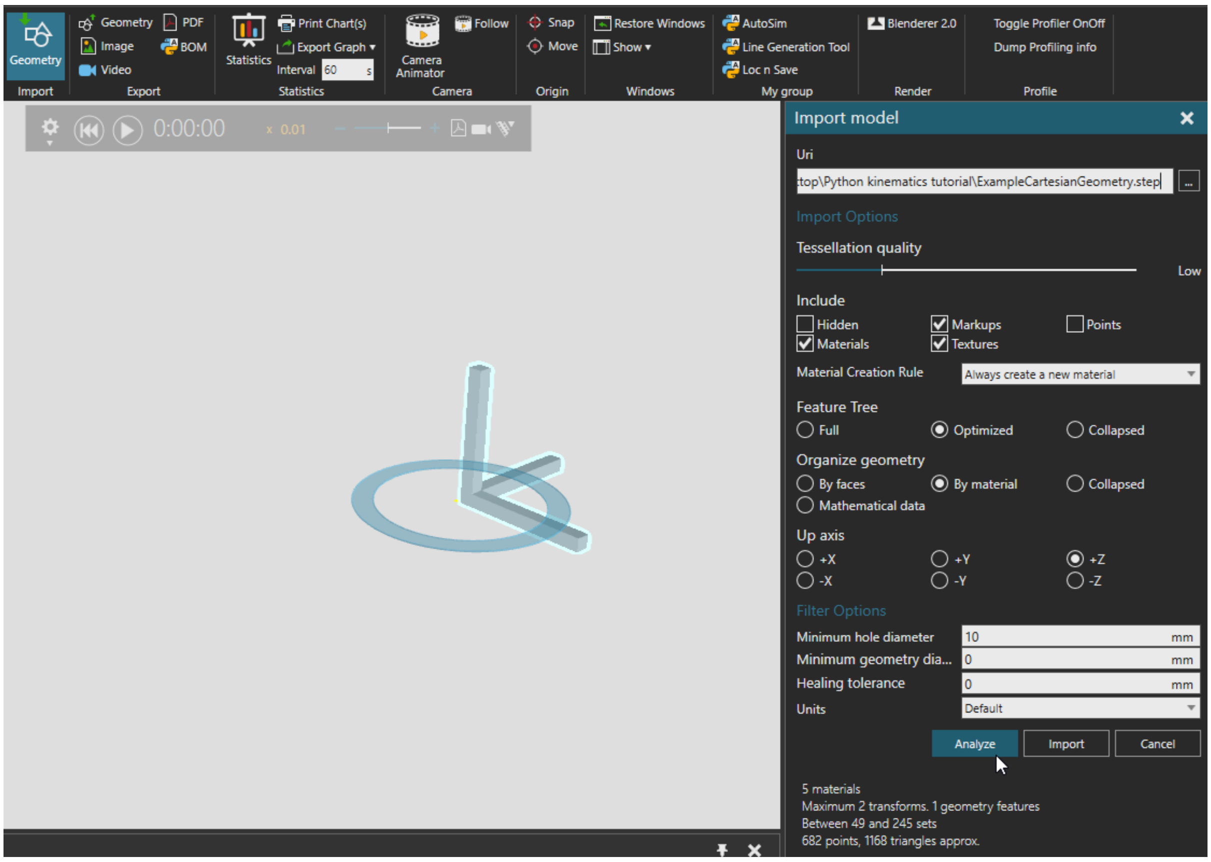
| | | |
|---|---|---|
| X 0 | Y 0 | Z 0 |
| Rx 0 | Ry 0 | Rz 0 |

| | |
|---|---|
| Name | Generic robot |
| Material | white |
| Visible | ✓ |
| BOM | ✓ |
| BOM Description | Example template for a robot |
| BOM Name | Generic Robot |
| Category | Template Components |
| PDF Exportlevel | Complete |
| Simulation Level | Detailed |
| Backface Mode | Feature |
| X | 0 |
| Y | 0 |
| Z | 0 |

The template robot has three translational axes for linear motion along X, Y, and Z coordinates. In this tutorial, the robot is a three-axis robot with only linear motion. Additional degrees of freedom (DOF) can be added for rotational joints if needed. Modeling cartesian robots with more than three DOF is shown in appendices.

## Import the robot geometries

Now that you have the template robot available, you can import the robot you wish to model. For this tutorial, a generic model of a cartesian robot is used, but you could import the 3D model of any cartesian robot and follow the same instructions. Download the .step model from the Academy by clicking the Download files button to the left of this document. Import the 3D model by pressing the **Geometry** button in the **Import** group in either the **HOME** or **MODELING** tab.



Import model

Uri: top\Python kinematics tutorial\ExampleCartesianGeometry.step

Import Options

Tessellation quality: Low

Include: [ ] Hidden [✓ Markups] [ ] Points [✓ Materials] [✓ Textures]

Material Creation Rule: Always create a new material

Feature Tree: Full [✓ Optimized] Collapsed

Organize geometry: By faces [✓ By material] Collapsed Mathematical data

Up axis: +X +Y [✓ +Z] -X -Y -Z

Filter Options

Minimum hole diameter: 10 mm
Minimum geometry dia...: 0 mm
Healing tolerance: 0 mm
Units: Default

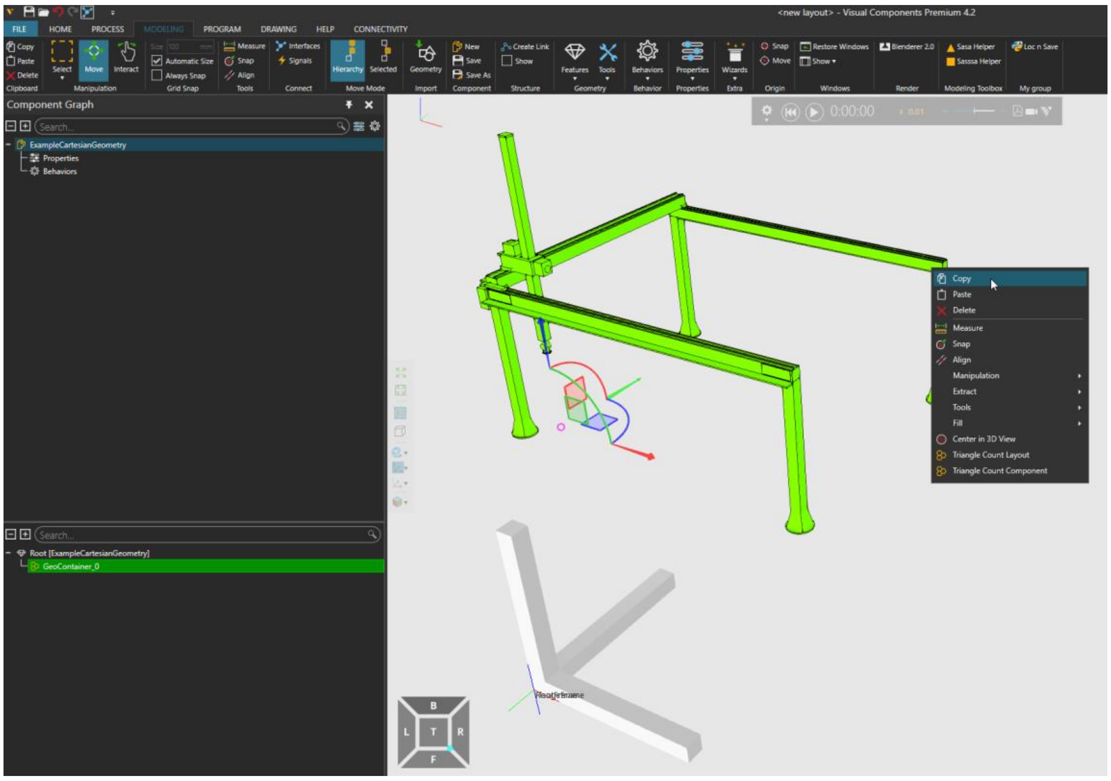
Analyze Import Cancel

5 materials
Maximum 2 transforms. 1 geometry features
Between 49 and 245 sets
682 points, 1168 triangles approx.

Select the *ExampleCartesianGeometry.step* file and set the import settings for your model. This example file is already simplified. When importing other geometries, it is good practice to simplify the geometry to make the model lighter for simulation.

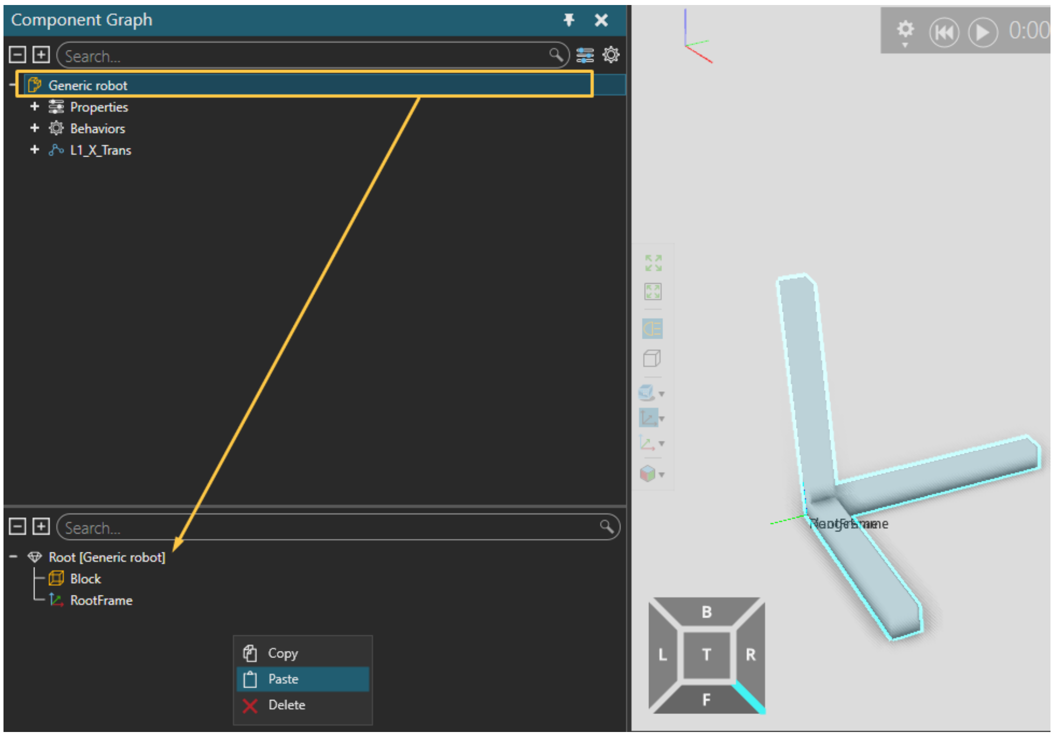
## Relocating the geometry features

The geometry model will be imported as a new component. Because this tutorial uses the template robot to speed up the modeling work, we will need to move the imported geometries to the template component. You can move the geometries by navigating to the **MODELING** tab and selecting the new component with imported geometries. Select the imported geometry features from the feature tree in the component graph located on your screen's left side.

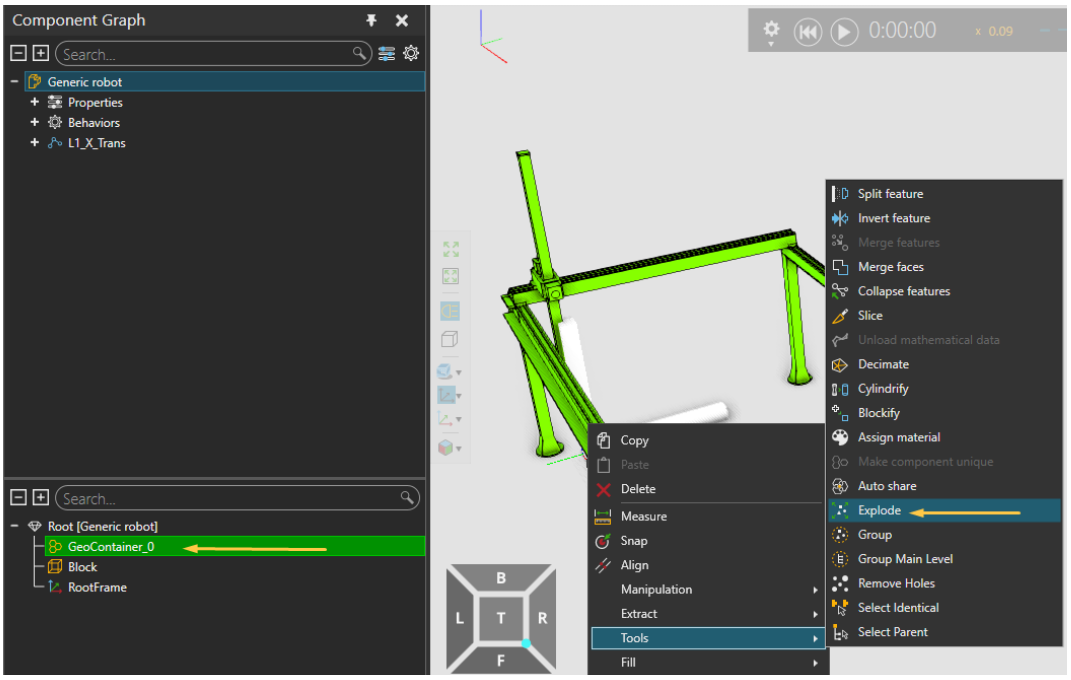


Once you have selected the features, the model should be fully green in the 3D world. Move the geometry features onto the template with the following steps:

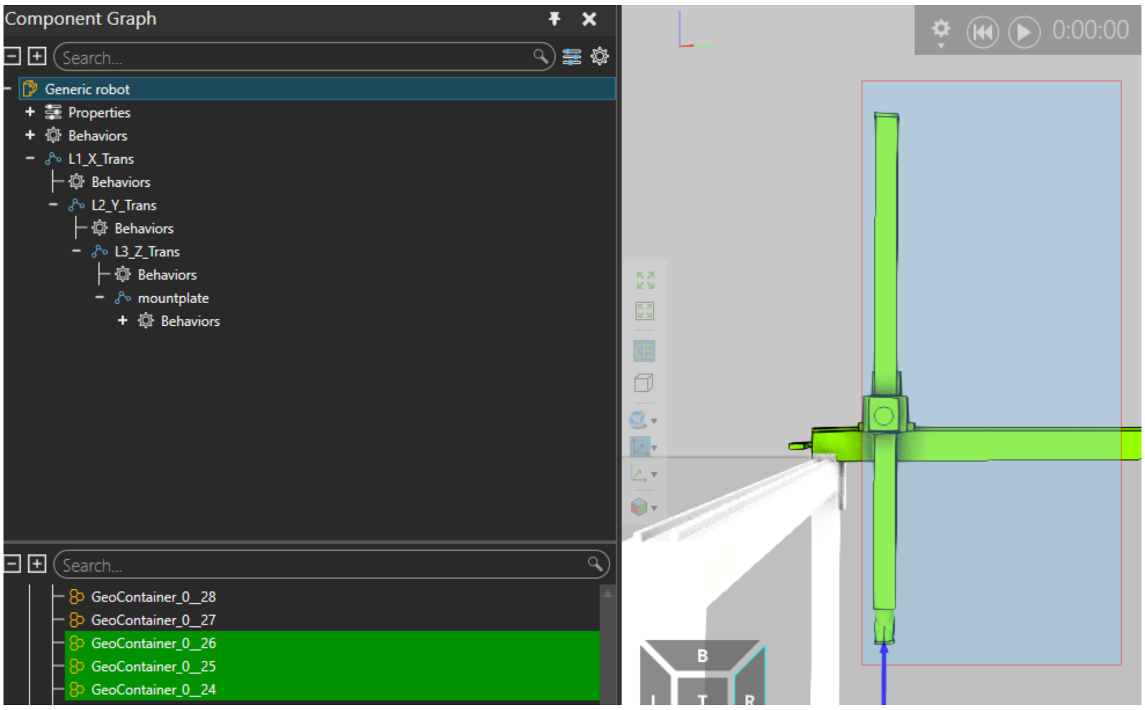
1. Copy the features by right-clicking either the feature tree or 3D world and press copy.
2. Select the Generic robot template Root node and paste the feature under the Root feature by right-clicking and selecting paste.
3. Delete the imported component and the original block features belonging to the template robot.



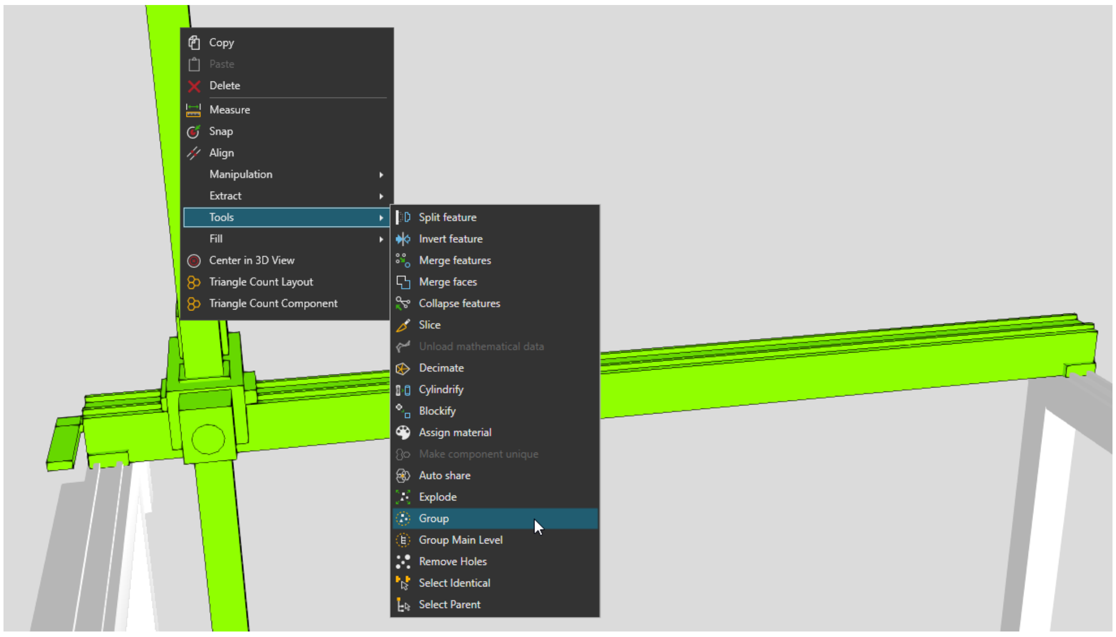
After moving the imported geometry features onto the generic template robot, you should divide the geometry features into individual links. If the geometry is in one feature, you must explode that feature to multiple smaller features. This is accomplished with the **Explode** function. Explode function is activated by selecting a feature to explode and then right-clicking on the feature to choose **Explode**. For more complex models, you might need to explode a feature multiple times to divide the feature into smaller pieces as needed.



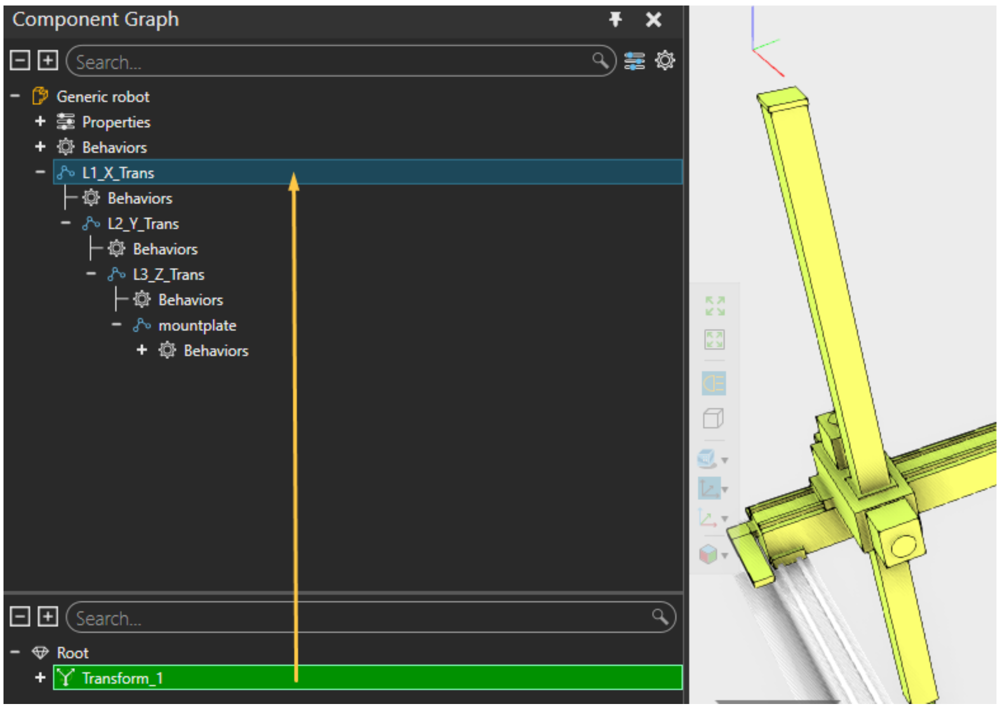
Once the geometry is split into multiple features you can start selecting groups of features that make a link, so features that should move together. In this case, select all of the top boom features, so everything except the four legs and the two support beams.



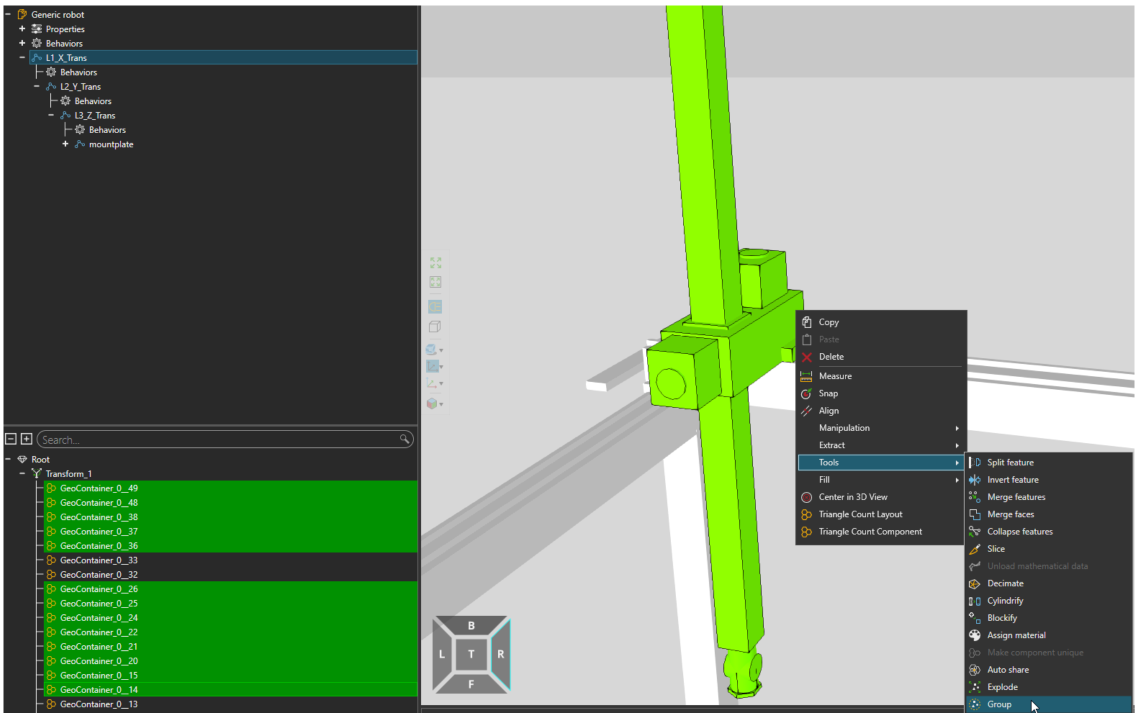
Select features either in the feature tree or directly in the 3D world. To select multiple features, hold down the **Control** key on your keyboard and click on the features. To perform rectangular selection, either use the Select tool or hold down **Control** and left mouse button in the 3D world. After you have all moving parts selected and the features are highlighted green, group the features together by right-clicking to select **Group**.



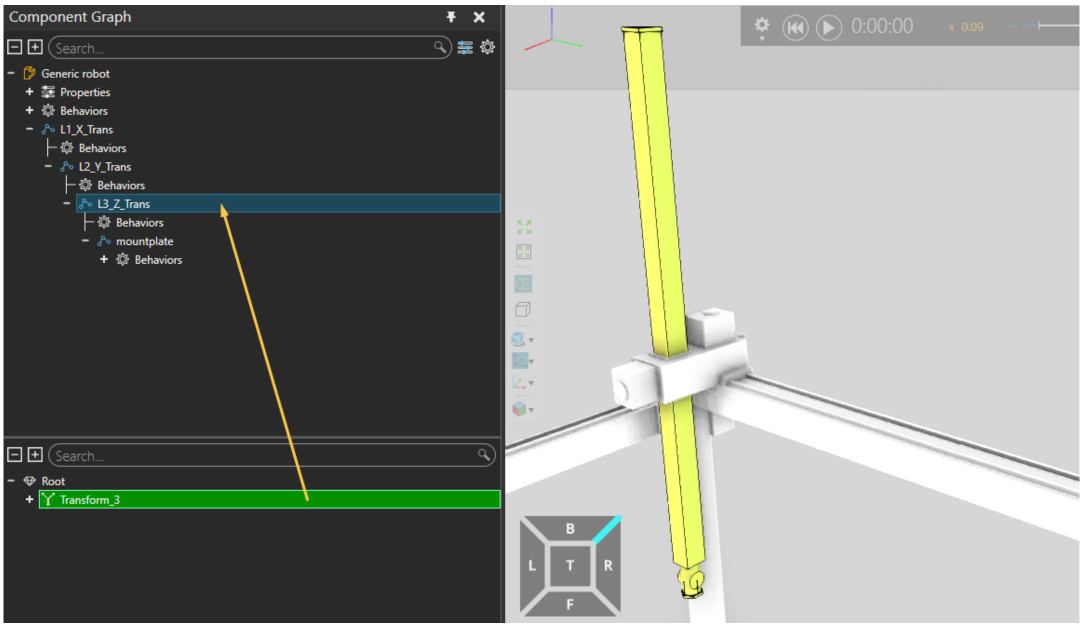
Once the features are grouped, it is easy to move the features to a moving link. Select the group of features and drag the group to the first link named L1_X_Trans. Note that the features will move relative to the parent coordinate system when moving features from one link to another. To retain the features' position in the 3D world, you can hold down the **Shift** key while dragging the features to a link. The link L1_X_Trans is configured with a translational joint that is moving along the +X axis.



Now that the top boom features have been moved to a moving link, it is good to test that no moving feature was left in the root node. Activate the **Interact** tool and try moving the boom forward and backward to inspect that all the robot's necessary parts are moving. If some features are not moving with the link, you can move those by dragging and dropping as in the previous step. After you are sure that the necessary features are in the first link, you can repeat the same process of selecting and grouping the features that should move along the +Y axis.

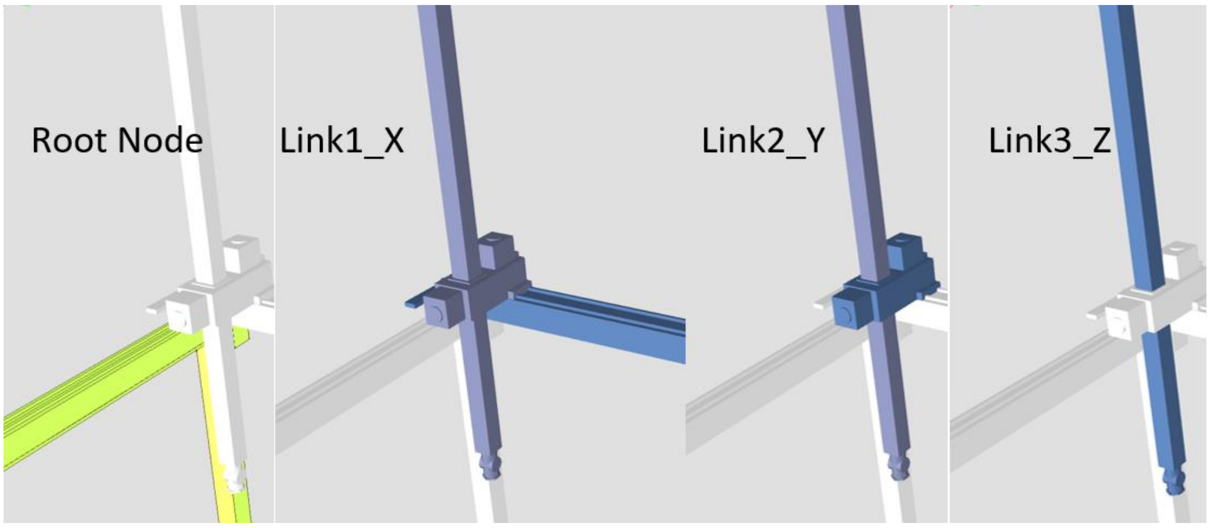


Move the features to the second link named L2_Y_Trans and use the **Interact** tool to move the link and verify that the correct features were moved to this link. Once you are happy with the second link, repeat the process once more to select the features that will move up and down, so along the +Z axis.

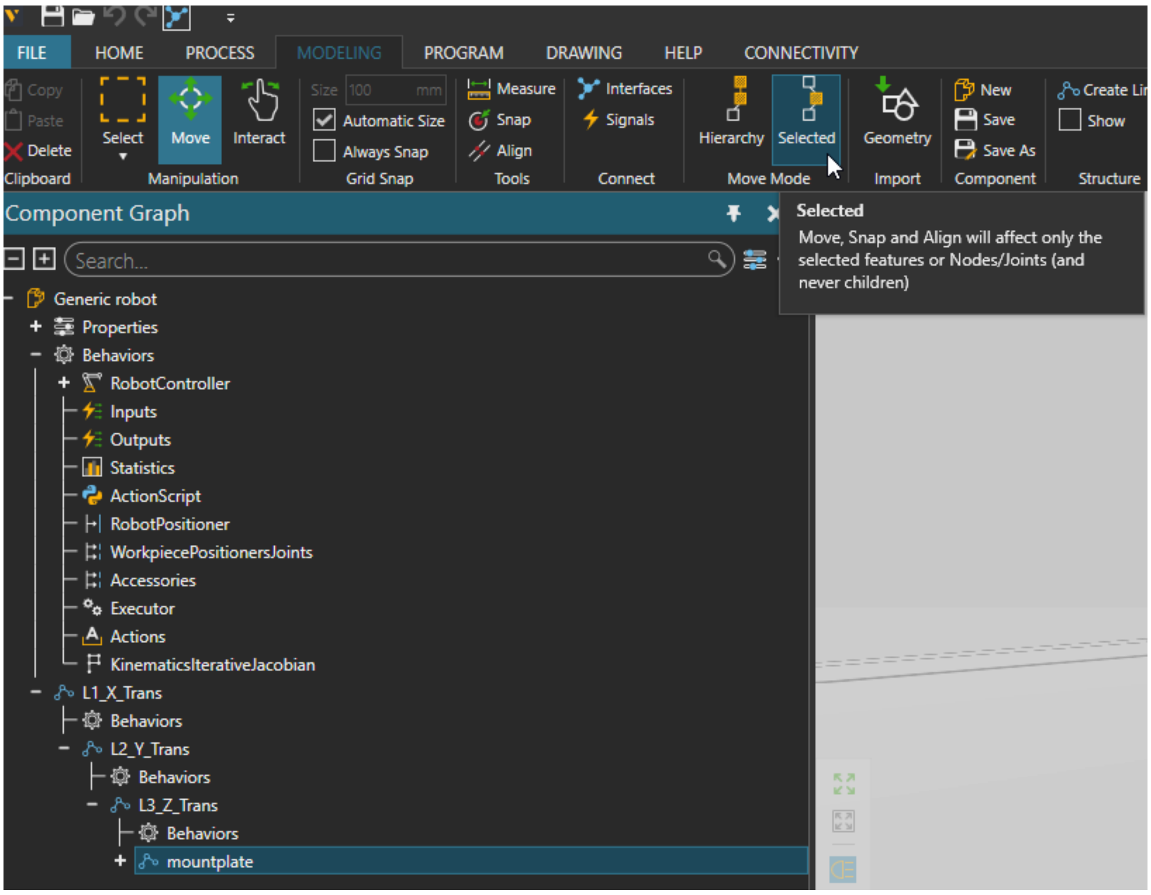


## Set the node locations

After moving the features to each link, you should see the following highlights when you select each link. Refer to the previous section if some of the features are still in incorrect links.



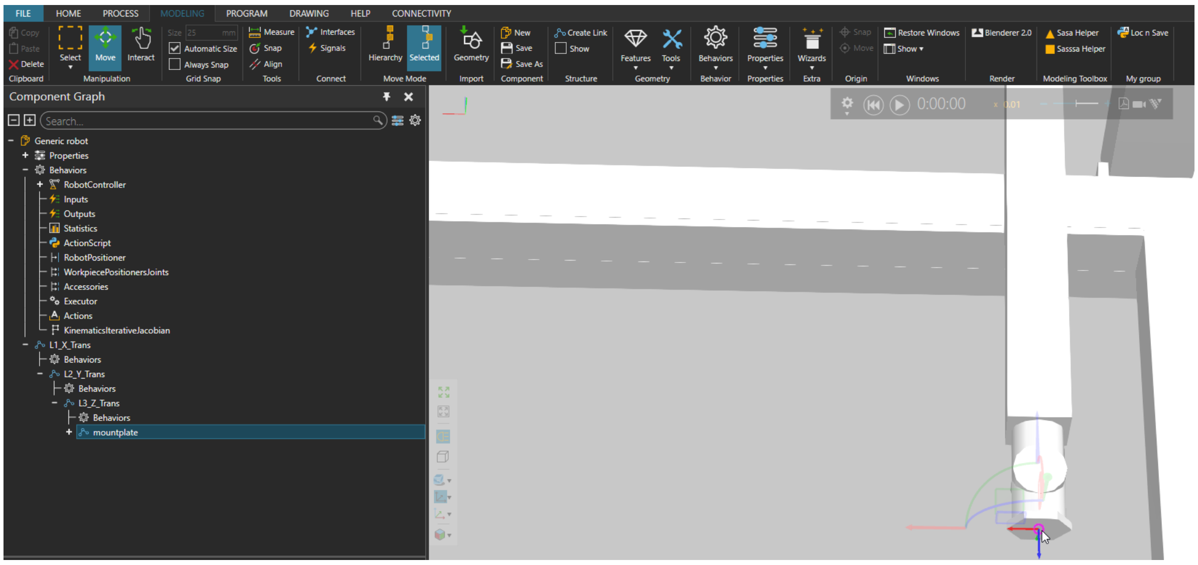
The next step in configuring the robot is to move the center point for each link to a correct position. The center point of each link defines the center of motion for that joint. Since this cartesian robot only has three joints, and each joint is configured to linear motion, the center of motion is not important. If the model has rotating joints, then the link origin must be precisely set to the center of rotation for that link.



Selected
Move, Snap and Align will affect only the selected features or Nodes/Joints (and never children)

- Generic robot
  - Properties
  - Behaviors
    - RobotController
    - Inputs
    - Outputs
    - Statistics
    - ActionScript
    - RobotPositioner
    - WorkpiecePositionersJoints
    - Accessories
    - Executor
    - Actions
    - KinematicsIterativeJacobian
  - L1_X_Trans
    - Behaviors
    - L2_Y_Trans
      - Behaviors
      - L3_Z_Trans
        - Behaviors
        - mountplate

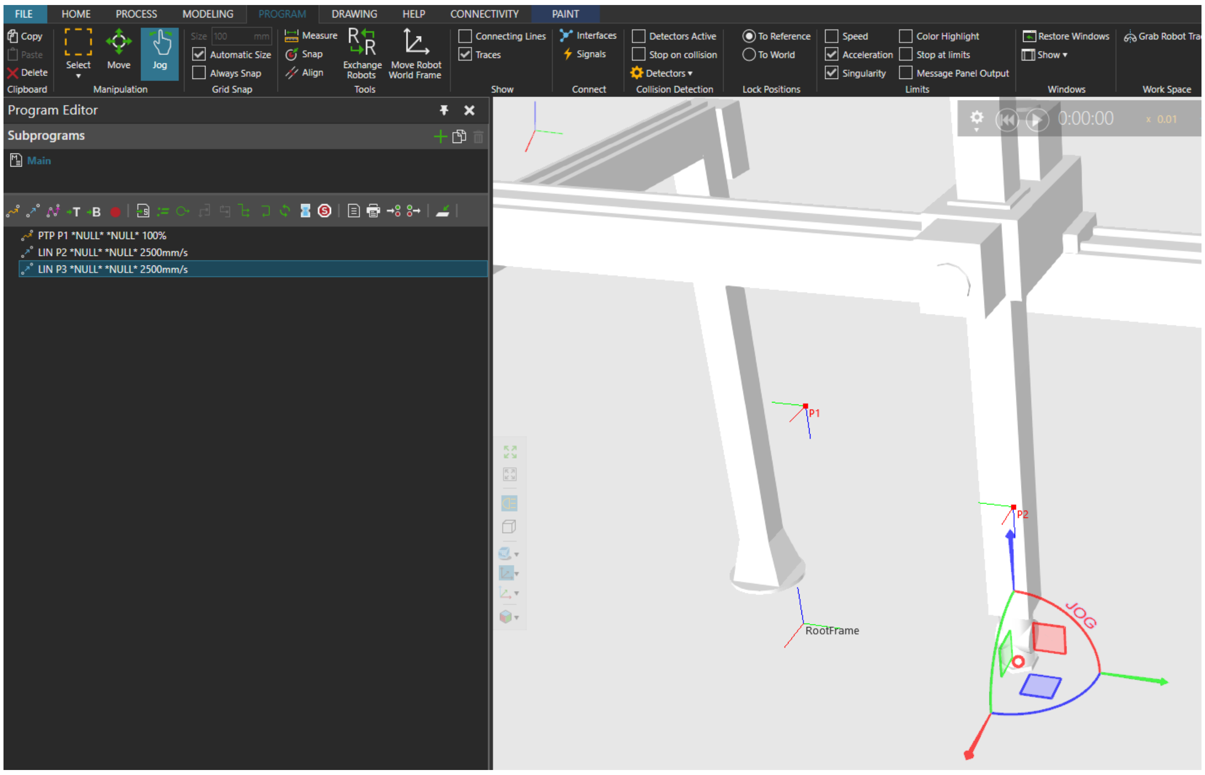
The most effective way to move link origin is to switch the **Move Mode** to **Selected** and use the **Move** tool. Move the *mountplate* link to the end effector location in the 3D world, as shown by the image below. Be sure also to move the FlangeFrame feature to the new end-effector location.


Component Graph

```
- Generic robot
  + Properties
  - Behaviors
    + RobotController
      Inputs
      Outputs
      Statistics
      ActionScript
      RobotPositioner
      WorkpiecePositionersJoints
      Accessories
      Executor
      Actions
      KinematicsIterativeJacobian
  - L1_X_Trans
      Behaviors
    - L2_Y_Trans
        Behaviors
      - L3_Z_Trans
          Behaviors
        + mountplate
```

## Setting the robot properties

After moving the *mountplate* link and the FlangeFrame feature, switch the Move Mode back to Hierarchy and hit the **Reset** button to refresh the robot's end-effector location. Now you can teach the robot motion commands in the **Program** tab. Activate the **Jog** tool, move the robot end effector, and try teaching point-to-point and linear motion commands for the robot.

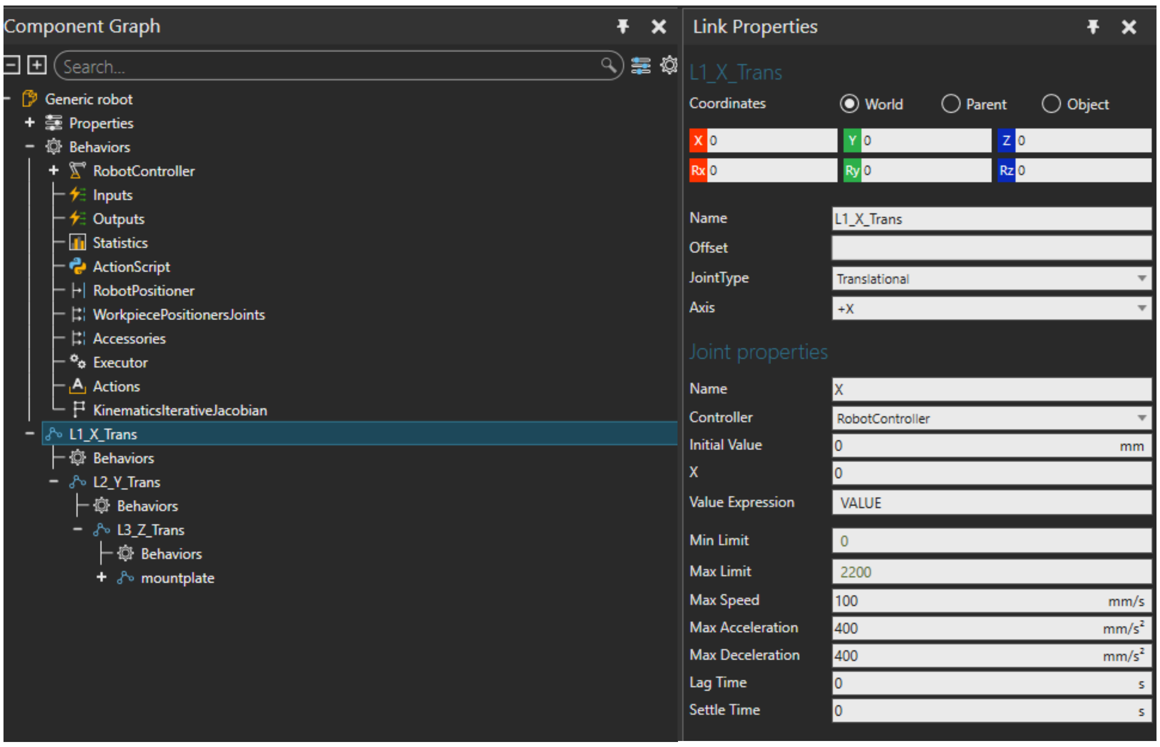

Program Editor

Subprograms

Main

```
PTP P1 *NULL* *NULL* 100%
LIN P2 *NULL* *NULL* 2500mm/s
LIN P3 *NULL* *NULL* 2500mm/s
```

Once you execute the robot program, the robot will move from target to target. The robot might be too fast, too slow, or reach impossible motion targets, as the robot has preset default values for joint velocities and limits. The individual joint properties can be set by selecting a link from the component graph and entering the desired properties, such as Minimum and Maximum Limit, Speed, and Acceleration, for the link. Jog the robot axes to determine suitable MinLimit and MaxLimit for each joint. These values are usually provided in the robot datasheet.



Component Graph

```
- Generic robot
  + Properties
  - Behaviors
    + RobotController
    ├ Inputs
    ├ Outputs
    ├ Statistics
    ├ ActionScript
    ├ RobotPositioner
    ├ WorkpiecePositionersJoints
    ├ Accessories
    ├ Executor
    ├ Actions
    └ KinematicsIterativeJacobian
  - L1_X_Trans
    ├ Behaviors
    - L2_Y_Trans
      ├ Behaviors
      - L3_Z_Trans
        ├ Behaviors
        + mountplate
```

Link Properties

L1_X_Trans

Coordinates [✓ World] Parent Object

| | | |
|---|---|---|
| X 0 | Y 0 | Z 0 |
| Rx 0 | Ry 0 | Rz 0 |

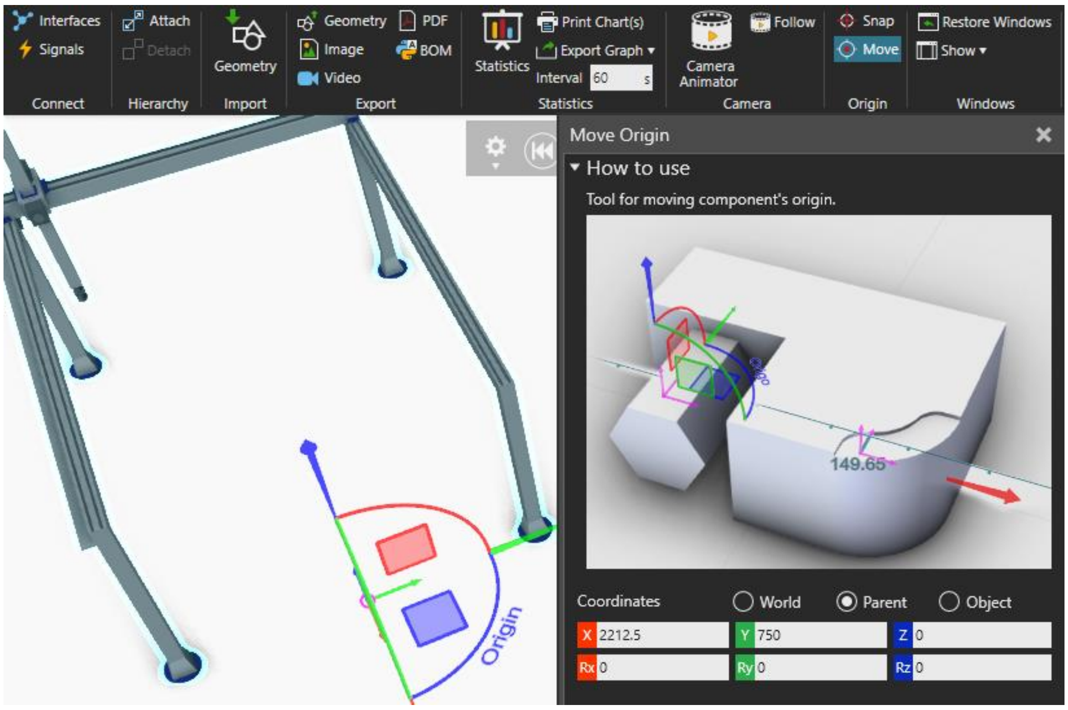
| | |
|---|---|
| Name | L1_X_Trans |
| Offset | |
| JointType | Translational |
| Axis | +X |

Joint properties

| | |
|---|---|
| Name | X |
| Controller | RobotController |
| Initial Value | 0 mm |
| X | 0 |
| Value Expression | VALUE |
| Min Limit | 0 |
| Max Limit | 2200 |
| Max Speed | 100 mm/s |
| Max Acceleration | 400 mm/s² |
| Max Deceleration | 400 mm/s² |
| Lag Time | 0 s |
| Settle Time | 0 s |

You can access all the component properties and behaviors in the modeling tab. For example, selecting the root node, it is possible to change the component name and material. The robot controller properties for maximum cartesian speed, Acceleration, and so on can be changed by selecting the RobotController behavior. It is not necessary to modify these values in this tutorial.

Robots usually have a well-defined origin and robot world frame. The robot world frame location is generally shown in the robot datasheet. For this robot, the origin and world frame can be centered at the front of the robot In the **MODELING** tab, go to the **Origin** section and use the **Move Origin** tool to relocate the robot origin. Press Apply from the **Move Origin** panel to save the changed position. After moving the robot origin, be sure to move the RootFrame feature from the robot root node to the new origin.



Move Origin

How to use

Tool for moving component's origin.

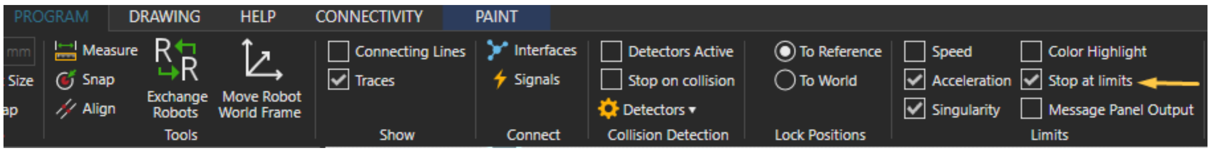
Coordinates: World [✓ Parent] Object

X 2212.5 Y 750 Z 0

Rx 0 Ry 0 Rz 0

To move the robot world frame, switch to the **PROGRAM** tab and select the **Move Robot World Frame** tool. With the tool open, select the Parent coordinate system and zero the coordinates.

The robot is now finished. You can further experiment with the robot by extending the robot program with additional motion commands, reading input signals, and writing output signals. Besides robot programming, the Program tab has numerous tools for debugging the robot program. Try enabling the Stop at limits to ensure that the robot does not move past the defined joint minimum and maximum values.



Connecting Lines ☐ Traces ✓

Detectors Active ☐ Stop on collision ☐

[✓ To Reference] To World

Speed ☐ Acceleration ✓ Singularity ✓

Color Highlight ☐ Stop at limits ✓ Message Panel Output ☐

## Review

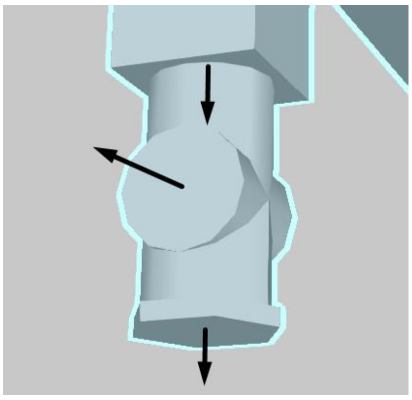
In this tutorial, you learned how to model a simple cartesian robot with Visual Components Professional or Premium. The tutorial result is a linear robot with translational joints along the X, Y, and Z-axes. The appendices show how to adapt the model for additional degrees of freedom and rotational joints.

Because python kinematics behavior is used to solve the inverse kinematics, the model works best for simple kinematic chains. This general-purpose python kinematics can be adapted to handle more complicated structures. The solver can be modified in the Modeling tab by double-clicking on the KinematicsIterativeJacobian behavior. More information about robot modeling and python kinematics is available in the Visual Components Help file.

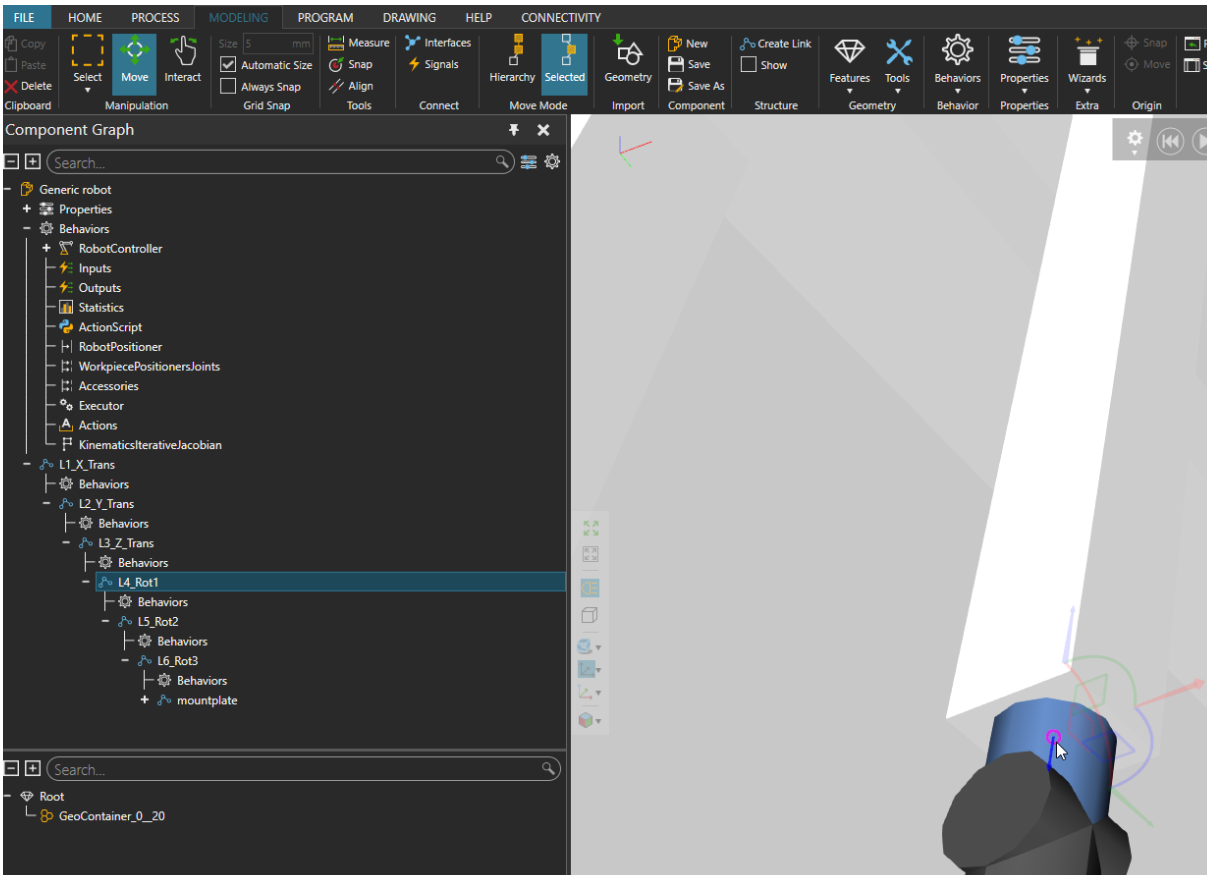
# Appendix

**Adapting the template robot to add more degrees of freedom**

It is possible to extend the model further by adding additional degrees of freedom to the robot. There is a swivel mechanism at the end of the third link, which can be configured for three rotational joints.



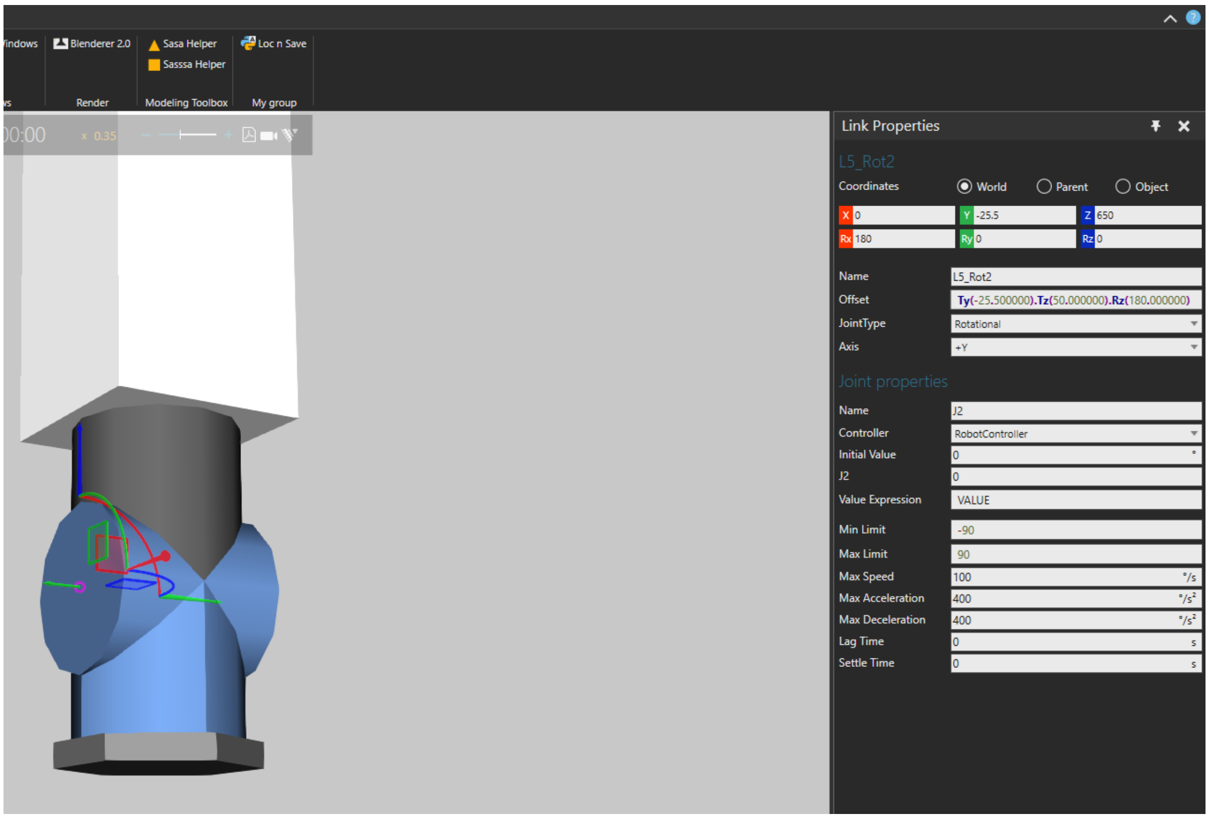
These rotational joints must first be added to the Component Graph. Use either the **Create Link** or **Extract Link** tool. Set the links to the correct location in the node hierarchy by dragging. Once the Component Graph is in order, move the geometry features to the correct links by dragging and dropping the features. Hold down the **Shift** key to preserve the feature's world location.



```
Generic robot
  Properties
  Behaviors
    RobotController
    Inputs
    Outputs
    Statistics
    ActionScript
    RobotPositioner
    WorkpiecePositionersJoints
    Accessories
    Executor
    Actions
    KinematicsIterativeJacobian
  L1_X_Trans
    Behaviors
    L2_Y_Trans
      Behaviors
      L3_Z_Trans
        Behaviors
        L4_Rot1
          Behaviors
          L5_Rot2
            Behaviors
            L6_Rot3
              Behaviors
              mountplate
```

```
Root
  GeoContainer_0_20
```

These joints need to have the correct axis of rotation; thus, it is necessary to relocate the node origins. Use the **Move** tool to move the node origins to match the geometry. Switch the Move Mode to **Selected** to keep the geometry features from moving.



Link Properties

L5_Rot2

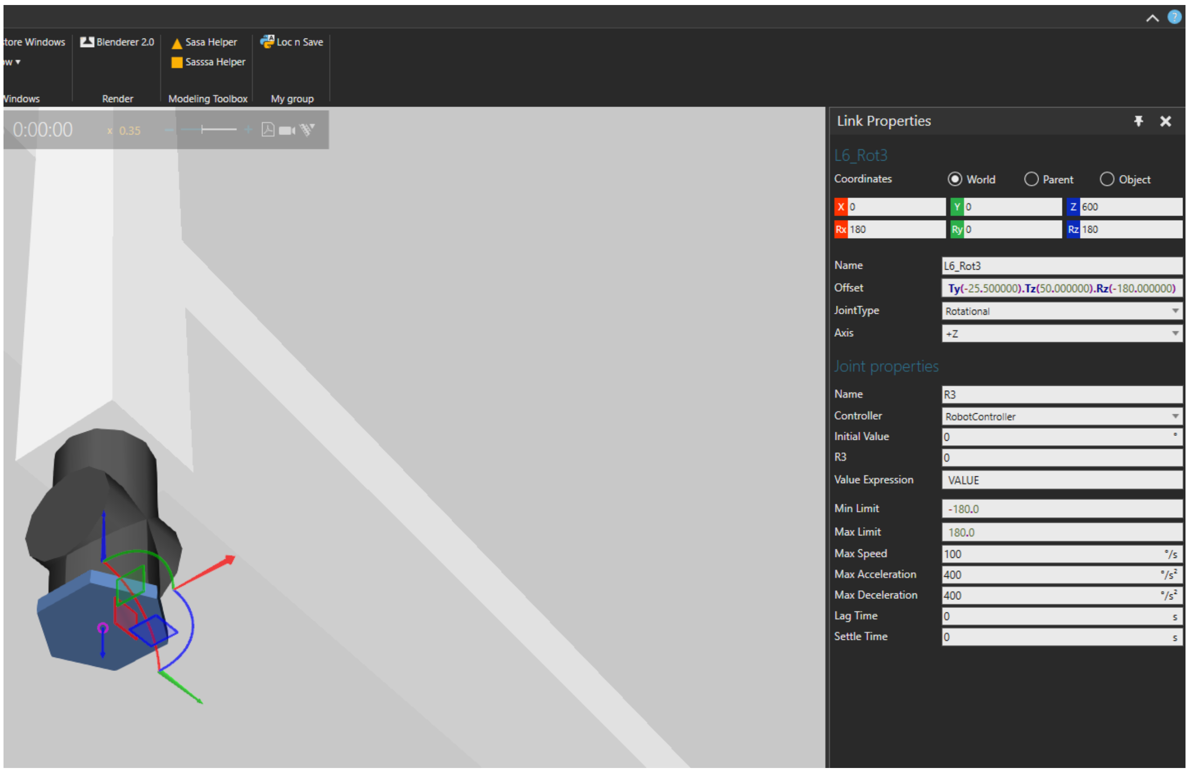
| | | | |
|---|---|---|---|
| Coordinates | [✓ World] | Parent | Object |
| X 0 | Y -25.5 | Z 650 | |
| Rx 180 | Ry 0 | Rz 0 | |

| | |
|---|---|
| Name | L5_Rot2 |
| Offset | Ty(-25.500000).Tz(50.000000).Rz(180.000000) |
| JointType | Rotational |
| Axis | +Y |

Joint properties

| | |
|---|---|
| Name | J2 |
| Controller | RobotController |
| Initial Value | 0 ° |
| J2 | 0 |
| Value Expression | VALUE |
| Min Limit | -90 |
| Max Limit | 90 |
| Max Speed | 100 °/s |
| Max Acceleration | 400 °/s² |
| Max Deceleration | 400 °/s² |
| Lag Time | 0 s |
| Settle Time | 0 s |

Configure the joint properties for each link as needed. Set the Axis, Controller, and other properties as with the linear joints.



Link Properties

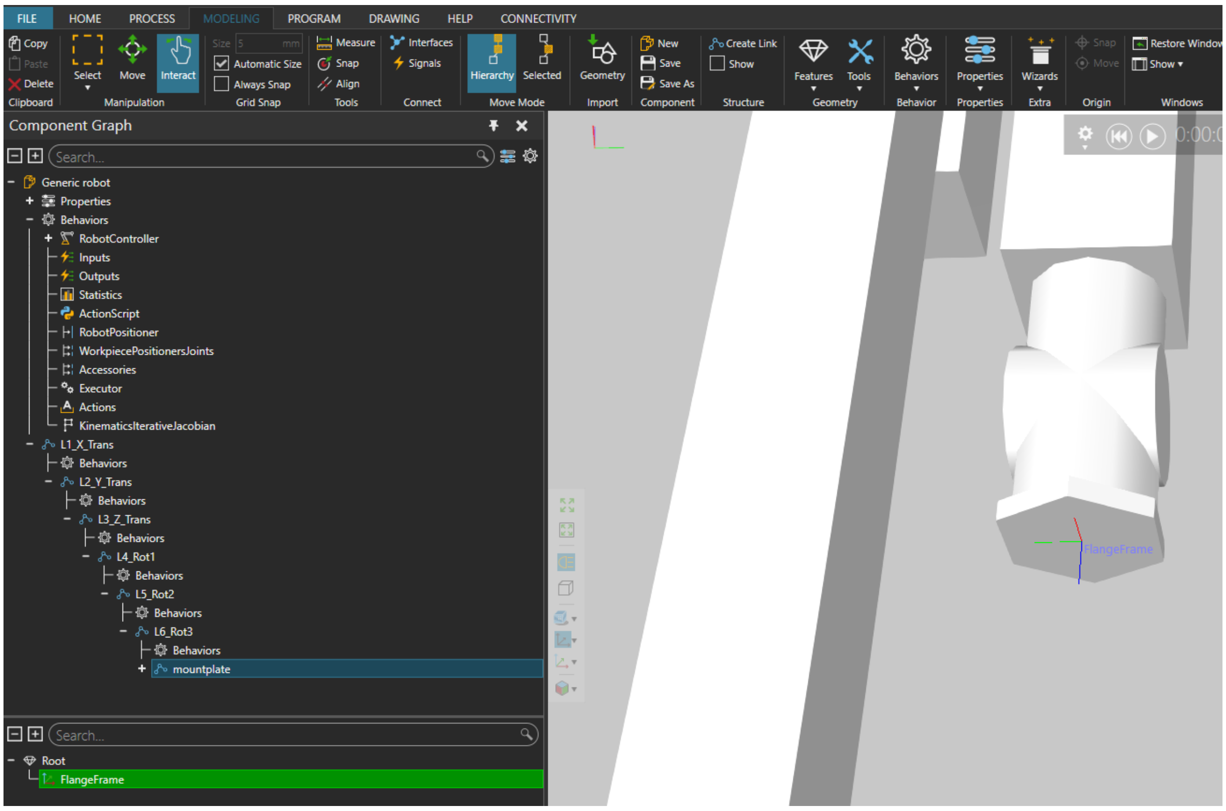
L6_Rot3

| | | | |
|---|---|---|---|
| Coordinates | [✓ World] | Parent | Object |
| X 0 | Y 0 | Z 600 | |
| Rx 180 | Ry 0 | Rz 180 | |

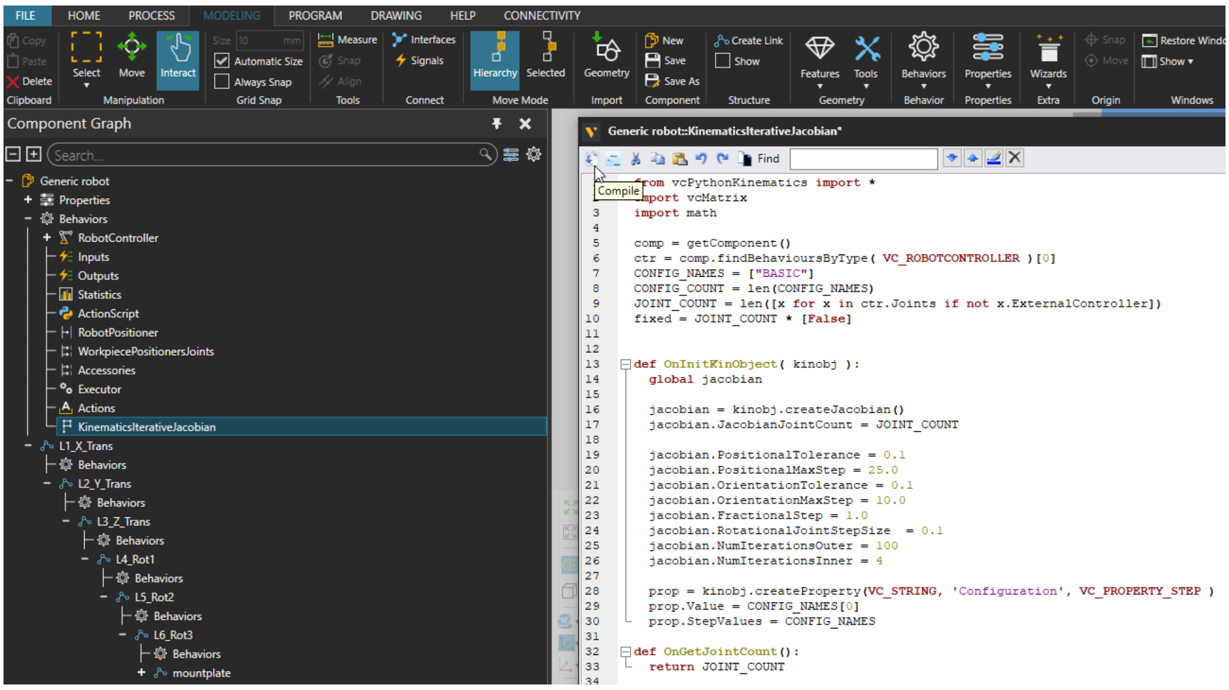
| | |
|---|---|
| Name | L6_Rot3 |
| Offset | Ty(-25.500000).Tz(50.000000).Rz(-180.000000) |
| JointType | Rotational |
| Axis | +Z |

Joint properties

| | |
|---|---|
| Name | R3 |
| Controller | RobotController |
| Initial Value | 0 ° |
| R3 | 0 |
| Value Expression | VALUE |
| Min Limit | -180.0 |
| Max Limit | 180.0 |
| Max Speed | 100 °/s |
| Max Acceleration | 400 °/s² |
| Max Deceleration | 400 °/s² |
| Lag Time | 0 s |
| Settle Time | 0 s |



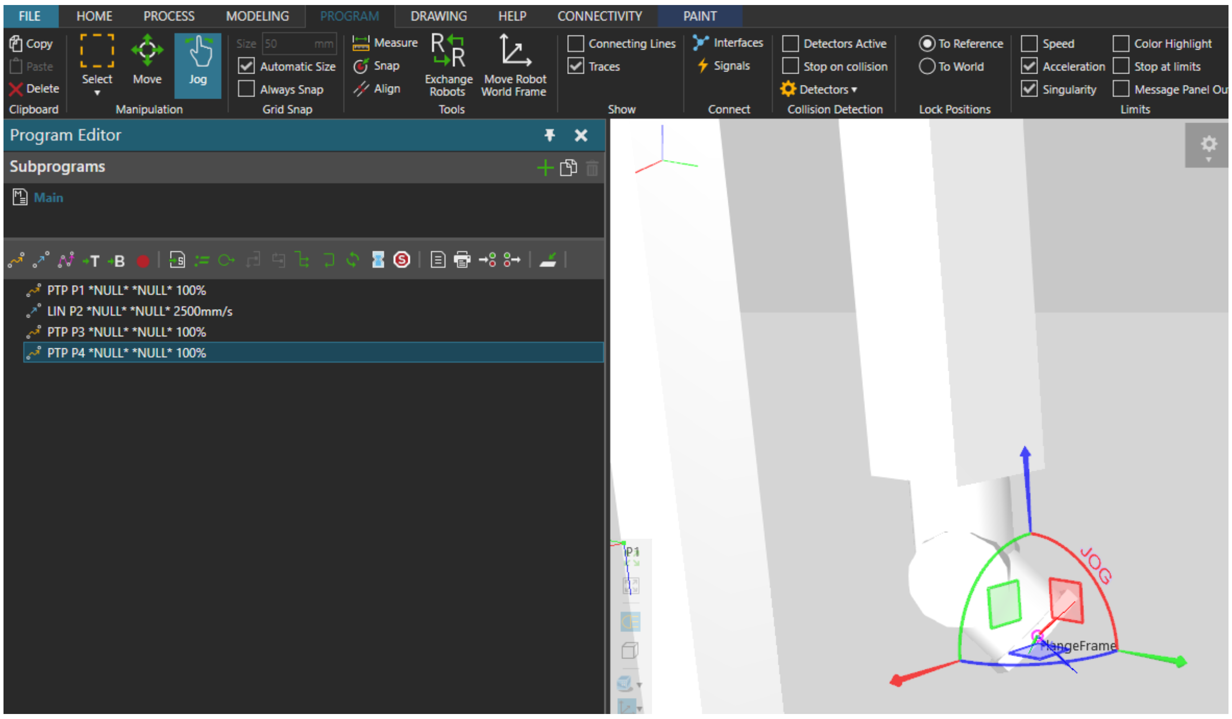
Once the additional joints have been configured, ensure that the *mountplate* link and the *FlangeFrame* feature are still in the correct position in the robot end-effector. Now that the robot's kinematic structure has been changed, the kinematic solver must be reset for the changes to take effect. First, open the KinematicsIterativeJacobian behavior by double-clicking it in the Component Graph. The script must be complied for the changes to apply, and an easy way to do this is to add an empty line somewhere in the script and press the **Compile** button.



```
from vcPythonKinematics import *
import vcMatrix
import math

comp = getComponent()
ctr = comp.findBehavioursByType( VC_ROBOTCONTROLLER )[0]
CONFIG_NAMES = ["BASIC"]
CONFIG_COUNT = len(CONFIG_NAMES)
JOINT_COUNT = len([x for x in ctr.Joints if not x.ExternalController])
fixed = JOINT_COUNT * [False]


def OnInitKinObject( kinobj ):
  global jacobian

  jacobian = kinobj.createJacobian()
  jacobian.JacobianJointCount = JOINT_COUNT

  jacobian.PositionalTolerance = 0.1
  jacobian.PositionalMaxStep = 25.0
  jacobian.OrientationTolerance = 0.1
  jacobian.OrientationMaxStep = 10.0
  jacobian.FractionalStep = 1.0
  jacobian.RotationalJointStepSize  = 0.1
  jacobian.NumIterationsOuter = 100
  jacobian.NumIterationsInner = 4

  prop = kinobj.createProperty(VC_STRING, 'Configuration', VC_PROPERTY_STEP )
  prop.Value = CONFIG_NAMES[0]
  prop.StepValues = CONFIG_NAMES

def OnGetJointCount():
  return JOINT_COUNT
```

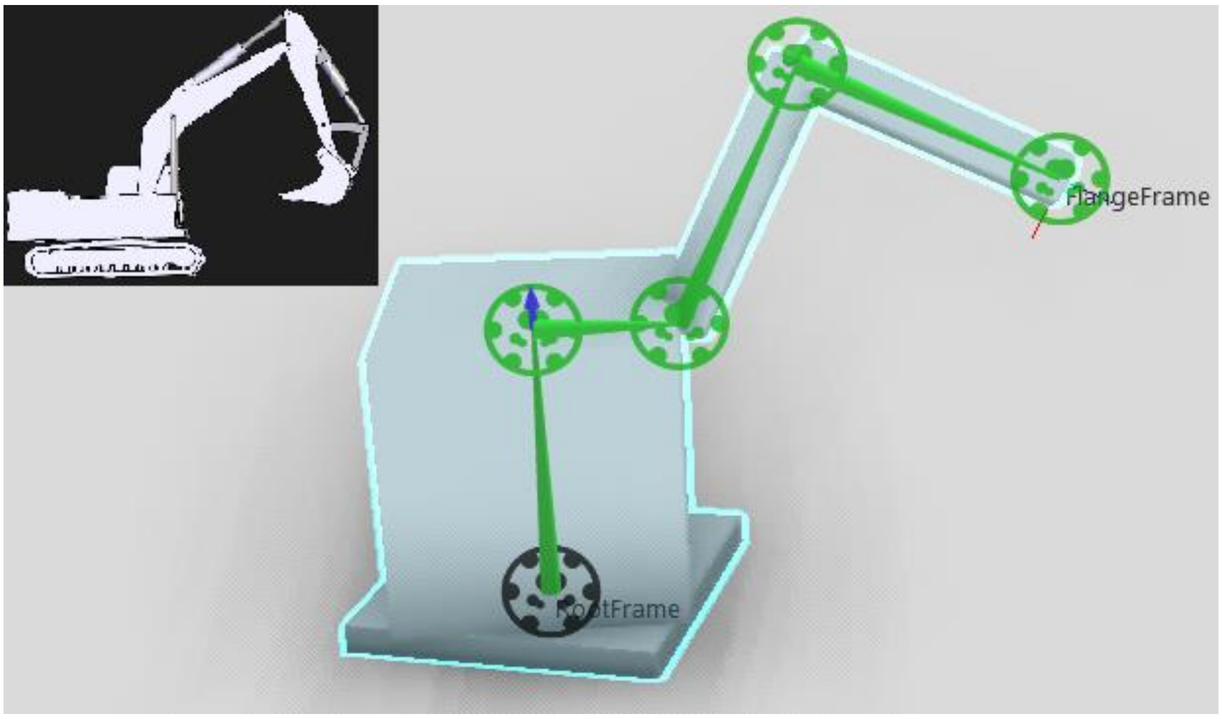
Close the code editor once you have compiled the script. It is possible to modify the script's solving algorithm, but that is not covered in this tutorial. See the Help file for vcPythonKinematics to learn more.

The robot model has now been updated with three rotating joints. Press the Reset button to ensure that the simulation is in the initial state and activate the Jog tool to test the robot.



```
PTP P1 *NULL* *NULL* 100%
LIN P2 *NULL* *NULL* 2500mm/s
PTP P3 *NULL* *NULL* 100%
PTP P4 *NULL* *NULL* 100%
```

That concludes this tutorial. As a final note, the python kinematics behavior can be adapted to handle a wide variety of different kinematic structures. By thinking through the kinematic structure and different joint types, it is possible to build more than simple cartesian robots.


FlangeFrame
RootFrame