HexGen® Hexapod Programming

USER GUIDE

Revision 2.01 (for Automation1 software versions 2.11.0 or newer)

GLOBAL TECHNICAL SUPPORT

Go to the Global Technical Support Portal for information and support about your Aerotech, Inc. products. The website supplies software, product manuals, Help files, training schedules, and PC-to-PC remote technical support. If necessary, you can complete Product Return (RMA) forms and get information about repairs and spare or replacement parts. To get help immediately, contact a service office or your sales representative. Include your customer order number in your email or have it available before you call.

Table of Contents

List of Figures

Safety Procedures and Warnings

IMPORTANT: Refer to the hexapod and electronic drive hardware manuals for warning and safety information about your system.

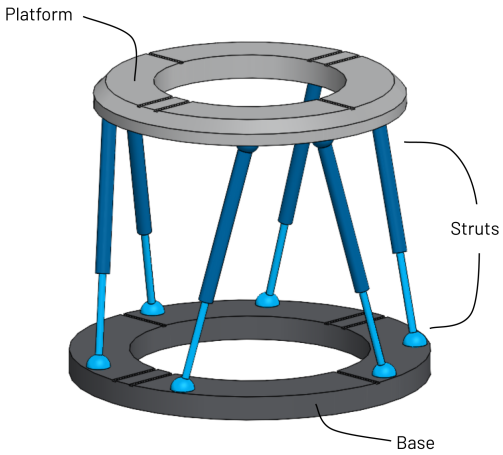
Optional Purchase Necessary: This guide describes methods to program hexapods in Automation1. To use these programming tools, you must purchase the Automation1 HX1 option. The controller shows the message that follows when there is not an HX1 option available in your system: “HX1 option not present. Cannot operate the Hexapod. Contact Aerotech for an updated license key.”

This page intentionally left blank.

Chapter 1: Overview: Hexapod Motion

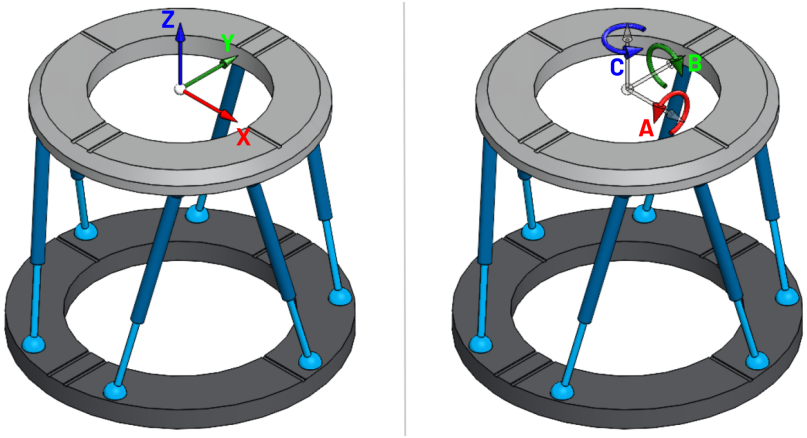
In this guide, hexapods are mechanical devices that can move a payload with six degrees of freedom. The hexapods in this document have a stationary base with a moving platform connected by six linear actuators called struts. Rotary joints connect the struts to the base and to the platform.

Figure 1-1: Basic Hexapod



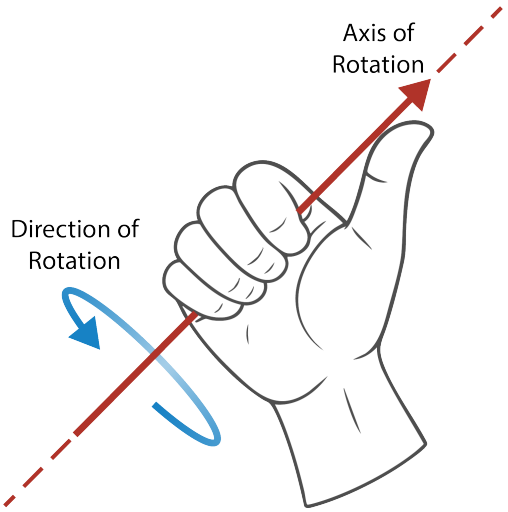
Hexapods are parallel-kinematic devices. The six struts of the hexapod change length during operation to control the position of the platform in all six degrees of freedom. The controller uses coordinate transformations to convert strut motion into platform motion. In Figure 1-2, X, Y, and Z are translational axes, and A, B, and C are rotational axes.

Figure 1-2: Hexapod Motion Axes



Rotations obey the right hand rule, which means that positive rotation is counterclockwise when looking along the axis of rotation in the negative direction. Refer to Figure 1-3.

Figure 1-3: Right Hand Rule



The Automation1 controller uses the Bryant Angle method to calculate the order of operations: A-B-C. The A axis rotates B and C, and then the B axis rotates C. The C axis does not rotate either A or B. Refer to Figure 1-4 and Figure 1-5.

Figure 1-4: Bryant Angle Stacked Stage Example

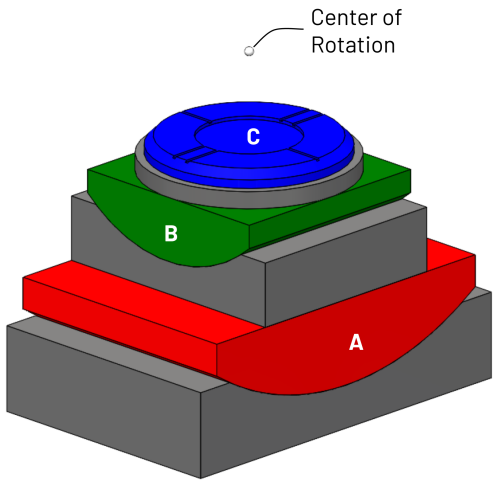
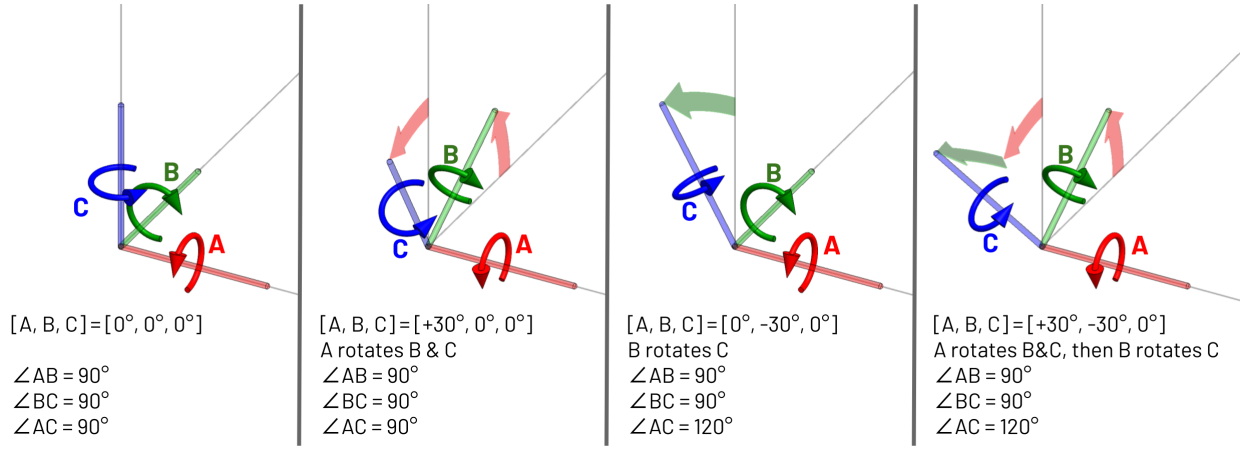


Figure 1-5: Bryant Angle Order of Operations



1.1. Interfacing with Hexapods in Automation1

Automation1 is available as a drive-based or a PC-based controller. The drive-based controller can control one hexapod and other non-hexapod axes. The PC-based controller can control two hexapods and other non-hexapod axes.

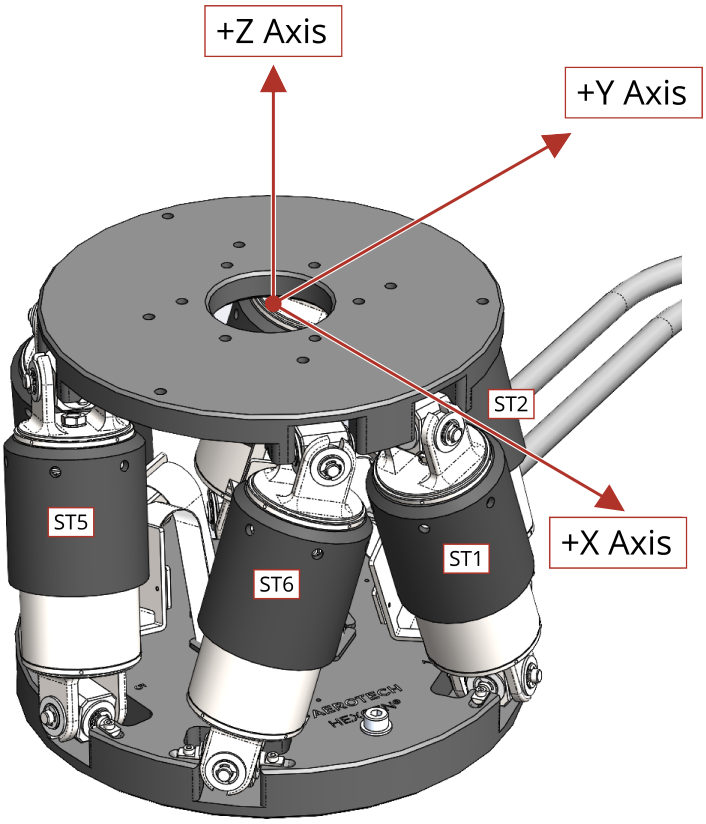
The standard Automation1 application programming interface (API) and a hexapod API command hexapod motion in Automation1. The syntax of the hexapod API is in Chapter 4: Programming Motion of this guide.

Several files and features make hexapod control possible in Automation1. Aerotech configures these files and features for you, which include:

- Transformation files: These convert between strut positions and platform positions.
- An initialization file, **HexCfg.ini**: This file contains geometric description and setup information.
- Geometric correction files: These improve the accuracy of the hexapod.
- A compiled AeroScript library, **HexCLibrary.a1lib**: The library includes the hexapod API commands.
- A restricted task program, **HexapodCStateMachine.a1exe**: This program manages the state of the hexapod.

Refer to your hexapod hardware manual to see the orientation of the motion axes in relation to the physical features of the hexapod (mounting holes, etc.). See Figure 1-6, which is from the HEX150-125HL hardware manual.

Figure 1-6: HEX150-125HL Axis Orientation Example



The default hexapod axis names are X, Y, Z, A, B, and C. The syntax in this guide assumes those axis names and in that order. You can change the names of the axes in Automation1 Studio and the **HexCfg.ini** file without having an effect on the axis index or the direction of motion. In Figure 1-6, commanding motion to the first axis of the hexapod moves the center of the platform toward the gap between strut one and strut six, even if the name of the motion axis is different.

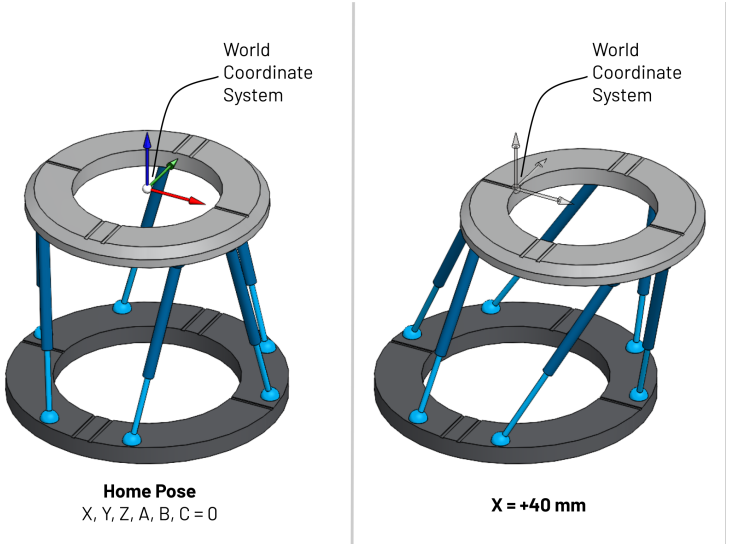
1.2. The Home Position

The home position is a special orientation, also known as a pose, of a hexapod. In the home pose, the struts are all the same length, and the platform is parallel to and centered above the base at a specified height. The range of motion in each degree of freedom is generally symmetrical from the home position. When a hexapod is not in the home position, struts can have unequal lengths, and the range of motion about the pose is asymmetrical. Because hexapod travel is interdependent, moving the platform in one direction reduces the available travel in other directions.

1.3. The World Coordinate System

The world origin of the hexapod is at the top center of the platform in the home pose. When the hexapod moves away from the home pose, the world origin remains fixed at its original position. The World Coordinate System does not move with the platform. Refer to Figure 1-7.

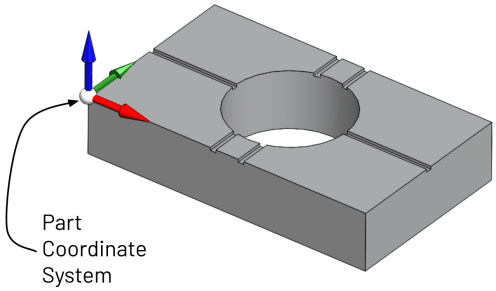
Figure 1-7: World Coordinate System



1.4. The Part Coordinate System

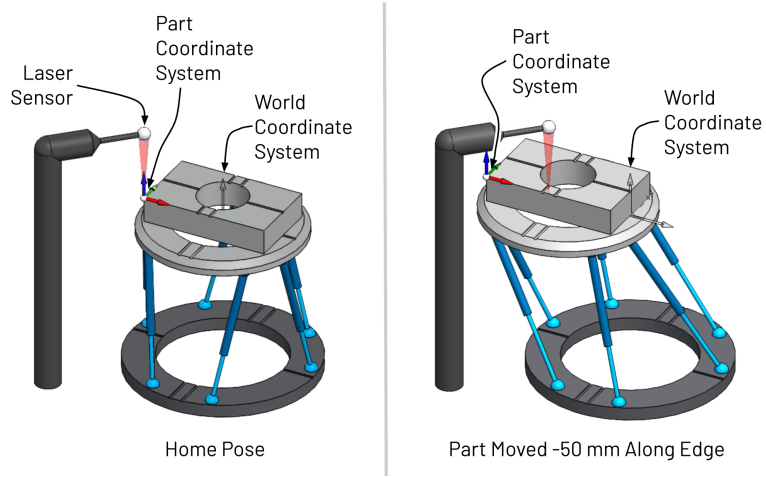
The part coordinate system is a reference frame at a point you specify on your part. In Figure 1-8, the origin is set at the lower-left corner of the top surface of the part.

Figure 1-8: Part Coordinate System



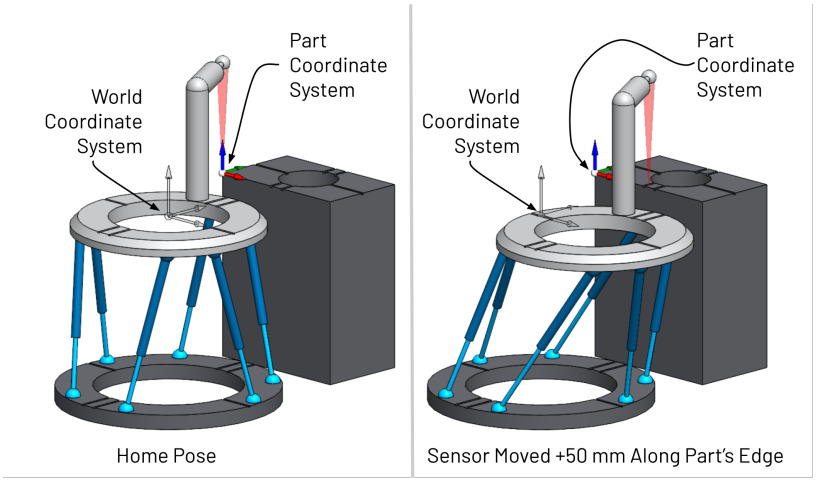
There are two types of interfaces between a part and the hexapod. First, you can mount the part to the platform of the hexapod and move it around. In this case, the part moves with the platform, so the coordinate system of the part also moves with the platform. Refer to Figure 1-9.

Figure 1-9: Part Mounted to Hexapod



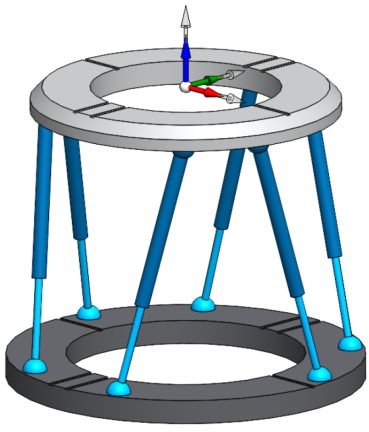
As an alternative, you can fix the part in space and mount a sensor, camera, tool, or other device to the platform and move it around the stationary part. In this case, the part coordinate system is stationary and does not move with the platform. Refer to Figure 1-10.

Figure 1-10: Sensor Mounted to Platform Interfacing with Stationary Part (Local Mode)



Immediately after homing, the part coordinate system is aligned with the world coordinate system. The commands in Chapter 2: Basic Operation let you move and orient the part coordinate system with respect to the world coordinate system.

Figure 1-11: Part Coordinate System Location after Homing



Chapter 2: Basic Operation

2.1. Operating Modes

There are two primary modes for commanding platform motion in Automation1:**Global Operating Mode** (see Section 2.6.) and **Local Operating Mode** (see Section 2.7.). In both modes, motion commands are sent to the hexapod platform through coordinate transformations.

There are two important differences between the Global and Local modes. The first difference is the effect of platform translations on the part coordinate system (refer to Section 2.5.). In Local mode, the part coordinate system is fixed in space. In Global mode, the part coordinate system moves with the platform. The second difference is how platform rotations have an effect on the direction of subsequent translations, as described in the sections that follow.

The hexapod API has a **Strut Operating Mode**. This mode is used during initial setup and troubleshooting. In this mode, commands to the hexapod axes move the struts and do not coordinate platform motion. Thus, the coordinate transformations are not active in Strut mode. The remaining content assumes users are operating in either Global or Local mode.

The hexapod API has a **Disabled Operating Mode**, where hexapod transformations are inactive. Kinematic calculations are disabled in this mode, but servo control of individual struts is possible. Commands in this mode are limited to the standard **`Enable()`** and **`Home()`** commands for the struts.

Use the commands that follow to enable the different operating modes:

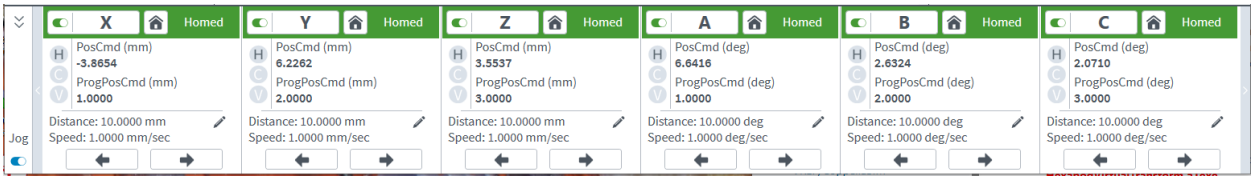
- **`EnableGlobalMode()`**
- **`EnableLocalMode()`**
- **`EnableStrutMode()`**
- **`DisableHexapod()`**

Issue these commands in the Immediate Command interface in Automation1 Studio Studio, in the **Develop** workspace in Automation1, or by using the API in a third-party programming language, like Python or .NET.

2.2. Strut Positions and Transformed Platform Position

In Automation1 Studio, the Axis Dashboard shows information about the hexapod axes. Two position values are shown, **ProgPosCmd** and **PosCmd**. Refer to Figure 2-1.

Figure 2-1: Axis Dashboard in Automation1 Studio



2.2.1. ProgPosCmd (ProgramPositionCommand): Platform Position

When operating in Global or Local modes, the **ProgPosCmd** values show the transformed platform position. The units of the **ProgPosCmd** values are in millimeters for translational axes and in degrees for rotational axes.

When in Strut or Disabled modes, transformations are inactive, so the **ProgPosCmd** values are the positions of each strut in millimeters.

2.2.2. PosCmd (PositionCommand): Strut Positions

In all operating modes, the **PosCmd** values show the position of each strut in millimeters. Strut positions are zero at the home position. Strut travel is symmetric about the home position.

2.2.3. Changing the Axis Dashboard Display

We recommend changing the **Axis Dashboard** display so that **ProgPosCmd** is the top entry (platform position) and **PosCmd** is the bottom entry (strut positions). Use the procedure that follows to change what the Axis Dashboard shows:

1. Click the gear icon in the upper-right corner of Automation1 Studio. The **Settings** screen comes into view.
2. In the selection tree on the left, select **Axis Dashboard**.
3. In the **Signal** column, select **ProgPosCmd** in the top drop-down and **PosCmd** in the bottom drop-down. Alternatively, you can set **ProgPosCmd** (platform position) as the top entry and **ProgVelCmd** (velocity) as the bottom entry to show the platform position and velocity.
4. Click the **Close** button to save your changes and to exit the screen.

2.3. Selecting between Global and Local Operating Mode

The relation of your part to the hexapod platform is how you select between Global and Local mode. Use Global mode when your part is attached to the hexapod platform and moves with it. For example, in laser processing, the part is attached to the platform, and the part origin is translated to the focal point of the laser beam at the start of the program. When Global mode is enabled, the X, Y, and Z directions rotate with changes in A, B, and C while keeping the focal point at the current programmed position.

Use Local mode when your part is fixed in space, and you are moving a sensor or device over it. In this case, the part coordinate system is stationary, and the X, Y, and Z motion directions do not change with platform rotation. For example, if a sensor positioned on the platform interacts with a fixed beam of light, rotating A, B, or C alters the angle of the sensor but not its direction of translation.

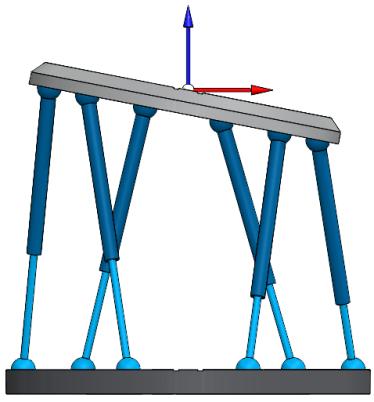
Table 2-1: Selecting Operating Modes

	Global	Local
Rotate around a fixed point in space	Yes	No
Rotate around a point attached to the platform	No	Yes
The part coordinate system rotates with A, B, and C motion	Yes	No

2.4. Center of Rotation

A common motion task is to rotate the platform around a user-defined center of rotation, or rotation point. Immediately after initialization, the default rotation point is at the world origin, which is the center of the platform at the home position.

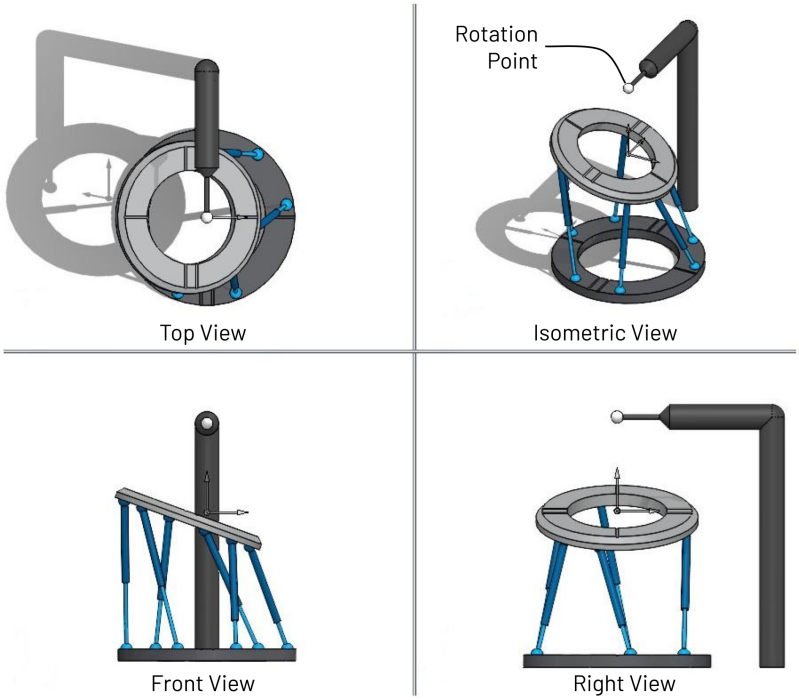
Figure 2-2: +20 Degrees Rotation in the B Direction from the Home Pose



The hexapod command **`SetRotationPoint()`** moves the rotation point relative to the world coordinate system. The syntax is **`SetRotationPoint(XOffset, YOffset, ZOffset)`**, where the offset arguments are the distance from the world origin in millimeters.

Figure 2-3 shows an example where the **SetRotationPoint()** command moved the rotation point above the center of the platform before a B-axis rotation. Note that in addition to the rotation, the platform shifted in the negative X direction because it rotated about a point high above itself.

Figure 2-3: Global Mode - +20 Degrees Rotation with a 75 mm Rotation Offset in Z Direction



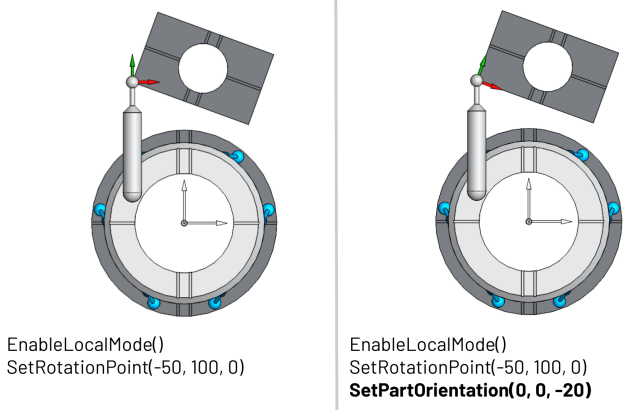
When you issue the **SetRotationPoint()** command, you also establish the origin of the part coordinate system. Motion does not occur when you issue the **SetRotationPoint()** command.

The rotational behavior of the platform and the values shown in ProgPosCmd depend on the operating mode. Refer to Section 2.6. and Section 2.7. for more information on Global and Local operating modes.

2.5. Part Coordinate System

In both Global and Local operating modes, translations occur along the coordinate system of the part. If the coordinate system of the part is not aligned to the world coordinate system at the home position, use the **SetPartOrientation()** command to align the part coordinate system with the part. The syntax is **SetPartOrientation (AOffset, BOffset, COffset)**, where the offset arguments are rotations about the world coordinate system in degrees. Refer to Figure 2-4.

Figure 2-4: SetPartOrientation() and the Part Coordinate System



2.6. Global Operating Mode

In Global operating mode, the rotational point is fixed in space, and the part coordinate system moves with the platform. The figures that follow show a part mounted to the platform (refer to Figure 2-5, Figure 2-6, and Figure 2-7). In Global mode, the part moves with the platform, and the coordinate system of the part is reoriented by rotational commands. Translational moves occur along the coordinate axes of the part.

Figure 2-5: Global Mode - Hexapod at Home, SetRotationPoint(-50, -30, 20)

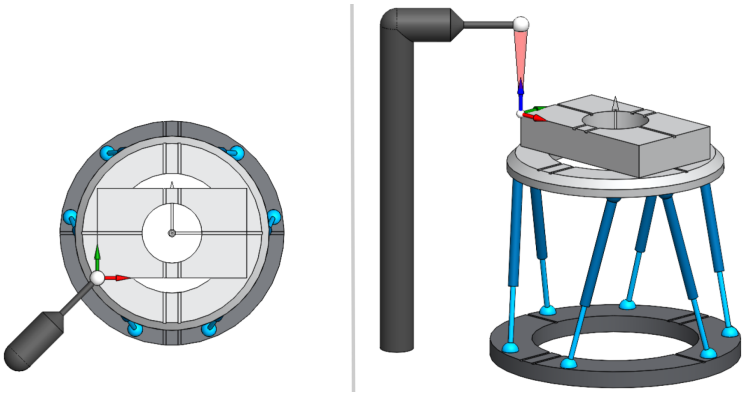


Figure 2-6: Global Mode - Rotation of C-axis by +20°, MoveIncremental(C, 20, 2)

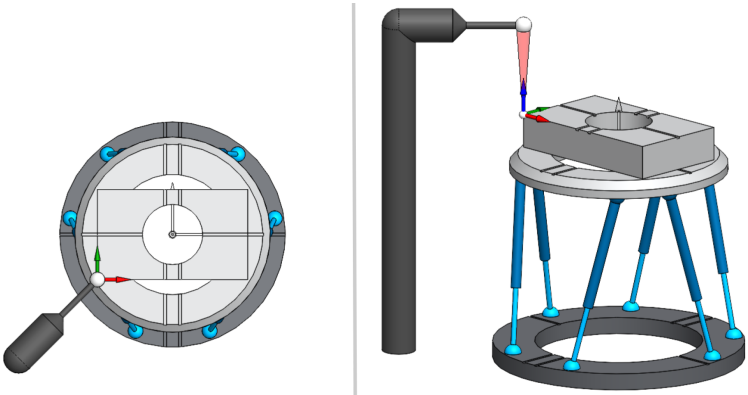
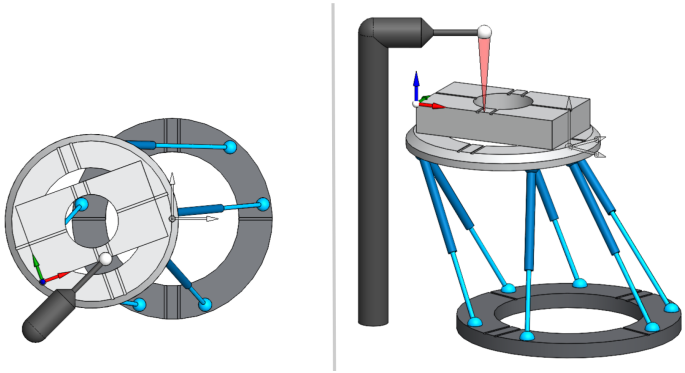
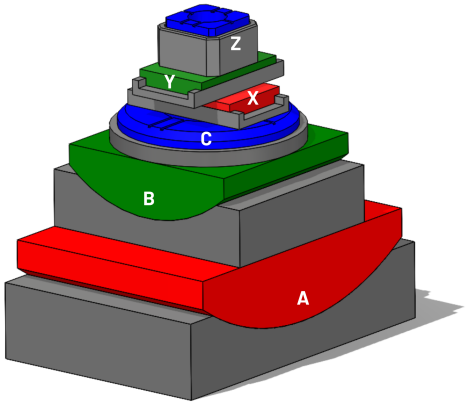


Figure 2-7: Global Mode - Translation of the X-axis by -50 mm, MoveIncremental(X, -50, 4)



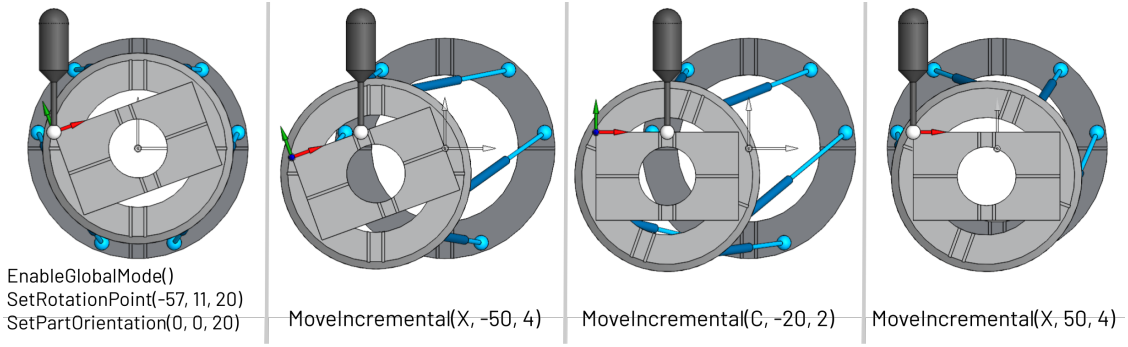
Motion in Global mode works the same as a group of stacked stages in the sequence A, B, C, X, Y, and Z. The A axis rotates the B, C, X, Y, and Z axes. The B axis rotates the C, X, Y, and Z axes. The C axis rotates the X, Y, and Z axes. Refer to Figure 2-8.

Figure 2-8: Global Mode - Stacked Stage Equivalent to Hexapod Motion



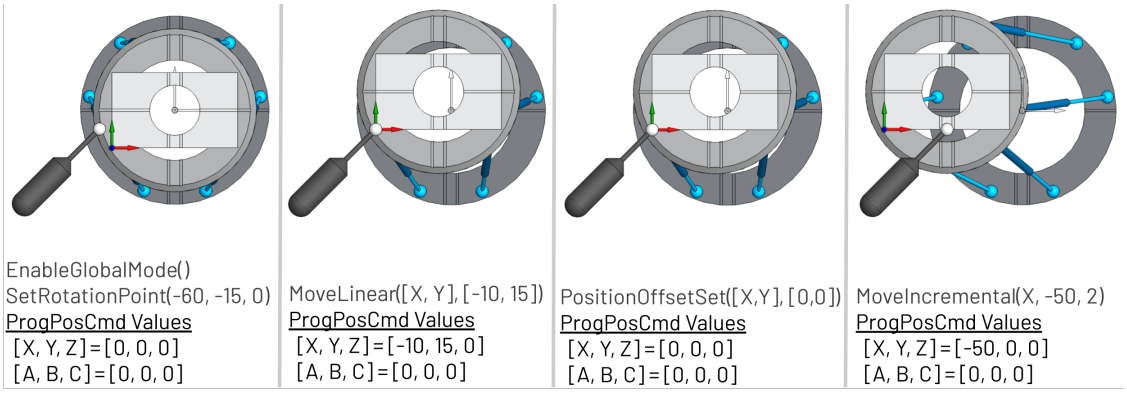
In Global mode, the **`SetPartOrientation()`** command defines the direction of motion along the X, Y, and Z axes when the A, B, and C program angles are all set to zero. The part coordinate system and direction of travel rotate in Global mode when you change the A, B, or C program angles. Refer to Figure 2-9.

Figure 2-9: Global Mode - SetPartOrientation() Rotations Change Direction of Subsequent Translations



In Global mode, the **`SetRotationPoint()`** command has no effect on the ProgPosCmd values. But, non-zero ProgPosCmd values often occur when you move the part coordinate system to the fixed rotation point. To zero the ProgPosCmd display after your part coordinate system is at the rotation point, use the standard **`PositionOffsetSet()`** command as shown in Figure 2-10. To remove position offsets, use the **`PositionOffsetClear()`** command.

Figure 2-10: PositionOffsetSet() Command in Global Mode



2.7. Local Operating Mode

In Local operating mode, the rotational point is attached to the platform, and the part coordinate system is fixed in space. Figure 2-11, Figure 2-12, and Figure 2-13 show a stationary part next to the hexapod.

In Local mode, the rotation point moves with the platform, while the part remains stationary. Thus, the coordinate system of the part is not reoriented by rotational commands. Translational moves occur along the coordinate axes of the part.

Figure 2-11: Local Mode - Hexapod at Home, SetRotationPoint(-50, 100, 0)

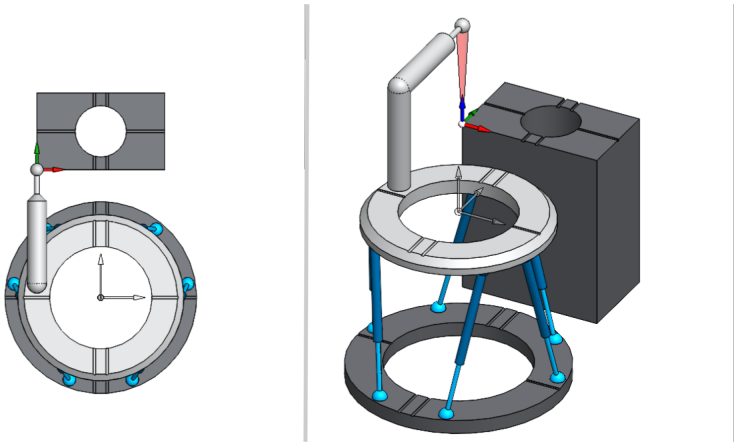


Figure 2-12: Local Mode - Rotation of C-axis by +20°, MoveIncremental(C, 20, 2)

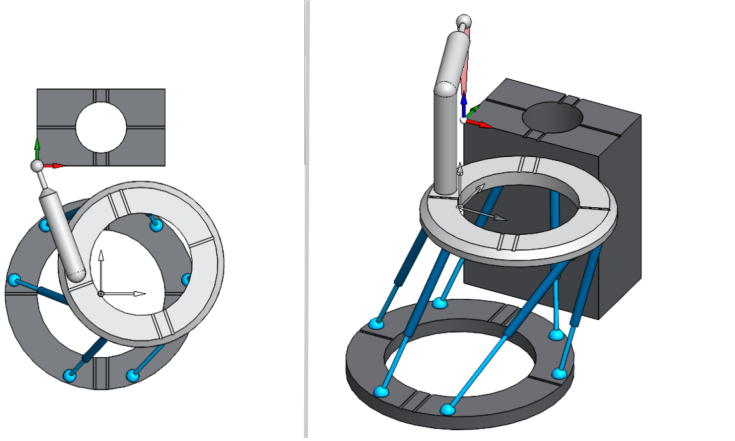
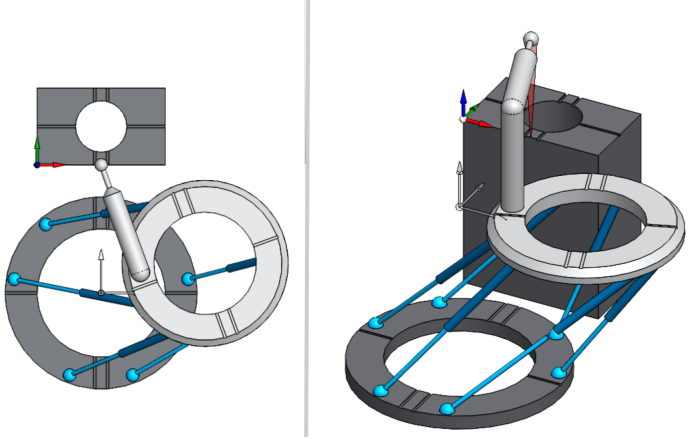
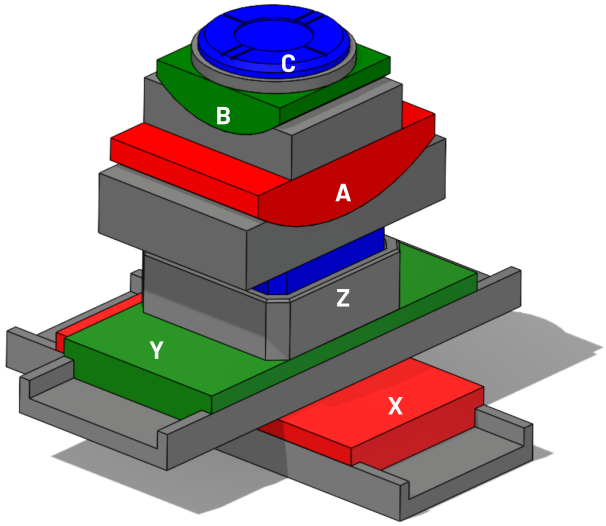


Figure 2-13: Local Mode - Translation of the X-axis by +50 mm, MoveIncremental(X, 50, 4)



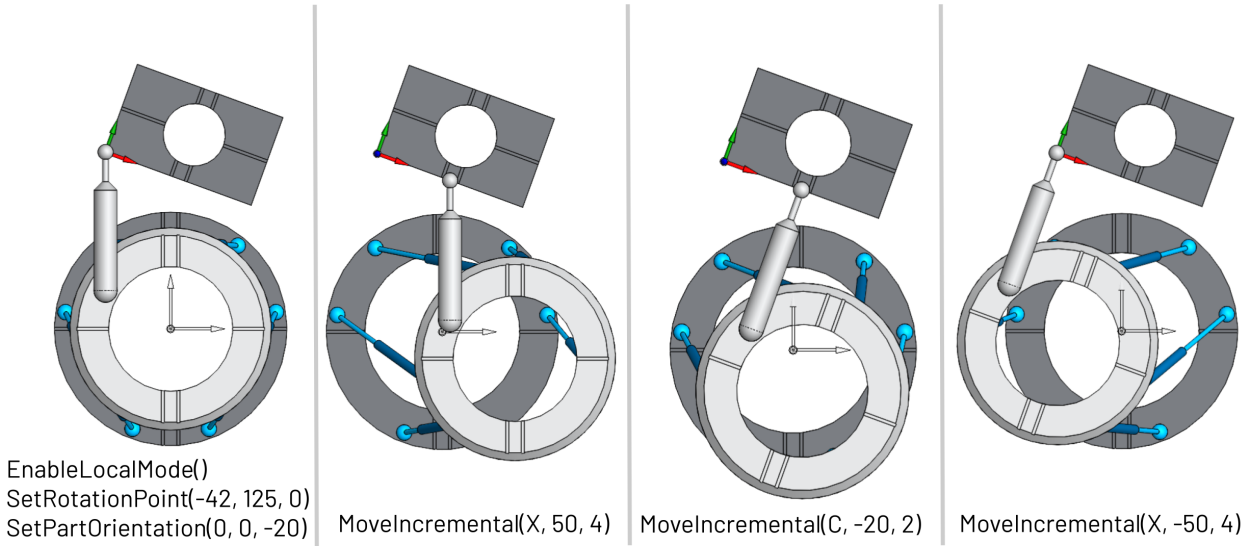
Motion in Local mode works the same as a group of stacked stages in the sequence of X, Y, Z, A, B, and C. The A axis rotates the B and C axes. The B axis rotates the C axis. Rotations of A, B, and C do not have an effect on the orientation of X, Y, and Z. Refer to Figure 2-14.

Figure 2-14: Local Mode - Stacked Stage Equivalent to Motion



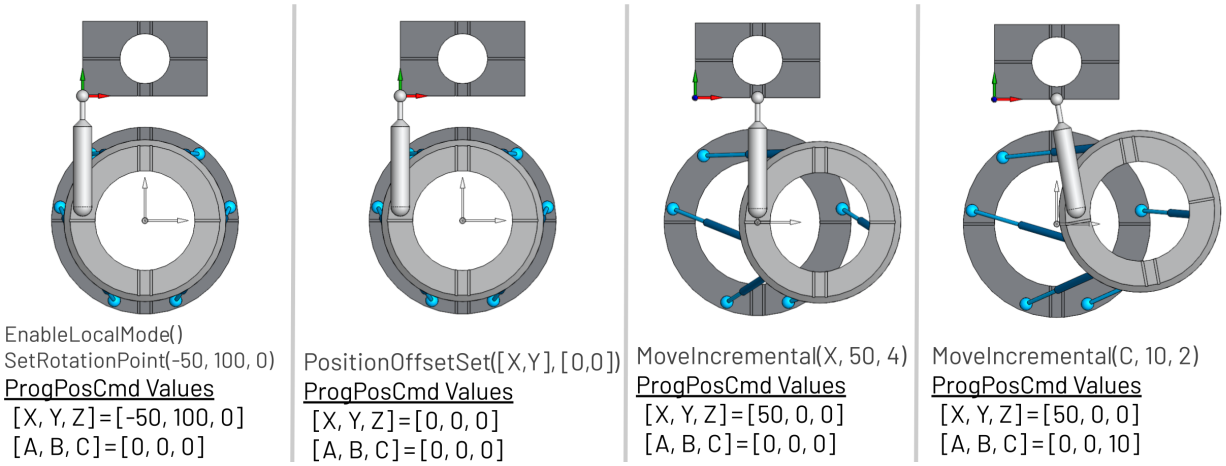
In Local operating mode, the coordinate system of the part does not reorient with rotations of A, B, or C. Thus, **`SetPartOrientation()`** defines the direction of motion for all commanded X, Y, and Z motion when in Local mode. Refer to Figure 2-15.

Figure 2-15: Local Mode - SetPartOrientation Rotations do not Change Subsequent Translations



In Local mode, the Program Position Command values are updated with the **`SetRotationPoint()`** command. To remove the offsets from the Program Position Command display, use the standard **`PositionOffsetSet()`** command as shown in Figure 2-16. To remove position offsets, use the **`PositionOffsetClear()`** command.

Figure 2-16: PositionOffsetSet() Command in Local Mode



2.8. Changing between Global and Local Operating Modes

You can change between Global and Local operating modes, but you should only do this in advanced cases. Because the part coordinate system rotates in Global mode but not in Local mode, non-zero angular positions or offsets cause the program position command values to change when you switch modes, even though motion does not occur. Established position offsets are usually not applicable in the new operating mode. You must reestablish new offset values after changing modes.

2.9. Getting Back to the Home Pose

Programming actions, such as setting rotation point offsets or switching operational modes, can cause unexpected values in the **ProgPosCmd**. When this occurs, moving the hexapod axes to absolute zero does not always get the hexapod back to home. In the home position, the struts are all at zero, which is shown by the **PosCmd** values in the Axis Manager.

The easiest way to get back to the home pose is to reset the controller. A controller reset puts the hexapod in the Disabled operating mode, clears all positional and angular offsets, and disables servo control of all struts. After you reset the controller, issue the **`EnableGlobalMode()`** or **`EnableLocalMode()`** command, depending on your application. The controller homes the hexapod with all offsets set to zero.

To move to the home pose without resetting the controller, use the procedure that follows:

1. Clear all of the offsets with the commands that follow:
   - **`SetRotationPoint(0, 0, 0)`**
   - **`SetPartOrientation(0, 0, 0)`**
   - **`PositionOffsetClear(X, Y, Z, A, B, C)`**
2. Enter Strut mode with the **`EnableStrutMode()`** command.
3. Issue one of the commands that follow to move the struts to their home positions:
   - **`Home([X, Y, Z, A, B, C])`**
   - **`MoveAbsolute([X, Y, Z, A, B, C], [0, 0, 0, 0, 0, 0], [velX, velY, velZ, velA, velB, velC])`**, where **vel** is the velocity for each axis.
4. After the motion completes and the PosCmd values are all at zero, issue the **`EnableGlobalMode()`** command or the **`EnableLocalMode()`** command, depending on your application.

2.10. Limiting Hexapod Travel

In both Local and Global modes, you can set travel limits for platform motion in the part coordinate system with the **SetHexNegativeTravelLimit()** and **SetHexPositiveTravelLimit()** commands. Activate the travel limits with the **EnableHexapodTravelLimit()** command. Deactivate the travel limits with the **DisableHexapodTravelLimit()** command.

The limit settings stay active in the controller until the user changes them or until the controller is reset. You must set both upper and lower bounds for any axis to which you want to apply motion limiting.

2.10.1. Operating Mode Requirements

Before you set and activate the hexapod travel limits, you must set the part coordinate system location and orientation with the **SetRotationPoint()** and **SetPartOrientation()** commands.

Hexapod limits are only valid in Global or Local modes. A fault occurs if you try to set the travel limits while the hexapod is in Disabled or Strut mode.

2.10.2. Safe Zone Fault

When hexapod travel limits are activated, the controller continuously monitors the commanded axis position. The controller throws a SafeZone fault when the commanded motion will exceed allowable limits. Use the Automation1 Studio fault acknowledge button or the **FaultAcknowledge()** command to clear the fault condition.

When a safe zone fault occurs, the controller stops motion before the hexapod exceeds the allowable travel range. To stop motion, the controller uses the deceleration rates defined in the Hexapod configuration file, **HexCfg.ini**.

2.10.3. Behavior Outside Limits

If an axis is already outside its travel range when limits are enabled, a safe zone fault does not occur. You can command motion towards the permitted zone without a fault. But, if you command motion away from the permitted zone, the controller triggers a SafeZone fault. Refer to the example program that follows to see how to set travel limits.

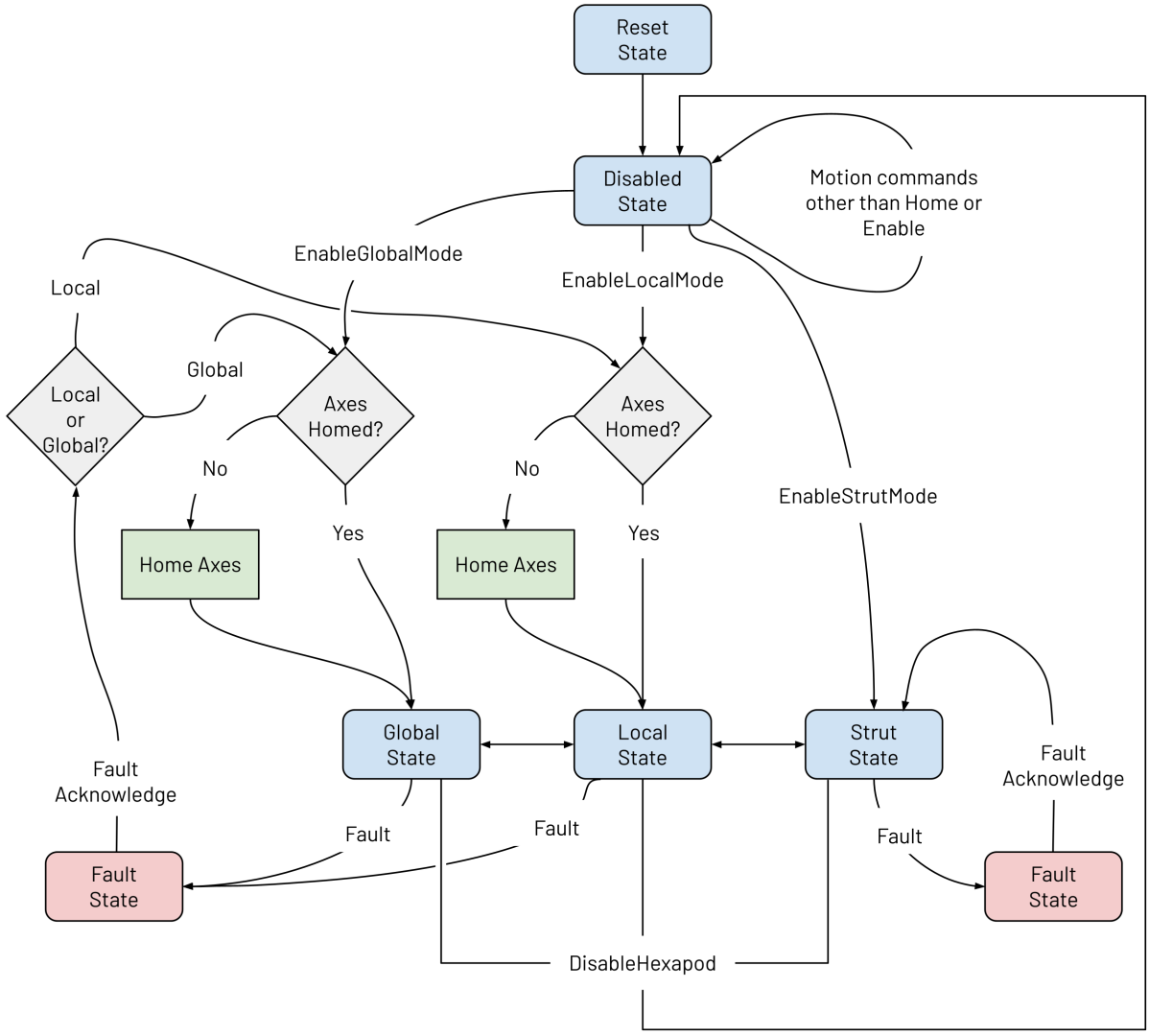
2.10.4. Example AeroScript Program for Setting Travel Limits

```
// Hexapod Travel Limit Program Example
// Declare variables
var $posLimit, $negLimit, $progPosCmd, $positionOffset as real

// Disable travel limits to allow homing
DisableHexapodTravelLimit([X, Y, Z, A, B, C])

// Clear rotation point, orientation, and position command offsets
SetRotationPoint(0, 0, 0)
SetPartOrientation(0, 0, 0)
PositionOffsetClear([X, Y, Z, A, B, C])

//Enter Local mode
EnableLocalMode()

// Set the task to program in absolute coordinates
SetupTaskTargetMode(TargetMode.Absolute)

// Move the platform to the home position
MoveLinear([X, Y, Z, A, B, C], [0, 0, 0, 0, 0, 0])

// Define the rotation point, orientation, and position offsets
// Note that these values are application-specific
SetRotationPoint(5.0, 10.0, 15.0)
SetPartOrientation(20.0, 0, 0)
PositionOffsetSet([X, Y, Z, A, B, C], [0, 0, 0, 0, 0, 0])

// Calculate the upper and lower bounds for the X axis as
// +/-2 mm from the current position.
$progPosCmd = StatusGetAxisItem(X, AxisStatusItem.ProgramPositionCommand)
$positionOffset = StatusGetAxisItem(X, AxisStatusItem.PositionOffset)
$posLimit = ($progPosCmd - $positionOffset) + 2.0
$negLimit = ($progPosCmd - $positionOffset) - 2.0

// Set the X-axis limit bounds and activate the limits
SetHexNegativeTravelLimit(X, $negLimit)
SetHexPositiveTravelLimit(X, $posLimit)
EnableHexapodTravelLimit(X)
```

2.11. The Hexapod State Machine

The AeroScript state machine program that runs in Task 2 controls the state of the hexapod. The program controls the modes of operation, starts hexapod control, and processes faults. Most of the state machine processes run in the background. Users only set the operation modes and acknowledge faults with the state machine. The state machine automatically starts after a controller reset.

Figure 2-17: Hexapod State Machine



After a controller reset, the state machine automatically moves from the Reset state to the Disabled state. In the Disabled state, you can issue **`EnableGlobalMode()`**, **`EnableLocalMode()`**, or **`EnableStrutMode()`** commands to switch states. You can also enable and home the struts using standard Automation1 functions (**`Enable()`**, **`Home()`**) and the **Axis Dashboard** buttons. Other motion commands to the hexapod will not execute while the system is in the Disabled state.

Before entering the Global or Local state, the state machine enables servo control of the struts. It then automatically homes the struts if they have not been homed yet. If any of the struts fail to home, a warning is issued, and the system enters the Disabled state.

IMPORTANT: The **`EnableGlobalMode()`** or **`EnableLocalMode()`** commands move the hexapod if the hexapod is not homed. Only issue those commands to an un-homed hexapod when it is safe for it to move.

While in the Global or Local states, the state machine monitors the servo control status of each axis. If the user disables servo control of an axis, the state machine re-enables it. To stop servo control of the struts, you must first exit the Global or Local state.

When entering the Strut state, the state machine does not enable servo control of the struts. It also does not automatically home the strut axes. In the Strut state, you must enable servo control and home the struts using standard Automation1 functionality (**`Enable()`**, **`Home()`**, and **Axis Dashboard** buttons).

You can change between Global, Local, and Strut modes by issuing the appropriate command. Motion will not occur when you change modes, but the program position command information can change.

When a fault occurs, the system enters a fault state. After you acknowledge the fault using the standard **`FaultAcknowledge()`** function or Automation1 Studio button, the controller returns to the state that was active before the fault. If the fault causes servo control to disable on any one strut, the controller moves to the Disabled state.

Chapter 3: Hexapod Library Commands

This section describes hexapod-specific AeroScript commands. Standard AeroScript commands also apply to hexapods. In Automation1 Studio, issue AeroScript commands in the immediate command window or as part of a program in the **Develop** workspace. Refer to the Automation1 AeroScript Programming Language help page to learn more about AeroScript.

Because PC-based controllers can control more than one hexapod, most of the hexapod commands in this section can take an optional index argument to specify which hexapod is being commanded. The first hexapod has an index of 0. The second hexapod has an index of 1. The hexapod index is determined by the order of the physical connection. If only one hexapod is connected, you do not need to use the optional index argument.

3.1. EnableGlobalMode()

```
EnableGlobalMode()
EnableGlobalMode(HexIndex)
```

The **EnableGlobalMode()** command enables hexapod kinematic transformations in Global operating mode. If servo control of the strut axes is disabled when **EnableGlobalMode()** is issued, the command enables servo control of the strut axes. If the hexapod is not homed, **EnableGlobalMode()** also homes the hexapod. Immediately after homing, the default rotation point coincides with the center of the top of the platform, and the **ProgPosCmd** values all read 0 in the **Axis Dashboard**.

To exit Global operating mode, do the step necessary for your application:

- Click the **Reset** button in Automation1 Studio. This exits Global mode and resets the positions of the axes.
- Issue the **EnableLocalMode()** command to change the operation from Global to Local mode. The **ProgPosCmd** values can change when going from Global to Local mode, depending on the orientation and location of the part coordinate system.
- Issue the **EnableStrutMode()** command to change the operation from Global to Strut mode. In Strut mode, the **ProgPosCmd** and the **PosCmd** values both show the strut positions in millimeters.
- Issue the **DisableHexapod()** command to exit Global mode and stop the transformation calculations. In Disabled mode, the **ProgPosCmd** and **PosCmd** values both show the strut positions in millimeters.

3.2. EnableLocalMode()

```
EnableLocalMode()
EnableLocalMode(HexIndex)
```

The **EnableLocalMode()** command enables hexapod kinematic transformations in Local operating mode. If servo control of the strut axes is disabled when you issue **EnableLocalMode()**, servo control of the strut axes is enabled. If the hexapod is not homed, **EnableLocalMode()** homes the hexapod. Immediately after homing, the default rotation point coincides with the center top of the platform, and the **ProgPosCmd** values all read 0 in the **Axis Dashboard**.

To exit Local operating mode, do the step necessary for your application:

- Click the **Reset** button in Automation1 Studio. This exits Local mode and resets the positions of the axes.
- Issue the **EnableGlobalMode()** command. The **ProgPosCmd** values can change when going from Local to Global mode, depending on the orientation and location of the part coordinate system.
- Issue the **EnableStrutMode()** command to change the operation from Local to Strut mode. The **ProgPosCmd** and **PosCmd** values both show the strut positions in millimeters.
- The **DisableHexapod()** command also exits Local mode and stops the transformation calculations. The **ProgPosCmd** and **PosCmd** values both show the strut positions in millimeters in Disabled mode.

3.3. EnableStrutMode()

```
EnableStrutMode()
EnableStrutMode(HexIndex)
```

The **EnableStrutMode()** command enables direct control of the strut lengths. This mode is for initial setup, optimization, and troubleshooting.

When strut mode is enabled, the program position command values show the position of each strut within its travel. The units are millimeters, even though the A, B, and C axes still show them in degrees.

When you issue **EnableStrutMode()**, strut servo control is not automatically engaged. The **EnableStrutMode()** command also does not automatically home the hexapod. Use the standard enable and home functionality to enable servo control and home the individual struts.

Exit Strut mode by resetting the controller or issuing the **EnableGlobal()**, **EnableLocal()**, or **DisableHexapod()** commands.

3.4. DisableHexapod()

```
DisableHexapod()
DisableHexapod(HexIndex)
```

Use the **DisableHexapod()** command to stop hexapod kinematic transformation calculations. Note that **DisableHexapod()** does not disable servo control of the struts. If you want to disable servo control, you must use the buttons in the **Axis Dashboard** or the standard **Disable()** AeroScript function after issuing the **DisableHexapod()** command.

IMPORTANT: When the hexapod is in Disabled mode, you can only enable and home the struts. If you try to command other types of motion, a fault occurs. To command motion, the hexapod must be in Global, Local, or Strut mode.

3.5. SetRotationPoint()

```
SetRotationPoint(Xoff, Yoff, Zoff)
SetRotationPoint(HexIndex,Xoff, Yoff, Zoff)
```

Use the **SetRotationPoint()** command to define the location of the rotation point for the specified hexapod. The offsets are from the world coordinate system, located at the top middle of the hexapod platform at the home pose. The units are millimeters.

You can issue the **SetRotationPoint()** command during any operating state (Local, Global, Strut, or Disabled). When in Local mode, the Xoff, Yoff, and Zoff values are added to the X, Y, and Z values of the program position command (**ProgPosCmd**). You can use the **PositionOffsetSet()** command to remove the **SetRotationPoint** values from the program position command display.

3.6. GetRotationPoint()

```
GetRotationPoint(Xoff, Yoff, Zoff)
GetRotationPoint(HexIndex, Xoff, Yoff, Zoff)
```

Use the **GetRotationPoint()** command to return the position of the rotation point through the **Xoff**, **Yoff**, and **Zoff** values. The values represent the location of the rotation point in the world coordinate system when the hexapod is in the home pose. The return values are real numbers.

3.7. SetPartOrientation()

```
SetPartOrientation(Aoff, Boff, Coff)
SetPartOrientation(HexIndex,Aoff, Boff, Coff)
```

Use the **SetPartOrientation()** command to define the orientation of the part coordinate system for the specified hexapod. Offset arguments are rotations about the world coordinate system in degrees. Rotations obey the Bryant Angle convention in the sequence of A-B-C, where A rotates B and C, and then B rotates C.

The **SetPartOrientation()** command can be issued in all operating states (Local, Global, Strut, or Disabled).

3.8. GetPartOrientation()

```
GetPartOrientation(Aoff, Boff, Coff)
GetPartOrientation(HexIndex, Aoff, Boff, Coff)
```

Use the **GetPartOrientation()** command to return the angular orientation of the part coordinate system relative to the world coordinate system through the **Aoff**, **Boff**, and **Coff** values. The values represent the orientation of the part coordinate system in the world coordinate system when the hexapod is in the home pose. The return values are real numbers.

3.9. GetHexapodMode()

```
GetHexapodMode()
GetHexapodMode(HexIndex)
```

Use the **GetHexapodMode()** command to get the current hexapod mode. The return values are:

0 = Disabled Mode

1 = Local Mode

2 = Global Mode

3 = Strut Mode

When the orientation and rotation point locations change on an enabled hexapod, the reported mode momentarily changes. This is because the hexapod is temporarily disabled during the offset change and then automatically re-enables after the change completes.

3.10. GetHexapodState()

```
GetHexapodState()
GetHexapodState(HexIndex)
```

Use the **GetHexapodState()** command to get the current operating state of the hexapod. The return values are as follows:

0 = Reset State

This state is active immediately after the controller is reset. The state machine automatically transitions from the Reset state to the Disabled state after a few moments.

1 = Enabled State

Either the Global or the Local operating mode is active.

2 = Fault Acknowledge State

The hexapod has a fault condition and is waiting for a Fault Acknowledge action from the user.

3 = Disabled State

Transformation calculations are inactive. In the Disabled state, you can enable servo control of individual struts with the standard **Enable()** command. You can also start hexapod control modes using the **EnableGlobalMode()**, **EnableLocalMode()**, or **EnableStrutMode()** commands.

4 = Strut State

Each strut is controlled independently. This mode is used for startup or to diagnose problems with the individual hexapod struts.

3.11. $HexTask

```
$HexTask[HexIndex]
```

Use **$HexTask[]** to get or set the task associated with the hexapod specified by **$HexIndex[0]** or **$HexIndex[1]**. The default task index for each hexapod is defined in the **HexCfg.ini** file, located in the controller file system. The state machine associates hexapod faults with the task set by **$HexTask[]**.

3.12. $HexapodCount

```
$HexapodCount
```

$HexapodCount returns the number of hexapods defined in the **HexCfg.ini** file. This is a read-only property.

3.13. WaitForHexapodAxes()

```
WaitForHexapodAxes()
WaitForHexapodAxes(HexIndex)
```

Use **WaitForHexapodAxes()** to wait for commanded motion on the axes of the specified hexapod to complete before moving to the next program line.

3.14. GetHexapodTransformVersion()

```
GetHexapodTransformVersion() as String
```

Use **GetHexapodTransformVersion()** to return the current hexapod software version as a string. For example:

```
var $version as string
$version = GetHexapodTransformVersion()
```

3.15. SetHexNegativeTravelLimit()

```
SetHexNegativeTravelLimit(axis, limit)
SetHexNegativeTravelLimit(HexIndex, axis, limit)
SetHexNegativeTravelLimit(axes[], limits[])
SetHexNegativeTravelLimit(HexIndex, axes[], limits[])
```

Use the **SetHexNegativeTravelLimit()** command to set the lower bound of allowable travel for the specified axis. The axis argument specifies a single axis for the limit. The **axes[]** argument specifies an array of axes, and a corresponding array of **limits[]** must be entered.

The rules that follow apply when using this command:

- The limit value must be a real number.
- The limit value specifies the minimum allowable Program Position Command for the axis.
- Offset values set with the **PositionOffsetSet()** command do not have an effect on the limit value. See Section 2.10.4. for an example program.
- You must also specify an upper limit with **SetHexPositiveTravelLimit()** before you can activate a limited travel zone.
- The upper limit of the travel zone must be greater than the lower limit.
- Global or Local mode must be active when the **SetHexNegativeTravelLimit()** command is issued. Otherwise, a fault will occur.
- The limit value stays the same until it is changed with the **SetHexNegativeTravelLimit()** command or until the controller is reset.

3.16. SetHexPositiveTravelLimit()

```
SetHexPositiveTravelLimit(axis, limit)
SetHexPositiveTravelLimit(HexIndex, axis, limit)
SetHexPositiveTravelLimit(axes[], limits[])
SetHexPositiveTravelLimit(HexIndex, axes[], limits[])
```

Use the **SetHexPositiveTravelLimit()** command to set the upper bound of allowable travel for the specified axis. The axis argument specifies a single axis for the limit. The **axes[]** argument specifies an array of axes, and a corresponding array of **limits[]** must be entered.

The rules that follow apply when using this command:

- The limit value must be a real number.
- The limit value specifies the maximum allowable Program Position Command for the axis.
- Offset values set with the **PositionOffsetSet()** command do not have an effect on the limit value. See Section 2.10.4. for an example.

- You must also specify a lower bound using **SetHexNegativeTravelLimit()** before you can activate a limited travel zone.
- The upper limit of the travel zone must be greater than the lower limit.
- Global or Local mode must be active when the **SetHexPositiveTravelLimit()** command is issued. Otherwise, a fault will occur.
- The limit value persists until it is changed with the **SetHexPositiveTravelLimit()** command or the controller is reset.

3.17. EnableHexapodTravelLimit()

```
EnableHexapodTravelLimit(axis)
EnableHexapodTravelLimit(HexIndex, axis)
EnableHexapodTravelLimit(axes[])
EnableHexapodTravelLimit(HexIndex, axes[])
```

Use the **EnableHexapodTravelLimit()** command to activate a limited travel zone. You must define the travel limits using **SetHexNegativeTravelLimit()** and **SetHexPositiveTravelLimit()** before using the **EnableHexapodTravelLimit()** command.

A safe zone fault occurs when commanded motion exceeds the travel limits. The controller decelerates all hexapod axes to zero velocity when a safe zone fault occurs.

If an axis is already outside its allowable travel range when limits are enabled, a safe zone fault does not occur. You can command motion towards the allowable zone without a fault. However, if you command motion away from the allowable zone, the controller triggers a SafeZone fault.

3.18. DisableHexapodTravelLimit()

```
DisableHexapodTravelLimit(axis)
DisableHexapodTravelLimit(HexIndex, axis)
DisableHexapodTravelLimit(axes[])
DisableHexapodTravelLimit(HexIndex, axes[])
```

Use the **DisableHexapodTravelLimit()** command to deactivate the restricted travel zone for the axis. When this command is issued, limit values stay in memory until you change them with the **SetHexNegativeTravelLimit()** and **SetHexPositiveTravelLimit()** commands or until you reset the controller.

Chapter 4: Programming Motion

In Automation1 Studio, use the **Develop** workspace to create, edit, and run motion programs in the AeroScript language.

Automation1 lets you run programs across multiple tasks at the same time, and hexapod control also uses this feature. With hexapod control, Task 1 is typically for programming, and Task 2 is only for running the state machine program.

The AeroScript program at the end of this section shows best practices for initializing and then commanding a hexapod. Refer to Section 4.1.

IMPORTANT: The commanded operation mode, rotation point, and position offsets stay active until the controller is reset or until the user changes them. If you change offsets and modes in a program, always set the operational offsets at the start of the program to make sure that the hexapod starts from a known operating state. The example program in this section (Section 4.1.) uses this best practice and sets the offsets and operating modes at the start of the program.

4.1. Example Program: Hexapod Programming

```
// Hexapod programming example

// ***************************************************************
// Start the main program block
program
	// Stop any hexapod transformation calculations that may be active
	DisableHexapod()

	// Enable servo control of the strut axes
	Enable([X, Y, Z, A, B, C])

	// Set the task to program in absolute coordinates
	SetupTaskTargetMode(TargetMode
	Absolute)

	// Clear any position offsets that may be active
	PositionOffsetClear([X, Y, Z, A, B, C])

	// Set the part orientation and rotation point to zero.
	// This causes subsequent motion to align with the world coordinate
	// system and places the part coordinate system coincident
	// with the world coordinate system
	SetRotationPoint(0, 0, 0)
	SetPartOrientation(0, 0, 0)

	// Enter Local operational mode. If any of the struts have not
	// been homed, the controller will home the hexapod.
	EnableLocalMode()

	// Move the hexapod to the starting point. For example,
	// this move might place a tool on the platform at the origin of a
	// stationary part. The position values are determined
	// by the user based on the machine configuration and
	// the work being performed.
	MoveLinear([X, Y, Z, AB, C], [-47.935, 81.014, 52.655, 0, 0, 0])

	// Set the location of the rotation point and the origin
	// of the part coordinate system within the world
	// coordinate system.
```

```
        SetRotationPoint(-75, -60, 100)


        // Set the orientation of the part coordinate system.
        // Angular offsets, when required, are determined by the user.
        // Bryant Angles: A rotates B & C, and then B rotates C.
        SetPartOrientation(10, 20, 30)

        // At this point, the program position command is the
        // location of both the rotation point and the origin of the part
        // coordinate within the world coordinate system. Set position
        // offsets to zero so that the program position represents the
        // relative difference between the rotation point and the part
        // coordinate system.
        PositionOffsetSet([X, Y, Z, A, B, C], [0, 0, 0, 0, 0, 0])

        // Set the vector speed to 5 mm/s for coordinated moves
        // (i.e., MoveLinear (G1), MoveCw (G2), and MoveCcw (G3) commands)
        SetupCoordinatedSpeed(5)

        // Perform various coordinated moves.
        // Rotate A 5 degrees about the rotation point.
        MoveLinear(A, 5)
        // Rotate B 3 degrees about the rotation point.
        MoveLinear(B, 3)
        // Rotate C 1 degree about the rotation point.
        MoveLinear(C, 1)
        // Translate the part within the part coordinate frame.
        MoveLinear([X, Y, Z], [5, 6, 7])
        // Move the back to the start position.
        MoveLinear([X, Y, Z, A, B, C],[0, 0, 0, 0, 0, 0])
end
```

Appendix A: Revision History

Revision	Description
2.01	General update with minor edits
2.00	Full revision from matrix transformations to C transformations in Automation1 Software version 2.11.
1.01	Full revision
1.00	New Manual

This page intentionally left blank.