COGNEX

AlignPlus® 4.3

User Manual

2020 December 16

Legal Notices

The software described in this document is furnished under license, and may be used or copied only in accordance with the terms of such license and with the inclusion of the copyright notice shown on this page. Neither the software, this document, nor any copies thereof may be provided to, or otherwise made available to, anyone other than the licensee. Title to, and ownership of, this software remains with Cognex Corporation or its licensor. Cognex Corporation assumes no responsibility for the use or reliability of its software on equipment that is not supplied by Cognex Corporation. Cognex Corporation makes no warranties, either express or implied, regarding the described software, its merchantability, non-infringement or its fitness for any particular purpose.

The information in this document is subject to change without notice and should not be construed as a commitment by Cognex Corporation. Cognex Corporation is not responsible for any errors that may be present in either this document or the associated software.

Companies, names, and data used in examples herein are fictitious unless otherwise noted. No part of this document may be reproduced or transmitted in any form or by any means, electronic or mechanical, for any purpose, nor transferred to any other media or language without the written permission of Cognex Corporation.

Portions of the hardware and software provided by Cognex may be covered by one or more U.S. and foreign patents, as well as pending U.S. and foreign patents listed on the Cognex web site at: cognex.com/patents.

The following are registered trademarks of Cognex Corporation:

Cognex, 2DMAX, Advantage, AlignPlus, Assemblyplus, Check it with Checker, Checker, Cognex Vision for Industry, Cognex VSOC, CVL, DataMan, DisplayInspect, DVT, EasyBuilder, Hotbars, IDMax, In-Sight, Laser Killer, MVS-8000, OmniView, PatFind, PatFlex, PatInspect, PatMax, PatQuick, SensorView, SmartView, SmartAdvisor, SmartLearn, UltraLight, Vision Solutions, VisionPro, VisionView

The following are trademarks of Cognex Corporation:

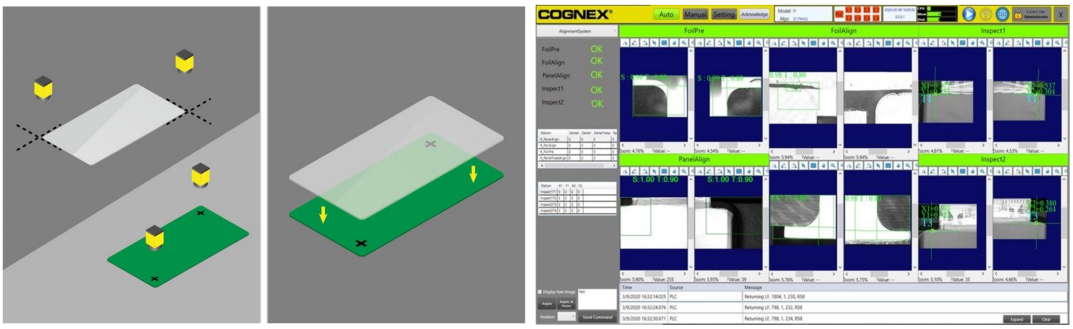
The Cognex logo, 1DMax, 3D-Locate, 3DMax, BGAII, CheckPoint, Cognex VSoC, CVC-1000, FFD, iLearn, In-Sight (design insignia with cross-hairs), In-Sight 2000, InspectEdge, Inspection Designer, MVS, NotchMax, OCRMax, PatMax RedLine, ProofRead, SmartSync, ProfilePlus, SmartDisplay, SmartSystem, SMD4, VisiFlex, Xpand

Other product and company trademarks identified herein are the trademarks of their respective owners.

Table of Contents

AlignPlus Overview

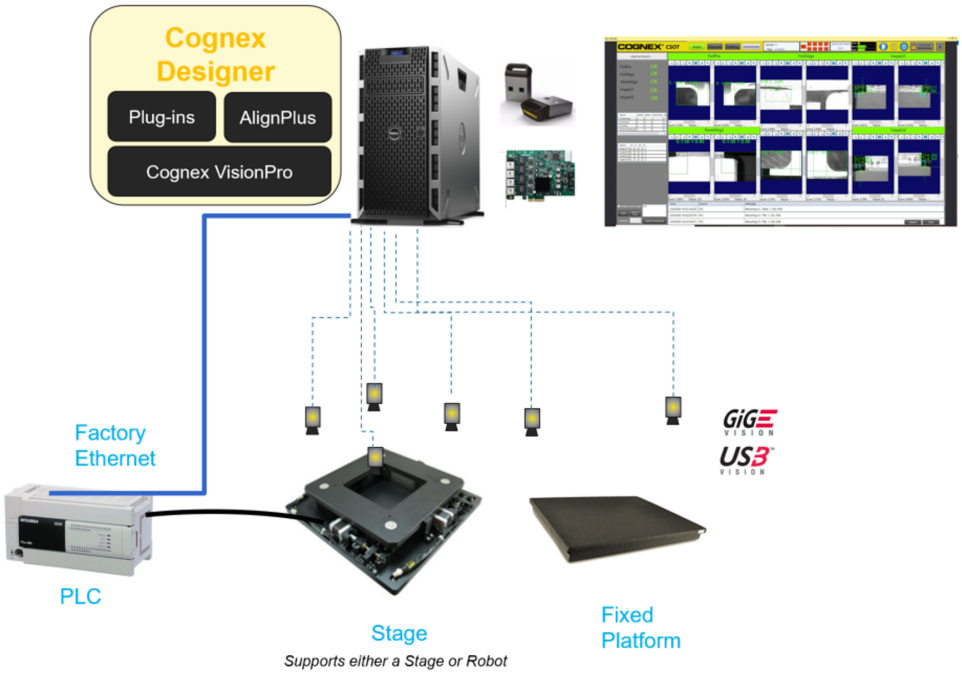
AlignPlus 4.3.0 is specifically designed for making it easy to develop & deploy multi-station, multi-camera 2D vision guided alignment applications. It provides a convenient graphical configuration that is flexible to handle a wide range of machine designs. Based on the user configuration selections, a deployment ready vision application is self-generated. For further application customization, user editable Cognex Designer project is also provided.



The automatically generated application consists of the following key elements:

- Vision Tasks
  - Calibration, Train time and Run Time tasks
- Pose Computing Tasks
  - Calculates the required motion to align or assemble part to the target pose
- Operator Interface
  - Manual trigger buttons
  - Setup & Run Time User Interfaces
- Common Alignment Functions
  - L-Check, Offset Compensation, Manual Alignment, Motion System Analysis
- System Utilities
  - Recipe management, User Management, Multi-Language, Data Logging, Image Saving
- Communication interface
  - TCP/IP commands for interfacing with external controllers such as PLCs and Robot controllers

In addition to cutting down project development time, the added benefit of the auto-generated project is standardization across different machine designs and projects. This makes it easier to hand-over & maintain projects between developers and between field engineers.



For OEMs who prefer to develop their own project from scratch, but want to utilize powerful alignment functionality, AlignPlus also provides calibration, feature finding and pose computing components that can be included in the OEM's Cognex Designer projects.

Release Info

About This Release

AlignPlus 4.3.0 introduces many new features. This release works with VisionPro 9.6 SR2 and Cognex Designer 4.3.2

Supported Operating Systems

AlignPlus 4.3.0 supports development and deployment on single or multiprocessor machines using native languages (English, Japanese, German, Korean, and Simplified Chinese) on a variety of Windows 64-bit operating systems.

- Windows 10 and Windows 10 IoT Enterprise
- Windows 7 Premium, Professional and Ultimate (Service Pack 1)

Supported Visual Studio Environments For Plugins Development

Designer 4.3.2 and AlignPlus 4.3.0 support development and deploying using the following compilers that target the .NET Framework version 4.7.2:

- Microsoft Visual Studio 2015 (v. 14.0), Update 3
- Microsoft Visual Studio 2017 (v. 15.0)

Security Key Firmware

AlignPlus uses a USB security key attached to your computer to ensure the software is properly authorized for use. This release supports firmware version 4.30 for AlignPlus security keys.

If you are using a security key with a previous version of firmware, contact Cognex Support for assistance in upgrading to the latest version. Keep your security key firmware up to date to take advantage of the latest features and improvements to Cognex software security.

Security License

Security License should include the following part:

1. VisionPro License

2. Cognex Designer License

3. AlignPlus License

The needed VisionPro and Designer licenses are included with the AlignPlus Products.

New Features

1. Mult-Part Pickup

   Supports single camera and single acquisition applications.

2. Inspection

   Supports measurement and gauging applications

3. One Time Calibration
4. Using Camera Center as Golden Pose
5. Independent Calibration Plate Configuration for different stations
6. Supports image acquisition from Basler USB 3.0 cameras, Hikvision GigE and USB 3.0 cameras
7. Documentation about AlignPlus Concept, user manual and communication protocol. (Complete documents for new features above will be released in next A+ release)

Improved Features

1. Enhanced manual trigger button for auto calibration
2. Enhanced recipe management for safety check
3. Improved layout of navigation tree
4. Added more exception messages for user to identify acquisition issues
5. Improved L-Check usability
6. Improved Offset compensation usability
7. Mechanism to avoid rerunning a on-going deployed application
8. Added ACB, AC commands to exported command sequence
9. Added a warning when left-handed coordinate is detected after hand-eye calibration

Known Issues in This Release

1. Issues related to calibration when separate calibration plates are used

   Cross calibration computation error.

   Hand-eye calibration result could be unreliable.
2. Program regeneration with wizard configuration changes takes as long as 5-10 minutes.
3. Part based calibration accuracy could be lowered when using a sparse/single feature tracking with an inaccurate motion device.
4. Feature finding task cannot run properly when feature finder component is configured to connect to more than one calibration components in configuration wizard.
5. Switching to display page whose live mode is on takes long time and could lead to an application freeze.

Fixed Issues

1. LFGP Command execution error: alignment task is executed before feature finding task finishes execution
2. GP command is supposed to run synchronously but runs asynchronously
3. Camera enable/disable status for feature finding task sometimes is not updated during recipe change.
4. ACB command execution error when real part is used for a vision guided hand-eye calibration.

Additional Documentation Online link

3rd Party Acquisition Development SDK

TCP/IP Commands Changes

New commands add in this release:

Command Key	Application	Description
MEA	AOI	Run inspection task synchronously after feature fining
MEAA	AOI	Run inspection task asynchronously after feature fining
LFMEA	AOI	Run feature fining task first, and then inspection task synchronously
MGP	Multiple Parts Pickup	Run multiple part alignment task synchronously after feature fining
MGPA	Multiple Parts Pickup	Run multiple part alignment task asynchronously after feature fining
LFMGP	Multiple Parts Pickup	Run feature fining task first, and then multiple part alignment task synchronously

Release History

Software Environment Support:

AlignPlus	VisionPro	Cognex Designer
3.0	9.1 SR1/SR2	2.6
4.0	9.2	3.0
4.0.1	9.5 SR1	4.2
4.2.0	9.6	4.3.1

AlignPlus 4.0

New functionality added in AlignPlus 4.0 include:

- Camera Simulator
- Screenshot Saving
- Part-based Hand-eye Calibration
- Part-based Cross Calibration
- Manual Calibration
- Disk Cleanup

Known Issues

1. Operation in features finder sometimes froze A+ program that lead to task forceful close.

2. CDB image player sometimes cannot play saved CDB files.

Fixed Issues

1. Error messages are very difficult for operators to understand.
2. Cognex Designer crashes by opening PatMax "All setting".
3. In runtime user interface, no recipe saving reminder before closing the application if there is any change in recipes.

AlignPlus 4.0.1

New functionality added in AlignPlus® 4.0.1 include:

- Improved Operator Interface include HMI customization and tree view navigation
- User Interface to manual set a feature finder Origin
- Support for live mode
- Camera connection status indicators
- Camera Position Adjustment in Home2D after Calibration to avoid re-calibration if camera shuttled for different work size
- Improved image playback and image archiving
- Enhanced logging features
- Camera acquisition disable/enable function in feature finding task.

Known Issues

1. Program developed in older version cannot run in this version.
2. Configuration wizard rerun compatibility issue:
    - Changes in tasks are overwritten
    - Some page layouts disposition
    - Calibration and alignment recipes cannot be loaded.

Fixed Issues

1. Feature finding tool block often causes processing time increase.
2. Memory increase when $LogData function is frequently called.
3. Recipe loading/unloading takes more than 1 minute during test mode which makes debugging in field too time-consuming.
4. Custom feature finder hang and crash occasionally in test mode.

AlignPlus 4.2.0

New functionality added in AlignPlus® 4.2 include:

1. Enhanced Recipe Management
2. Master Image Display
3. Exporting and Importing Setup wizard Configuration.
4. Manual Control Buttons
5. Many side functions including:
    - L-Check
    - Enhanced Image Saving and Playback
    - Motion Analysis
    - Manual trigger buttons
    - Offset Compensation
    - Placement Limit Check
    - Camera Simulator Advanced Mode
6. Localization
7. Hand-eye calibration result judgment and moving path graphic display

Fixed Issues

1. Recipe corruption after IPC is cut off electricity while AlignPlus program was running.
2. Image Corrector's cycle time increases by 2-3 times after user reconfigures scripting in custom tool block feature finder.
3. AlignPlus program developed under older version cannot run in new version.

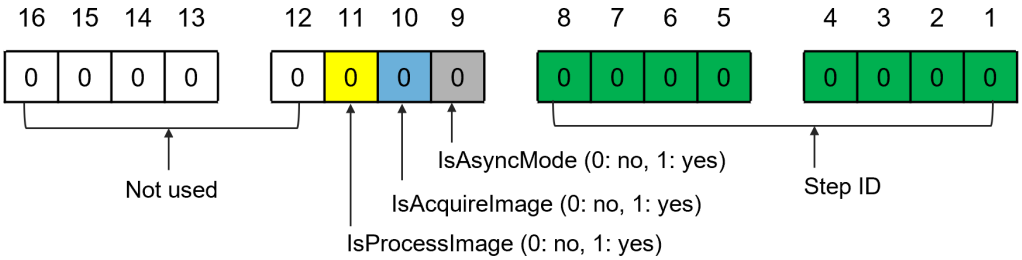
4. Configuration wizard rerun compatibility issue:
    - Changes in tasks are overwritten
    - Some page layouts disposition
    - Calibration and alignment recipes cannot be loaded.
5. Camera reconnection issue

TCP/IP Commands Changes

1. New commands added in this release

Command Key	Application	Description
LFA	Alignment/Assembly	Run feature fining task in asynchronous mode
GPA	Alignment/Assembly	Run pose computer task in asynchronous mode
LFGP	Alignment	Run feature fining task first, and then pose computer task synchronously

2. EncodedID definition is changed as below
    - EncodedID in A+ 4.0.1



    - EncodedID in this release

Execution Mode	EncodedID
Acquire Only	StepID + 1000
Acquire and Process	StepID
Process Only	StepID + 2000

Installation

Install Steps

To install AlignPlus 4.3, please install the following programs following the displayed instructions.

1. Install VisionPro 9.6 SR2. Follow the instructions on the wizard screens.
2. Install Cognex Designer 4.3.2. Follow the instructions on the wizard screens.
3. Install AlignPlus 4.3.

For the most up-to-date installation file, please refer to the Cognex online support site:

VisionPro: https://support.cognex.com/en/downloads/visionpro

Cognex Designer: https://support.cognex.com/en/downloads/cognex-designer

AlignPlus: https://support.cognex.com/en/downloads/alignplus

External plugin installation

For external plugin installation, follow the guidance of Cognex Designer user manual:

In order for a plugin to be usable by a Designer application in development or test mode, it must meet the following requirements:

All of the .DLL files that implement the plugin must be stored in a single plugin directory within %ProgramData%\Cognex\Designer\Plugins\ on the PC upon which you are running Designer.

The directory that contains the plugin .DLL files must also contain a Plugin Configuration file that describes the characteristics and compatibility information for the plugin.

Cognex recommends that you adopt the following convention for naming and arranging plugin directories within %ProgramData%\Cognex\Designer\Plugins\:

%ProgramData%\Cognex\Designer\Plugins_company_name_\ _plugin_name__version_ \ _architecture_ \

where:

- _company_name_ is the name of the company that created the plugin.
- _plugin_name_ is the name of the plugin.
- _version_ is the version of the plugin
- _architecture_ is Any, x86, or x64

Using this naming scheme allows you to manage the concurrent installation of multiple versions of a given plugin.

Note: The use of this naming scheme is optional as long as the two requirements listed above are met. Whenever you create a new project in Designer, Designer will search all directories within %ProgramData%\Cognex\Designer\Plugins\. Any plugins within the directory, or any of its sub-directories, will be added to the project's list of plugin references.

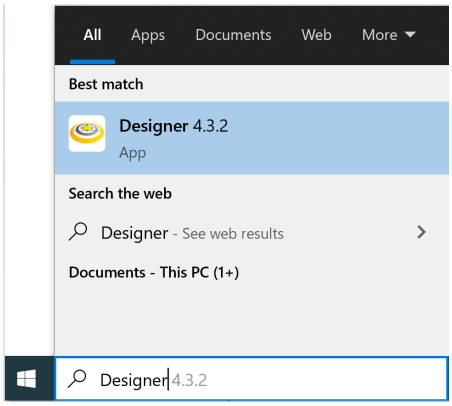
For more information about plugin, please refer to **Cognex Designer User Manual/How To.../How To... Plugins/Working with Plugins**.

Create an AlignPlus Project

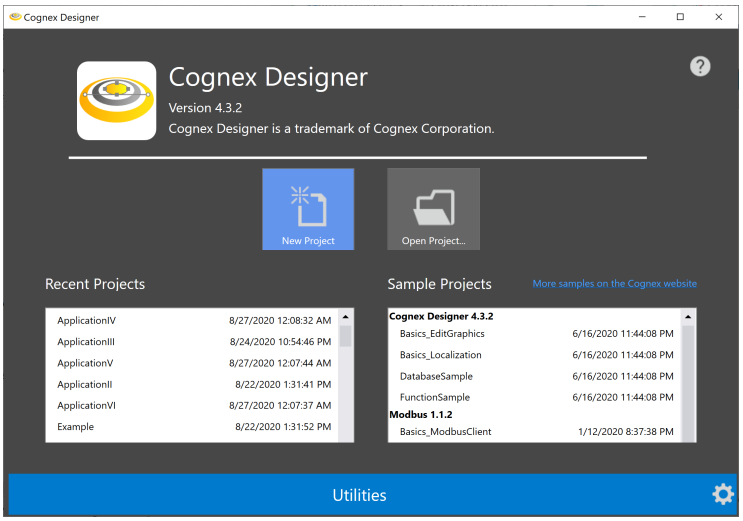
Create an Empty Project

The first step to create an AlignPlus project is to create an empty AlignPlus project:

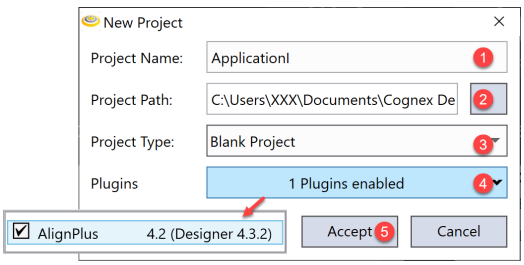
1. Type "Designer" in command window and select the suggested Cognex Designer application.



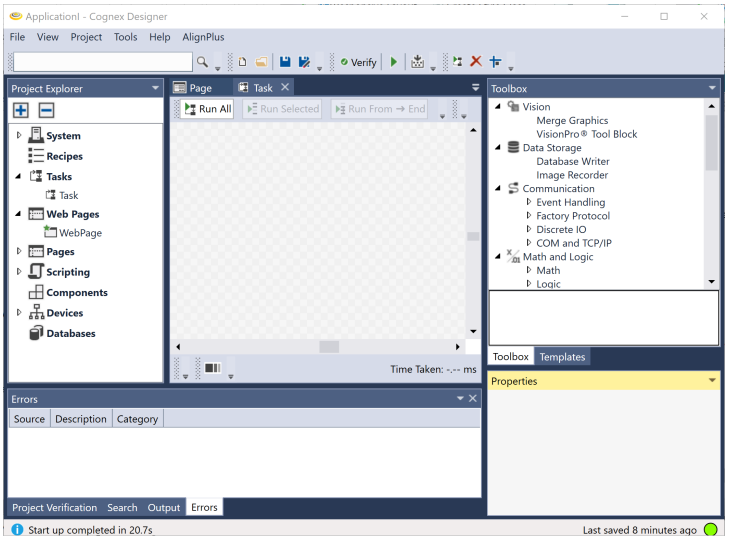
2. In the opened dialog, click "New Project" icon, an new dialog will pop up for you to confirm project name and its path.



3. In the pop-up dialog, input a project name and choose a path for it (the default path is "C:\Users\<UserName>\Cognex Designer\Projects"). Keep the default project type as "Blank Project" and the default Plugins as default as "AlignPlus", click "Accept" button to confirm. After this, Designer will run a process to create the empty project.



4. A empty AlignPlus project has no difference with other empty Designer projects except that it has AlignPlus plugins added.



After the empty AlignPlus project is created, Configuration Wizard then can be used to generate the contents of the project based on user's configuration.

What is Configuration Wizard

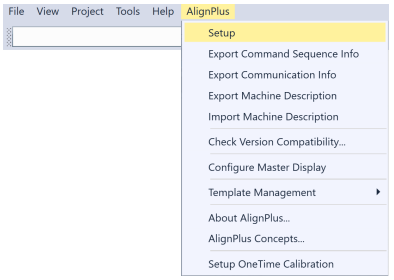
AlignPlus Configuration Wizard provides an interface for user to configure cameras, calibrations, feature finders, and pose computers for an application, and then automatically generates the project in few minutes. Configuration Wizard supports multiple cameras, multiple stations, stage/robot alignment, assembly, or inspection applications. It is very convenient and flexible for user to generate projects based on machine descriptions.

An automatically generated project includes the following functions:

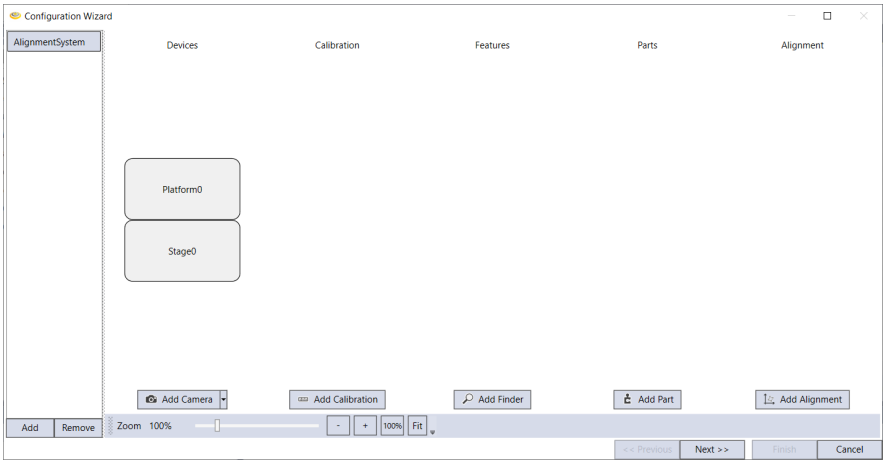
1. Vision tasks including calibration, train time and run time feature finding tasks
2. Pose computing tasks which compute the required motions to align or assemble parts to the target poses
3. User interface for user to setup the machine, test, run, and deploy
4. Common alignment functions such as L-Check, Offset Compensation, Manual Alignment, and Motion Analysis
5. System utilities such as recipe management, user management, multi-Language, data Logging, and Image saving and play back
6. Communication interface for TCP/IP commands to interface with external controllers such as PLCs and Robot controllers

Note: For basic knowledge about a designer program, please refer to **Help\Cognex Designer User Guide\Developing a Cognex Designer Project.**

To open the Configuration Wizard, choose the "Setup" option under "AlignPlus" menu in the opened designer project.



An empty Configuration Wizard is as blew.



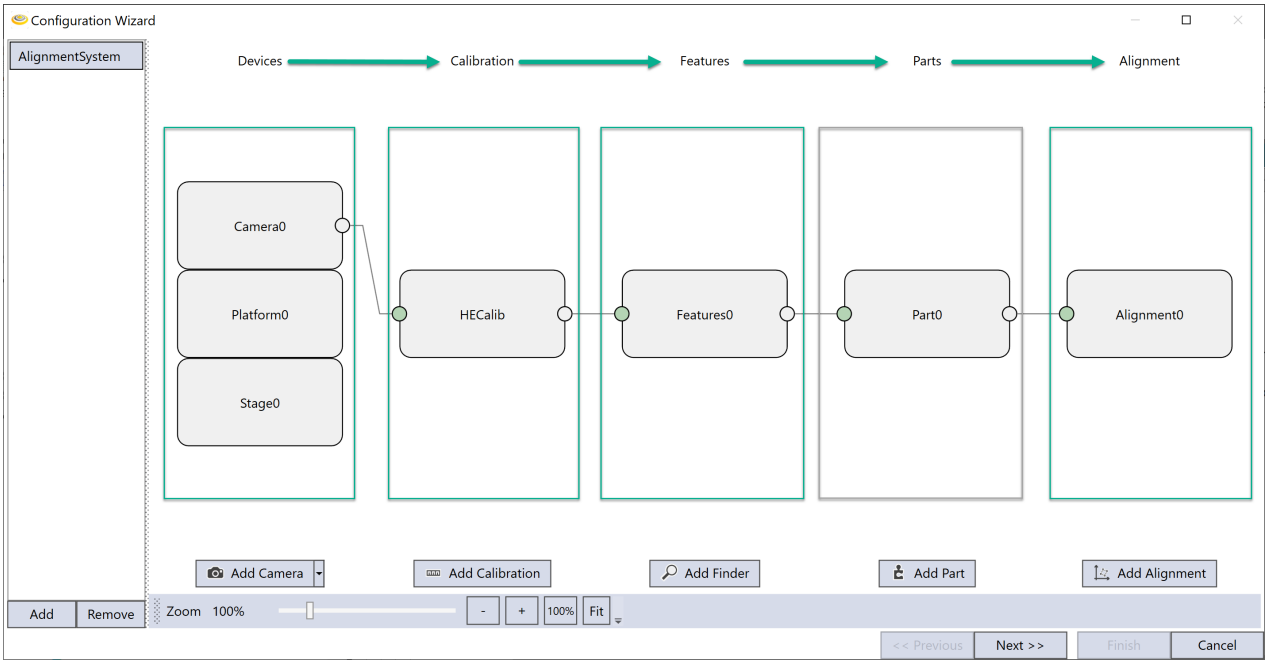
Configuration Wizard provides configurations for the following aspects of an AlignPlus program:

- Alignment System on page 18
- Devices on page 20
- Calibrations on page 21
- Finders on page 29
- Parts on page 31
- Alignments on page 32
- Settings on page 34

Alignment System

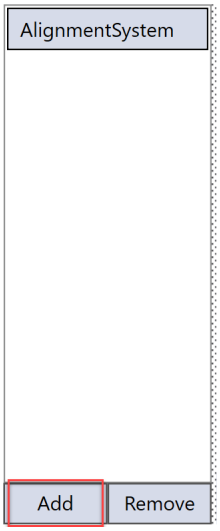
An alignment system is a system that communicates or controls related hardware (such as camera, lights, motion devices), runs vision tasks, and computes results to guide motion devices to achieve alignment/assembly goals. One alignment system manages five type of components: Devices, Calibrations, Feature Finders, Parts, and Alignments. Each type could have multiple components depending on the application.

Devices are used to acquire images, communicate with motion devices and control lights. Calibrations calibrate images acquired from the corresponding cameras; Feature finders locate features on calibrated images; Parts specify which part rests on which station; Alignments compute target poses for motion devices based on found features. Each component takes inputs from the components (except Devices) on the left side and produces outputs for components (except Alignments) on the right side. During the wizard configuration, configure from left to right so that each component receives its proper inputs.



Add

By default, the Configuration Wizard generates only one alignment system named "AlignmentSytem". However, it allows user to add more alignment systems. After the application is generated, each alignment system will run independently without interfering with the others. To add one, click "Add" button at the lower left corner of set up wizard window.



To rename the added system, double-click it to enter edit mode and input the new name (only numbers and alphabetic characters are allowed). Once it is done, hit "Enter" key to confirm.

Remove

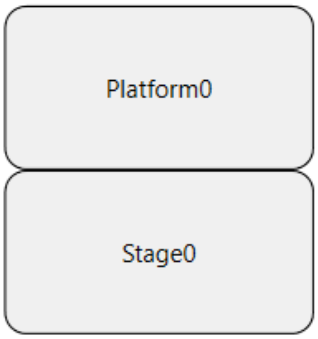
To remove an alignment system, select it from the list and then click "Remove" button. Note that there should be at least one alignment system remains.

Devices

Devices in Configuration Wizard are representatives of real hardware devices related to vision system in an application, including motion devices, cameras, and light controllers. Their quantities should match with the numbers of corresponding hardware devices in that application.

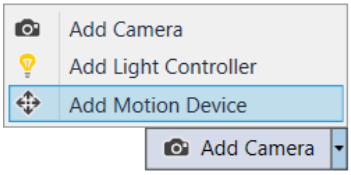
Motion Device

A stage and a platform are created by default for an alignment system. Stage is a motion device that can move part on it, such as a XYT stage or a robot. Whereas platform is another type of motion device on which part cannot be moved, such as a stationary table.



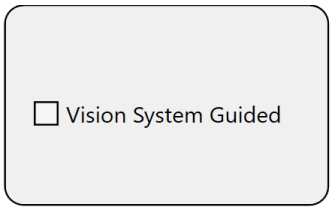
Add

If there are more stages or platforms in the system, you can click **Add Motion Device** to add them.



Set

All motion devices including the default stage and platform have one parameter within: **Vision System Guided**. Click "  " at the upper right corner of the component, or right click the component and select **Expand/Collapse** to open this parameter.



If the "Vision System Guided" is checked, the Configuration Wizard will generate a calibration loop that can be used to guide motion device to move during hand-eye calibration. If it is unchecked, the calibration loop will not be created, motion device and hand-eye calibration task need to interface step by step to finish the hand-eye calibration process.

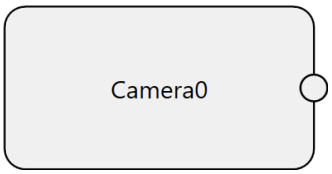
Delete

Besides the two default devices (Stage0 and Platform0), the rest of the devices can be deleted if they are not used. To delete one, select it and press "Delete" button on keyboard.

Camera

Add

Click **Add Camera** to add a camera device to the alignment system.



The first one has a default name of **Camera0**, the following ones will be Camera1, Camera2, and so on. To rename a camera device, double-click it or select it and hit "F2" key to enter edit mode, then input the new name.

Note: The renaming method mentioned above applies to all other components in the alignment system.

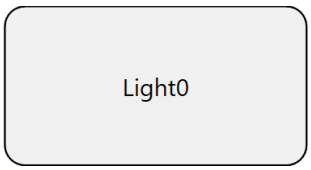
Delete

Select the camera to be deleted, and press "Delete" button on keyboard.

Light Controller

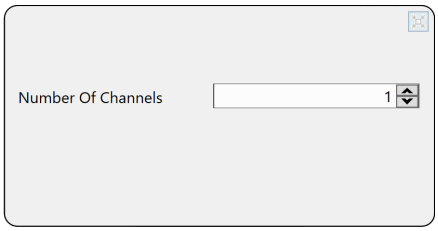
Add

Click **Add Light Controller** to add a light controller to the alignment system.



The naming scheme for light controller device works the same as it does for cameras.

A light controller could have multiple channels, each channel controls one light. After the project is generated, the vision system can control through light controller which channel to turn on/off, what intensities to use to achieve desired imaging effect. To input the number of channels, click the "  " icon to open the edit box and enter the number.



CAUTION: This is the only place to change the number of channels of a light controller. If this number should be changed after the project is generated, one need to come back to the Configuration Wizard to change it, and rerun the wizard.

Delete

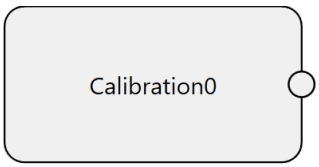
Select the light controller to be deleted and hit "Delete" button on keyboard.

Calibrations

Calibration establishes shared coordinate among different cameras and motion devices. There are six types of calibrations: stationary hand-eye calibration, moving hand-eye calibration, cross calibration, manual calibration, checkerboard calibration, and hand-eye camera mount calibration, each serves a different purpose. To understand more about different types of calibrations, please refer to Calibration Introduction and its following topics.

Add

Click **Add Calibration** to add a calibration component. The first component has a default name of **Calibration0**.



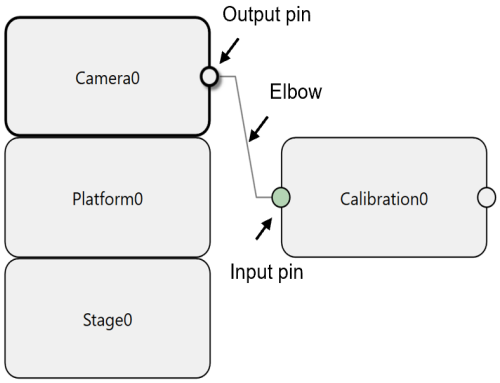
Connect

Before configuring its calibration type, the calibration component needs to connect to at least one camera device. Here are steps to connect:

1. Select one camera device.

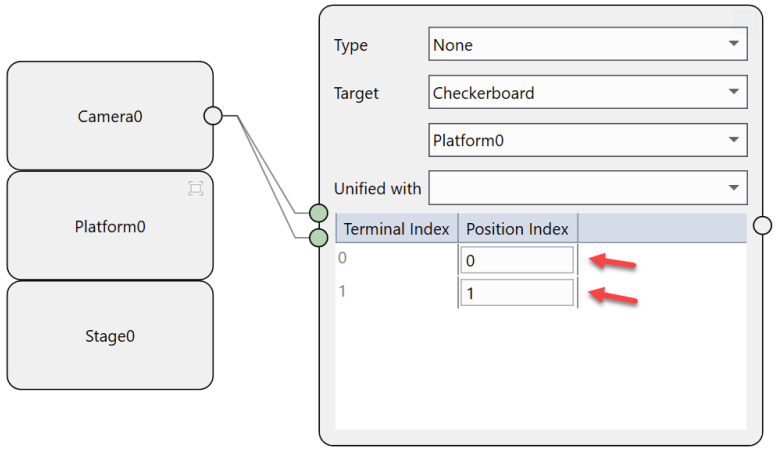
2. Click the output pin of the selected camera and drag a line from it to the calibration component on the right.

4. When you see the calibration component generates an input pin to receive the line, release mouse to finish connection.

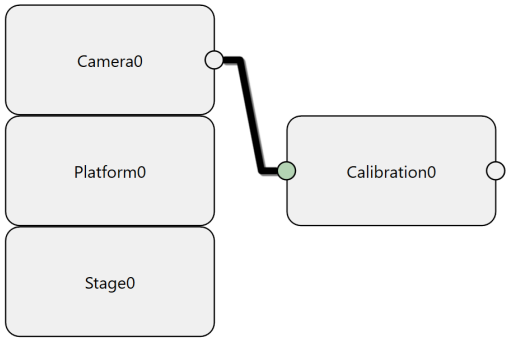


Note: It is the same way to connect a calibration component to a feature finder, a feature finder component to a part, or a part component to an alignment.

If a calibration requires two image acquisitions from the same camera at two different positions(such as camera shuttling case), then user need to drag two lines from that camera device to the calibration component. The calibration component will automatically generate two input pins with different position indexes.

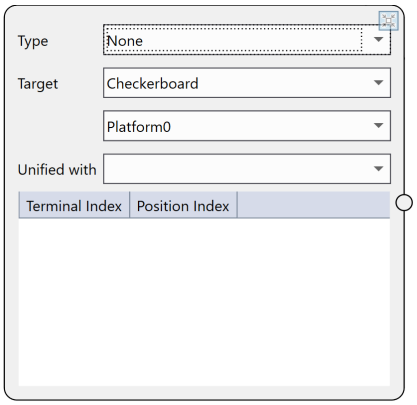


To remove a link between two components, first select the line until it becomes bold, then press "Delete" button on keyboard.



Set

After connecting to camera device(s), user can start configuring parameters inside a calibration component. Click "  " at the upper right corner of the component, or right click it and select **Expand/Collapse** to open its inside parameters.



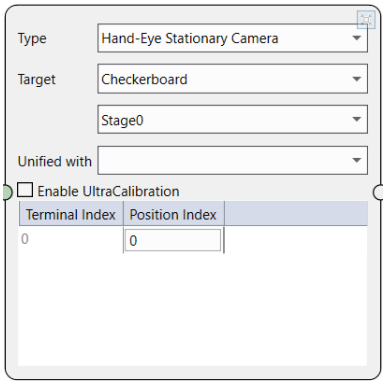
Type

Here are all options for a calibration type:

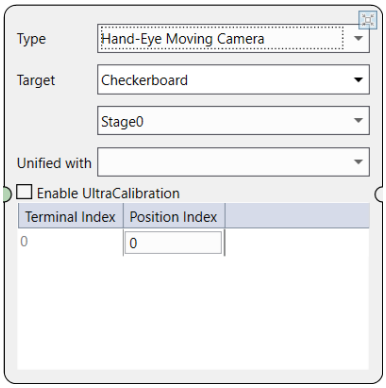
Option	Description
None	Upon this select, there will be no calibration task added to the auto-generated project. Images are merely passed through this calibration component from cameras to connected feature finders; these finders will find their features in pixel space.
Hand Eye Stationary Camera	Select this option to be able to perform stationary hand-eye calibration in which calibration target moves together with a motion device under the FOVs of stationary cameras. The connected feature finders will find features in Home2D.
Hand Eye Moving Camera	Select this option to perform moving hand-eye calibration of in which cameras moves together with a motion device over a stationary calibration plate. The connected feature finders will find features in Home2D.
Cross Camera	Select this option to be able to perform cross calibration. The connected feature finders will find features in Home2D generated by the unified hand-eye calibration.
Pixel to Physical Units	Select this option to be able to perform checkerboard calibration. The connected feature finders will find features in Plate2D.
Manual Calibration	Select this option to be able to perform manual calibration. The connected feature finders will find features in Home2D that is generated by manual calibration.
Hand-Eye Camera Mount	Select this option to be able to perform one-time calibration for every connected cameras. These calibration results will be used to avoid recalibration when cameras are repositioned to fit new model size.

Hand-eye Calibration

Hand-eye Camera Stationary and Hand-eye Moving Camera have similar parameters.



Stationary Camera



Moving camera

Target

Option	Description
Part	Use part features to perform hand-eye calibration
Checkerboard	Use calibration plate features to perform hand-eye calibration
Hybrid	Use calibration plate to calibrate cameras intrinsic parameters and generates distortion-free images, and then perform hand-eye calibration by tracking feature(s) on a calibration part.

Motion Device

All available motion devices in the wizard. Choose the one that will move the calibration target/cameras during the hand-eye calibration.

Unified with

Enables a cross calibration to be unified with current hand-eye calibration.

Enable UltraCalibration

Select this check box to make the generated application be able to perform UltraCalibration.

Related Data

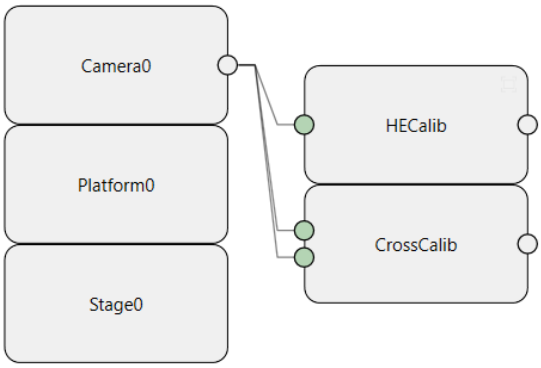
Table Column	Description
Terminal Index	The index of the input terminal. The upper most terminal has an index of 0.
Position Index	To acquire images of regions of a part, many alignment systems use multiply cameras to acquire images from multiple regions of a part. These cameras acquire images simultaneously when application uses the position index . If position indexes are different, then cameras acquire images at different times.

Cross Calibration

It takes three steps to add a cross calibration component: connect it to cameras, , and unify it with hand-eye calibration block.

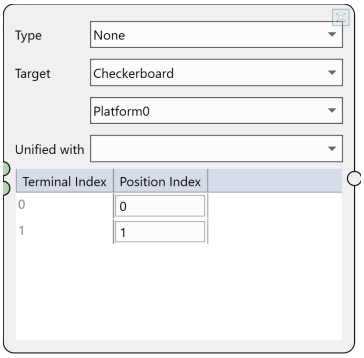
1. Connect to cameras

   Create a new calibration component and rename it (for example, "CrossCalib"), and connect cameras to this component according to its acquisition requirements.



2. Check position indexes

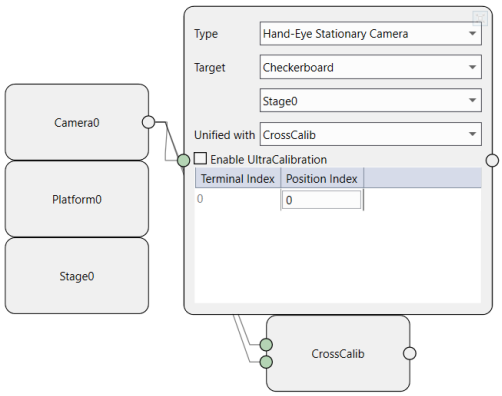
   Check whether the position indexes are correctly configured. The first position index is 0, second is 1, and so on. For cross calibration, there should at least two different position indexes.



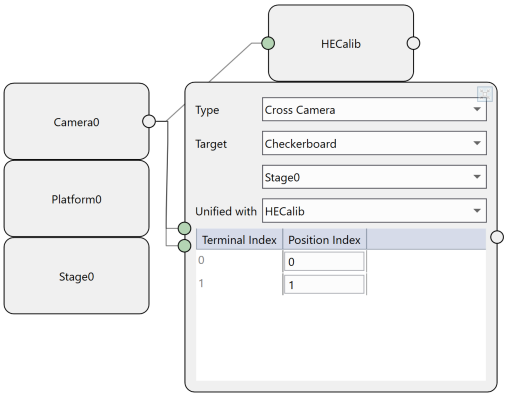
3. Unify with a hand-eye calibration component

   There are two ways to unify a cross calibration component with a hand-eye calibration component, either way will work.

   - In the opened hand-eye calibration component, select the name of the cross calibration component to be unified in the **Unified with** drop list, and click **Yes** in pop-up dialog.



   - In the opened cross calibration component, select the name of the hand-eye calibration component to be unified in the **Unified with** drop list , and click **Yes** in pop-up dialog.



   However, if there are more than one cross calibrations which needs to bind with the same hand-eye calibration, only the second way can realize it.

Target

Option	Description
Part	Use part features to perform cross calibration
Checkerboard	Use calibration plate features to perform cross calibration
Hybrid	Use calibration plate to calibrate cameras intrinsic parameters and generates distortion-free images, and then perform cross calibration using feature(s) of a part on corrected images

Unified with

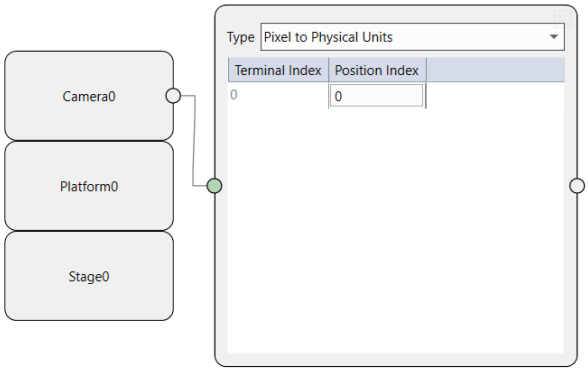
Leave it as empty the corresponding hand-eye calibration component has already selected current cross calibration component in its **Unified with** drop list. Otherwise, select the name of the hand-eye calibration component to unified with and then click **Yes** in the pop-up dialog.

Related Data

Table Column	Description
Terminal Index	The index of the input terminal. The upper most terminal has an index of 0.
Position Index	To acquire images of regions of a part, many alignment systems use multiply cameras to acquire images from multiple regions of a part. These cameras acquire images simultaneously when application uses the position index . If position indexes are different, then cameras acquire images at different times. There must be more than one position index in cross calibration.

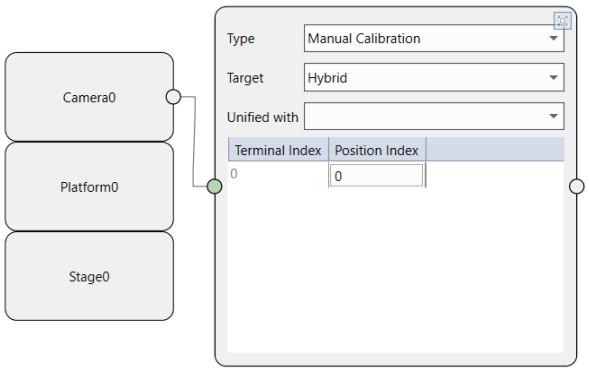
Checkerboard Calibration

Checkerboard calibration does not require any motion devices, nor needs to be unified with any other calibrations, it only requires a calibration plate to be placed stationary in the FOVs of cameras before the calibration. It takes two simple steps to configure a checkerboard calibration component: after connecting to the related cameras, select **Pixel to Physical Units** in the type field.



Manual Calibration

Manual calibration also does not require any motion devices, nor any other calibrations. To add it, choose **Manual Calibration** type after its connection to cameras.



Target

Option	Description
Hybrid	Use calibration plate to calibrate cameras intrinsic parameters and generates distortion-free images, and then perform manual calibration using feature(s) of a part from corrected images
None	User directly input parameters of transform from Raw2D to Home2D without the need of feature information

Unified with

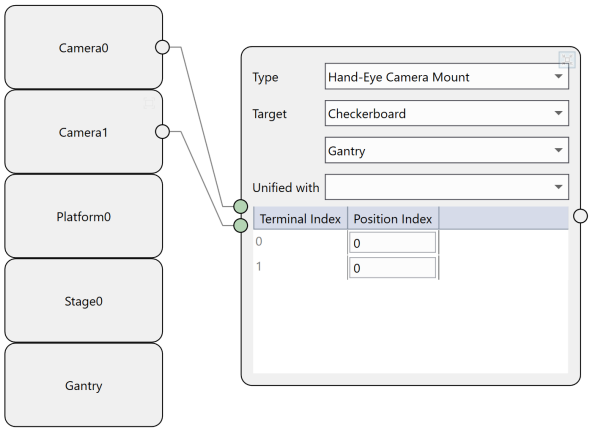
Leave it as empty.

Related Data

Table Column	Description
Terminal Index	The index of the input terminal. The upper most terminal has an index of 0.
Position Index	To acquire images of regions of a part, many alignment systems use multiply cameras to acquire images from multiple regions of a part. These cameras acquire images simultaneously when application uses the position index . If position indexes are different, then cameras acquire images at different times.

One-time Calibration

One time calibration performs moving hand-eye calibration for each connected camera individually using the gantry where the camera is mounted to as the motion device, to get the transforms from gantry spaces to Home2D. To add an one-time calibration, choose **Hand-Eye Camera Mount** type after its connection to cameras.



Target

One-time calibration only uses calibration plate as calibration target.

Unified with

Leave it as empty.

Related Data

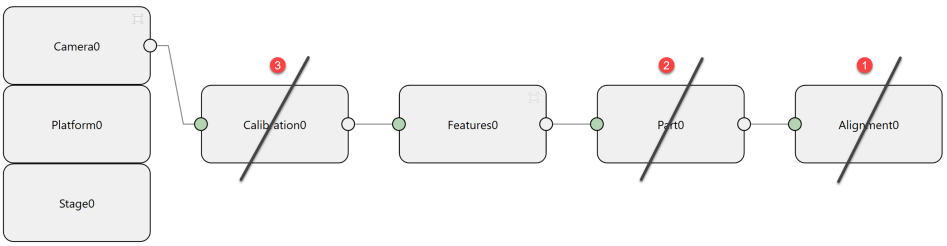
Table Column	Description
Terminal Index	The index of the input terminal. The upper most terminal has an index of 0.
Position Index	Always be 0.

Note:
One-time calibration is only used for camera offsets adjustment to avoid recalibration during model change, its result can not be used to correct images directly, therefore it can not be connected to a feature finder component.

Delete

If a calibration component is bounded with another calibration component (such as a cross calibration is unified with a hand-eye calibration), you need unhook them first by choosing empty in their **Unified with** fields, and then delete the one you would like to delete.

If a calibration component is connected to a finder, and the finder links to a part, the part links to an alignment, then you need to delete alignment component first, then delete the part, and at last the calibration component can be deleted.



Finders

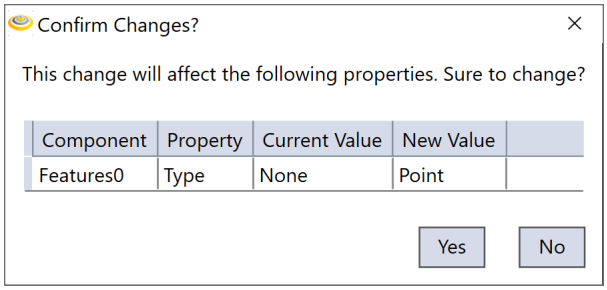
Finders are used to locate features (such as fiducial marks, corners, etc.) on calibrated images. One finder will have one feature finding task in the generated project. This task can be used for both train time and run time feature finding.

Add

Click **Add Finder** to add one finder component.

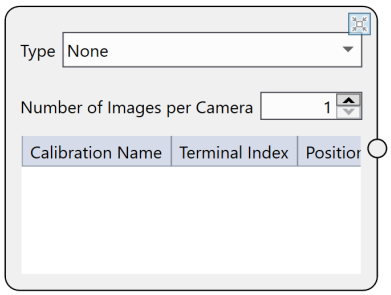
Connect

A finder component need to be linked to a calibration component before it can be configured inside. Once the link is made, a dialog will pop up saying the finder type will be changed to **Point**, click "Yes" to continue.



Set

In the opened dialog box of the finder component, user can configure the feature type and number of images per camera.



Type

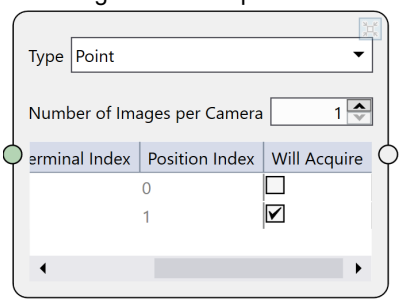
Option	Description
Custom	When custom is chosen, the FeatureExtractorSubTask inside the corresponding feature finding task generated by the Configuration Wizard will be empty. The idea is that the users can customize this sub task. One advantage of selecting 'Custom' type is that when the wizard is rerun, the user customization contents in the FeatureExtractorSubTask will be maintained. For the other types, AlignPlus will overwrite the sub task contents to default settings.
None	Default option when finder is not connected to any calibration component.
Point	Select this option to have the feature finding task and the corresponding configuration HMI to find point features.
Line	Select this option to have the feature finding task and the corresponding configuration HMI to find line features.
Generic Feature	Select this option to have the feature finding task and the corresponding configuration HMI to find generic features.
Multiple Parts	Select this option to have the feature finding task and the corresponding configuration HMI to locate multiple parts using point features. Current multiple parts function only supports single camera single position pickup applications.
AOI Features	Select this option to have the feature finding task and the corresponding configuration HMI to locate generic features for inspection applications. The auto-generated corresponding feature finder will be a generic features finder.

Tip: It is not recommended to make any changes to auto-generated contents in FeatureExtractorSubTask of a feature finding task to avoid possible damage to it. However, if the user is confident enough to make modifications in it, and then would like to keep these modifications when the wizard rerun is needed, it can be done in this way: 1) choose point/line/generic feature type in finder component for the first time of project generation; 2) make modifications inside the FeatureExtractorSubTask that are very necessary for the application; 3) change the feature type to 'Custom' in the finder component and rerun the wizard.

Number of Images per Camera

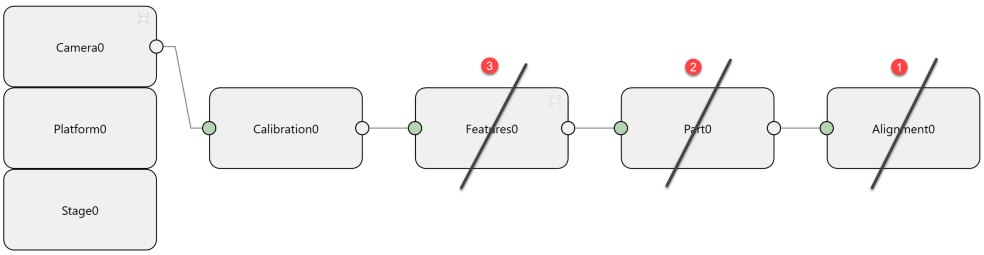
Specifies the number of images that the corresponding feature finding task will capture from each camera at each acquisition position. The default number is 1, increase it if the same camera needs to capture multiple images of the part under different exposure and lighting settings.

Related Data

Table Column	Description
Calibration Name	Name of the calibration component whose output terminal is connected to current finder component
Terminal Index	Index of the corresponding input terminal of the calibration component.
Position Index	Corresponding position index in the calibration component
Will Acquire	A check box that indicates if the image at the corresponding Calibration Name, Position Index, and Terminal Index will be acquired during the feature finding process. If an image are not required for feature finding, you can check this option off. 

Delete

If the finder component is connected to a part, and the part links to an alignment, then you need to first delete alignment component, then the part, and the finder component can be deleted.



Parts

Part is used to specify which part rests on which stage or platform. Part component has no corresponding task, nor needs any further configuration in the generated project. It only provides necessary for pose computation in alignment task.

Add

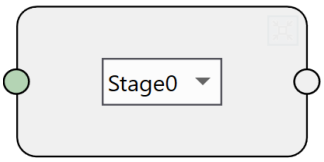
Click **Add Part** to add a part component.

Connect

Connect it with one finder component on the left side.

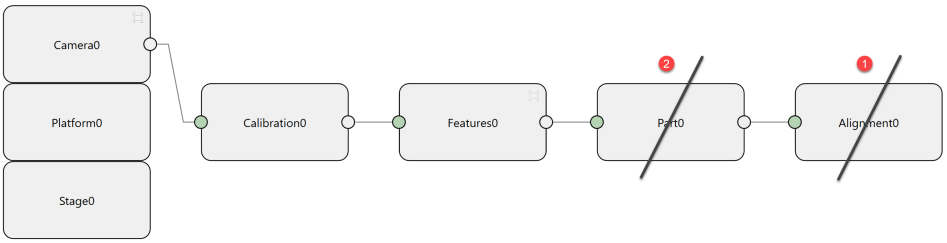
Set

Open the dialog box, and select the device where the part (whose features are found by the connected finder) rests on according to real situation.



Delete

If the part component is connected to an alignment component, delete the alignment first, then delete the part.

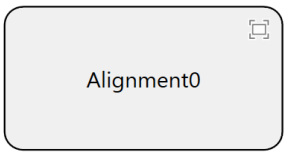


Alignments

Alignment component allows user to configure the techniques for alignment, assembly, or inspection computation. One alignment in Configuration Wizard has one corresponding task in the auto-generated project.

Add

Click **Add Alignment** to add an alignment component in the wizard workspace.

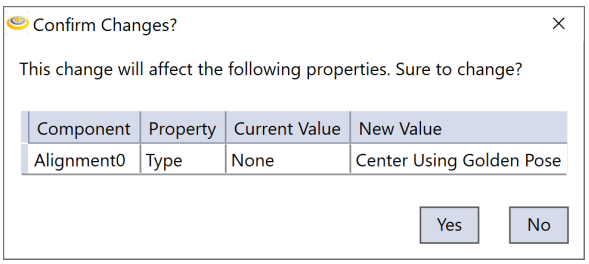


To change the name of it, double-click **Alignment0** to edit it.

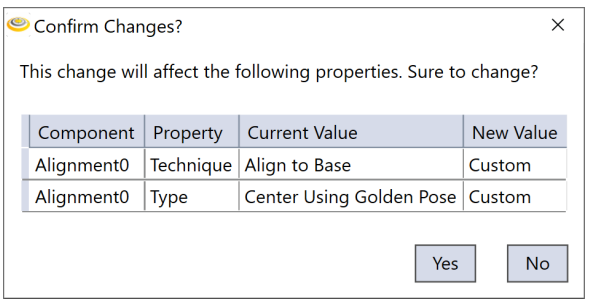
Connect

Alignment needs to be connected to part component first before its configuration. For alignment pose computation, the component only needs to connect to one part. For assembly pose computation, it should connect to these two parts to be assembled.

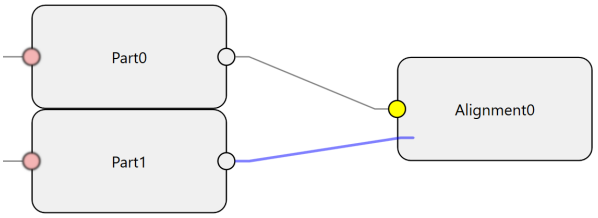
When the component connects to the first part, the wizard will set its default type as "Center Using Golden Pose"; click "Yes" in the pop-up dialog to continue.



When the component connects to the second part, the wizard will change its type to "Custom"; click "Yes" in the pop-up dialog to continue, you can change the type afterward.



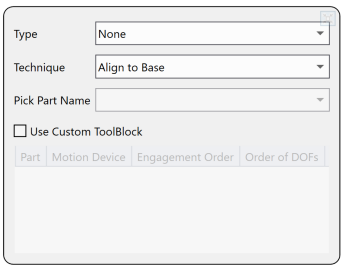
There could be cases when you try to connect the second part to an alignment component, but the component does not accept:



The cause of this phenomena is both parts are on the same station, which is apparently a mistake. To solve it, check and reconfigure the motion devices in both parts, make sure the two devices are different, then reconnect them to alignment component.

Set

Open the alignment dialog box, configure the following parameters according to real application.



Type

Option	Description
Custom	Select this option to create an empty PoseComputer subtask inside the corresponding alignment task in generated project. Users can customize this subtask, and when the Configuration Wizard reruns, the customized content in this PoseComputer subtask will be maintained.
None	Select this option when an alignment component has no input connections.
Center Using Golden Pose	This option can be used either for one part alignment application or two parts assembly application where each part aligns to its target pose defined during train time. For more information, please refer to Alignment Procedure and Using Golden Pose.
Center Using Paired Features	Select this option to align parts using one part's features as target pose for another part. For more information, please refer to Using Paired Features.
Multiple Parts	Select this option to align multiple parts to multiple trained target poses.
AOI	Select this option to run inspection on run time part, including measurement and gauging. AOI function allows users to get features from more than one part, however, the feature names should be unique among all parts.

Tip: It is not recommended to make any changes to auto-generated contents inside a PoseComputerSubTask of an alignment task to avoid possible damage to it. However, if the user is confident enough to make modifications in it, and would like to keep these modifications when the wizard rerun is needed, it can be done in this way: 1) choose other types in alignment component for the first time of project generation to let the wizard generate the needed pose computer contents; 2) make modifications inside the PoseComputerSubTask that are very necessary for the application; 3) change the alignment type to 'Custom' in the alignment component and rerun the wizard.

Technique

Option	Description
Custom	Select this option to keep the PoseComputer subtask empty. Users can customize this subtask, and when the Configuration Wizard reruns, the customized content in PoseComputer subtask will be maintained.
Align To Gripper	Select this option when a part paced on a stationary platform need to be picked up by a gripper. The gripper will first adjust its own position based on the vision system's feedback, then pick the part up at a fixed relative pose to the part. For more information, please refer to Align to Gripper.
Align To Base	Select this option when a part is attached to a motion device(such as a stage), after alignment pose computation, the part is moved by the motion device to the part's trained golden pose. For more information, please refer to Align to Base.

Option	Description
Assembly Blind Transfer	Select this option when the parts are handled in this way: the first part is placed on and moved by a motion device at the feedback of the vision system to the expected relative pose of the second part; one part is blindly transferred to the other part's side, and the two parts are assembled. For more information, please refer to Assembly Blind Transfer.
Assembly Guided Pick	Select this option when the parts are handled in this way: the picking device such as robot or gripper adjusts its own position first before picking up the first part based on the vision system's guidance; the picking device picks the first part up and moves it over to the second part where the two parts are assemble together. For more information, please refer to Assembly Guided Pick.
Assembly Guided Place	Select this option when the parts are handled in this way: the picking device such as robot or gripper blindly picks up the first part and then adjust its pose together with the part to fit it to the second part's pose based on the vision system's guidance; the picking device moves the first part over to the second part's side, and the two parts are assembled together. For more information, please refer to Assembly Guided Place.
Split Axis	Select this option when three elements of moving freedom(X/Y/Theta) are split to 2 or 3 different stages in machine.
Measurement	Select this option to run measurement in alignment task.

Pick Part Name

The name of the part whose pose will be adjusted by a motion device during alignment/assembly process. It is automatically selected by the wizard based on user's configuration.

Use Custom Tool Block

Select this check box to have the run-time application present a VisionPro tool block edit control that you can use to implement.

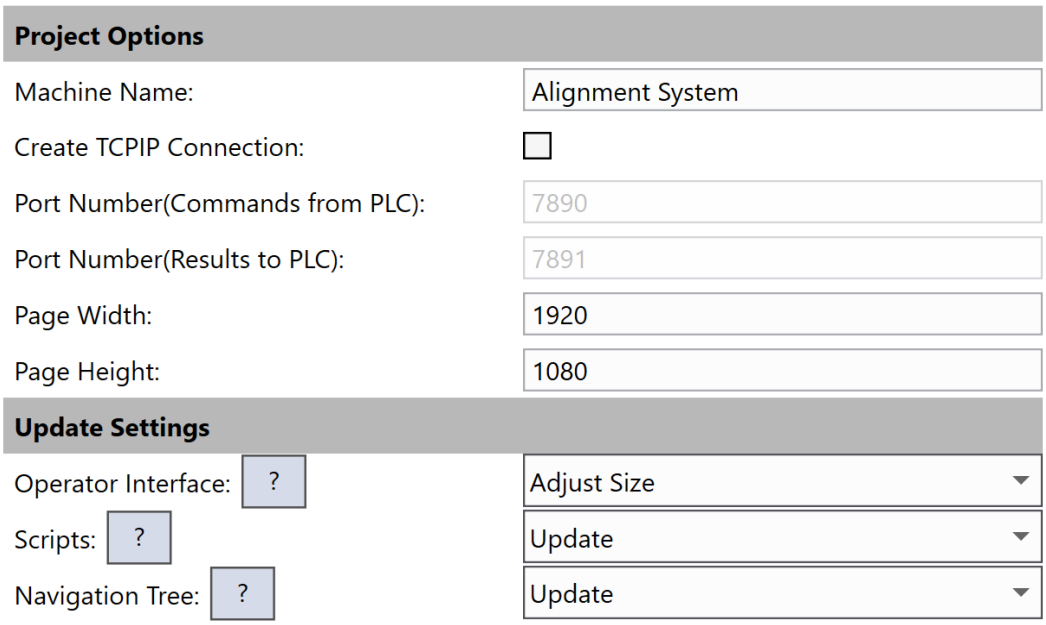
Delete

Select the alignment component, and hit "Delete" button on keyboard.

Settings

After all components are well configured, click **Next** in the lower right of the Configuration Wizard to continue.

Then a setting page will show up:



Element	Description
Machine Name	Displays the machine name that the header of the run-time application displays.
Create TCPIP Connection	If it is checked, Configuration Wizard will automatically create two TCP/IP server devices (CommandsFromPLC and ResultsToPLC) as well as their callback scripts during the project generation. Otherwise, these two devices and their callbacks will not be created.
Port Number (Commands from PLC)	The port number of CommandsFromPLC TCP/IP server device.
Port Number (Results to PLC)	The port number of ResultsToPLC TCP/IP server device.
Page Width	Enter the width of the deployed application. If you stretch the width of the deployed application and rescale it, the controls on the application will be stretched by the same scale factor.
Page Height	Enter the height of the deployed application. If you stretch height of the deployed application and rescale it, the controls on the application will be stretched by the same scale factor.
Operator Interface	Operator interface update settings can help fix interface issues after a upgrade • Minimal: this option will add missing pages or buttons only. If page width or page height have changed, the operator interface layout may break. • Adjust Size: this mode will resize and reposition all wizard-created pages and controls, and add back any missing controls. User-created controls are not affected. • Adjust All Properties: this option will resize, reposition and reset properties like color or style of wizard-created controls and pages. User-created controls are not affected. • Reset Pages: this option will remove all operator interface customizations from wizard-created pages. User-created pages are unaffected. Note that this option resets all scripts attached to buttons and controls as well
Scripts	Script Update Settings determine how AlignPlus updates scripts when the wizard runs. • Minimal: (Not recommended) this option performs no script changes other than adding any essential scripts that are missing. • Update: this option makes an attempt to update scripts in the project to the latest version without discarding user changes. • Reset: this option will remove all user changes to scripts and reset all scripts to their fresh state. Use this option if your project is having a lot of scripting errors.
Navigation Tree	Navigation Tree Update Settings determine how AlignPlus updates the Navigation Tree • Update: this option will preserves all edits to the navigation tree except those made invalid by configuration changes. • Reset: this option will remove all user changes to the navigation tree and restore it to its default state.

Note: If it is the first time of the project generation, keep the default settings for **Operator Interface**, **Scripts** and **Navigation Tree**; if it is wizard rerun, pick carefully what you want wizard to do for the already generated UI, scripts, and navigation tree.

Click **Finish** to let the wizard generate the project after all settings are done, click **Previous** to go back to Configuration Wizard workspace if you would like to modify anything on it, or click **Cancel** if you do not want to generate a program.

CAUTION: If **Cancel** button is clicked, all settings you have done including the components configurations and the settings on current page will be cleared out. To save current configurations and settings, run the wizard first, and then you can export the configurations into a .json file which can be imported and reused later. See more on Export and Import Wizard Configuration on page 45

Things to Consider before Configuration

To configure a Wizard Configuration correctly, one needs to fully understand the application about devices used, stations, parts, feature types, alignment methods, etec.

Devices

These are questions to be answered before configuring the Devices in Configuration Wizard.

About motion devices:

1. How many stations are there in this application?
2. Whether the station is movable or stationary?
3. Whether calibration loop task is needed for hand-eye calibration?

About cameras:

1. How many cameras used for each station?

About light controllers:

1. How many light controllers used for each station?
2. How many channels each light controller has?

Calibration

The first thing to consider about calibration is whether a calibration plate (such as calibration mylar sheet) can be used as calibration target in the machine. For cross calibration which requires calibration target to be transferred from one station to another, is the mechanical device capable of transferring either a calibration plate or a real part? If calibration plate can be transferred, choose it as the first option. If not, then use real part as calibration target. If both are not, then use manual calibration instead.

Calibrations for movable station

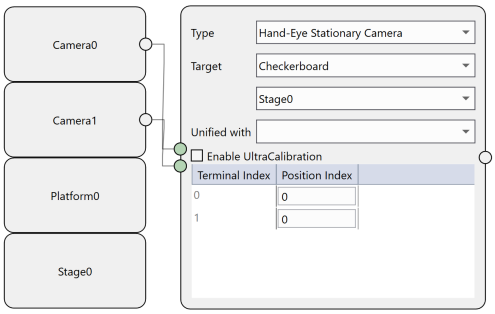
For movable station, use hand-eye calibration type.

Here are aspects to consider for hand-eye calibration.

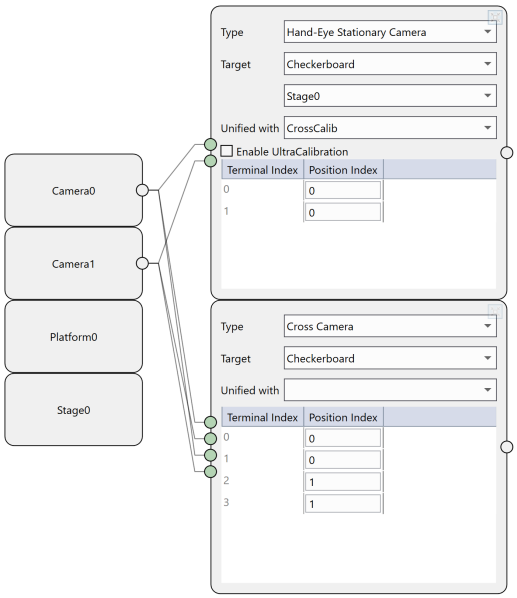
1. Will cameras move or stay stationary during hand-eye calibration?(moving hand-eye or stationary hand-eye)

2. Can calibration plate be moved by the motion device during hand-eye calibration? (Choose real part if not)

Here is an example of a hand-eye calibration configuration for two stationary cameras, using checkerboard as target.



In the movable station, cameras may need to be shuttled to a different position to acquire images of other regions of the run time part. In this case, a cross calibration is needed. Here is an example of adding a cross calibration for the same movable station.



Calibrations for platform

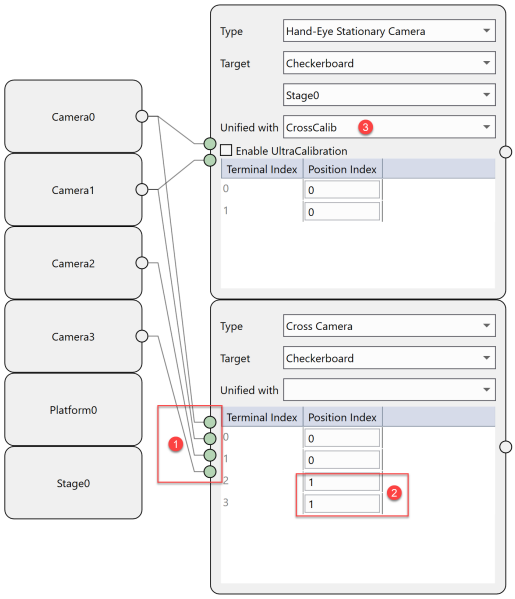
For stationary platform, check first whether it would be used for assembly or inspection application.

Assembly application

To establish a shared coordinated between the movable station and the platform, a cross calibration is needed between the two stations.

In the application example below, a calibration plate is first placed on **Stage0**, and two images of it are acquired by **Camera0** and **Camera1**. After that, then calibration plate is transferred by mechanical device from **Stage0** to **Platform0**, where other two images are acquired by **Camera2** and **Camera3**.

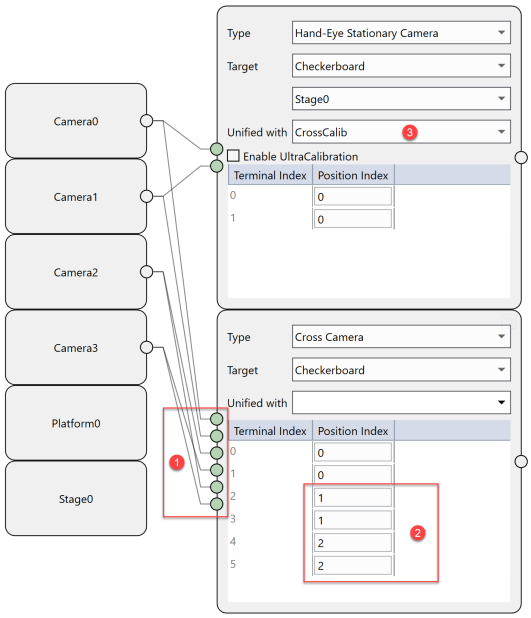
So the cross calibration component should connect to four cameras: the first two acquire at one time, and the second two acquire at another time. However, cross calibration component cannot know whether those images should be captured at the same time or not. By default it sets all their position index as 0. Therefore, user needs to manual change the position indexes of **Camera2** and **Camera3** from 0 to 1. Following that, unifies the cross calibration component with the hand-eye calibration component.



If current platform has camera shuttling within it, then the cross calibration should cover three acquisition positions: one is the position when calibration target is at the movable station, two are when calibration target is at the platform station.

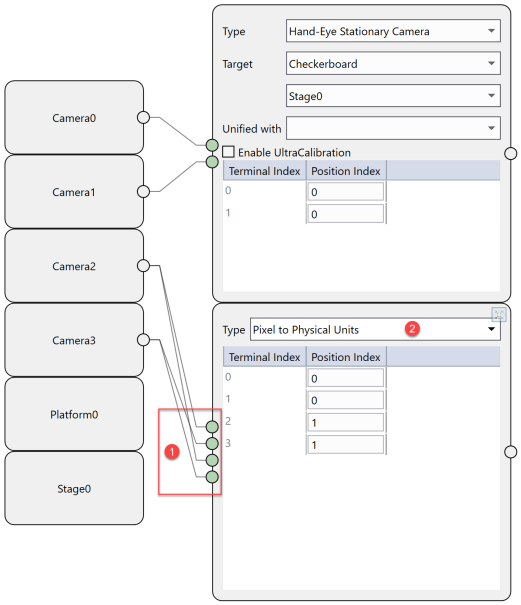
In the application example below, calibration target is first placed under FOVs of **Camera0** and **Camera1** so the two cameras can acquire images at position 0, then moved by mechanical transfer from **Stage0** to **Platform0** so that **Camera2** and **Camera3** can capture images of it first at position 1, second at position 2. For more information about cross calibration, please refer to Applications of Cross-Calibration on page 1.

Therefore, the cross calibration component needs 6 images from three different positions, link the cameras as blow and change the position indexes of the last four input terminals, and then unify it with the hand-eye calibration component.



Inspection application

Inspection applications does not require interacting with motion device, therefore, checkerboard calibration will be enough. If camera shuttling is involved, then it can be configured as below:



However, if that platform already has a checkerboard-based cross calibration component, then the following inspection finder can reuse the component without creating additional checkerboard calibration.

Feature Finding

The following questions should be considered before configuring a finder component:

1. What the feature type of current part?

   Point, Line, Generic Feature, Multiple Parts, AOI, or Custom type.

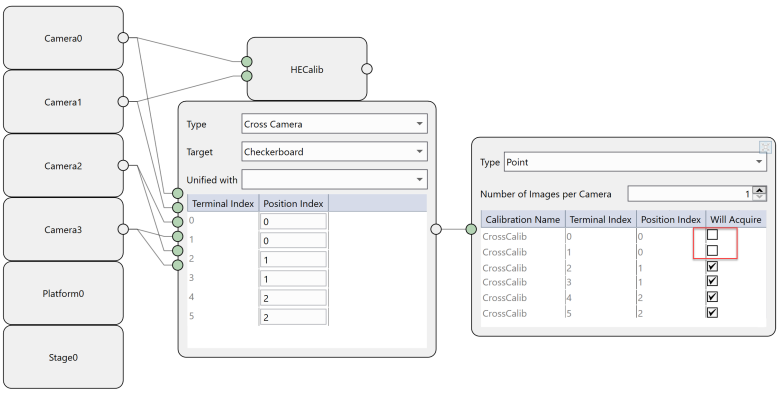
2. How many images should be taken by each camera at each position for feature finding?

   In most cases, **Number of Images per Camera** is set as 1. However, when multiple images should be taken from the same camera at one position, this parameter should be changed accordingly.

3. Whether image acquisitions needed in the connected calibration are all required for the finder?

   By default, a finder inherits all the acquisition settings from its connected calibration component. However, users can choose whether images from given input terminals should be acquired for the finder.

   In the example below, image acquisitions from **Camera0** and **Camera1** are only required for cross calibration to establish the shared coordinate between stage and platform. However, once the coordinate is established, the finder does not need to acquire image from **Camera0** nor **Camera1** anymore, but only images from cameras within the platform. Therefore, the user needs to manually uncheck the "Will acquire" check box for the first two input terminals in the finder component.



Part

Check which part rests on which station.

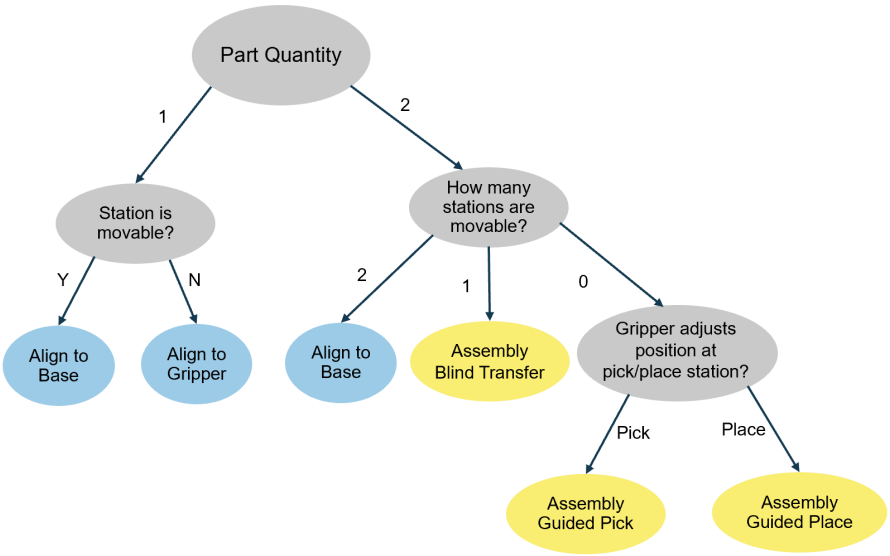
Alignment

Here is a decision tree for your reference about how to check an application type:

If it is an one-part application, then it is either Align to Base or Align to Gripper depending on whether the station is movable or not.

If it is a two-part application, there are four cases:

- **Align to Base** if both are placed on movable stages, the application is an two-part alignment application in which each part has its own alignment component and align to its own golden pose in run time.
- **Assembly Blind Transfer** if only one part is placed on movable stage.
- **Assembly Guided Pick** if both are placed on stationary platform, and motion device will adjust its position before picking up one part.
- **Assembly Guided Place** if both are placed on stationary platform, and motion device will adjust its position after blindly picking up one part.



Setup Examples

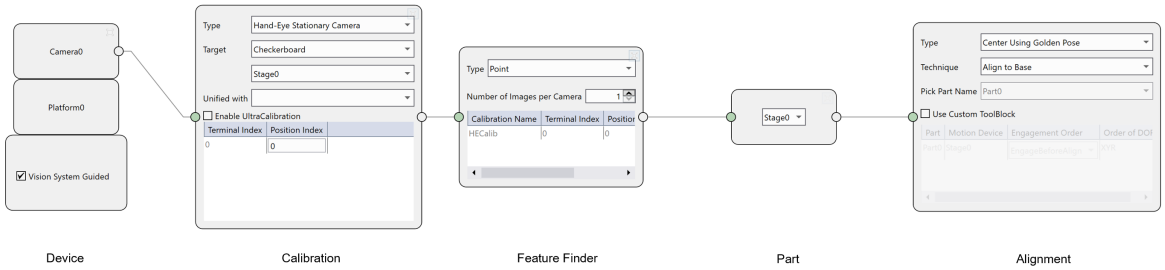
Application I_Align to Base

In this application, one camera is looking at a part which is affixed onto a stage, the goal of this application is to bring part to its golden pose in run time.

Category	Item	Parameter
Part	Quantity	1
Camera	Camera Quantity	1
	Camera Shuttling	/

Category	Item	Parameter
Hand-eye Calibration	Calibration Type	Stationary Camera
	Calibration Target	Checkerboard
	Vision Guided or Motion Guided	Vision Guided
Feature Finding	Feature Type	1 Point
Alignment	Align Technique	Align To Base
	Align Type	Center Using Golden Pose

The configuration of this application is as follow:

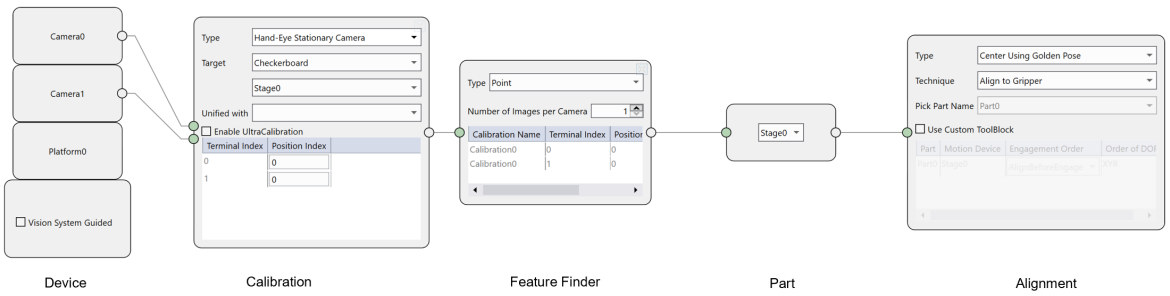


Application II_Align To Gripper

In this application, two cameras are looking at a part which is affixed onto a stationary platform, after the vision system detects the part's pose, a robot will adjust its position based on the vision system's feedback and then pick the part up.

Category	Item	Parameter
Part	Quantity	1
Camera	Camera Quantity	2
	Camera Shuttling	/
Hand-eye Calibration	Calibration Type	Stationary Camera
	Calibration Target	Checkerboard
	Vision Guided or Motion Guided	Motion Guided
Feature Finding	Feature Type	2 Point
Alignment	Align Technique	Align To Gripper
	Align Type	Center Using Golden Pose

The configuration of this application is as follow:

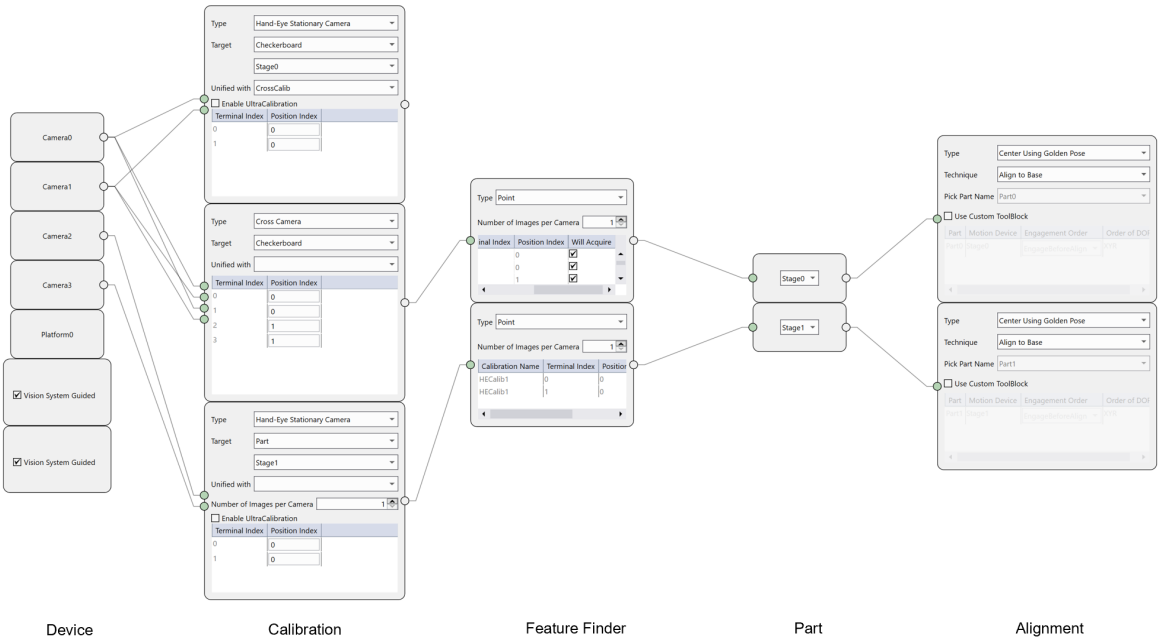


Application III_Two Stages Alignment

In this application, two parts are affixed onto two stages, two cameras fore each stage. The first part is aligned to its golden pose using 4-point features which requires camera shuttling, the second part is aligned to its golden pose using 2-point features. After both are aligned respectively, the two parts are assembled together.

Category	Item	Stage1 Parameter	Stage2 Parameter
Part	Quantity	1	1
Camera	Camera Quantity	2	2
	Camera Shuttling	Pos0, Pos1	/
Hand-eye Calibration	Calibration Type	Stationary Camera	Stationary Camera
	Calibration Target	Checkerboard	Part
	Vision Guided or Motion Guided	Vision Guided	Vision Guided
Cross Calibration	Type	Camera Shuttling	/
	Calibration Target	Checkerboard	/
Feature Finding	Feature Type	4 Point	2 Point
Alignment	Align Technique	Align To Base	Align To Base
	Align Type	Center Using Golden Pose	Center Using Golden Pose

The configuration of this application is as follow:



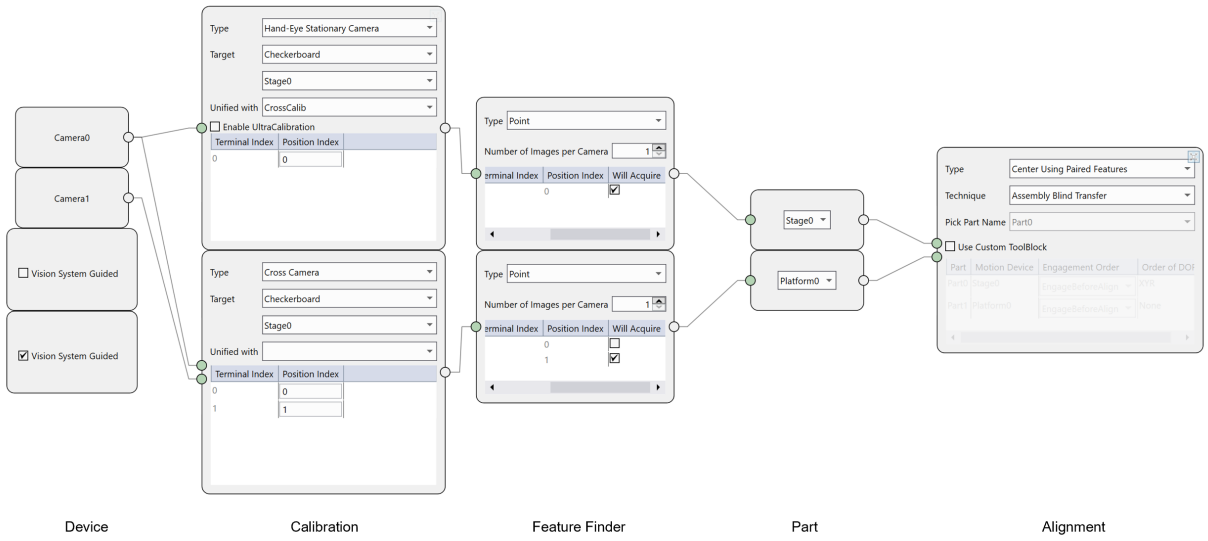
Application IV_Assembly Blind Transfer

In this application, one part is place on a stage, the other is placed on a stationary platform. The stage has only one camera, so does the platform. The first part aligns its pose based on the second part's run time pose using paired features, and then blindly transferred to the second part's side where they will be assembled.

Category	Item	Stage1 Parameter	Stage2 Parameter
Part	Quantity	1	1

Category	Item	Stage1 Parameter	Stage2 Parameter
Camera	Camera Quantity	1	1
	Camera Shuttling	/	/
Hand-eye Calibration	Calibration Type	Stationary Camera	/
	Calibration Target	Checkerboard	/
	Vision Guided or Motion Guided	Vision Guided	/
Cross Calibration	Type	/	Cross Station Calibration
	Calibration Target	/	Checkerboard
Feature Finding	Feature Type	1 Point	1 Point
Alignment	Align Technique	Assembly Blind Transfer	
	Align Type	Center Using Paired Features	

The configuration of this application is as follow:



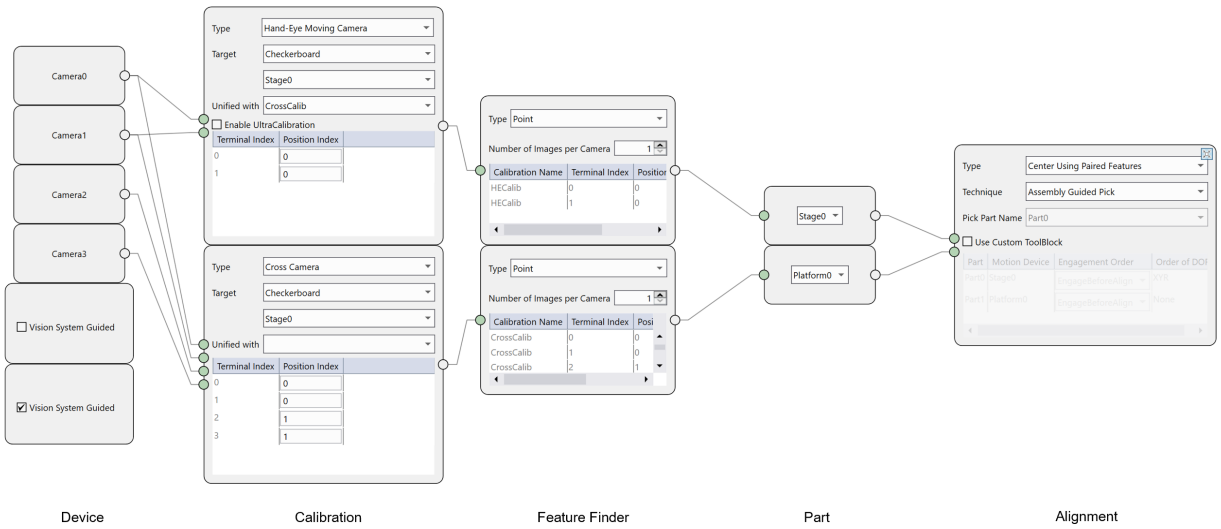
Application V_Assembly Guided Pick

In this application, both parts are placed on stationary platforms, the first part at the picking station, the second part is at the placing station. In calibration time, robot run moving hand-eye calibration over the picking station, and then runs a cross calibration to bridge two stations. In run time, the vision system calculates the target pose for robot to pick up the part at correct position. Then the part is moved to the placing station where the two parts will be assembled together.

Category	Item	Stage1 Parameter	Stage2 Parameter
Part	Quantity	1	1
Camera	Camera Quantity	2	2
	Camera Shuttling	/	/
Hand-eye Calibration	Calibration Type	Moving Camera	/
	Calibration Target	Checkerboard	/
	Vision Guided or Motion Guided	Vision Guided	/
Cross Calibration	Type	/	Cross Station Calibration
	Calibration Target	/	Checkerboard

Category	Item	Stage1 Parameter	Stage2 Parameter
Feature Finding	Feature Type	2 Point	2 Point
Alignment	Align Technique	Assembly Guided Pick	
	Align Type	Center Using Paired Features	

The configuration of this application is as follow:

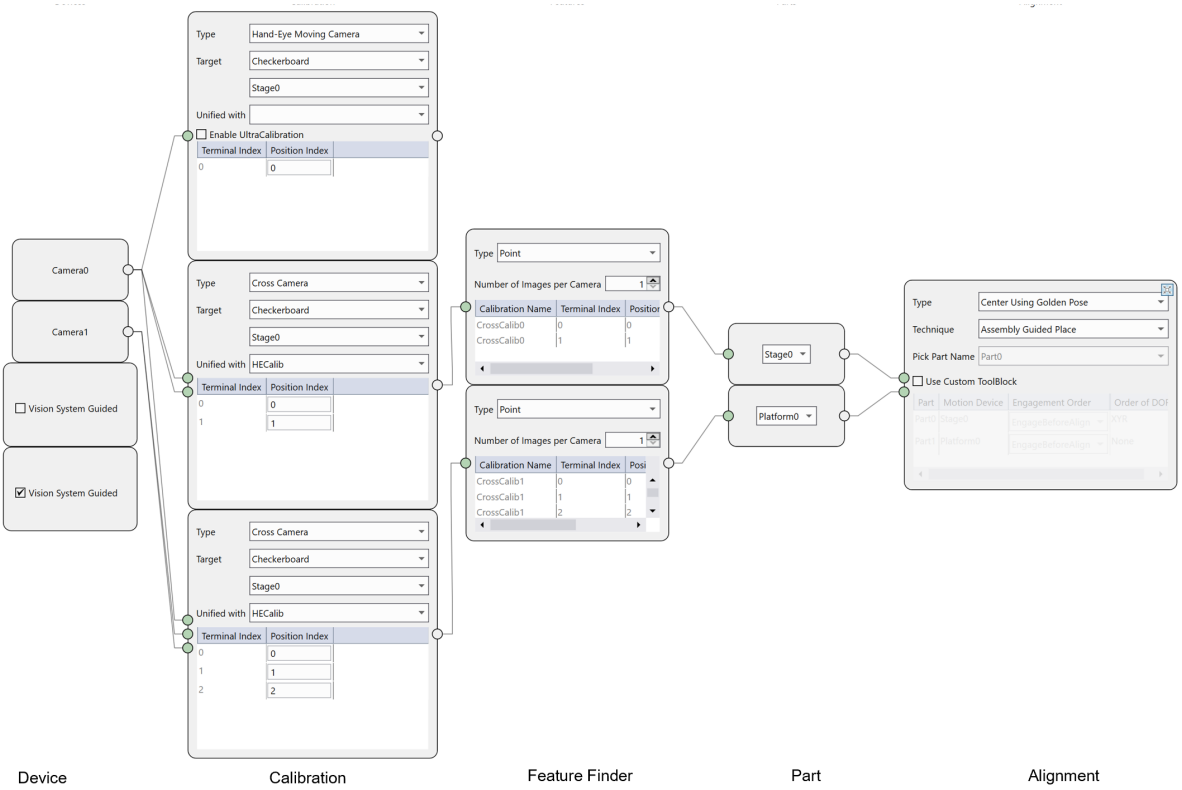


Application VI_Assembly Guided Place

In this application, both parts are placed on stationary platforms, the first part is the picking station, second part is at the placing station. During calibration, robot runs a moving hand-eye calibration over the picking station, and then runs a cross calibration to bridge two stations. In run time, the vision system calculates the target pose of robot for it to adjust its position after blindly picking up the first part. The part is then moved to the second part's side where the two parts will be assembled.

Category	Item	Stage1 Parameter	Stage2 Parameter
Part	Quantity	1	1
Camera	Camera Quantity	1	1
	Camera Shuttling	Pos0, Pos1	Pos0, Pos1
Hand-eye Calibration	Calibration Type	Moving Camera	/
	Calibration Target	Checkerboard	/
	Vision Guided or Motion Guided	Vision Guided	/
Cross Calibration	Type	Camera Shuttling	Mixed Cross Calibration
	Calibration Target	Checkerboard	Checkerboard
Feature Finding	Feature Type	2 Point	2 Point
Alignment	Align Technique	Assembly Guided Place	
	Align Type	Center Using Golden Pose	

The configuration of this application is as follow:

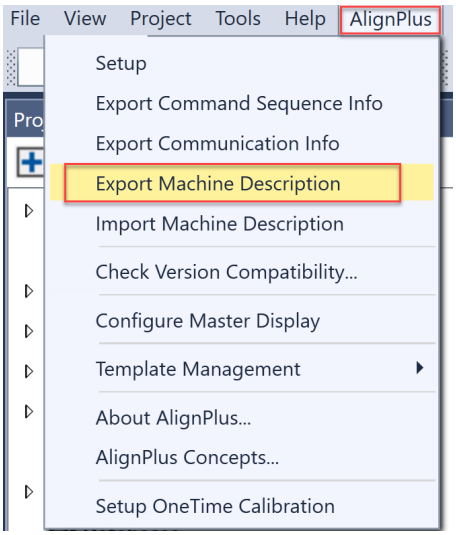


Export and Import Wizard Configuration

After an application is generated, its configuration remains in the Configuration Wizard for users to review, modify, or rerun. For users who would like to generate a new project based on a previous project's configurations, Machine Description export function can help on this.

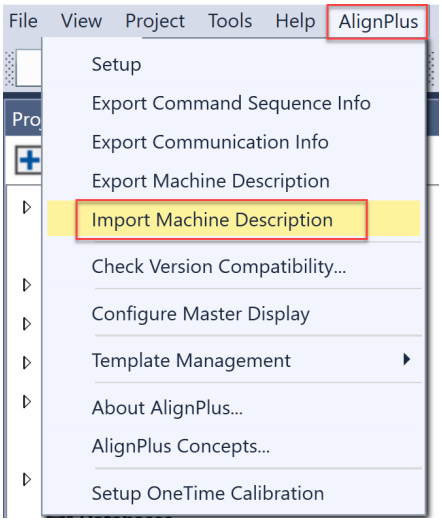
Export wizard configuration

Click "AlignPlus" menu and select "Export Machine Description" option, there would be a pop-up dialog asking you to provide a location for an .json file in which the wizard configuration will be saved.



Import wizard configuration

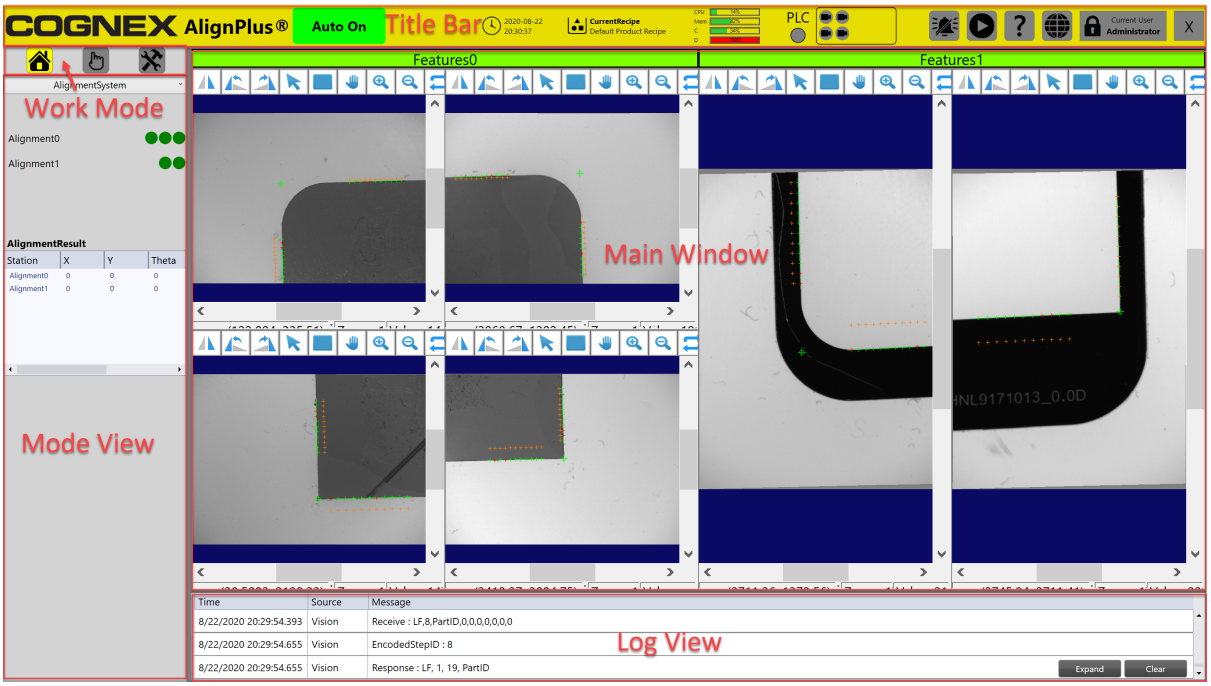
Create a new empty AlignPlus project first, then click its "AlignPlus" menu and select the "Import Machine Description" option. There will be a pop-up dialog looking for the .json file that you would like to import. Navigate and select the previously exported .json file and click "OK" , then the configuration file will be imported.



Click "Setup" option under "AlignPlus" menu to check the imported wizard configuration and modify it to meet the new project's requirements. After the modification, run the wizard, then the new project will be generated.

Home Page

Main user interface is the default interface when AlignPlus program starts. It's consisted of five areas: title bar, work mode, mode view, main window and log view.



Title Bar

Title bar monitors hardware and system status, switches online/offline status, enables language and user change, provides functions such as image play back, etc. For more information, please refer to Title Bar on page 50

Work Mode

AlignPlus program is always in one of the of three modes: auto mode, manual mode and setup mode. Work mode provides three buttons to switch.

	Auto mode	Under auto mode, program runs vision tasks automatically in response to commands received from external devices.
	Manual mode	In manual mode, user can manually trigger program to run vision tasks such as calibration, feature finding or pose computing. Manual mode only works in offline status. When this mode is chosen, program will pop up a dialog to confirm whether you want to switch to offline.
	Setup Mode	Setup mode is used to set up cameras, lights, calibration parameters, vision tasks and side functions (such as image saving).

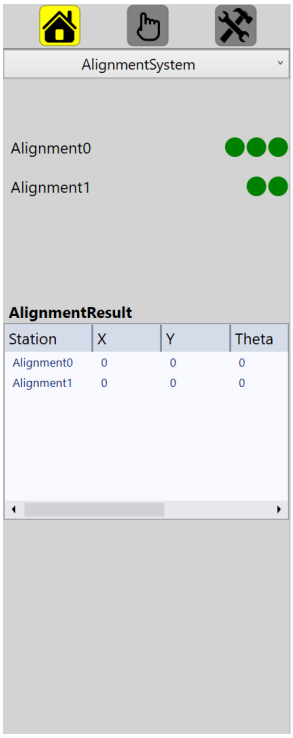
To setup a newly generated program, you should first go to setup mode to setup cameras, lights, calibration and feature finding parameters, second go to manual mode to trigger the program to finish calibration, feature finder training, then come back to setup mode to save product recipe, and at last go to auto mode to make program ready to run automatically.

Mode View

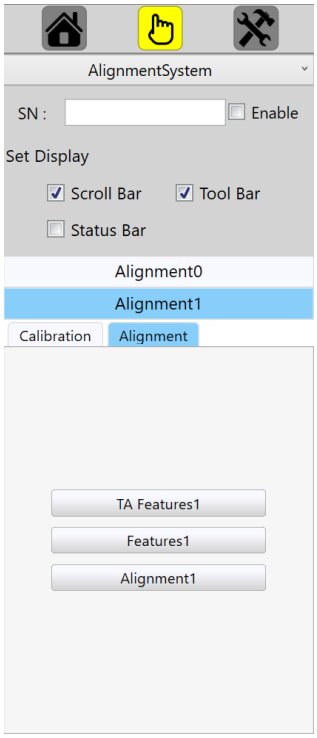
Mode view will change according to different work mode. When it's auto mode, mode view will monitor each stations' OK/NG status, and the X, Y, Theta values that vision feeds back to motion devices. When it's manual mode, mode view will show many command buttons that allows user to trigger the program to run calibration, feature finding or pose computing. When

it's setup mode, mode view will show a navigation tree that user can quickly navigate each page and setup the program from up to down.

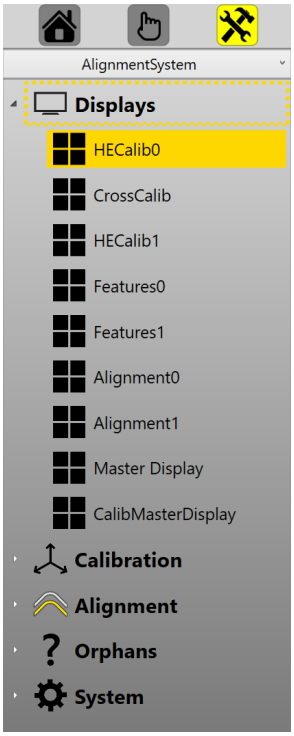
Auto Mode



Manual Mode



Setup Mode

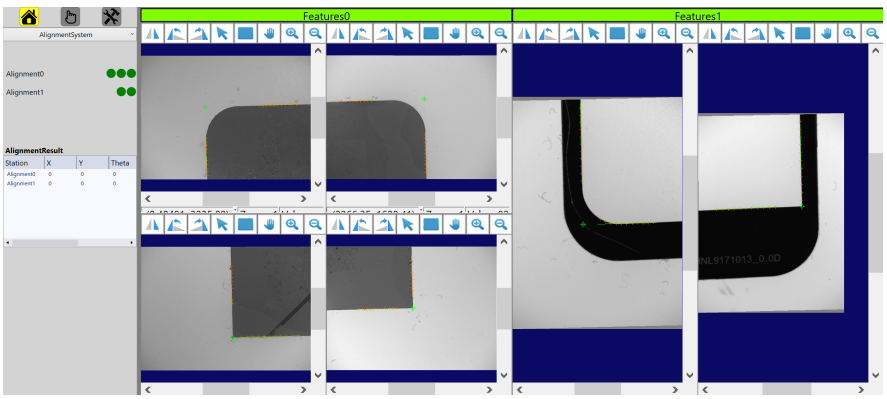


Main Window

Main window displays different contents under different mode.

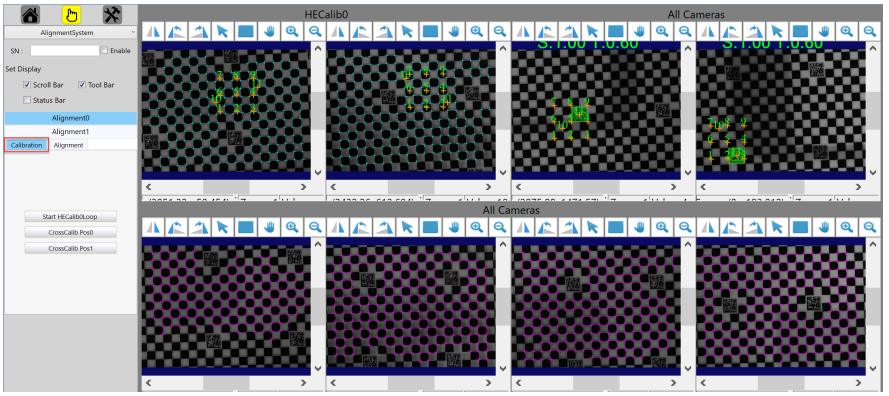
1. In auto mode, it displays the alignment master display which shows all stations' feature finding result images in one window.

- Alignment Master Display

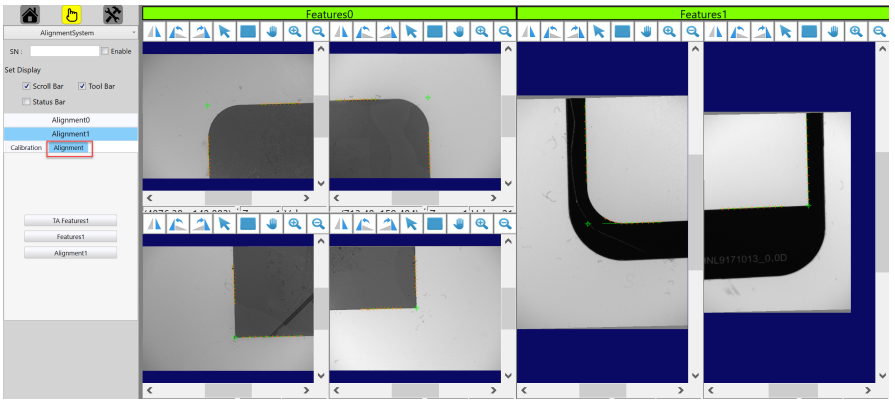


2. In manual mode, it shows either alignment master display or calibration master display depending on whether user is choosing calibration or alignment button group on work mode. Like alignment master display, calibration master display shows all stations' calibration images in one window.

- Calibration Master Display



- Alignment Master Display

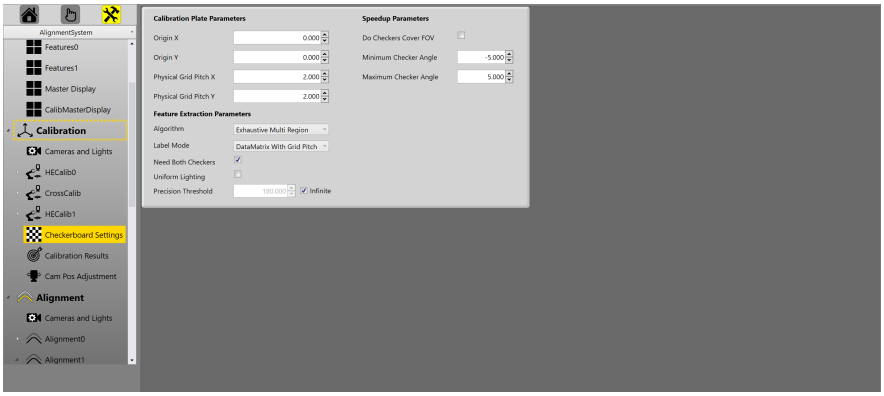


3. In setup mode, main window shows each calibration or feature finding image display separately on different pages, so that user can monitor each station's images in big window.

- single feature finder display

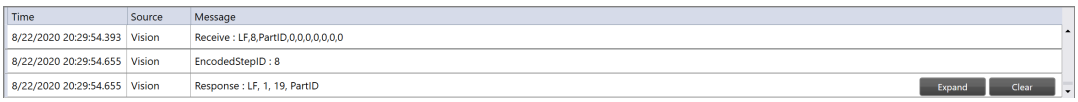
It also shows corresponding pages if you're setting up other parameters.

- calibration parameter display

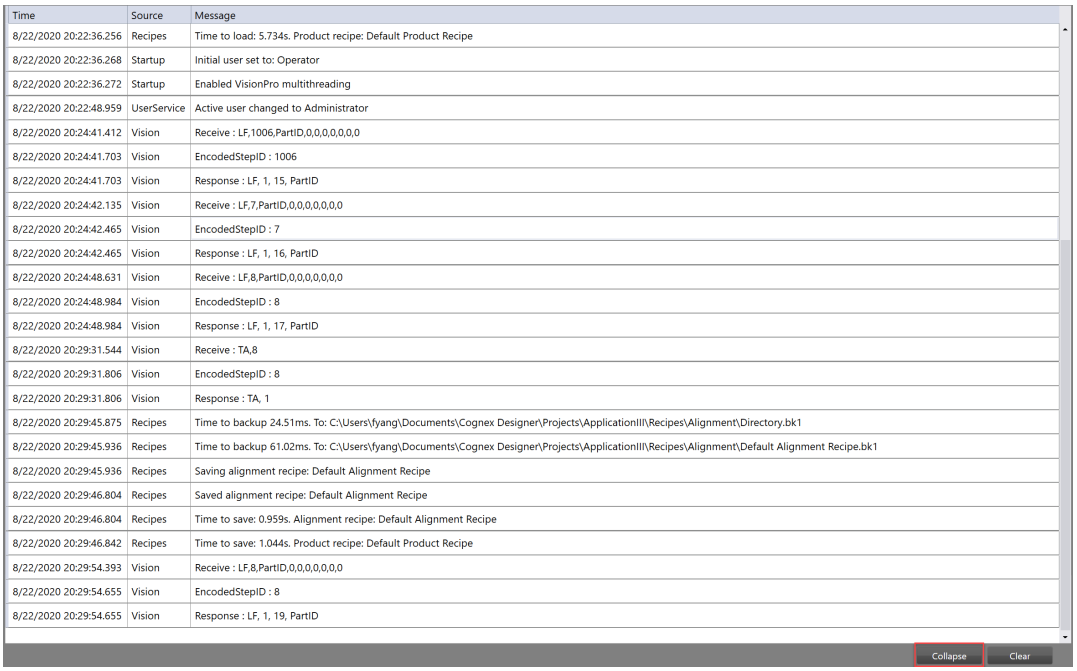


Log View

Log view lies at the button of main user interface. It displays latest system log which tracks operational changes, records input commands and output results in the format of string.



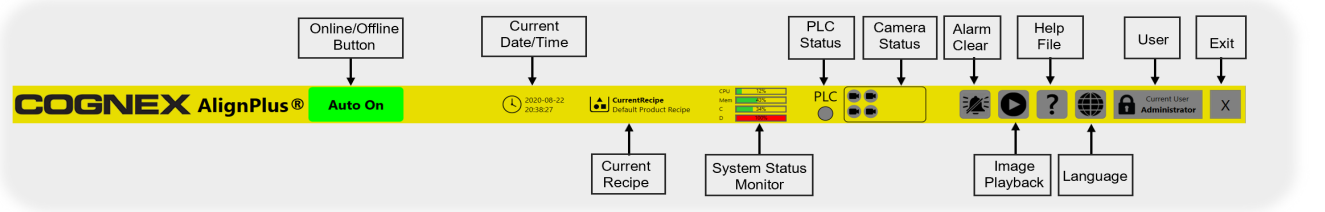
You can click **Expand** button to expand log view to main window so that more information would be displayed, and click **Collapse** to withdraw it back to original area. It is suggested to collapse it before switching to other pages so that the latest log messages will not be blocked.



The system log displayed here is the same as Message Viewer on page 159and would be saved into .csv file as configured in Logging Setup on page 172.

Title Bar

Title bar shows the program's online/offline status, current time and selected recipe, monitors hardware and system status, and provides some frequently used functions such as image play back, user change, language change, etc.



Auto On

Auto on/off button shows whether program is online or offline. If it is online, the button will be green, otherwise red. Click the button can switch between online and offline status.

- **Online**

  Online status is used for production. When it's online, the program will be on standby waiting for commands from external devices. Every time when commands are received, the program will process vision tasks and feedback the results automatically if the requested tasks are not busy.

  The program will automatically change to online mode when "  " is chosen in work mode panel.

- **Offline**

  Offline status is used for machine setup and program testing. When it's offline, user can set up cameras, run calibrations, test feature finders or verify other functions.

  The program will automatically change to offline status when "  " or "  " is chosen in work mode panel.

Status Monitor

Current Date/Time

A clock shows current system date and time.

Current Recipe

Shows current product recipe name. A product recipe contains acquisition information, calibration data and alignment data. The first created recipe name is "Default Product Recipe", you can add your own recipes with more meaningful names. For more information about recipe, please refer to Product Recipe on page 160.

System Status Monitor

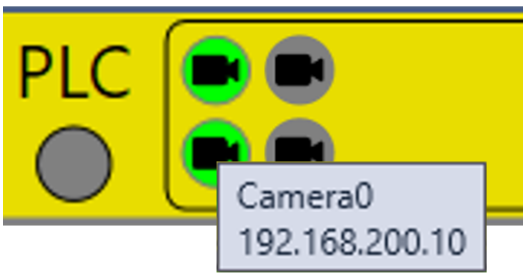
System status monitor shows status of the computer on which AlignPlus program is running including CPU utilization, memory usage and disc capacity. When they are under safe values, the status bars will be green, otherwise will be orange or red depending on degrees above/below safe values.

PLC Status

Shows whether the PLC (Programmable logic controller) which controls the machine is connected to AlignPlus program. Green shows it is well connected and ready to communicate. Gray indicates it is not connected, in this case you need to first check hardware connection, then go to Communication on page 170 to check whether PLC's IP address and port number are configured correctly or not.

Camera Status

Camera status monitors whether all physical cameras are connected to AlignPlus program. Each camera icon stands for one camera device set in the configuration wizard. When mouse-hovering over an icon, there will be a tip showing its camera name, and the camera's IP address or serial number if it's connected.



When the icon is green, it means that camera is connected, otherwise disconnected. After the camera is reconnected, the program will automatically put it online and turn the corresponding camera icon into green.

Alarm Acknowledge

When an alarm event is raised, the title bar will flash between red and gray to remind user to check on alarm. After the alarm is checked, one can click "  " icon to clear the alarm so that title bar will stop flashing. For more information about alarm, please refer to Alarms and Status on page 158.

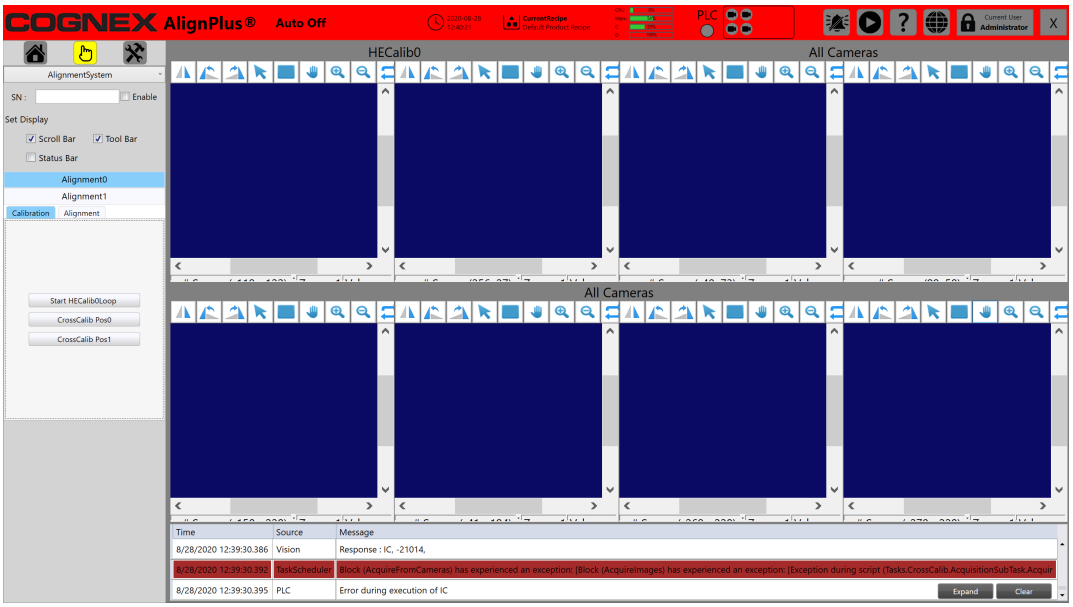
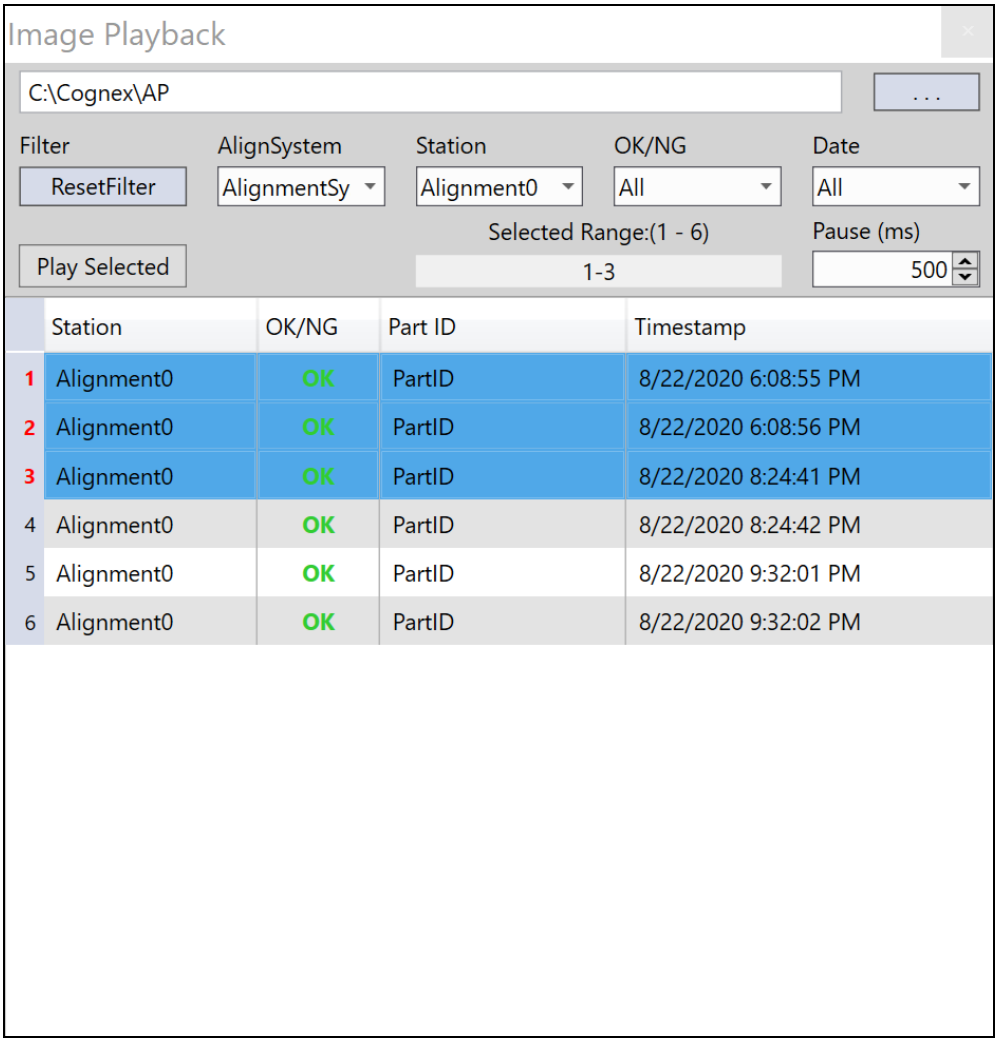


Image Play Back

Image playback function reruns saved raw images as if they were triggered though commands. To play back images, you need to first save some raw images using Image Saving on page 176 function.

Image play back is only available in manual and setup mode. Click "  " to open the play back dialog and follow the steps below:



1. Choose the directory where the raw images are saved.
2. Use filter to specify the alignment system, the station, the part disposition (OK/NG/Both), and time based filters. Once filters are applied, all qualified images will be list out in the table below. Click **ResetFilter** if you want to revert to default filters.
3. Select the images you want to play back, using Ctrl + or Shift + if there are more than one items to select or Ctrl + A to choose all. The **Selected Range** will display the indexes of selected images.
4. Click **Play Selected**, then the image play back will run through all those images from top to down, output each result and display every graphics at the same time.
5. Increase **Pause** time if you want to monitor each result and its images and graphics in a slower speed.

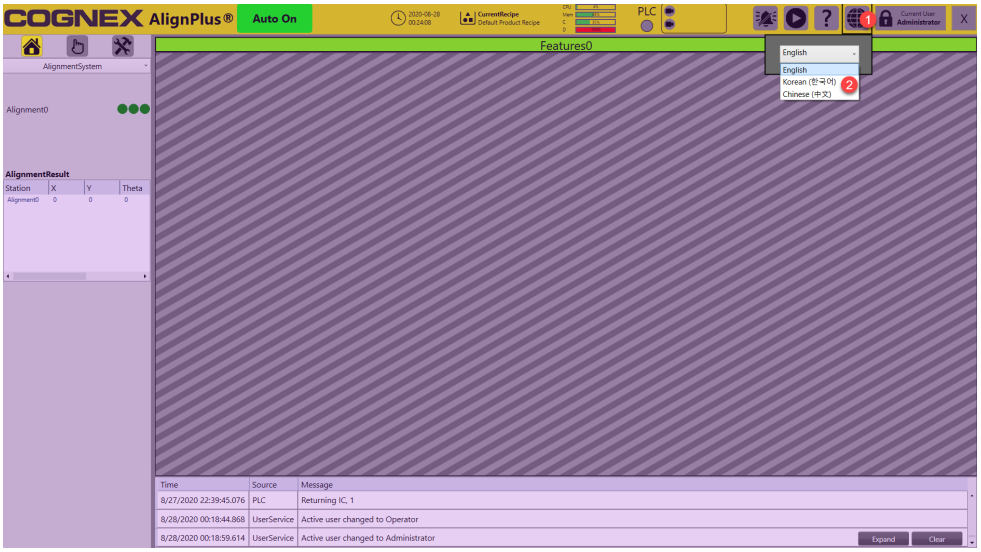
Help

Displays all available help files.

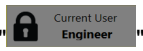
Language Change

Click "  " icon to choose the language you want.

AlignPlus supports three languages: English, Chinese and Korean.

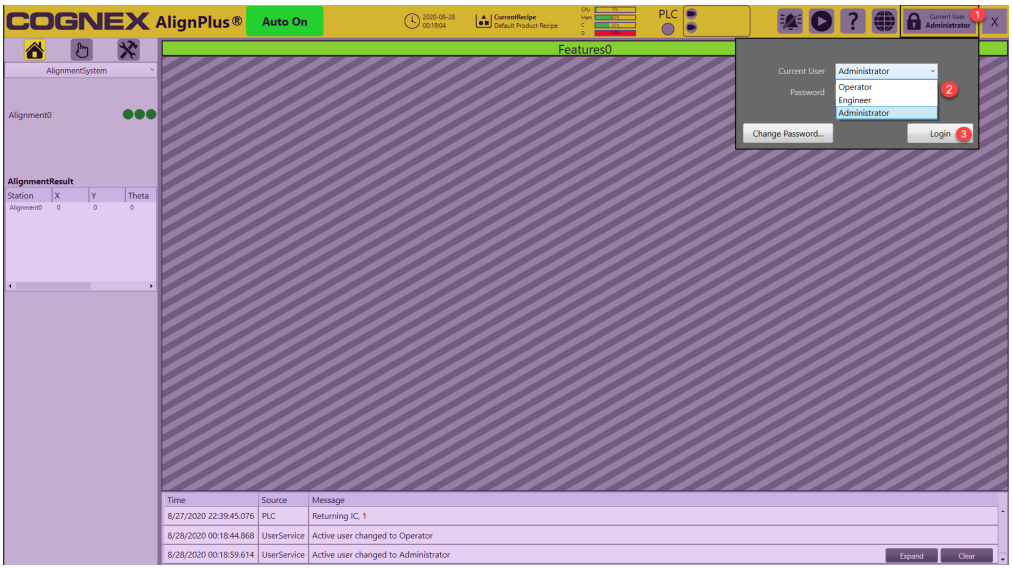


User Change

Click " 

" icon, and choose user from the three options: Administrator, Engineer or Operator. The default is **Operator**.

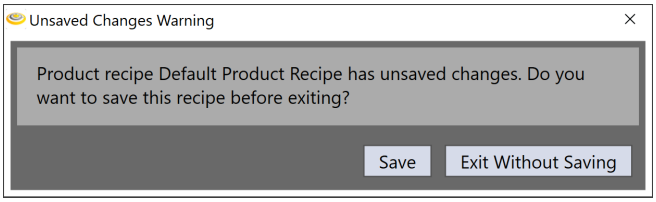
Administrator has full access to all the functions in program. Engineer shares the same access of administrator except setting up feature finders. Operator can monitor program on auto mode, play back images and monitor displays, alarms, and logs in setup mode.



Exit

Click " 

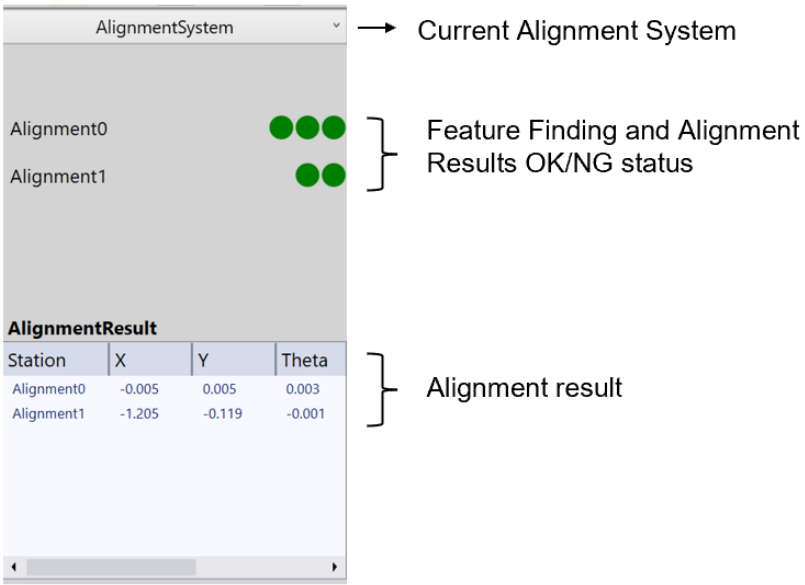
" icon to exit current program. The program will double confirm whether you want to exit. If there is any unsaved recipe, the program will pop up another dialog prompting recipe saving choices. Click "Save" if you want to save, or "Exit Without Saving" if you want to discard the changes.



Auto Mode

Click "  " button to enter auto mode.

In auto mode, the mode view will display result status of alignment stations and their output x, y, theta values.

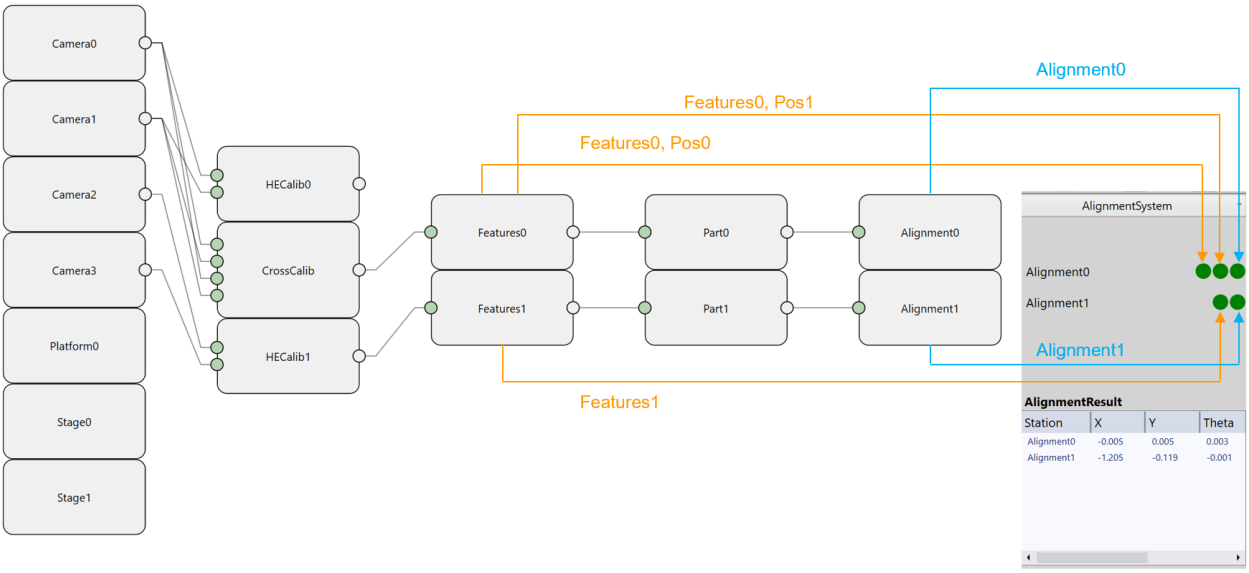


Here is an example project and its auto generated result status on mode view. In this project, there are two pose computers: **Alignment0** and **Alignment1**. Alignment0 uses feature finder **Features0**'s results, and Alignment1 uses **Features1**' results. Features0 needs to acquire images at both position0 and position1, while Features1 needs only to acquire at position0.

Correspondingly, on mode view, For Alignment0, the first light represents whether feature finding of Features0 at position0 is successful or not, the second light shows feature finding at position1 is successful or not, and the third one shows whether the alignment result is within align limit or not.

For Alignment1, the first light indicates whether all features in Features1 at position0 are found successfully, and second one displays whether alignment result of Alignment1 is within align limit or not.

To understand more about how to configure align limit, please refer to Placement Limit Checker on page 185.



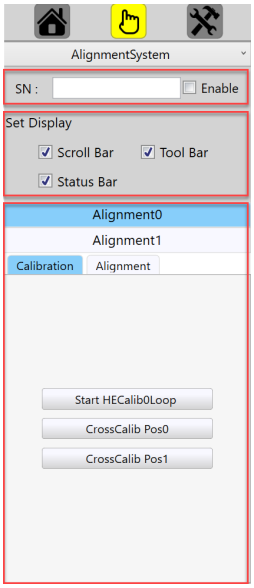
The AlignmentResult part shows each alignment station's **X, Y, Theta** results. These values will be sent to motion device right after it has been calculated out. Note that **Theta** here is in degree.

Manual Mode

Click "  " button to change to manual mode.

In manual mode, you can send commands to run calibration, feature finding or pose computing by clicking the corresponding buttons which are convenient to set up or test the program.

Manual mode control panel has three parts: SN input, display setting, and command buttons.

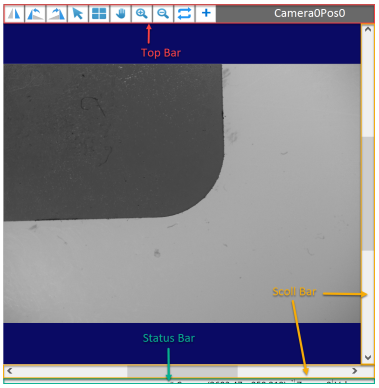


SN Input

Input the current serial number of the part.

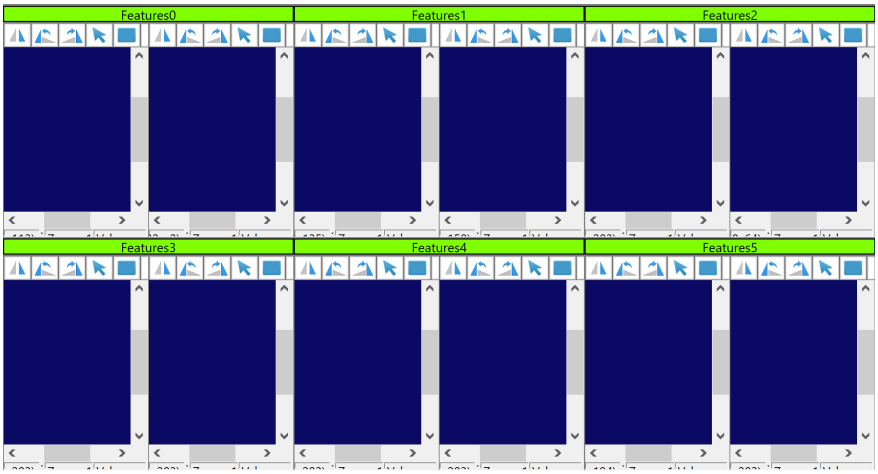
Set Display

Disables or enables scroll bar, top bar or status bar of all image displays.

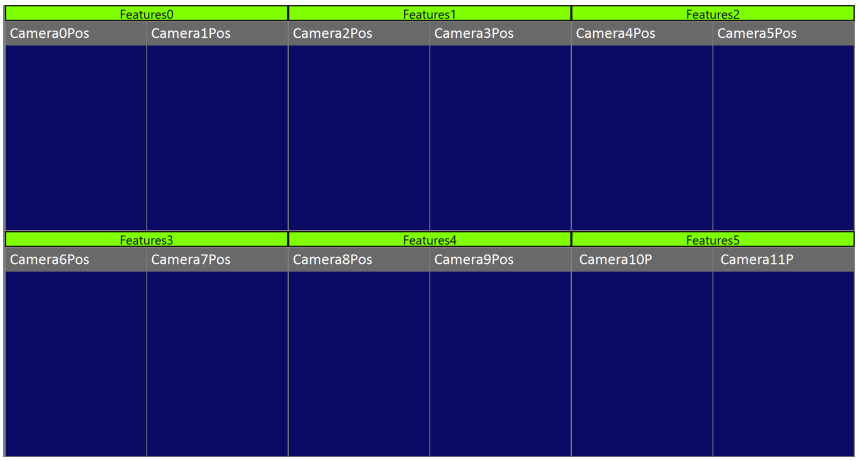


When there are more than two stations in one alignment system, the alignment master display will look very crowded with all those scroll bars, top bars and status bars on. In this case, we can disable these bars to make more spaces for images.

Before disable:



After disable:

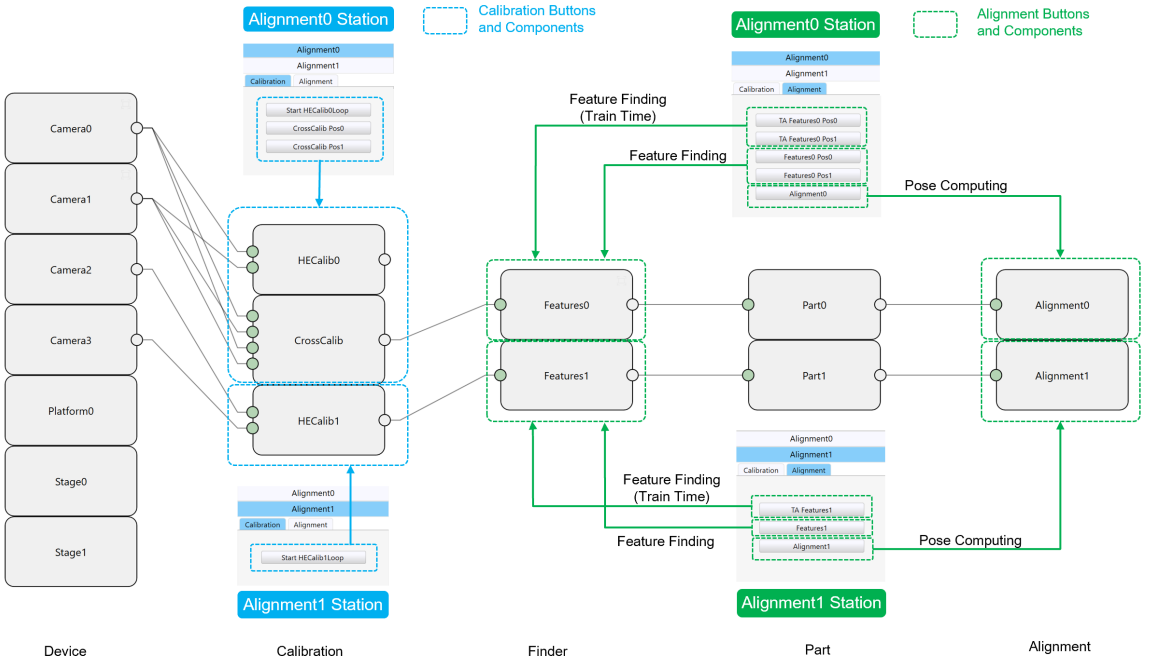


Command Buttons

Button Categories

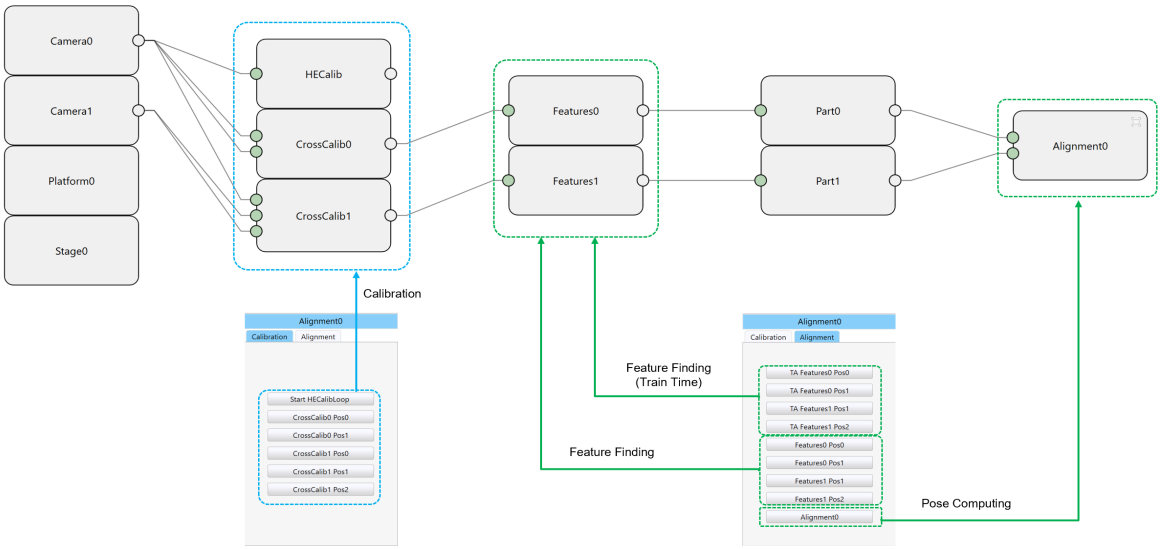
Command button panel provides the necessary buttons to trigger vision tasks including calibrations, feature finders, and pose computers. These buttons are first categorized in terms of alignment stations, then under each station, they are divided into two groups: calibration and alignment. Once calibration group is selected, only calibration buttons under current station will be shown. Or if alignment group is selected, only feature finding train time and run time buttons as well as alignment button will be shown.

Command buttons are all automatically generated based on the configuration wizard. Here is an example of a configuration and its all corresponding command buttons:



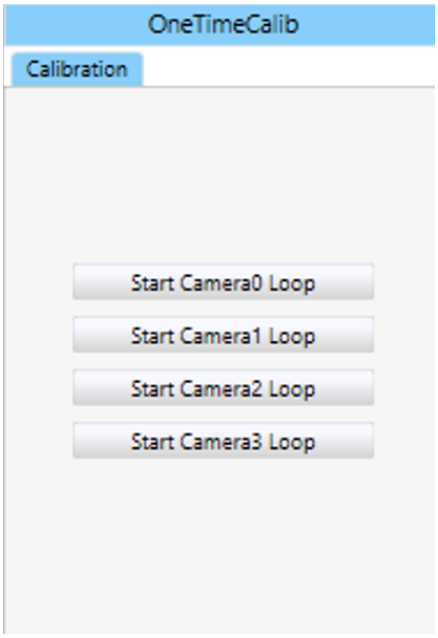
In this example, there are two alignment stations: **Alignment0** and **Alignment1**. In Alignment0, the calibration group includes both **HECalib0** and **CrossCalib** buttons, alignment group includes **Features0**'s train time, run time buttons and alignment0 button. In Alignmen1, the calibration group only has one **HECalib1** button since there is no other calibration in Alignment1 station, the alignment group has **Feature1**'s train time, run time buttons and alignment1 button.

If it is assembly application, there will be only alignment, then all calibration buttons will be put together under calibration group, and all feature finding buttons and one alignment button will be put together under alignment group. Here is another example of it:



During test mode of this assembly program, button **Alignment0** should be only triggered after both Features0 and Features1 have finished running.

Camera Mounted Calibration Buttons



1. The relationship between

Button Commands

Each button has one command attached, when the button is the clicked, the command will be sent to CommandHandler which later will call corresponding tasks. The command follows the same structure as defined in Command String Structure which has CommandKey, EncodedID and certain parameters such as current stage's x, y, theta.

During program test, to get current stage's x, y, theta before the command is sent out, command button panel will call **GetStagePosition** call back function first to get x, y, theta, then compile it into the command string and send it out to CommandHandler. For more information about command button panel call back function, please refer to How to get stage's current pose for manual buttons on page 315.

If **GetStagePosition** is not implemented, then the panel will use default value (0,0,0) for x, y, theta input. This works when x, y, theta does not matter for specific testing, such as: to test whether vision tools find features correctly or not.

Once a command is sent out, the system log will record the command string.

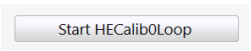
9/1/2020 17:32:35.303	Vision	Receive : LF,1006,PartID,0,0,0,0,0,0,0

Or you can also use command button panel's **CommandSentCallBack** call back function to get the command string.

Button Status

After a button is triggered and command is sent out, the button will be grayed out before the requested task finish running, such as:

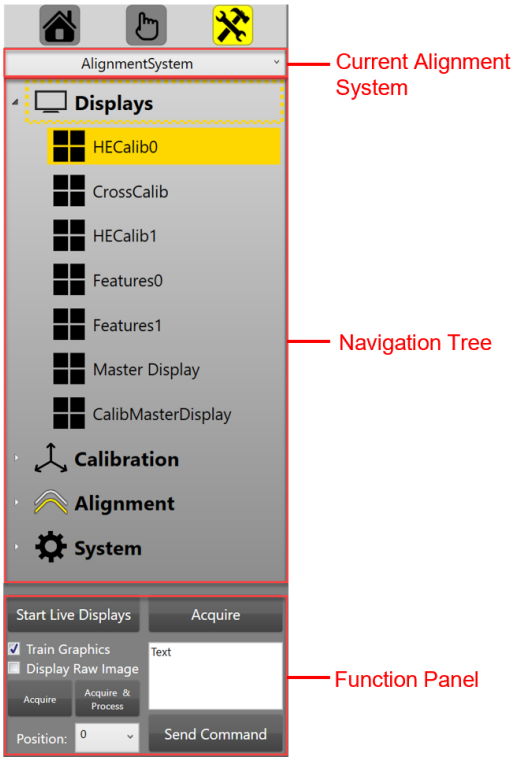
Once the task finishes, the button will turn back to normal:



Setup Mode

Click "  " to enter set up mode.

The setup mode has three blocks: Current Alignment System, Navigation Tree and Function Panel.



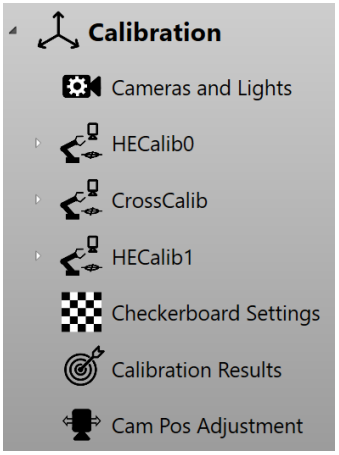
Current alignment system allows user to choose different alignment systems. Navigation tree displays all settings in a top down tree structure, it has four parts: **Displays**, **Calibration**, **Alignment** and **System**. Function panel appears for relevant selections in the navigation tree. It provides live display, image acquire and process functions.

Displays

Display each calibration, feature finder' images and graphics in separate Multiple Display on page 62, or in one master calibration or alignment display.

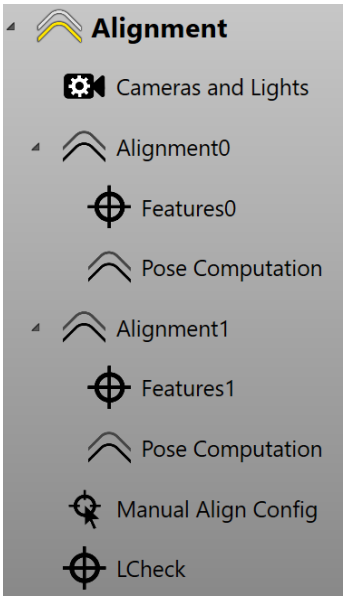
Calibration

Calibration category includes cameras and lights settings, each calibration's settings, checkerboard settings, calibration results and camera pose adjustment function. See more information at Calibration Navigation on page 71.



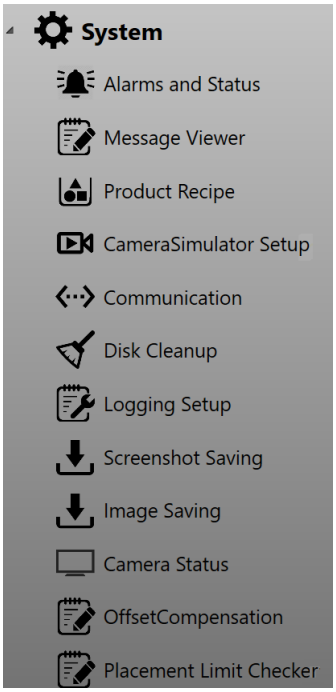
Alignment

Alignment category includes camera and lights setting for alignment, each feature finder's vision tool setting, pose computation if custom pose computer is used, manual alignment configuration and LCheck(length check between two features) function. See more information at Aligment Navigation on page 107.



System

System category contains settings such as recipe, image saving, align limit checking, etc. See more information at Alarms and Status on page 158 and the following topics.

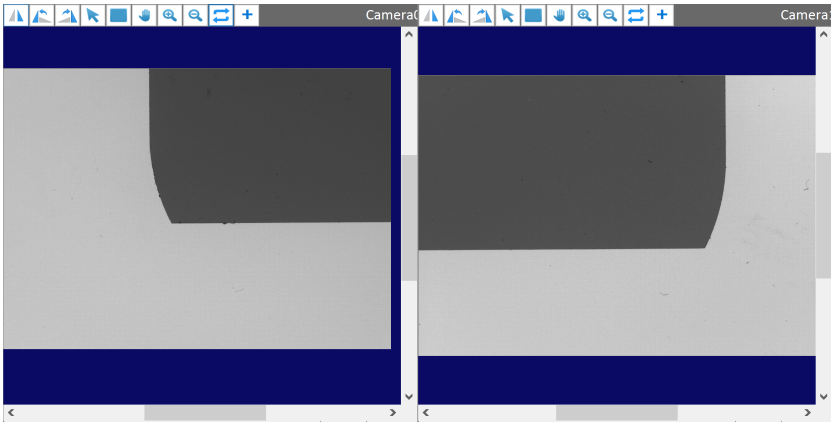


Display

Multiple Display

Multiple display is an UI control in AlignPlus which can display all images and graphics for one calibration or feature finder. In AlignPlus program, each calibration or feature finder in the configuration wizard will have one independent multiple display under **Displays** category of navigation tree in setup mode.

Here is an example of one feature finder display:

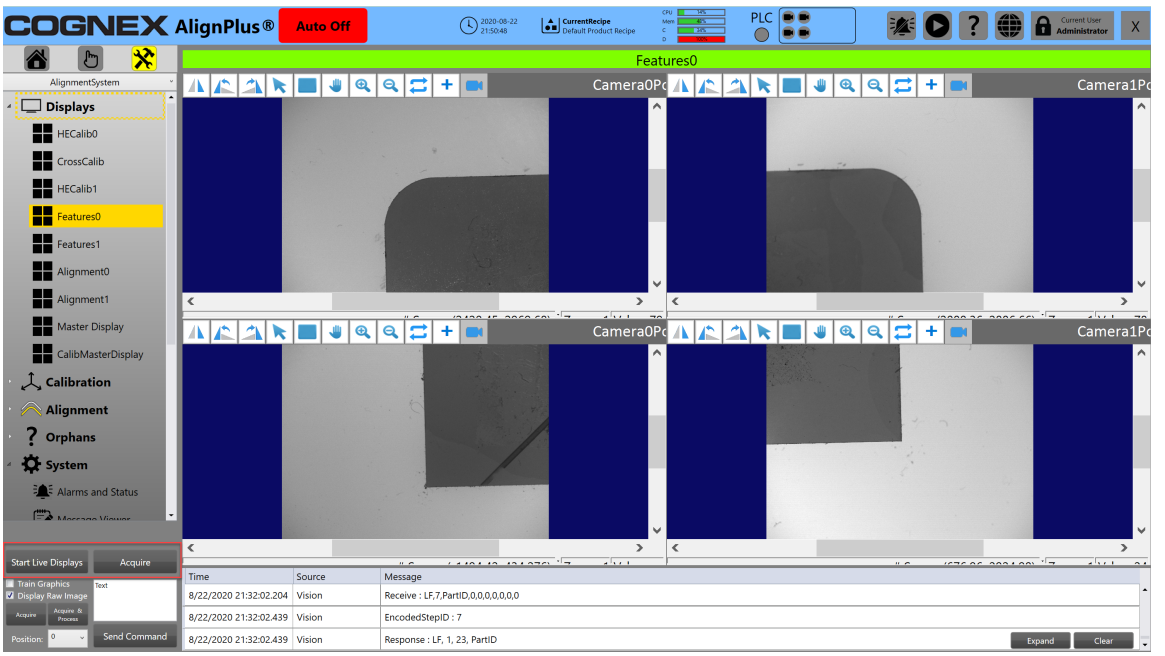


Item	Description
	Mirror image horizontally
	Rotate image 90 degrees to left side
	Rotate image 90 degrees to right side

Item	Description
	Choose pointer
	Show only current image in MultipleDisplay, after it is clicked, the icon will be come , by clicking it again, it will change back to and show all images
	Pan image
	Zoom in image
	Zoom out image
	Reset all settings
	Show cross line centering at image center.

Live Image or Acquire Once

Multiple display goes with function panel in setup mode which provides live or acquire once function.



- Live Image

  Choose one multiple display, then click **Start Live Display** button to trigger live image display for selected display, once it is in live mode, the button will change to **Stop Live Display** for you to stop anytime.

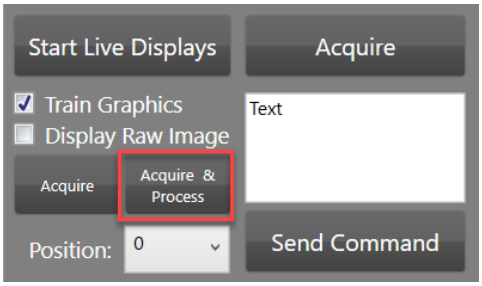
- Acquire

  Choose one multiple display, then click **Acquire** to trigger selected display to acquire images once from collected cameras.

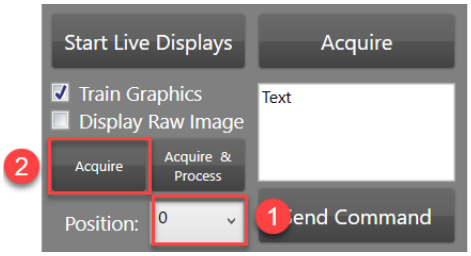
Image Acquire and Process

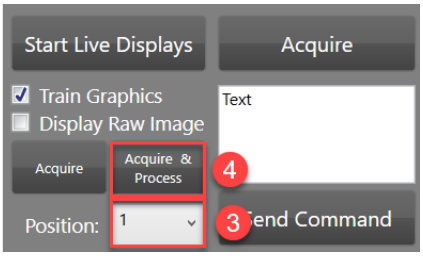
Without sending a command, a calibration or feature finder task can be triggered using images acquire and/or process function in functional panel under multiple display.

If current multiple display only needs to acquire images at one position, then just click **Acquire & Process** to run trigger the calibration or feature finder task behind to acquire images and run vision task.



If current multiple display requires camera shuttling, then choose a position index first and click **Acquire** button if it's not the last position and click **Acquire & Process** at the last position. Take a cross calibration display for example, it needs to acquire images first at position0, and acquire and process at position1 to extract features and run cross calibration calculation, then follow the steps marked below to run cross calibration manually.

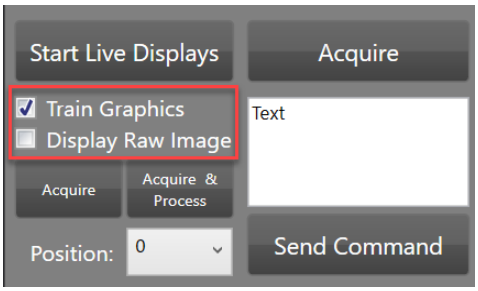




Note: After step 2 above, there will be no images showing on display though they're already acquired at position0, they'll be shown after all images are acquired and processed.

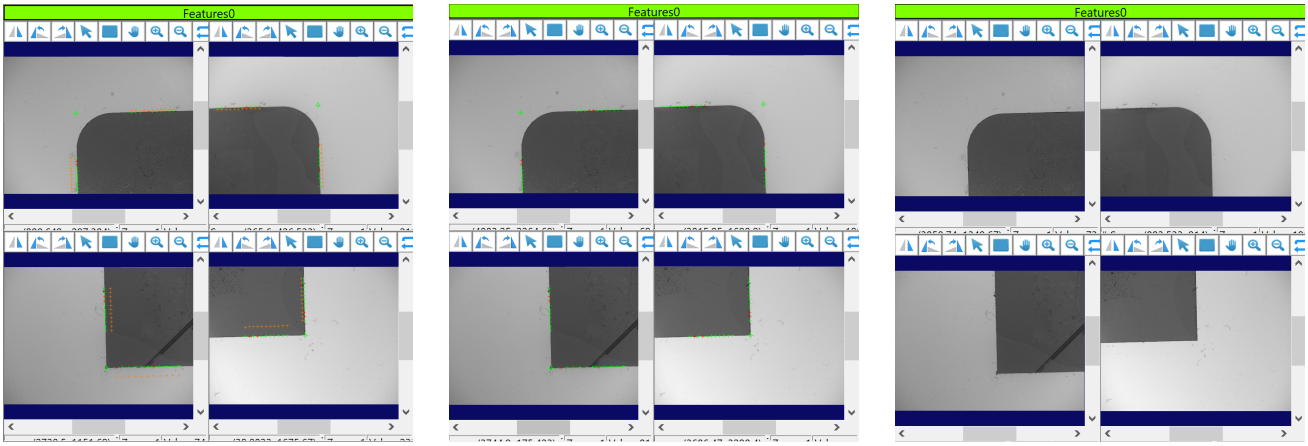
Show Graphics

By default, multiple display will display both trained features and run time features during run time. If you want to hide trained features, check **Train Graphics** off on function panel. Or if you want to show only raw images, check on **Display Raw Image**, then all graphics will be hidden.



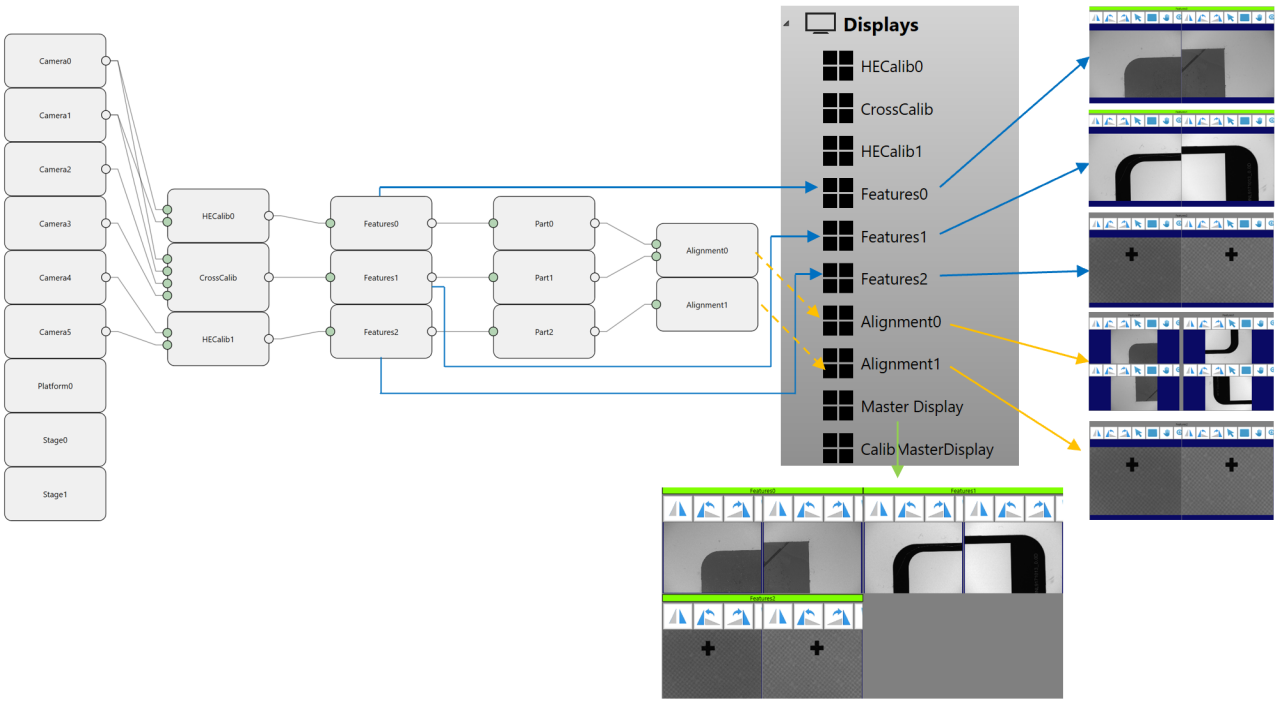
Here is an example of the different graphic displays:

Default	Hide Train Graphics	Display Raw Images



Alignment Master Display

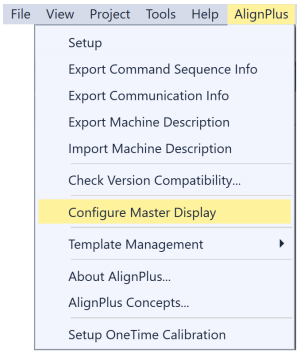
In setup mode, each feature finder component in the configuration wizard will have an individual multiple display, and each alignment component will have one display which merges all its connected finders' multiple displays. Besides that, all feature finder displays from all stations will be merged into one alignment master display. The master display will also be used in manual mode and auto mode.



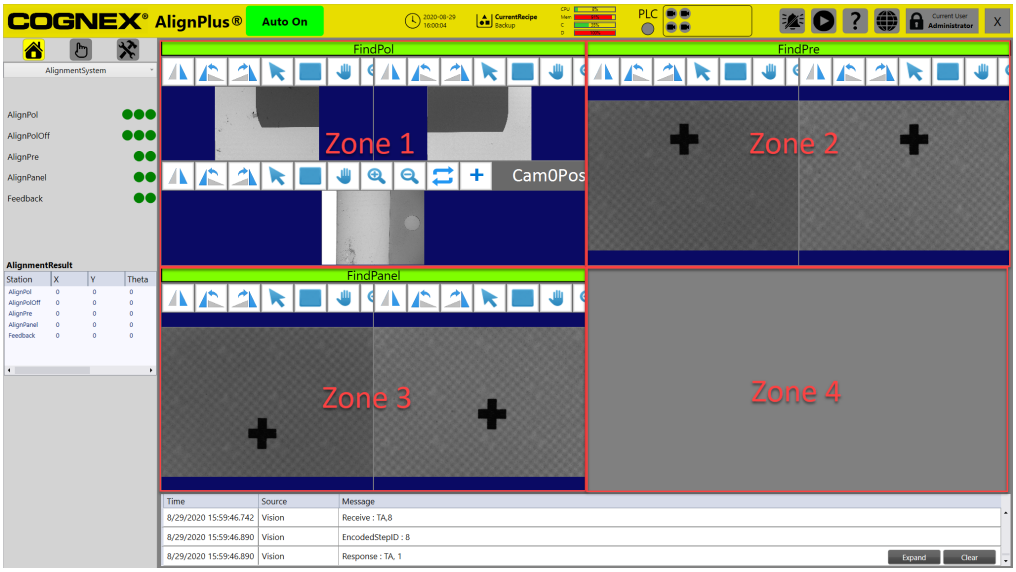
Master Display Layout

The default alignment master display layout may not be well arranged by default. In this case you can use master display configuration function to adjust it.

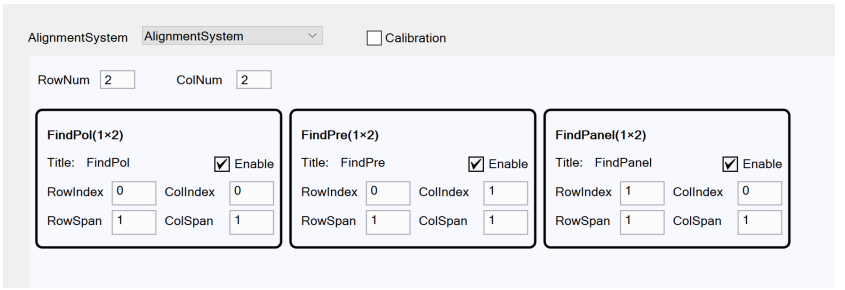
Click **Configure Master Display** option under **AlignPlus** menu to open master display configuration function, it can be opened when program is running in test mode so that you can see the effect quickly after layout is adjusted.



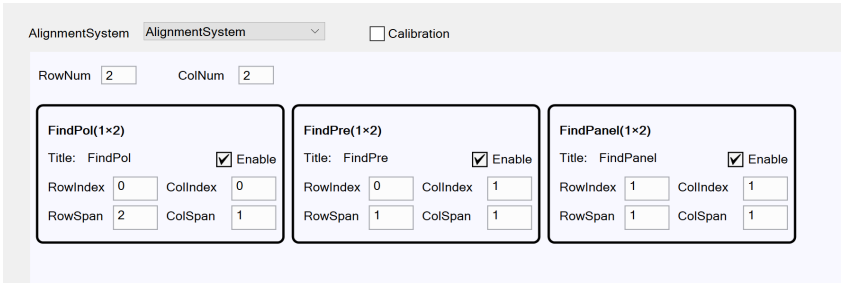
In the example below, the whole display area are divided into a 2X2 zones, **FindPol**, **FindPre** and **FindPanel** occupy the first three zones. However, the last one is left empty.



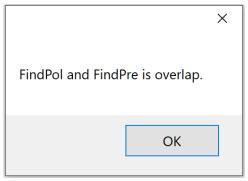
Here is the current display parameters.



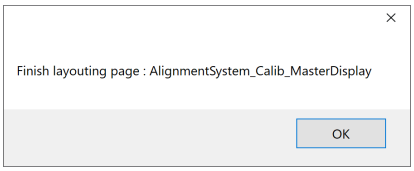
To utilize the last zone, we can swap move FindPanel from zone3 to zone4, and then expand FindPol to the second row. Here is the way to configure it.



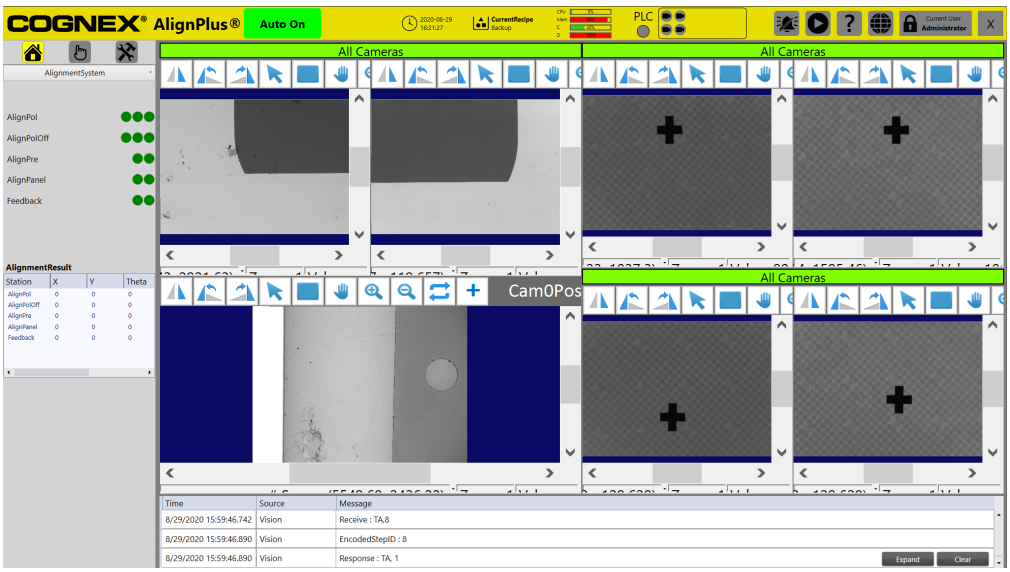
Click **Apply** button at the button of configuration page, there will be an dialog pop up to confirm it's a success or not. If it's not successful, the dialog will have a message to show what the error is such as:



If it says layout finished, then it is successfully applied.

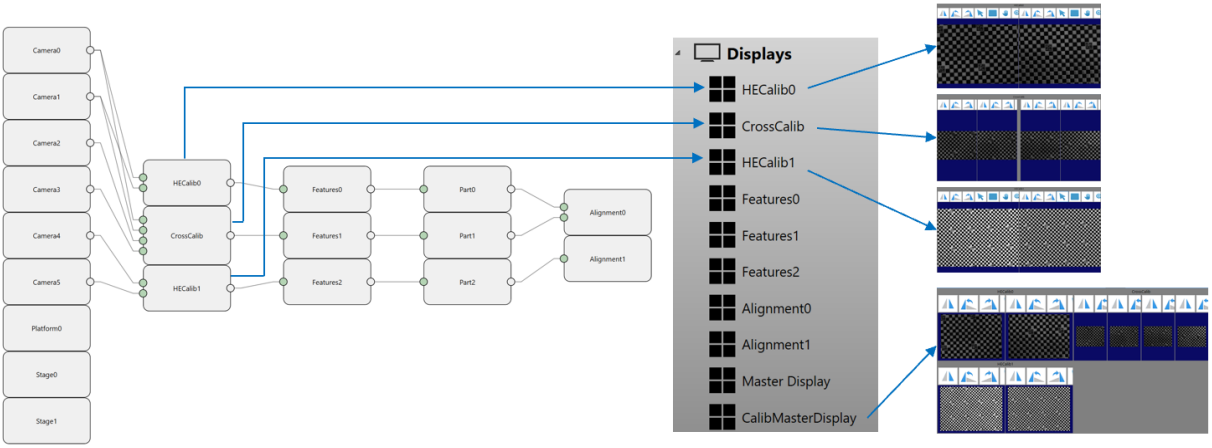


The new alignment master display will look like below after the change.



Calibration Master Display

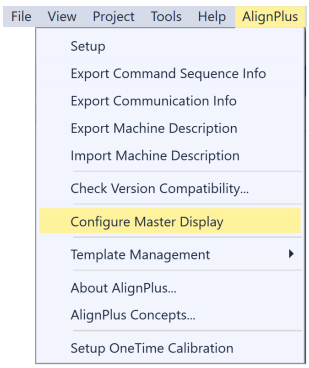
Each calibration component in the configuration wizard will have an individual calibration multiple display in setup mode. Besides that, all those multiple displays will be merged into one window as calibration master display in setup mode. The calibration master display will also be used in manual mode.



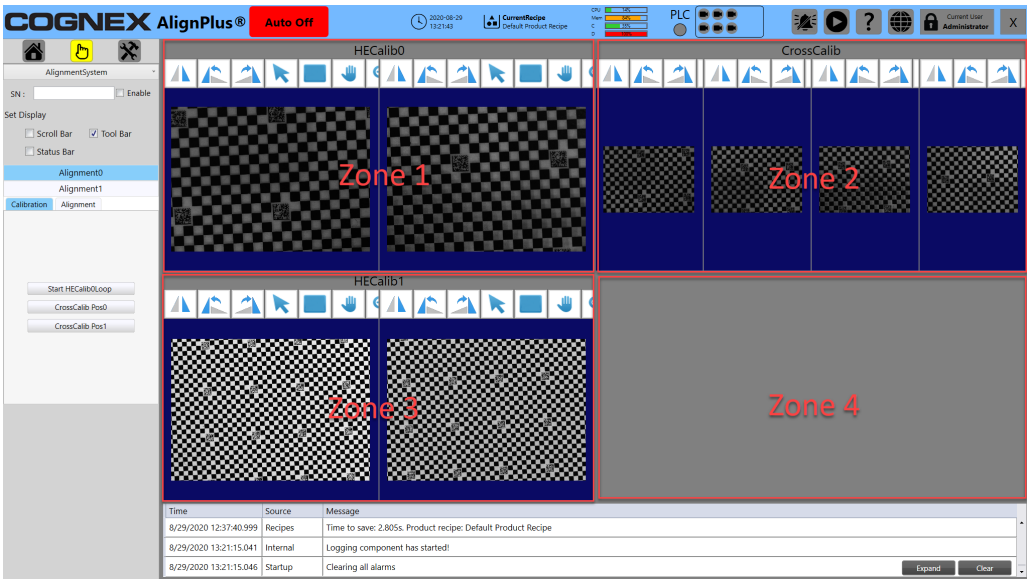
Master Display Layout

The default calibration master display layout may not be well arranged by default. In this case you can use master display configuration function to adjust it.

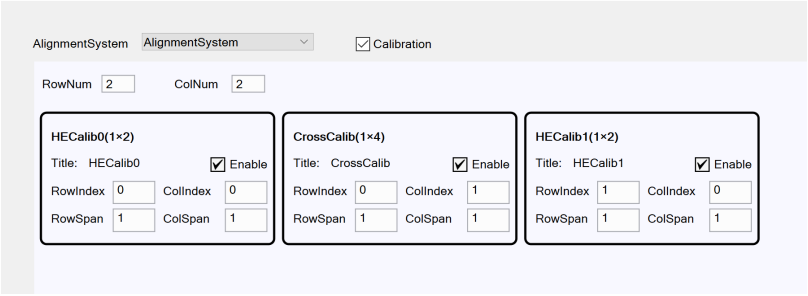
Click **Configure Master Display** option under **AlignPlus** menu to open master display configuration function, it can be opened when program is running in test mode so that you can see the effect quickly after layout is adjusted.



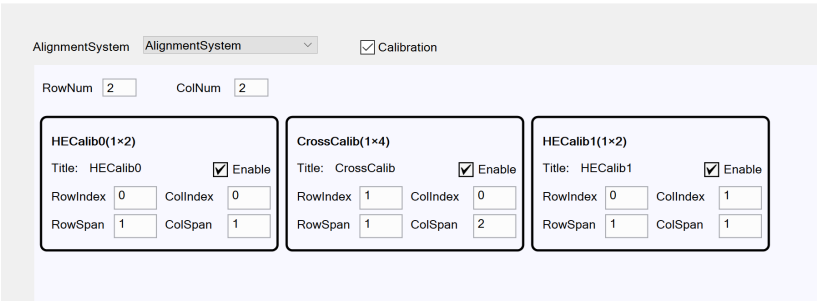
In the configuration page, check on **Calibration** first to enter into calibration display setting.



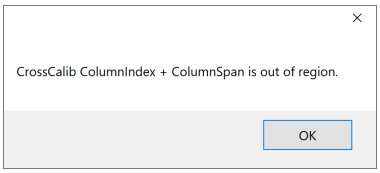
In this example, we can see that the whole display area are divided into a 2X2 zone, **HECalib0** occupies the first zone, whose row and column indexes are both 0, and spans are both 1; **HECalib1** takes the second zone, whose row index is 0 and column index is 1; **CrossCalib** uses the third zone with row and column indexes both as 1. However, the last zone is unused.



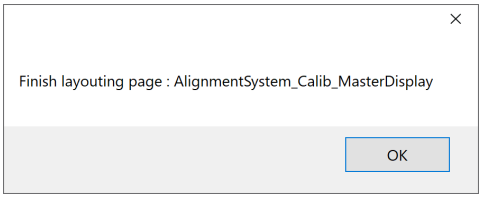
To utilize the last zone, we can swap zones of CrossCalib with HECalib1 display, and then expand CrossCalib display to take both third and fourth zone, here is the way to configure it.



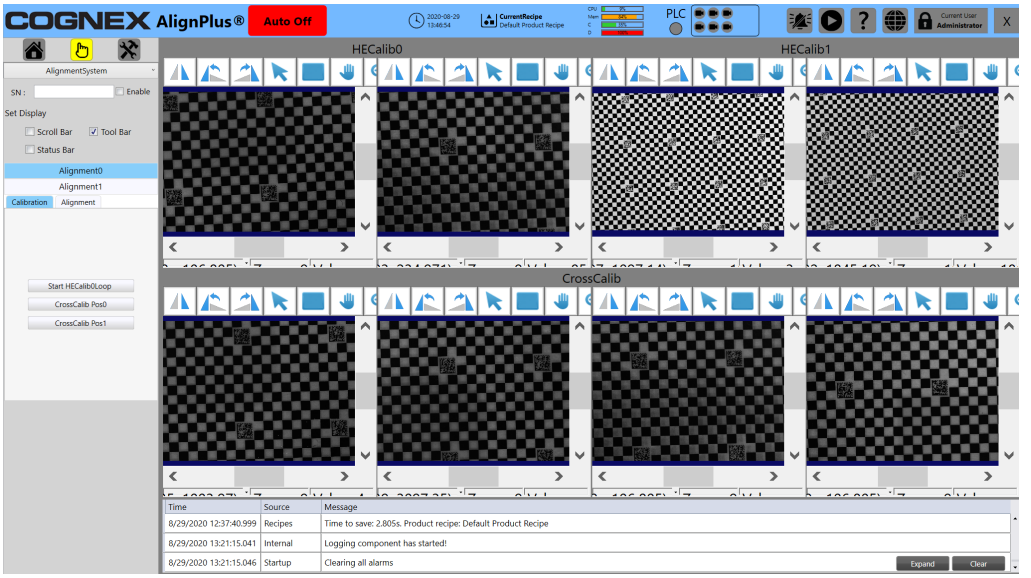
Click **Apply** button at the button of the configuration page, there will be a dialog pop up to confirm it's a success or not. If it is not successful, the dialog will have a message to show what the error is such as:



If it says layout finished, then it is successfully applied.



The new calibration master display will look like below after the change.

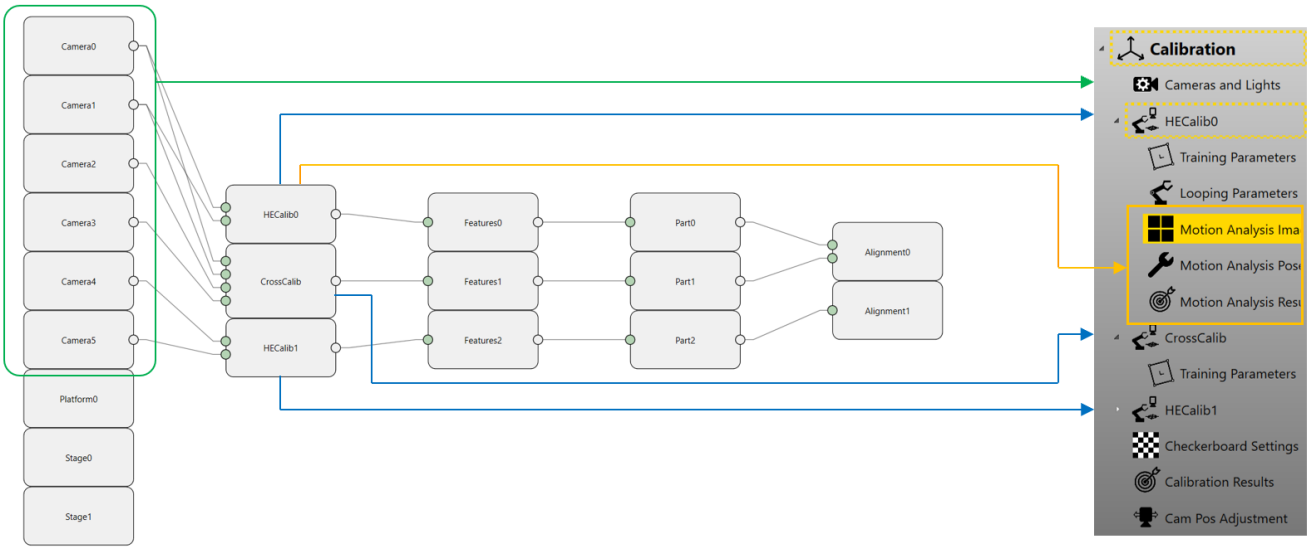

COGNEX AlignPlus®
Auto Off
PLC
Current User
Administrator
AlignmentSystem
SN :
Enable
Set Display
Scroll Bar
Tool Bar
Status Bar
Alignment0
Alignment1
Calibration
Alignment
Start HECalib0Loop
CrossCalib Pos0
CrossCalib Pos1
HECalib0
HECalib1
CrossCalib
Time
Source
Message
8/29/2020 12:37:40.999
Recipes
Time to save: 2.805s. Product recipe: Default Product Recipe
8/29/2020 13:21:15.041
Internal
Logging component has started!
8/29/2020 13:21:15.046
Startup
Clearing all alarms
Expand
Clear

Calibration

Calibration Navigation

Calibration category in navigation tree is used to enter and maintain settings for calibrations which include: Hand-eye calibration and motion analysis, other general calibration settings, checkerboard settings, camera pose adjustment and calibration results.

Here is one example of wizard configuration and its corresponding calibration pages in navigation tree:



Cameras and Lights

This page is used to set camera exposures and light intensities for each calibration. For more information please refer to Cameras and Lights for Calibration on page 75.

Hand-eye Calibration

Each hand-eye calibration component in the configuration will have one group in calibration category which includes settings for hand-eye calibration and related motion analysis. Here are the sub pages under each hand-eye calibration group:

- Training Parameters

  Set parameters for hand-eye calibration
- Looping Parameters

  Set hand-eye calibration looping parameter
- Motion Analysis Image Display

  Display for motion analysis task
- Motion Analysis Pose Gen

  Set parameters for motion analysis

- Motion Analysis Result

  Display results of motion analysis

Cross Calibration

Each cross calibration will have one parameter setting page to set the lens distortion mode. See more information at Cross Calibration Training Parameter on page 101.

Calibration Results

Calibration results collect all calibrations' results, you can check one by one their report data about camera location, overall residual, etc. See more information in Calibration Results on page 104.

Cam Pos Adjustment

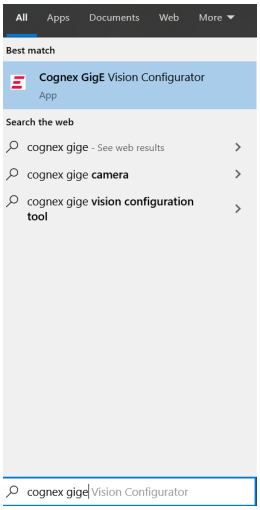
Camera pose adjustment allows user to input camera position offsets instead of recalibrate all cameras during mode change. See more information in Cam Pos Adjustment on page 105.

GigE Camera Configuration

This topic specifies GigE Camera hardware configuration. This should be done before running AlignPlus program.

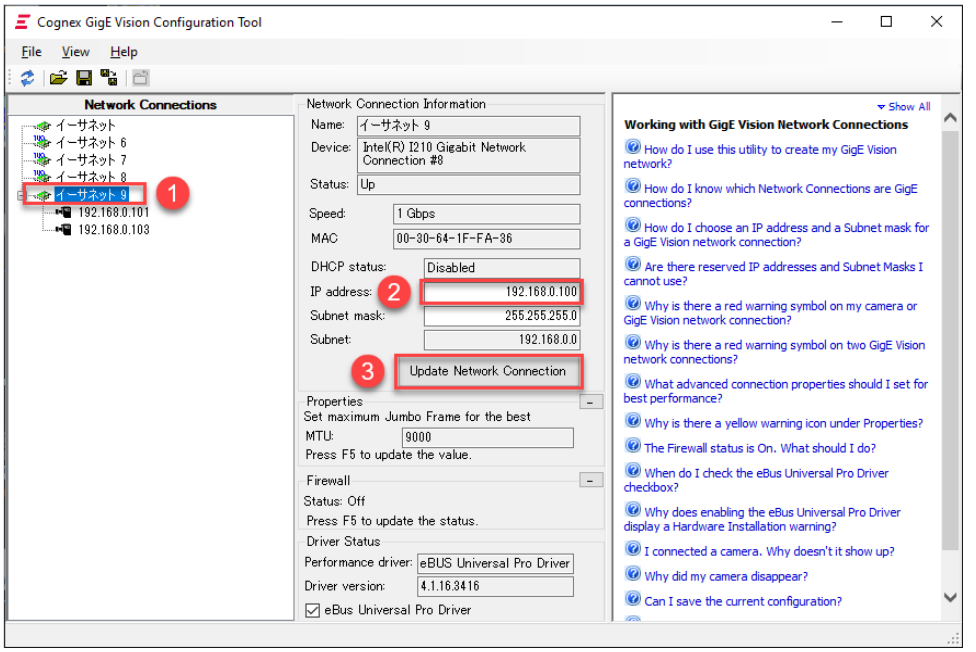
First, make sure that camera is connected to network card whose speed (link up speed) is 1Gpbs since image acquisition requires high speed data exchange, then follow the steps below to set up the camera IP address and configure the network card parameters to make image acquisition smooth and stable.

1. Type "Cognex GigE Vision Configurator" in windows command window and open **GigE Vision Configurator**.

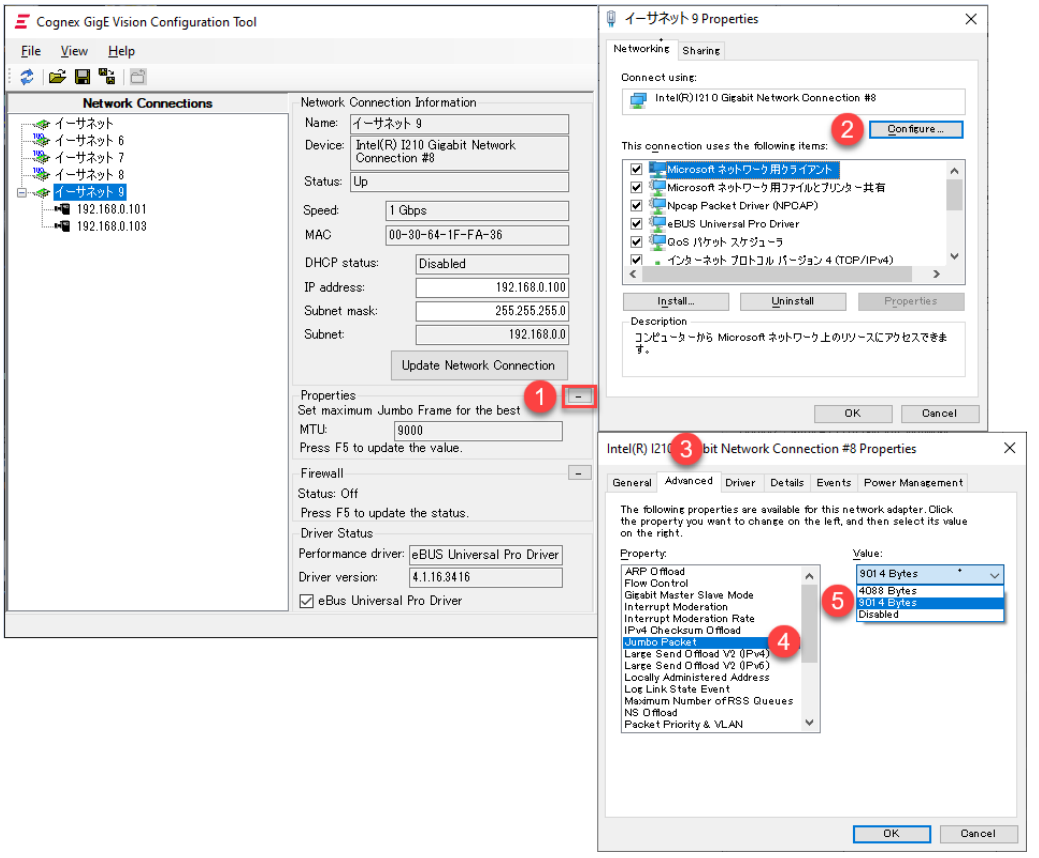


2. In the opened GigE Vision Configurator UI, change camera's IP address or Ethernet card IP address to make them in the same subnet.

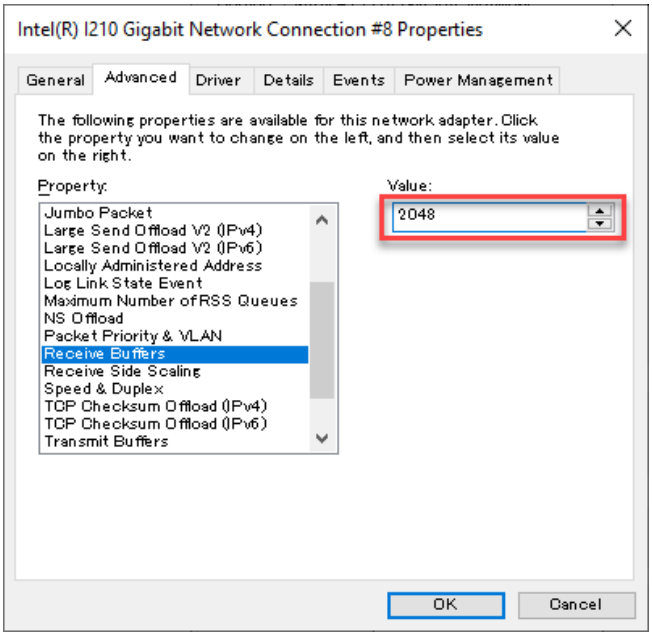
Here is the example to change Ethernet card IP address.



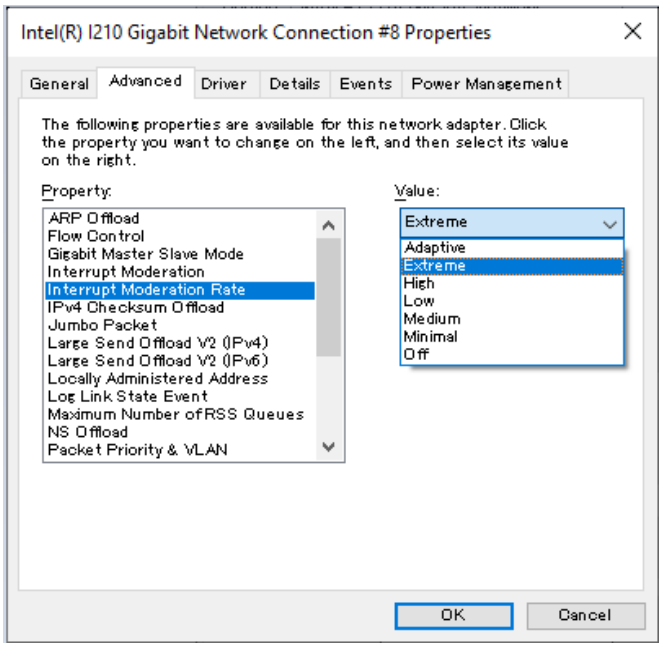
3. Set maximum Jumbo Frame value as highest in Ethernet card properties.



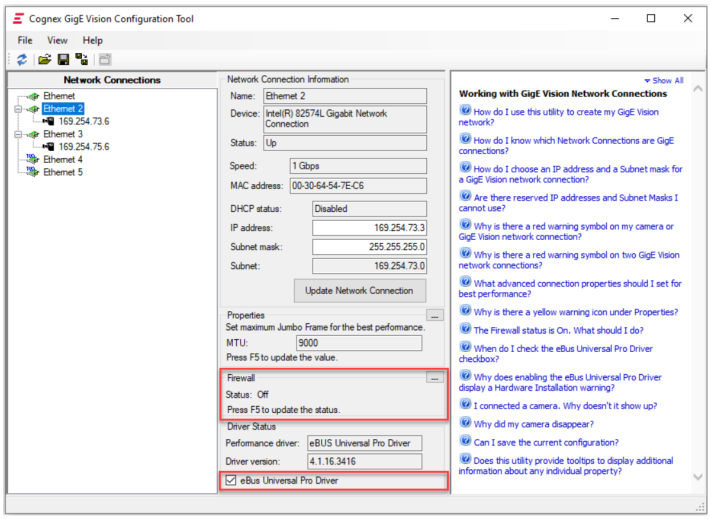
4. Change the **Receive Buffers** property and choose the highest possible value in the list.



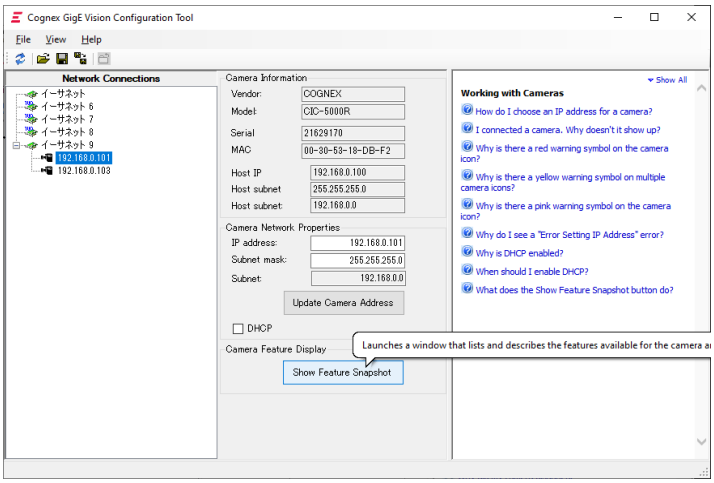
5. Change the **Interrupt Moderation Rate** property to **Extreme** in the list.



6. Double check that the firewall is off and eBus Driver is enabled.

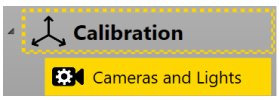


7. Click “Show Feature Snapshot” shown below in order to confirm that the control layer of GigE Vision protocol is working.



Cameras and Lights for Calibration

Camera and lights page, located under **Calibration** category, is used to configure cameras and lights for calibrations.



This page consists of two sub pages: Global Settings and Exposure Settings. Global Settings page is for camera hardware selection and global parameters setting. Exposure Settings page is for individual parameters setting for each calibration when the corresponding global parameters are disabled.

Global Settings

Global Settings include camera settings and lighting settings.

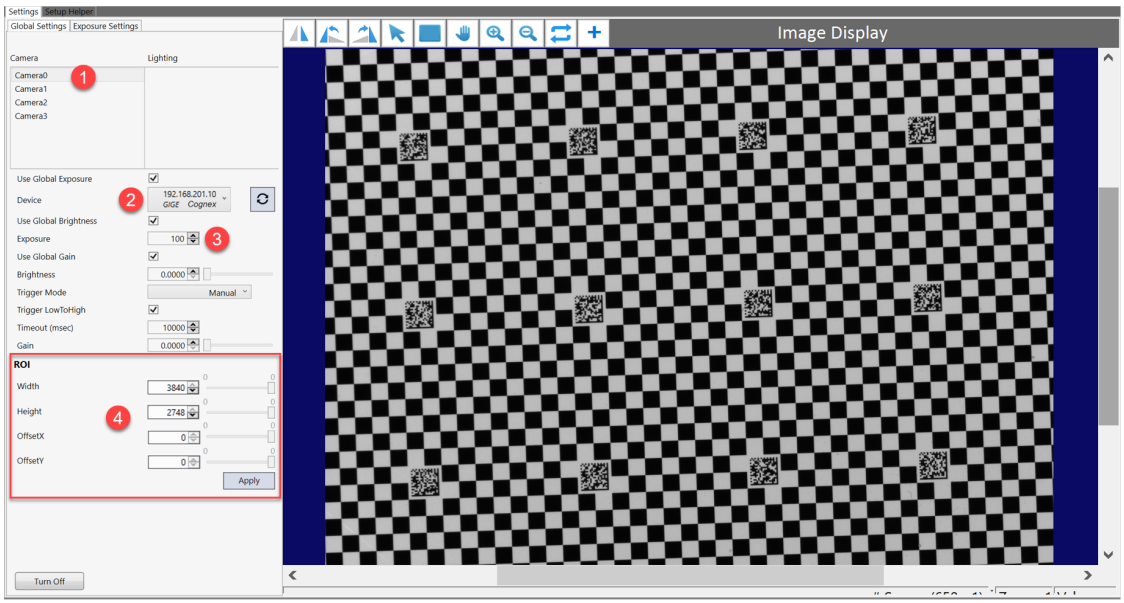
Camera Settings

Global Settings allow users to select camera hardware devices for calibration, and set their global exposures/brightnesses/gains.

The steps are:

1. Select a camera device from camera list (These are camera devices configured in the Configuration Wizard).
2. Select the corresponding camera hardware device from **Device** list.
3. Set proper exposure/brightness/gain for each camera to obtain images of appropriate contrast in live mode.
4. Specify ROI (Region Of Interest) of the select camera which allows the camera acquires images only from within the configured ROI.

   By default, ROI uses the whole image area. However, users can make it smaller to save image acquisition and processing time. ROI setting is a setting of camera hardware, therefore it effects all images acquired by this camera no matter it is used for calibration or feature finding.

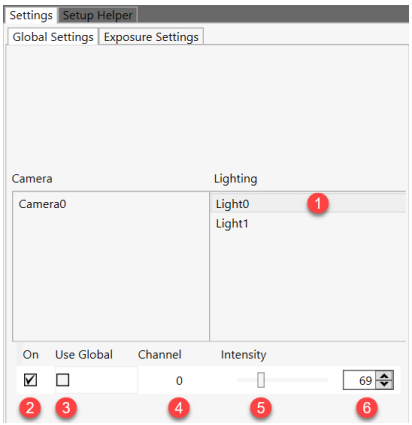


Item	Content/Type	Description
Use Global Exposure	Boolean	True: use the same exposure time for this camera whenever it is used for a calibration False: use local exposure times for different calibrations when this camera is used
Device	/	List of all connected camera hardware devices with information of their IP addresses (if they are GIGE Cameras) or serial numbers (if they are USB3 cameras), and brand names. Click button to update if a connected camera is not listed
Use Global Brightness	Boolean	True: use the same brightness for this camera whenever it is used for a calibration False: use local brightnesses for different calibrations when this camera is used
Exposure	Double(in ms)	Set exposure time for the selected camera
Use Global Gain	Boolean	True: use the same gain for this camera whenever it is used for a calibration False: use local gains for different calibrations when this camera is used
Brightness	Double	Set the brightness value for each image acquisition.
Trigger Mode	Manual/Auto/Semi	Manual: Software triggering. Auto: Acquires when it detects a transition on an external line. Semi: Acquires when the software is running and the external line detects a transition.
Trigger LowToHigh	Boolean	Image acquisition is triggered on the rising edge of the trigger signal when Trigger Mode is set as auto or semi-auto
Timeout (msec)	Integer	Set a timeout period that determines how long the acquisition device waits for an image to become available before the application generates a timeout error.
Gain	Double	Increasing brightness of image by amplifying both signal and noise received from each pixel. Not recommended to increase this value when there are other options to improve image brightness.

Item	Content/Type	Description
Width	Unsigned Integer	The width of the ROI
Height	Unsigned Integer	The height of the ROI
OffsetX	Unsigned Integer	The x-value of the origin of the ROI
OffsetY	Unsigned Integer	The y-value of the origin of the ROI

Lighting Settings

A light controller added in the Configuration Wizard can be configured on this page for its global parameters.



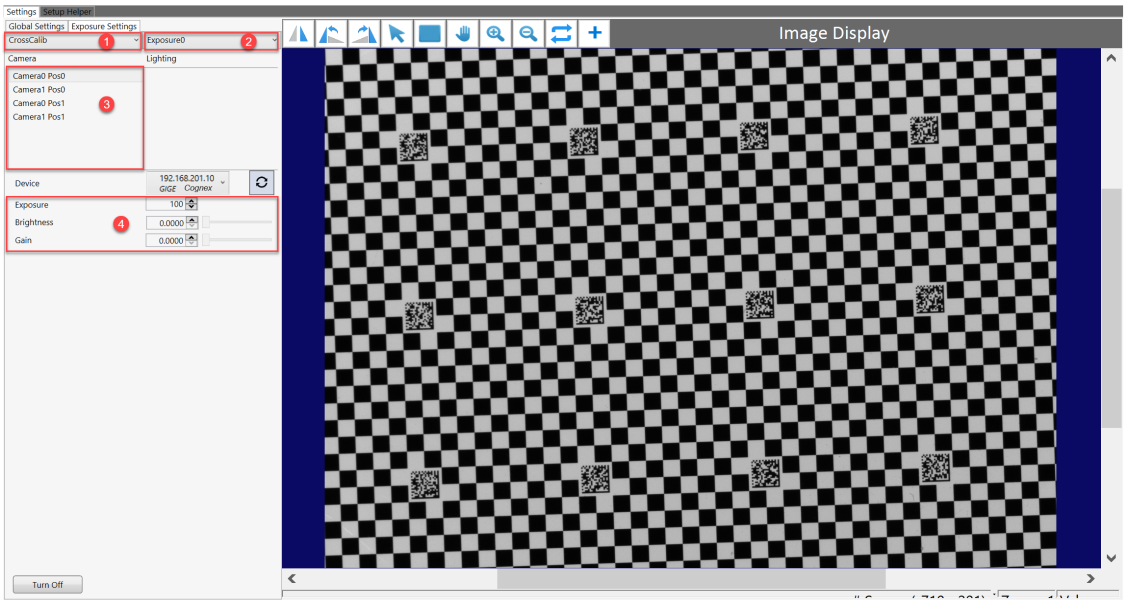
Item	Description
Lighting	Available light controllers
On/Off	Turn the selected channel's light on/off for image acquisition of the selected camera
Use Global	True: use the same intensity for this channel of light whenever it is used for a calibration. False: use local intensities for different calibrations when this light is used
Channel	The channel index of selected light controller (the total number of channels for each controller is configured in the Configuration Wizard).
Intensity slide bar	Increase or decrease intensity of this channel by sliding the bar right or left (this function is convenient to monitor intensity changes on live image).
Intensity edit box	Input the intensity of current channel manually, the minimum value is 0, maximum is 255.

Exposure Settings

When a global exposure/brightness/gain of a camera or a global intensity of a light channel is disabled, the corresponding local values for different calibrations should be configured individually on Exposure Settings page. Otherwise, this page can be skipped.

Camera Settings

The setting steps are:

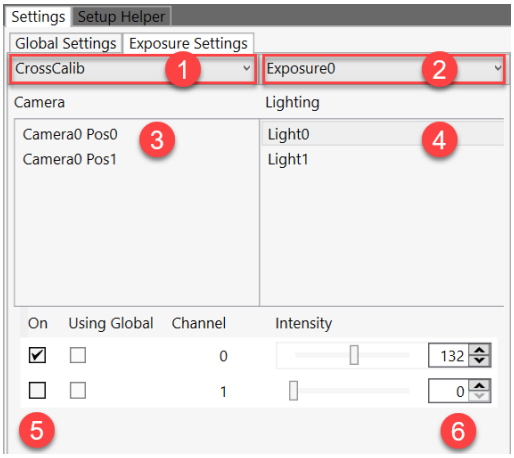


1. Choose one calibration.
2. Choose acquisition.

   There is only one acquisition (only Exposure0) for one position in a calibration.
3. Choose camera position.
4. Set exposure/brightness/gain for current position.
5. Walk through all calibrations and all positions to set exposures/brightnesses/gains individually.

Lighting Setting

The lighting setting steps are:



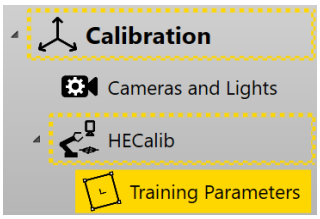
1. Choose a calibration from the calibration list
2. Select an acquisition
3. Select a camera position
4. Select a light controller, configure the on/off status and intensity for each of its channels
5. Choose another light controller, configure the on/off status and intensity for each of its channels
6. Walk through all calibrations and all acquisitions to set up their lighting respectively.

Tip: In the middle of lighting setting for different positions of a calibration, if you would like to turn off all lights before moving on to the next position, you can click the "Turn Off" button at the lower left corner of this page to turn all lights off.

Hand-eye Calibration

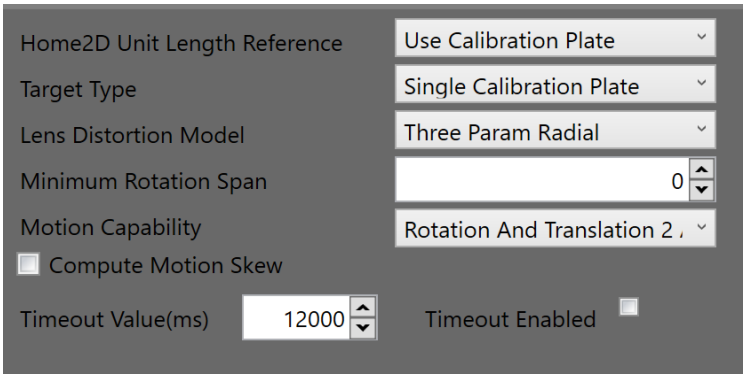
Hand-eye Calibration Training Parameters

Hand-eye calibration parameters influence the computation of the relationship between the camera and the motion device. It is used to provide the underlying tool information about the system being calibrated. The tool uses this information to generate valid and accurate result. Hand-eye calibration parameters setting is located under Calibration category.



Checkerboard Based Hand-eye Calibration

The following section describes the various hand-eye calibration parameters that can be adjusted when a checkerboard is used as a calibration target. It briefly describes how each parameter effects the computed calibration result.



Home2D Unit Length Reference

- Use Calibration Plate

  Indicates that an accurate calibration target is being used and that the parameters of the calibration target can be assumed to be the ground truth during calibration. When an accurate calibration target is used, the relative positions of the cameras computed during hand-eye calibration is accurate.

- Use Motion Stage

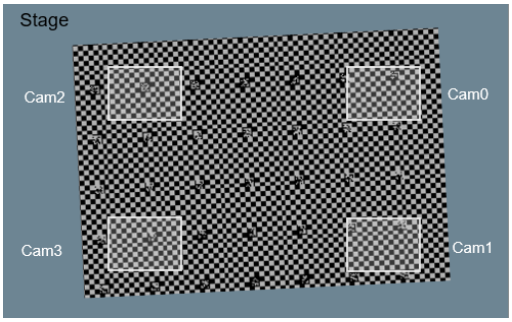
  Indicates that the motion device is accurate. In this mode, the relative positions of the cameras are computed by the rotation of the motion device. If the rotation is inaccurate or not sufficient, the estimated positions of the cameras would be inaccurate and integrating data across cameras would become unreliable.

Target Type

- Single Calibration Plate

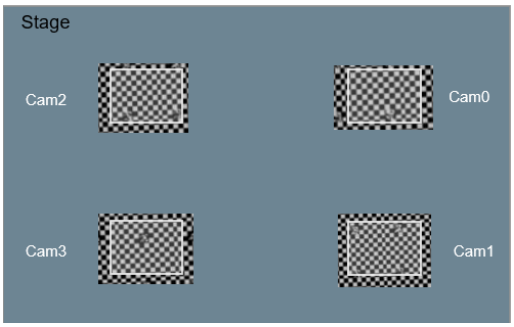
  Indicates that a single calibration target is being used to hand-eye calibrate the multiple cameras. Following calibration, all cameras would have the ability to map their features to a common Plate2D. The relative position of the

cameras are estimated accurately. The absolute cameras positions in Home2D is influenced by the accuracy and extent of rotation during the hand-eye calibration process.



- Separate Calibration Plates

  Indicates that a separate calibration target is being used for each camera. In this mode, the checkerboard is used to compute lens and perspective distortions, but the motion device is used to compute the relative and absolute positions of each camera in Home2D. The accuracy of this computation depends upon the accuracy and extent of the rotation during the calibration process. The user should use this option only if a single calibration target cannot be used for the application.



Lens Distortion Mode

- Three Param Radial

  This model calibrates for nonlinear optical distortion and perspective distortion. When compared with PerspectiveAndRadialWarp, this mode adds additional coefficients that properly model the location of the optical center. This mode is recommended for lenses with minimal to moderate distortion, typically those with focal lengths greater than 6mm.

- SineTanLaw Projection

  This model calibrates for nonlinear optical distortion and perspective distortion. When compared with ThreeParamRadialWarp, this model uses a computation model that is appropriate for lenses with moderate to severe distortion, typically those with focal lengths less than 6mm.

- No Distortion

  This mode assumes that the relationship between pixels and Home2D is an affine transform. This mode does not compensate for lens or perspective distortion. One benefit of this mode is it reduces processing time of image correction significantly.

Minimum Rotation Span

This parameter is only available as a protection mechanism, to ensure that the user does not accidentally generate poses with a smaller rotation angle span. During the hand-eye calibration process if the rotation angle span is less than this value, there is no guarantee that the hand-eye calibration result would be accurate.

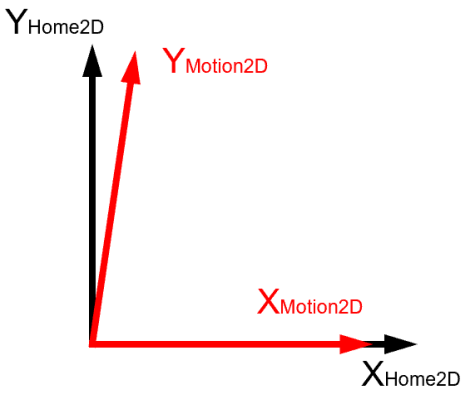
Motion Capability

Specify motion device's capabilities in moving calibration target.

Options	Degree Of Freedom		
	X	Y	Theta
Rotation And Translation 2 Axes	√	√	√
Rotation Only	x	x	√
Translation Only 1 Axis	√	x	x
Translation Only 2 Axes	√	√	x
Rotation And Translation 1 Axis	√	x	√

Compute Motion Skew

Enable or disable motion skew computation. Some lower accuracy motion stages may exhibit a skew between the X motion axis and Y motion axis as shown in the figure below. Hand-eye calibration can compute the skew and correct it in transforms between Home2D and the motion stage coordinate space. However, if there is not enough data to compute, the calculated results may deviate far away from motion's real parameters. To avoid that, it is recommended to disable motion skew computation when there are limited features to compute.



Timeout Enabled

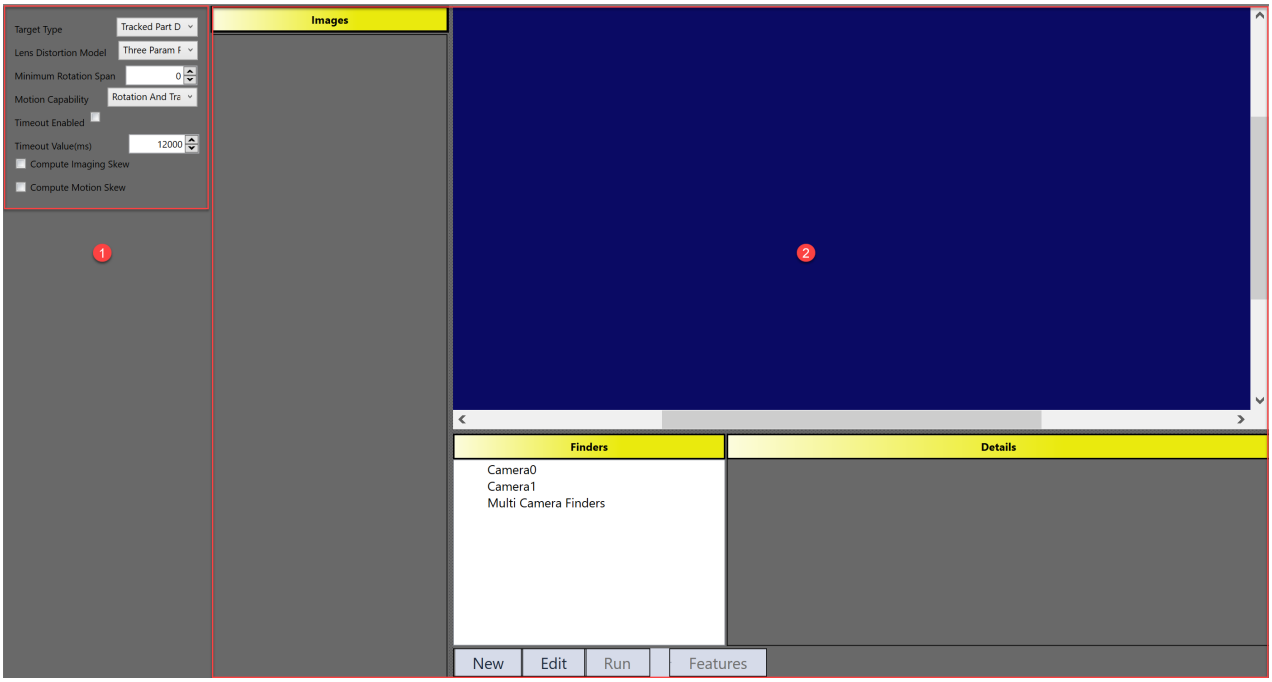
Enable or disable timeout for calibration.

Timeout Value(ms)

The calibration time out value in milliseconds.

Part-based Hand-eye Calibration

Part-based hand-eye calibration parameters page has two significant user interfaces. The first, **Calibration Parameters**, is for setting the hand-eye calibration parameters. The second, is for setting up feature finders that track features on the part during the hand-eye calibration process.



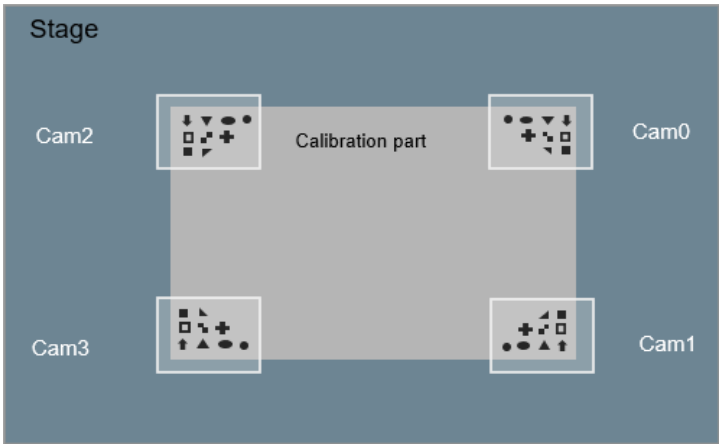
Calibration Parameters

The calibration parameters for part-based hand-eye calibration are almost the same as checkerboard based hand-eye calibration. This section simply outlines differences:

1. Part-based hand-eye calibration does not have Home2D Unit Length Reference parameter. This is because the calibration process depends upon the accuracy of the motion stage to provide a length reference.
2. The Target Type parameter that is described below:

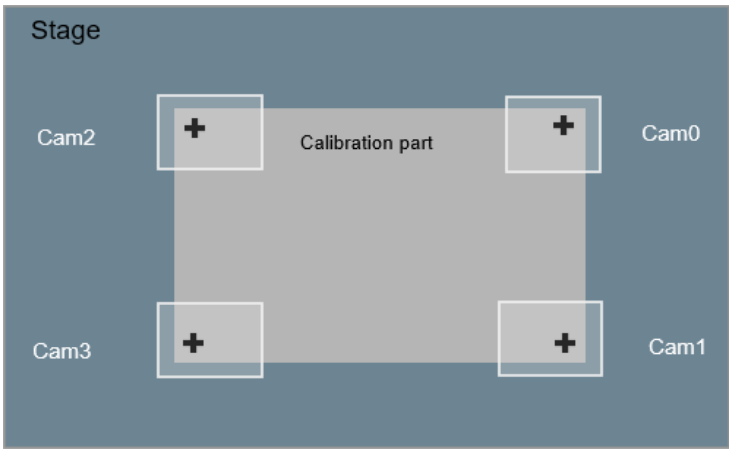
- Tracked Part Dense Features

  Used when the number of fiducial tracked during the hand-eye calibration process is more than three.



- Tracked Part Sparse Features

  Used when the number of fiducial marks is at-least one but less than three.



Compute Image Skew

Enable or disable image skew computation. Image skew makes each pixel acquired on image drift away in y direction proportionately to its distance to the top or bottom line of the image. Image skew could cause accuracy problems for both feature locating and measurement, so it is better to get it corrected. However, when there are not enough features in part based calibration, enabling image skew computation may result in incorrect compute result. Therefore, it is recommended to disable it when features are sparse.

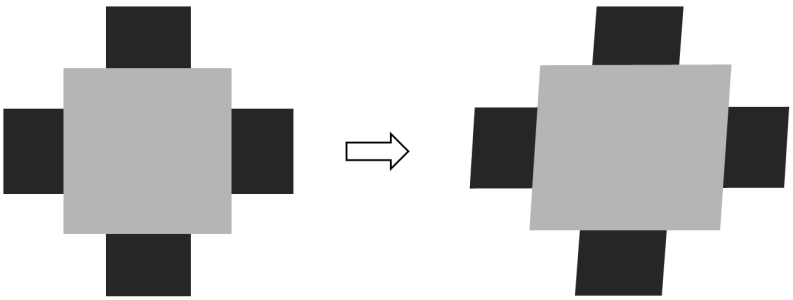
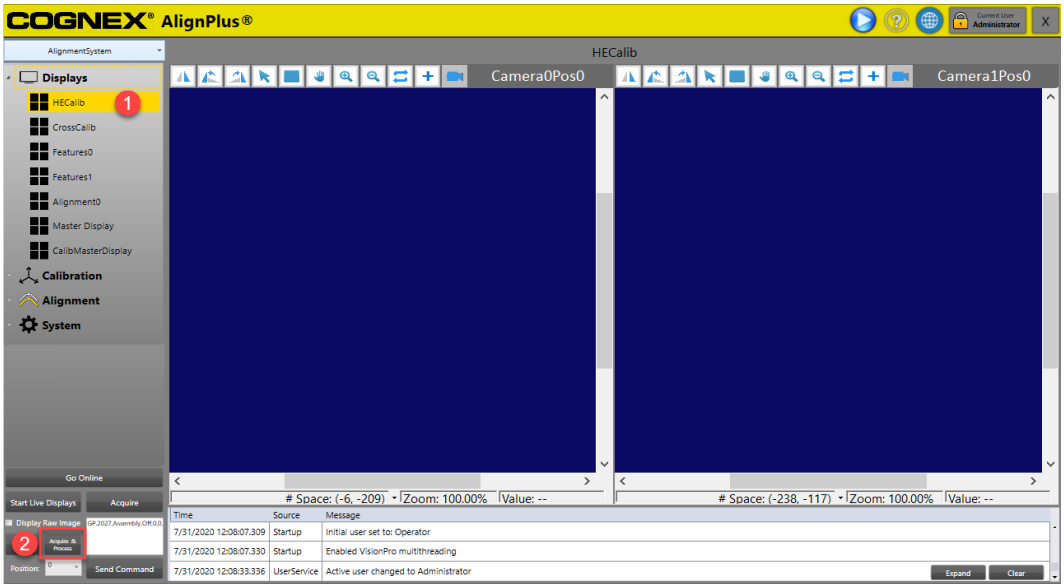


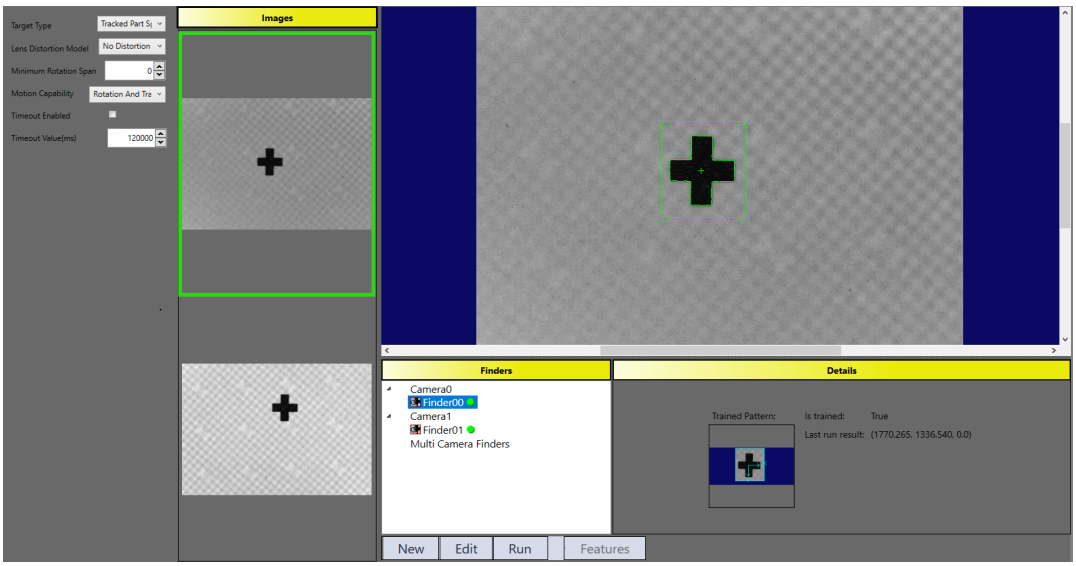
Image without skew Image with skew

Feature Finding User Interface

Feature Finding UI is a point features finder UI that is used to set feature finding vision tasks for part-based hand-eye calibration. Before start adding vision tools, the feature finding UI should first have images for each camera. To get those images, one can first select the related part-based hand-eye calibration display which is under **Display** category, and then click **Acquire & Process** button in function panel below.



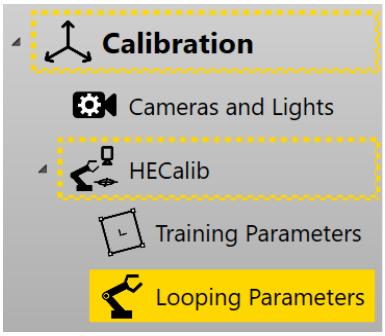
After the images have been acquired, the next step in the setup to be completed is in the training parameters page where feature finders should be added for part-based hand-eye calibration. Below is an example of tracking one fiducial mark using PatMax Finder on page 113.



If there are more features to be tracked, you can add more feature finders under each camera. One thing to pay attention is that hand-eye calibration will track each feature through the whole hand-eye calibration process, accumulating its positions at every motion pose for final computation. So, during feature finder setting, make sure those features will be correctly and uniquely found under each FOV during hand-eye calibration process, and their output positions should always be within FOV.

Looping Parameters

Looping parameters is used to set hand-eye calibration poses. It is located in certain hand-eye calibration group under **Calibration** category in setup mode.



AlignPlus can generate applications where the application generates the poses during hand-eye calibration. The poses contain the recommended translation only poses and rotation only poses. The poses are determined by the following parameters

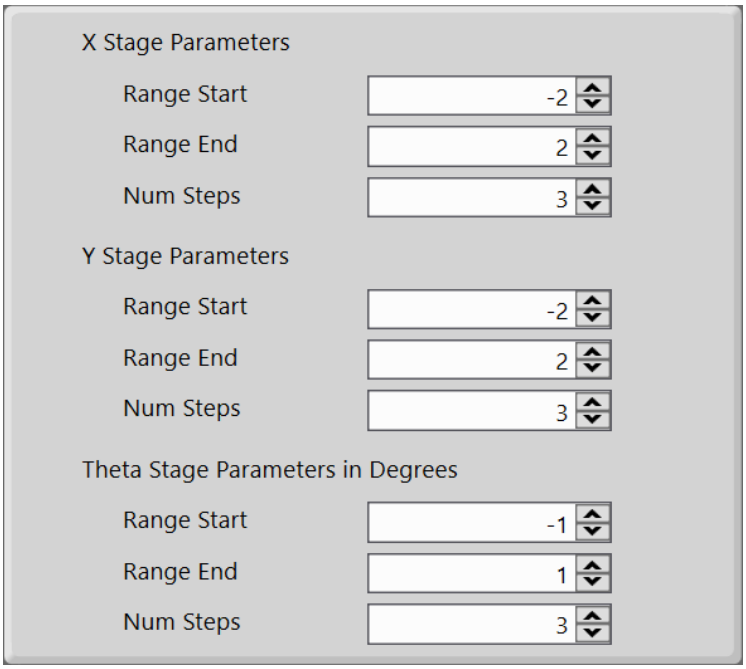
• The maximum and minimum limits for the X axis, and the number of samples when moving between the two limits

• The maximum and minimum limits for the Y axis, and the number of samples when moving between the two limits

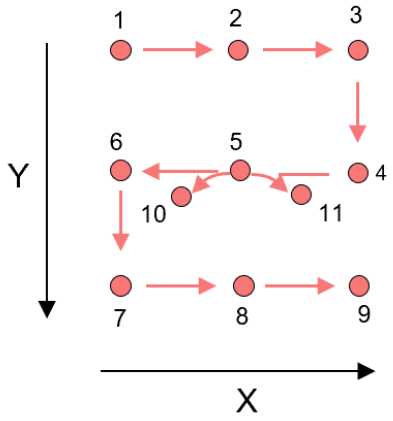
• The maximum and minimum limits for the Theta axis, and the number of samples when moving between the two limits(The rotation angle here is right-handed, which means positive angle is from positive x axis to positive y axis).

Default Loop

The setting that is shown below shows the default values for the above parameters, it requires motion device to move along X, Y and Theta axes to run through all sample points.



The figure below shows the path that will be taken by the motion device for default motion parameters.



The coordinates of each point are listed below:

Calibration Point	X	Y	Theta
1	-2	-2	0
2	0	-2	0
3	2	-2	0
4	2	0	0
5	0	0	0
6	-2	0	0
7	-2	2	0
8	0	2	0
9	2	2	0
10	0	0	-1
11	0	0	1

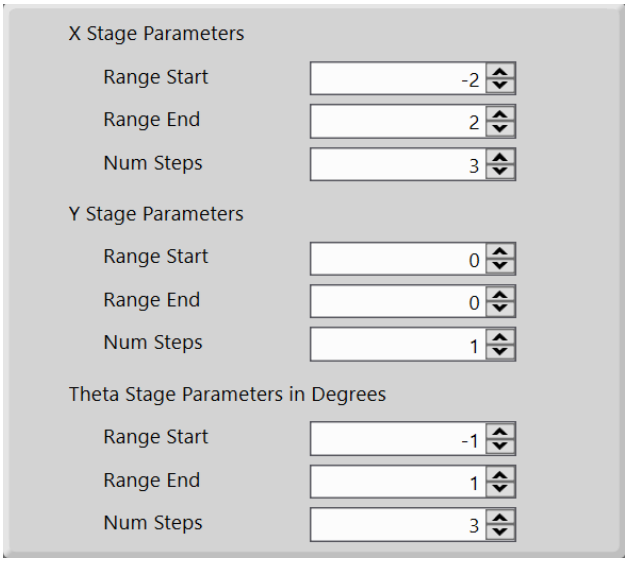
Note:

- The second point of rotation is (0,0,0) which is the same of point 5, thus this point is ignored when for rotation iteration.

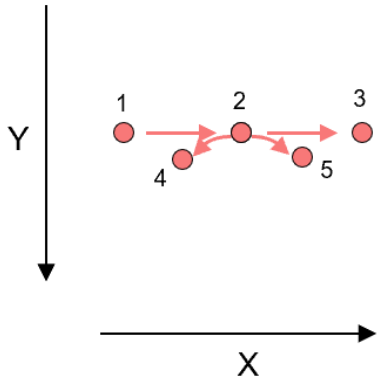
Loop With Two Axes

X-Theta Loop Parameters

When there are only X and Theta axes in the motion device, the maximum and minimum limits of Y axis should be set both as 0, and number of steps should be set as 1.

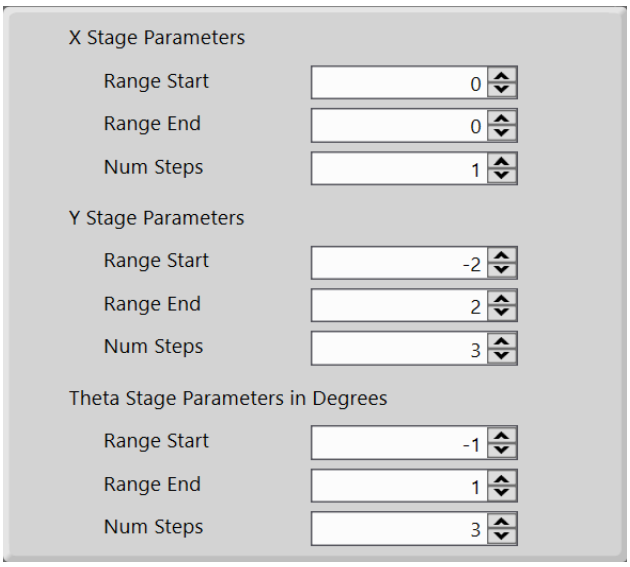


The figure below shows the path that will be taken by the motion device for above motion parameters.

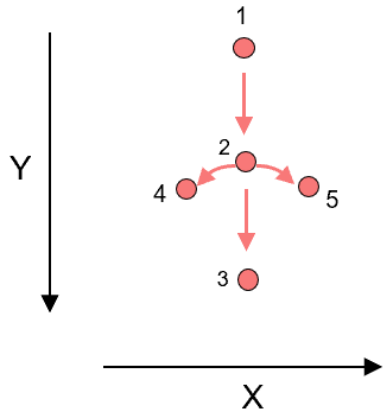


Y-Theta Loop Parameters

If it's a Y-Theta motion device, then the setting should be as below:

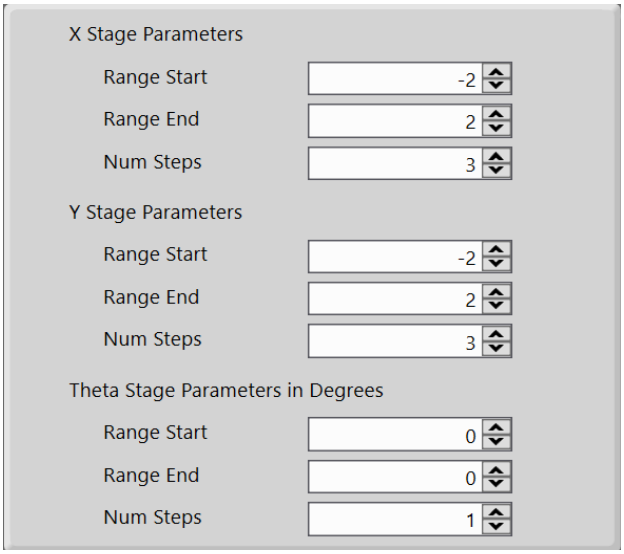


The figure below shows the path that will be taken by the motion device for above motion parameters.

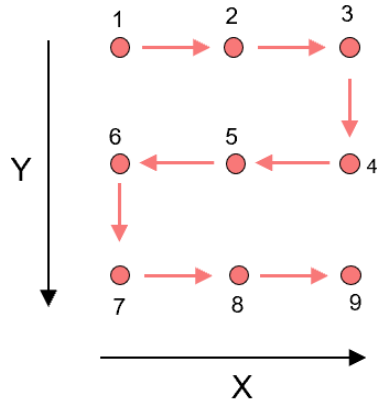


X-Y Loop Parameters

If the motion device has only X and Y axes, then the limits of Theta should be set as 0, and steps should be set as 1.

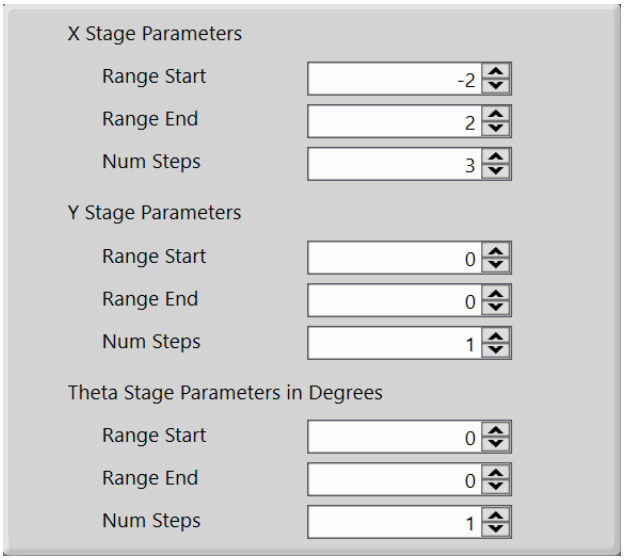


The figure below shows the path that will be taken by the motion device for above motion parameters.

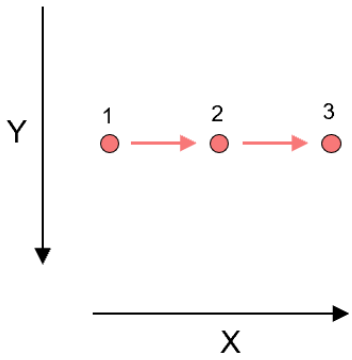


Loop With Single Axis

When there are only one axis in a motion device, then just set the all limits of unavailable axes as 0, and their number of steps as 1. Here is an example of only X axis.



The figure below shows the path that will be taken by the motion device for above motion parameters.



Note: The motion capability setting(Rotation And Translation 2 Axes, Rotation And Translation 1 Axis, Translation Only 2 Axes, etc) in hand-eye calibration training parameter should match the motion capability indicated in looping parameters.

Motion Analysis

In alignment or assembly applications, the final alignment or assembly performance is dependent on not only the repeatability and accuracy of feature finding and pose computation in the vision system, but also depends on the motion device's performance. The motion device's condition significantly affects the final alignment or assembly results, therefore, to evaluate motion device's performance is often a recommended step during machine setup.

Motion analysis is a function in AlignPlus that is designed to serve this purpose. It tracks certain calibration targets while motion device moves the target or camera to a set of given locations to evaluate motion's stability and accuracy, therefore, detects any motion issues including but not limited to vibration, repeatability, or drift.

Motion analysis function is automatically generated under each hand-eye calibration component without having the user configure it during wizard configuration. In run time, motion analysis uses the same motion device that is used in its corresponding hand-eye calibration.

Motion Analysis Process

Motion analysis process resembles vision-guided hand-eye calibration process in which motion device moves at command of the vision system. The steps are as follows:

1. The calibration target is affixed to the motion device for stationary-camera applications. For moving camera applications, the calibration target is placed on the surface that would contain the part to be aligned
2. The motion device is moved to working position provided by the vision system.
3. At working position, following a settling time, the image of the calibration target is captured, checkers or trained feature are located, and their coordinates are accumulated.
4. The motion device moves to one of the target positions in the position list provided by vision system.
5. After the steps 2- 4 have been repeated sufficient times, the motion analysis results are computed.

Motion Analysis Setup

Motion analysis setup consists of the following:

1. Setup tracking features
2. Choose coordinate space
3. Chose the test type for motion device performance analysis.

Setup tracking features

Motion analysis tracks one feature point on calibration target. The calibration target type is dependent on its hand-eye calibration target type set in Setup wizard. If it is calibration plate, motion analysis tool will automatically extract checker features on plate and use the mass center of all extracted features in train time as the point to track; If it is a real part, motion analysis tool will use the origin of trained feature set by user as the point to track.

Choose coordinate space

There are three different spaces options that motion analysis can choose.

- Raw2D

  When the perspective distortion and lens distortion are negligible in acquired image, one can track feature in Raw2D (raw image space) directly. The motion analysis result report is based on pixel, however this can be converted manually into millimeter using pixel resolution to better interpretation of the results.

- Checkerboard Calibration

  Checkerboard calibration corrects lens distortion and perspective distortion on acquired image and output a distortion free image with selected space as "Checkerboard Calibration". In Checkerboard Calibration space, features can be located more accurately if those distortions are not negligible.

- Home2D

  With hand-eye calibration result applied, the input images for motion analysis will have Home2D space available. If Home2D is chosen, the coordinates of extracted features will be in Home2D.

Performance Analysis

There are four ways to define how motion device should move during evaluation, each with different evaluation purposes:

- **Static Acquisition**

  In this mode, motion device stay stationary(which means there is no target positions), and let vision system locates features repeatedly at working position to evaluate whether there is a motion vibration issue.

- **Static Motion**

  In this mode, motion device moves between working position and a fixed target position, repeatedly or moves between working position and some boundary target positions. This mode is used to evaluate motion device's repeatability.

- **Dynamic Motion**

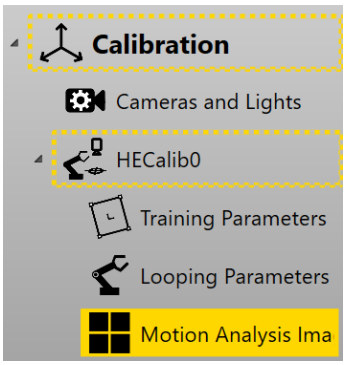
  In this mode, motion device moves back and forth between working position and some customized target positions. This is used to check whether there is a backlash in motion device.

- **Random target**

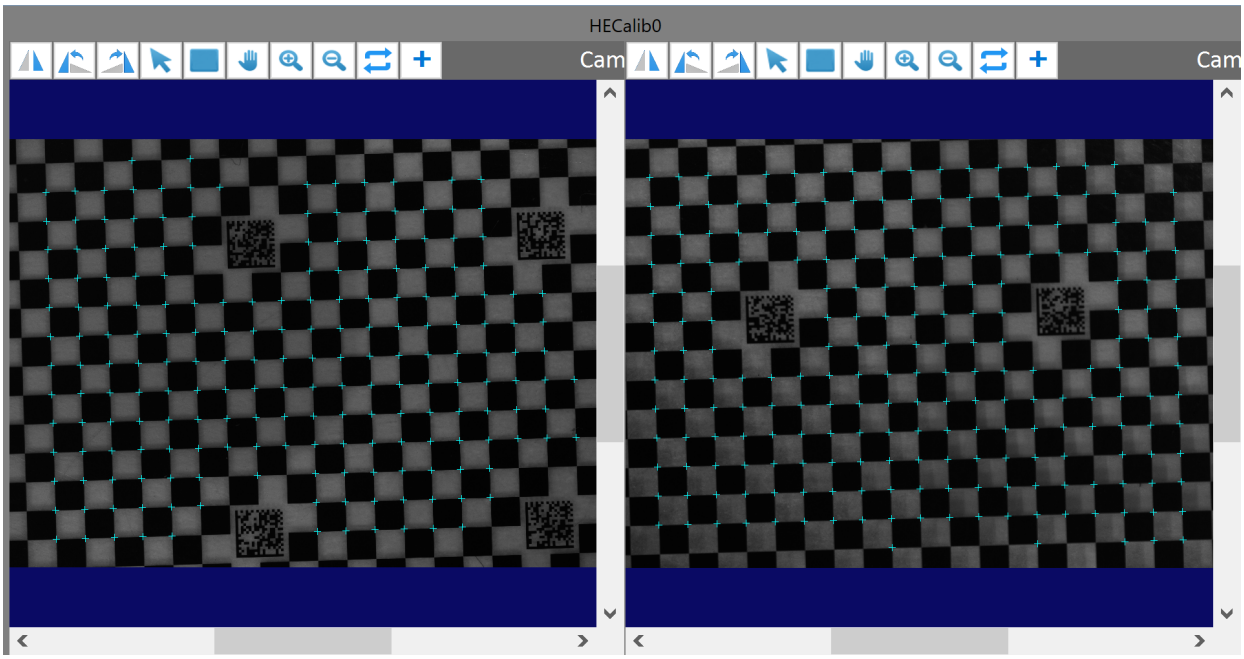
  In this mode, motion device moves back and forth between working position and some random target positions to check the accuracy of absolute position of motion device.

Motion Analysis Image Display

Motion analysis image display shows the images and their graphics of motion analysis task on a Multiple Display on page 62.

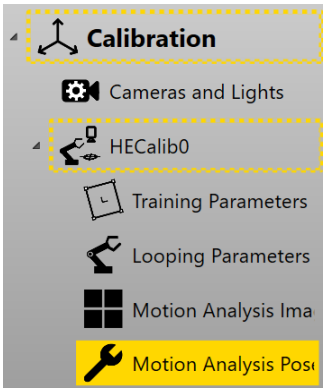


Motion analysis task uses the same cameras and their acquisition settings as those used in corresponding hand-eye calibration task. Therefore, you do not need to configure camera and their exposures for motion analysis separately once they have already been set for hand-eye calibration.



Motion Analysis Pose Generator

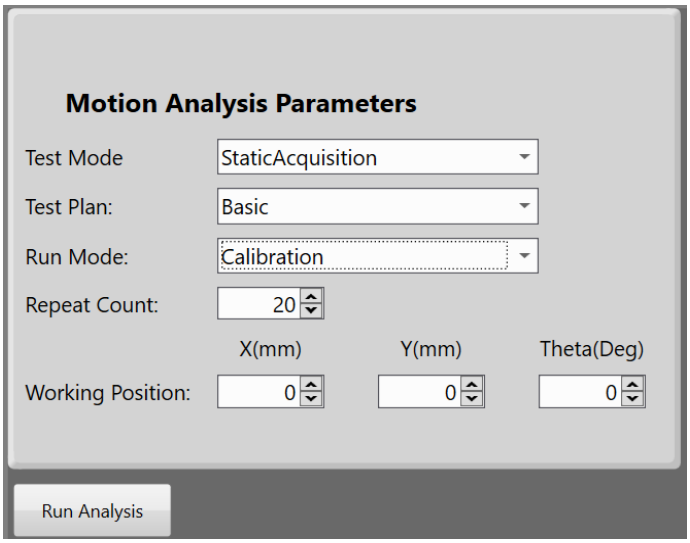
Motion analysis pose generator is used to set up parameters, train features, and run motion analysis. It can be found under its hand-eye calibration group.



In the setting page, choose one of the four options (Static Acquisition, Station Motion, Dynamic Motion, or Random Target) from "Test Mode" drop-down list to set up certain evaluation.

Static Acquisition

Static acquisition moves motion to one specific position and then acquire images repeatedly. This purpose of this test is to evaluate whether there is a vibration issue in current machine.



Settings

- Test Mode

  Choose **Static Acquisition**.

- Test Plan

  There is only one option (**Basic**) under this mode.

- Run Mode

Options	
UncorrectedTrain	Train features in Raw2D
UncorrectedRun	Run task in Raw2D
Calibration	Run checkerboard calibration at working position
CorrectedTrain	Train features in Plate2D
CorrectedRun	Run task in Plate2D
HECorrectedTrain	Train features in Home2D
HECorrectedRun	Run task in Home2D

- Repeat Count

  how many times user want to do the test, the default is 20.

- Working Position

  Specify the position where images will be acquired, features being tracked and accumulated. Motion analysis will only acquire images at working position.

Train Feature

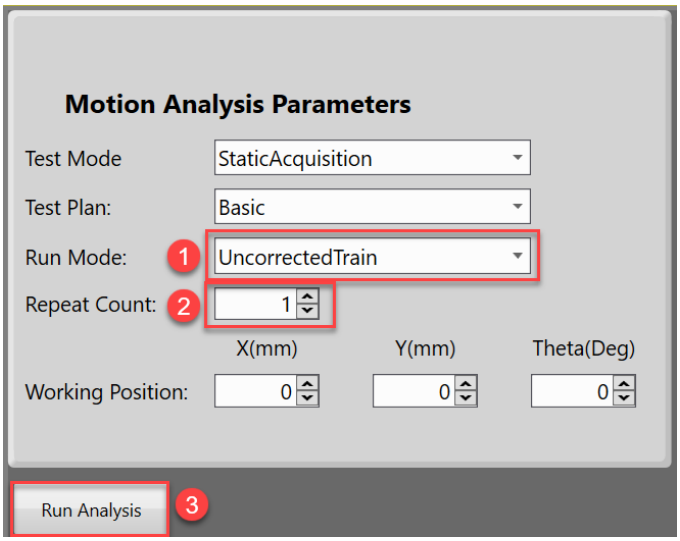
Before running motion analysis, target feature should be trained first regardless of feature type.

Checkerboard Feature

Under checkerboard-based hand-eye calibration, motion analysis can only use checkerboard for feature tracking. Checkerboard plate does not need extra feature finder since motion analysis tool will automatically select the mass point of extracted checkers as target feature to track. User only needs to select the space for feature and then train.

1. Raw2D Train Process

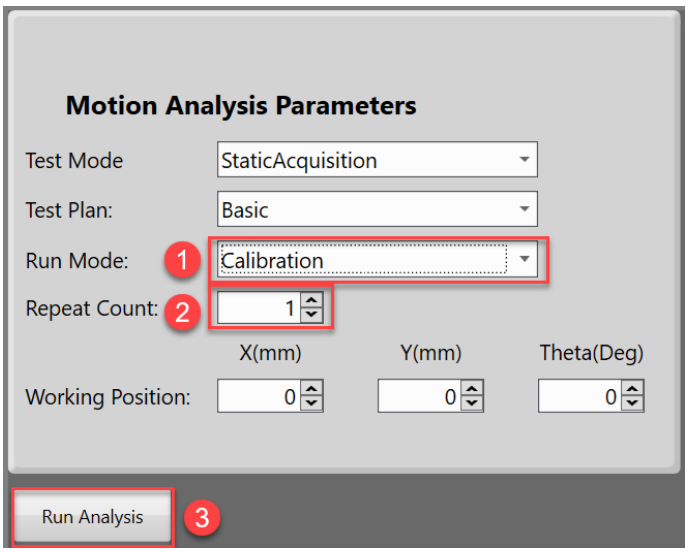
   For Raw2D space training: first select **UncorrectedTrain** in Run Mode drop-down list, then input repeat count as 1, and then click **Run Analysis** to train the feature.



   It is the same process when Home2D space is used.

2. Plate2D Train Process

   For Plate2D space training, an extra step to run checkerboard calibration is needed: Choose **Calibration** in Run Mode drop-down list, input repeat count as 1 and then click **Run Analysis** to run checkerboard calibration for all cameras used in motion analysis task.



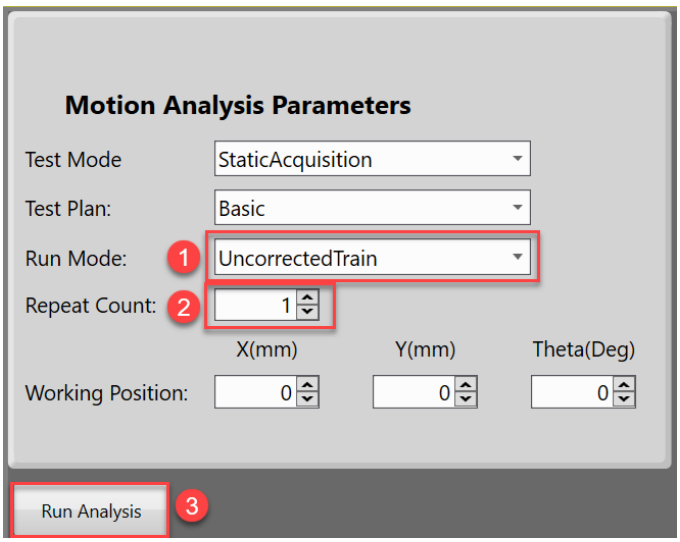
   After checkerboard calibration is done, select **Calibration Train** in Run Mode and click **Run Analysis** again to train feature.

Part Feature

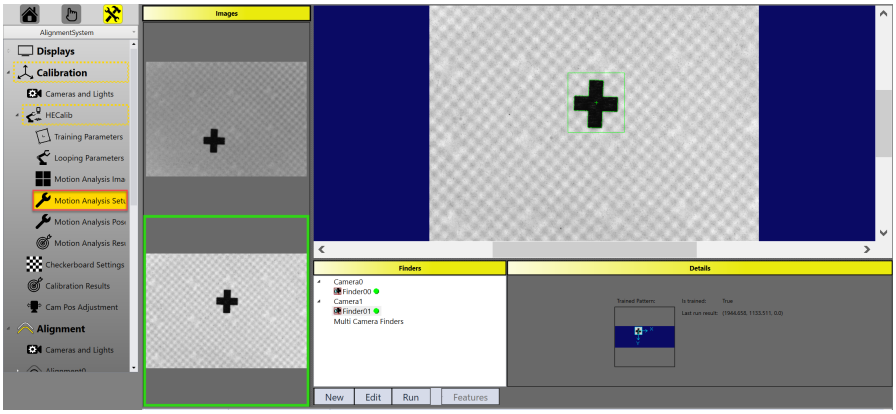
Under part-based hand-eye calibration, motion analysis can only use real part for feature tracking. When the program is generated, there will be an extra page named "Motion Analysis Setup" on HMI for user to set up feature finder for motion analysis task.

Here are the steps to train features:

1. Acquire images by clicking **Run Analysis** in train mode.



2. Set feature finders in **Motion Analysis Setup** page. Since motion analysis only tracks one feature for each camera, only one-point feature finder should be added under each camera.



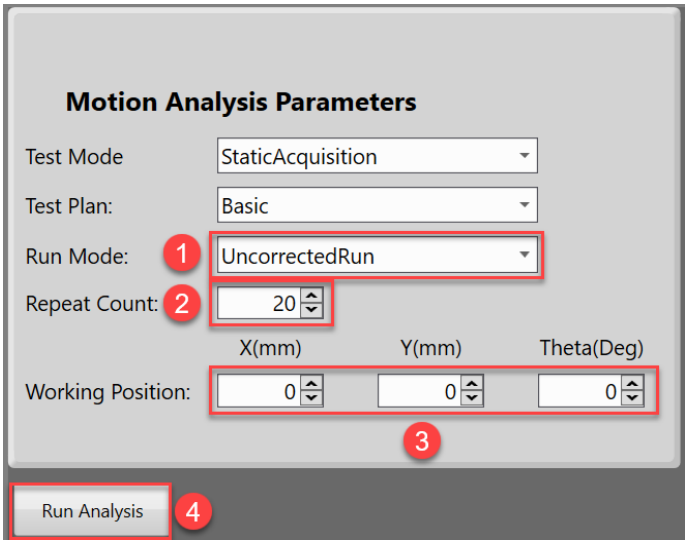
3. Run Step1 again to train feature.

The same process above applies to feature training in either Raw2D or Home2D space.

Note: Even Repeat Count is set as larger than 1, motion analysis will only run once when Run Mode is "UncorrectedTrain", "Calibration", or "CalibrationTrain".

Run

After features is trained, change Run Mode to UncorrectedRun/CorrectedRun/HECorrectedRun depending on in which space feature is trained; Set parameters such as repeat count, working position, and then click **Run Analysis** button to run the motion analysis procedure.

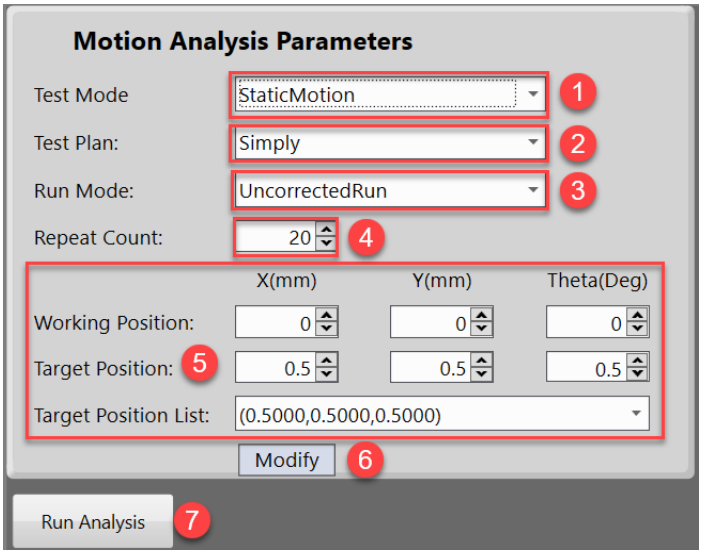


Static Motion

Static motion mode moves motion device back and forth between working position and target position until it has repeated this move sufficient times. The target position could either be a fixed position or one position from a set of boundary positions depending on which test plan user choose: **Simply**, **Moderate**, or **Fully**.

Simply

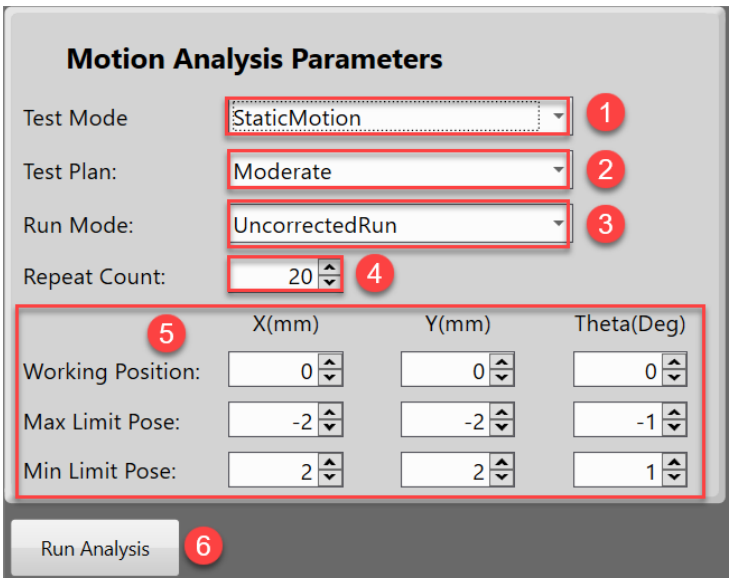
In simply test plan, motion analysis function will control the motion device to move back and forth between working position a fixed target position repeatedly. The train time process is the same as described in Static Acquisition. Here are steps to set up for run:



1. Select **Static Motion** in test mode.
2. Select **Simply** in test plan.
3. Choose UncorrectedRun/CorrectedRun/HECorrectedRun in run mode depending on in which space features were trained because run time space and train time space should be the same.
4. Input repeat count, the default value is 20.
5. Input working position and target position.
6. Click **Modify** to confirm target position if it has been changed.
7. Click **Run Analysis** to run the motion analysis.

Moderate

In moderate test plan, motion analysis function will move motion device back and forth between working position and a serial of boundary positions within inputted **Max Limit Pose** and **Min Limit Pose**. The train time process is the same as described in Static Acquisition. Here are steps to set up for run:



1. Select **Static Motion** in test mode.
2. Select **Moderate** in test plan.

3. Choose UncorrectedRun/CorrectedRun/HECorrectedRun in run mode depending on in which space features were trained because run time space and train time space should be the same.
4. Input repeat count, the default value is 20.
5. Input working position, max limit pose and min limit pose.
6. Click **Run Analysis** to run the motion analysis.

Let us assume the max limit pose is (X_2, Y_2, Θ_2) and min limit pose is (X_1, Y_1, Θ_1), then the generated set of poses will be as listed in table below.

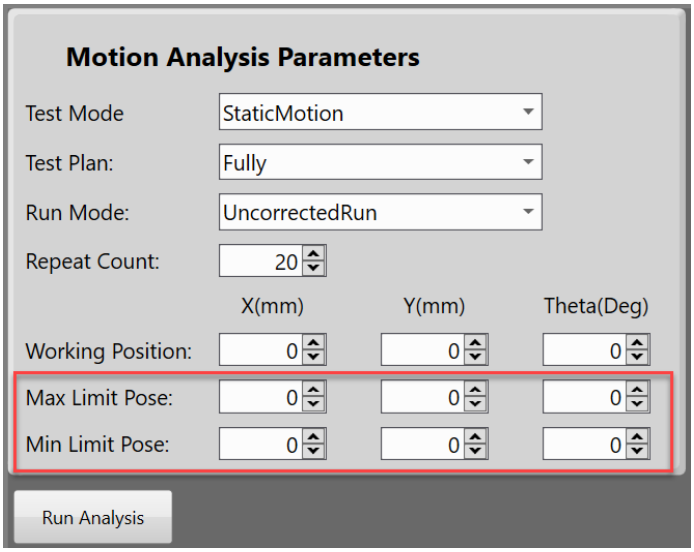
Index	Acquisition Position
1	(X_2, Y_2, Θ_2)
2	(X_2, Y_2, Θ_1)
3	(X_2, Y_1, Θ_2)
4	(X_2, Y_1, Θ_1)
5	(X_1, Y_2, Θ_2)
6	(X_1, Y_2, Θ_1)
7	(X_1, Y_1, Θ_2)
8	(X_1, Y_1, Θ_1)

Once the **Run Analysis** button is clicked, motion device will move to working position to let vision take a set of images, and then move to the first position in the table and come back to working position where vision takes another set of images, then move to the second position in the table and come back again to working position to acquire images, and so on. After the last position has been moved to, the target position will go back to the first position of the table until repeated moves have reached specified count.

Fully

In fully test plan, motion analysis function will move motion device back and forth between working position and a full set of boundary positions. The full set of positions are consisted of three sub sets which moves motion device along single axis move, two axes or three axes within inputted maximum and minimum positions.

The train time process is the same as described in Static Acquisition. The run time process is the same with moderate test plan's excepting changing test plan to **Fully**.



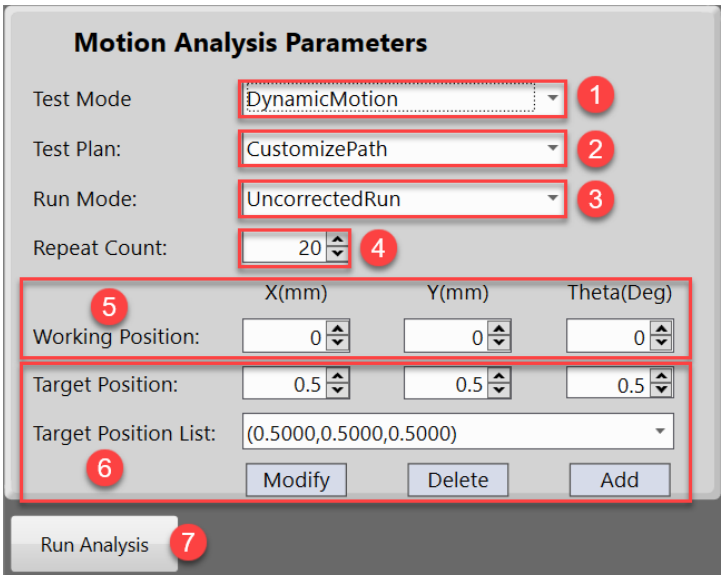
Let's assume the max limit pose is (X_2, Y_2, Θ_2) and min limit pose is (X_1, Y_1, Θ_1), the single axis move will have 6 poses, two axes move will have 12 poses, and three axes move will have 8 poses.

Index	Single Axis Move	Two Axes Move	Three Axes Move
1	$(X_2, 0, 0)$	$(X_2, Y_2, 0)$	(X_2, Y_2, Θ_2)
2	$(X_1, 0, 0)$	$(X_2, Y_1, 0)$	(X_2, Y_2, Θ_1)
3	$(0, Y_2, 0)$	$(X_1, Y_2, 0)$	(X_2, Y_1, Θ_2)
4	$(0, Y_1, 0)$	$(X_1, Y_1, 0)$	(X_2, Y_1, Θ_1)
5	$(0, 0, \Theta_2)$	$(X_2, 0, \Theta_2)$	(X_1, Y_2, Θ_2)
6	$(0, 0, \Theta_1)$	$(X_2, 0, \Theta_1)$	(X_1, Y_2, Θ_1)
7	/	$(X_1, 0, \Theta_2)$	(X_1, Y_1, Θ_2)
8	/	$(X_1, 0, \Theta_1)$	(X_1, Y_1, Θ_1)
9	/	$(0, Y_2, \Theta_2)$	/
10	/	$(0, Y_2, \Theta_1)$	/
11	/	$(0, Y_1, \Theta_2)$	/
12	/	$(0, Y_1, \Theta_1)$	/

Once the **Run Analysis** button is clicked, motion device will move the same way as in moderate test plan except that fully test plan has a larger set of target positions to iterate.

Dynamic Motion

Under dynamic motion test mode, motion device moves between working position and a set of customized target positions. Dynamic motion mode is used to analyze backlash of motion system. The train time process is the same as described in Static Acquisition. Here are the steps to set parameters for run:



1. Select **DynamicMotion** in test mode.
2. There will be only one option for test plan: **CustomizedPath**
3. Choose UncorrectedRun/CorrectedRun/HECorrectedRun in run mode depending on in which space features were trained because run time space and train time space should be the same.
4. Input repeat count, the default value is 20.
5. Input working position where images will be acquired.

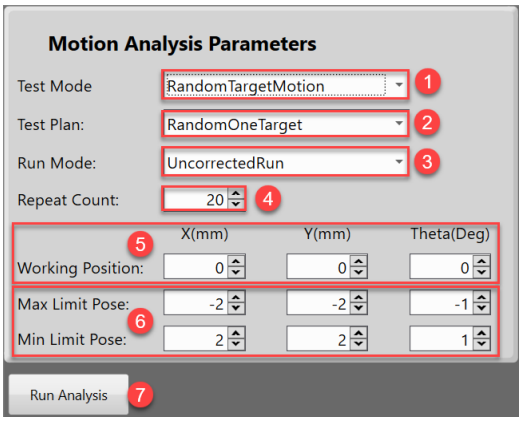
6. Input one target position and click **Add** to add it to moving path. Add more target positions if you like. If certain target position needs to be modified, select that position in the drop down list, modify it in edit boxes of target position, and click **Modify** to save the changes. If that position should be removed, select it from drop down list, then click **Delete** button.
7. Click **Run Analysis** to run the motion analysis.

Once the **Run Analysis** button is clicked, motion device will move to working position to let vision take a set of images, and then move to the first position in the target position list and come back to working position where vision takes another set of images, then move to the second position in the list and come back to working position to acquire images again, and so on. After the last position in the list has been moved to, the target position will go back to the first position of the list until repeated moves have reached specified count.

Random Target Motion

In random target motion mode, motion device moves between working position and random target positions repeatedly. The random positions are generated by motion analysis tool within inputted max and min limit poses. Random target motion is used to evaluate motion device's absolute position repeatability.

The train time process is the same as described in Static Acquisition. Here are the steps to set parameters for run:

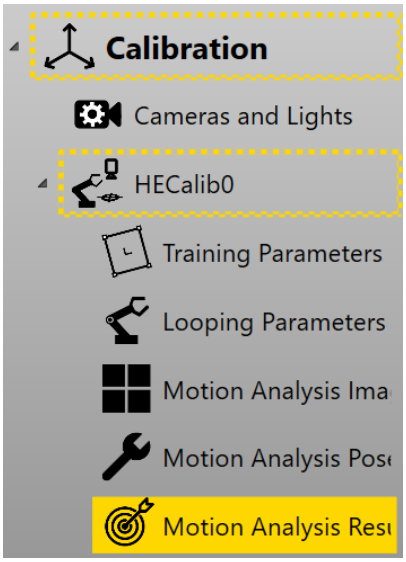


1. Select **RandomTargetMotion** in test mode.
2. There will be only one option for test plan: **RandomOneTarget**
3. Choose UncorrectedRun/CorrectedRun/HECorrectedRun in run mode depending on in which space features were trained because run time space and train time space should be the same.
4. Input repeat count, the default value is 20.
5. Input working position where images will be acquired.
6. Input max limit pose and min limit pose where random position will be generated within.
7. Click **Run Analysis** to run the motion analysis.

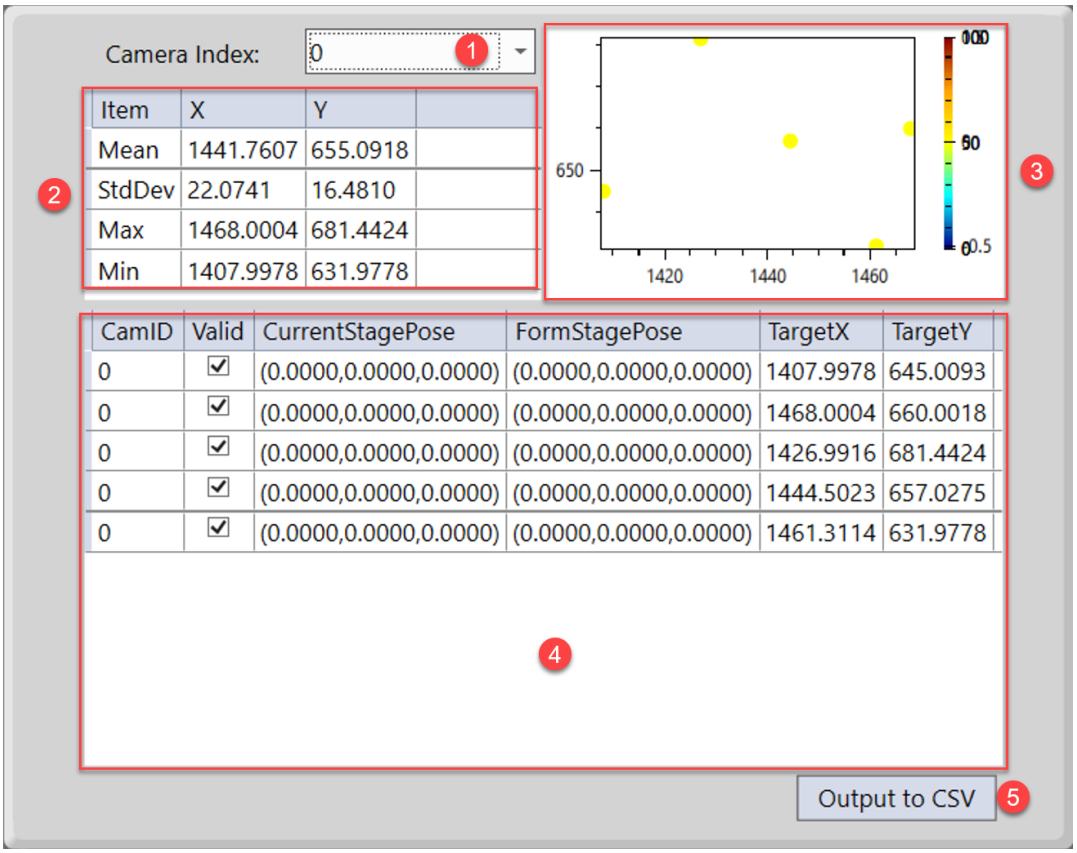
Once the **Run Analysis** button is clicked, motion device will move to working position to let vision take a set of images, and then move to a random position generated by motion analysis tool and come back to working position where vision takes another set of images, then move another random position and come back to working position to acquire images again, and so on. Repeat this process until it reaches specified times.

Motion Analysis Result

This page shows result of motion analysis, it is located in hand-eye calibration group under **Calibration** category in setup mode.



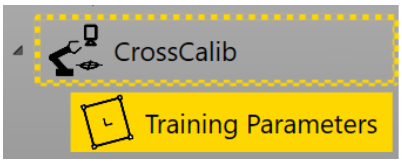
After motion analysis finishes running, switch to motion analysis result page, and select one camera from the camera index list, then the result for selected camera will be shown as below:



- The first table will show the statistics data like mean value, standard deviation, maximum and minimum values of the positions of tracked features.
- The graphic on the right side shows all tracking points' scatter plot.
- The second table shows all tracking points' X and Y coordinates in selected space.
- Click **Output to CSV** button to save the data to a CSV file.

Cross Calibration Training Parameter

Cross Calibration Training parameters lies under **Calibration** category of Setup page.



This page only have one parameter to configure: Lens Distortion Model

Lens Distortion Mode

- Three Param Radial

  This model calibrates for nonlinear optical distortion and perspective distortion. When compared with PerspectiveAndRadialWarp, this mode adds additional coefficients that properly model the location of the optical center. This mode is recommended for lenses with minimal to moderate distortion, typically those with focal lengths greater than 6mm.

- SineTanLaw Projection

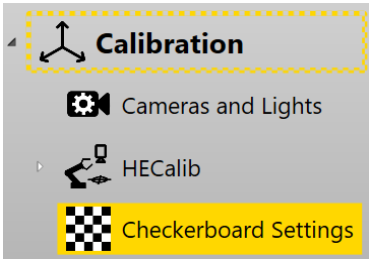
  This model calibrates for nonlinear optical distortion and perspective distortion. When compared with ThreeParamRadialWarp, this model uses a computation model that is appropriate for lenses with moderate to severe distortion, typically those with focal lengths less than 6mm.

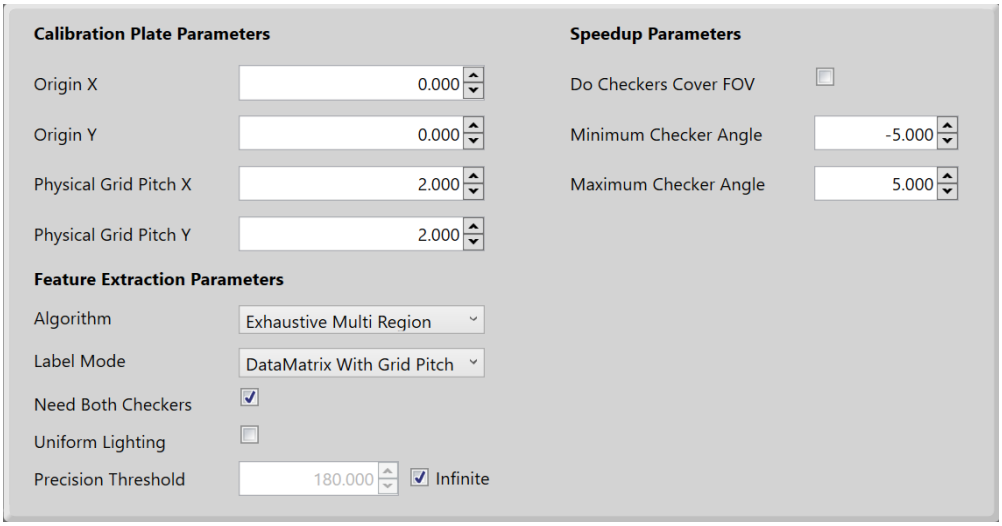
- No Distortion

  This mode will model perspective distortion only; any nonlinear optical distortion is ignored. By comparing the residual error values produced using this computation mode with the residual error values from ThreeParamRadialWarp or SineTanLawProjectionWarp you can improve your understanding of the individual sources of residual error.

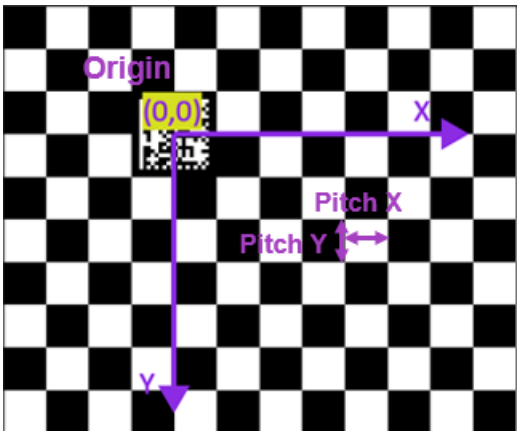
Checkerboard Settings

This page set parameters for checkerboard-based calibration, it's located under Calibration category.





Calibration Plate Parameters



- OriginX

  Gets/sets the the x value of the designated origin to be used for labeling of returned feature points. The vertex closest to point (OriginX, OriginY) will be used as the origin for point correspondence when Label Mode is CogCalibFeatureExtractorCheckerboardLabelModeConstants.UseOrigin.

  When not operating in CogCalibFeatureExtractorCheckerboardLabelModeConstants.UseOrigin mode, this property is ignored.

- OriginY

  Gets/sets the the x value of the designated origin to be used for labeling of returned feature points. The vertex closest to point (OriginX, OriginY) will be used as the origin for point correspondence when Label Mode is CogCalibFeatureExtractorCheckerboardLabelModeConstants.UseOrigin.

  When not operating in CogCalibFeatureExtractorCheckerboardLabelModeConstants.UseOrigin mode, this property is ignored.

- Physical Grid Pitch X

  Gets/sets the physical units of grid pitch along the x-axis of the calibration plate coordinate system (Plate2D, see the remarks section of Cognex.VisionPro.CalibFix.CogCalibFeatureExtractorCheckerboardLabelModeConstants). It is the distance between any two adjacent checker vertices whenever the line joining them is parallel to the x axis of Plate2D.

  Throws System.ArgumentOutOfRangeException: The value is less than or equal to 0.

- Physical Grid Pitch Y

  Gets/sets the physical units of grid pitch along the y-axis of the calibration plate coordinate system (Plate2D, see the remarks section of Cognex.VisionPro.CalibFix.CogCalibFeatureExtractorCheckerboardLabelModeConstants). It is the distance between any two adjacent checker vertices whenever the line joining them is parallel to the y axis of Plate2D.

  Throws System.ArgumentOutOfRangeException: The value is less than or equal to 0.

Feature Extraction Parameters

For **Algorithm** and **Label Mode**, please refer to Feature Extractor.

- Need both checkers

  Gets/sets the flag to indicate whether the tool should find only vertices shared by two interior light checkers.

  True: the extractor will find only those vertices belonging simultaneously to two interior light checkers(An interior checker is one that does not touch the image boundary or the border of the calibration plate).

  False: the extractor will attempt to find all vertices of all interior light checkers.

- Precision Threshold

  Gets/sets the threshold for discarding vertices with excessive positional uncertainty, specified in pixels.

  Due to noise and distortion, there are errors in the computed vertex positions. The algorithm internally estimates the position uncertainty for all found vertices, and excludes those from the final result whose position uncertainty estimates exceed the threshold specified here.

  Throws System.ArgumentOutOfRangeException: If the input value is less than 0 in the setter.

- Uniform Lighting

  Gets/sets the flag to indicate whether the checkerboard is expected to be uniformly illuminated in the run-time images.

  True: the extractor expects the light checkers to be uniformly illuminated, and uses an efficient technique for finding the vertices which can improve the speed performance. However, if in fact the illumination is not uniform, this technique may not find certain vertices that are severely affected by the non-uniform lighting.

  False: the tool performs better in presence of severe non-uniform lighting, and may find more vertices in these cases.

Speed Up Parameters

- Do Checkers Cover FOV

  Gets/sets whether the checkers are expected to entirely cover the field of view. Note that the purpose of this property is to improve speed performance when the checker coverage is known. This property should only be set to true if it is known beforehand that the checkers will cover the entire image for each camera at each pose. Note that Do Checkers Cover FOV is only used when Algorithm is Cognex.VisionPro.CalibFix.CogCalibFeatureExtractorCheckerboardAlgorithmConstants.Exhaustive.

- Minimum Checker Angle

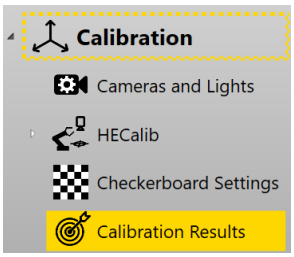
  Gets/sets the minimum expected angle of checker orientations. Note that if the new value is larger than Maximum Checker Angle, then Maximum Checker Angle will be changed to the new Minimum Checker Angle. The purpose of Minimum and Maximum Checker Angle is to improve speed performance when the checker orientations are known. Minimum and Maximum Checker Angle are only used when Algorithm is Cognex.VisionPro.CalibFix.CogCalibFeatureExtractorCheckerboardAlgorithmConstants.Exhaustive.

- Maximum Checker Angle

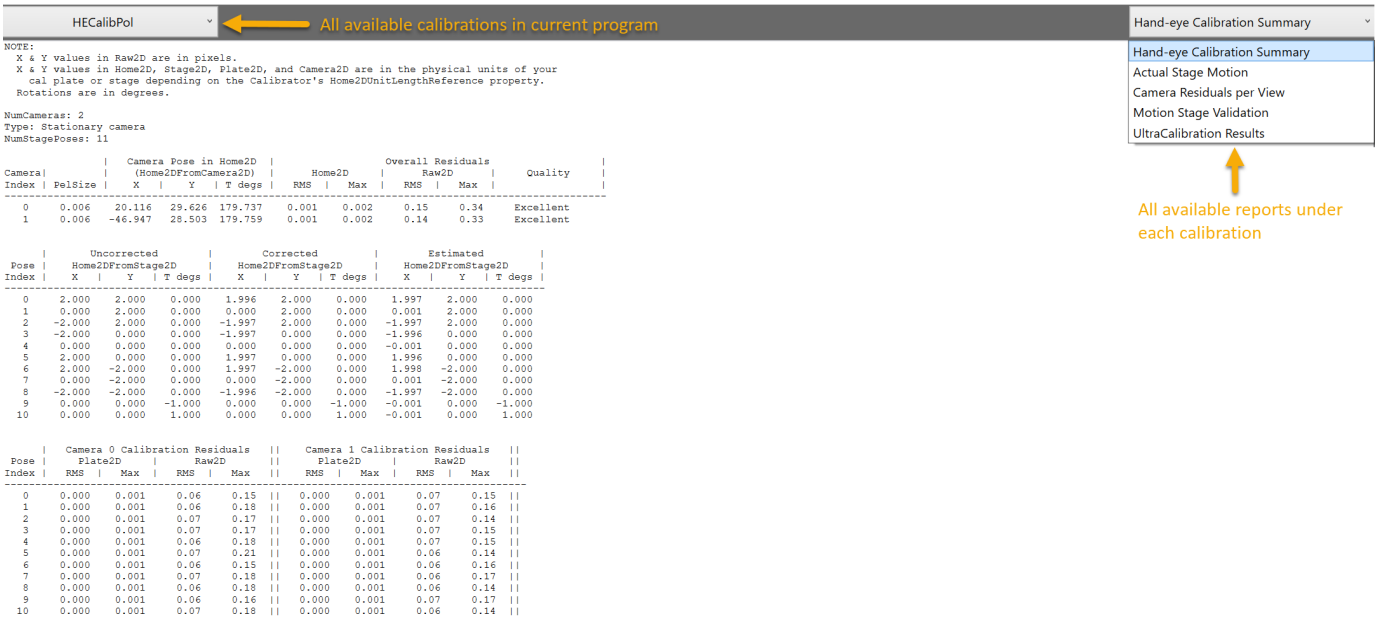
  Gets/sets the maximum expected angle of checker orientations. Note that if the new value is smaller than Minimum Checker Angle, then Minimum Checker Angle will be changed to the new Maximum Checker Angle. The purpose of Minimum and Maximum Checker Angle is to improve speed performance when the checker orientations are known. Minimum and Maximum Checker Angle are only used when Algorithm is Cognex.VisionPro.CalibFix.CogCalibFeatureExtractorCheckerboardAlgorithmConstants.Exhaustive

Calibration Results

Calibration Results page shows all calibration's results, it is located under **Calibration** category



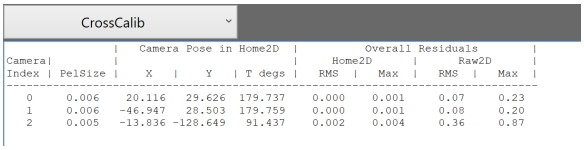
Select one calibration from the drop-down list at the upper left corner of page.



For hand-eye calibration, it includes reports:

- Hand-eye calibration summary
- Actual Stage Motion
- Camera Residual per View
- Motion stage validation
- UltraCalibration Results (only available when UltraCalibration is enabled in configuration wizard)

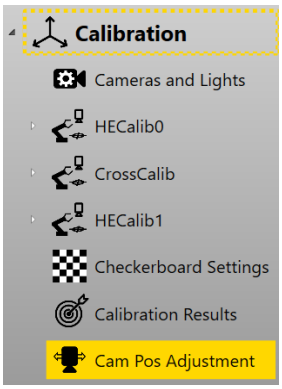
For other calibration, it will only have the **Camera Residual Per View** report.



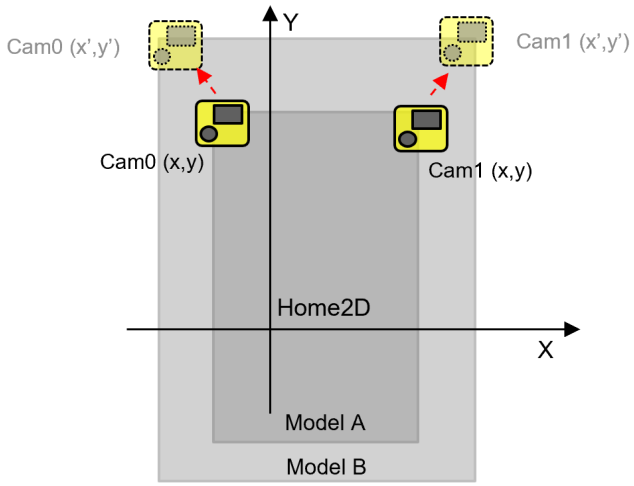
For more information about each item in the report, please refer to **Hand-eye Calibration Result Report** and **Stage Validation Result** under **Calibration\Calibration Types\Hand-eye Calibration Overview** category in AlignPlus Concept document.

Cam Pos Adjustment

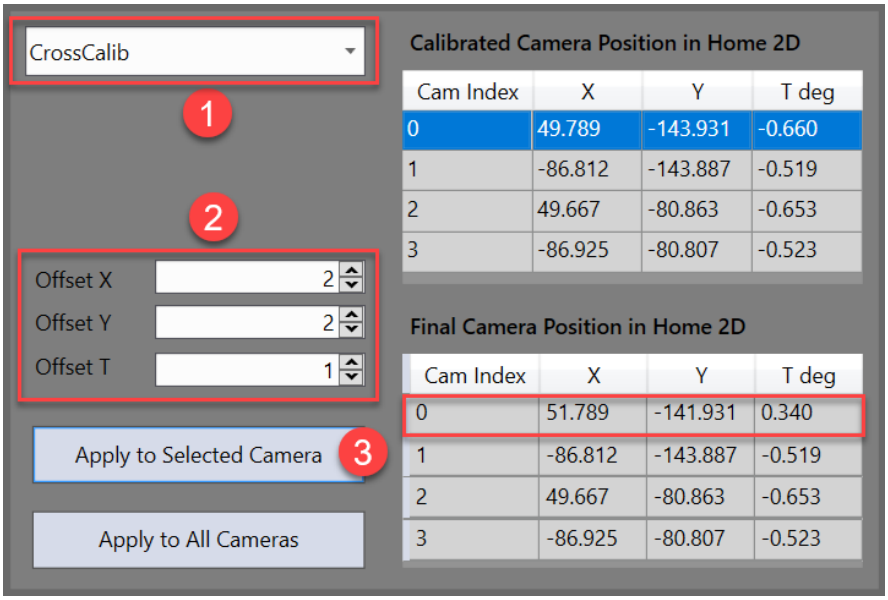
Camera pose adjustment function adjusts camera position during mode change to avoid recalibration. It's located under **Calibration** category in setup mode.



To save setup time during model change, sometimes customer does not want to rerun every calibration though cameras' positions have been changed to fit new mode size. Camera pose adjustment function can help on this. Camera pose adjustment function allows user to modify cameras' positions in Home2D, these new positions will be used to create new transforms between Camera2D and Home2D so that the locations of found features under new FOV will be updated.



In the setting page, offsets can be added to the original position of each camera to adjust its position if the camera has already been calibrated once. Here are the steps:



1. Choose one calibration.

   Offsets will only be added to specific camera under selected calibration. For the same camera under different calibrations, offsets should be added separately under each calibration.

2. Select one camera in **Calibrated Camera Position in Home2D**.

   The selected camera should have already been calibrated with valid Home2D coordinates.

3. Input X, Y, Theta offsets.

   Offset X, **Offset Y** and **Offset T** are offsets in Home2D, not the offsets along gentries where cameras are amounted to.

4. Click "Apply to Selected Camera" to save the offsets to the selected camera.

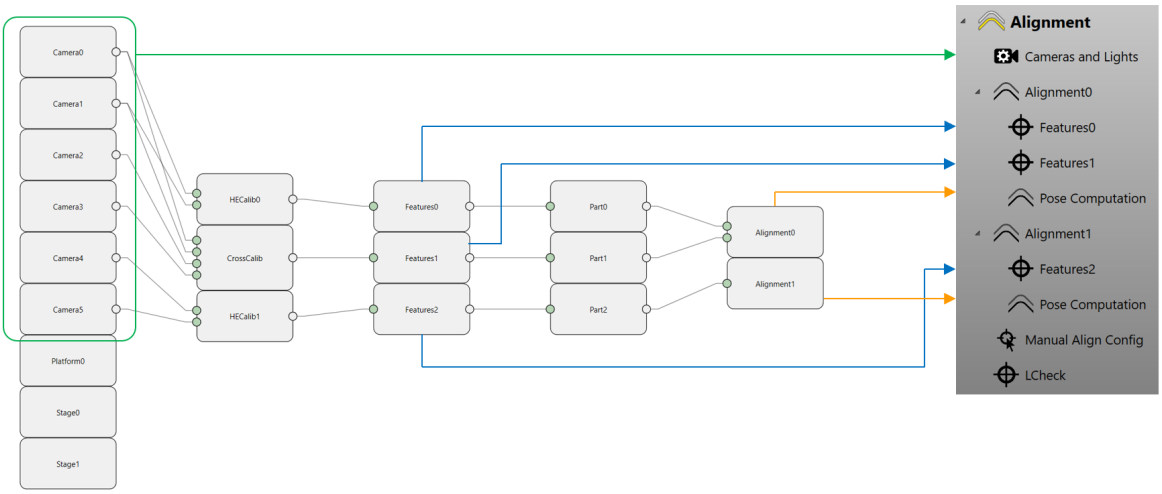
   After apply, the selected camera's new position will be updated to **Final Camera Position in Home2D** table.

   If all cameras share the same offsets, you can click "Apply to All Cameras" to update their positions at one time.

Alignment

Aligment Navigation

Alignment category in navigation tree is used to input and maintain settings for feature finding and alignment which include: camera and lights settings, vision tool settings for each feature finder, customization for pose computer, configurations for manual align and L-Check.

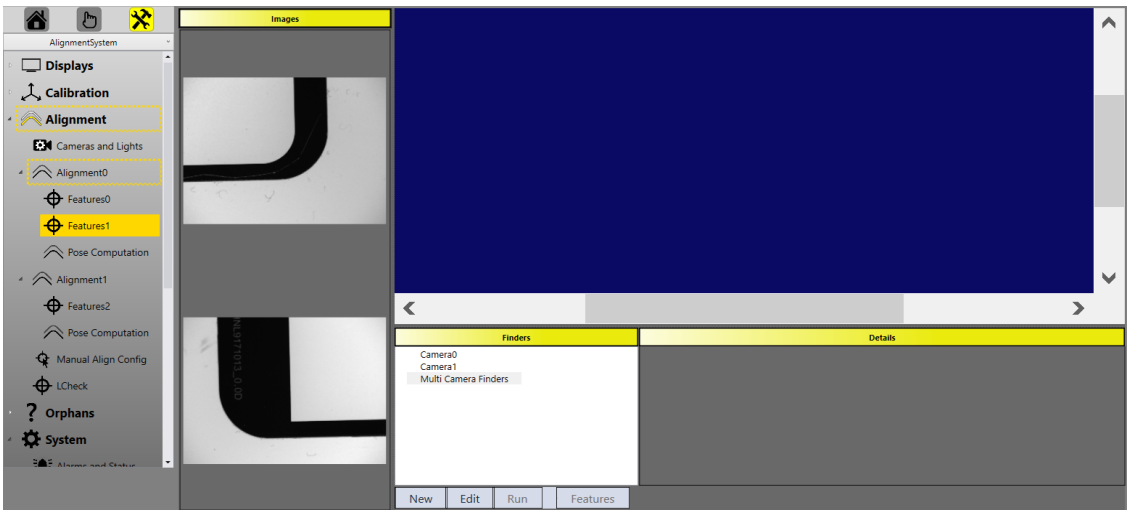


Camera and Lights

This page is used to set camera exposures and lights intensities for each feature finder. For more information please refer to Camera and Lights for Alignment on page 108.

Feature Finders

Each finder component added in the Configuration Wizard has one setup page on HMI after the application is generated. The page name is the same as the component name, for example, "Features0", "Features1". For more information, please refer to Configure Features Finder on page 111.



Pose Computation

Each alignment component configured in the Configuration Wizard has one pose computation page on HMI after the application is generated. This page is editable only if "Use Custom ToolBlock" option is selected for the alignment

component. For more information about how to customize pose computation on this page, please refer to Custom Pose Computation on page 146.

Manual Align Config

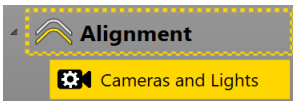
Manual Align function allows operators to manually select the feature location on a pop-up window when a run time feature is not found. For more information, please refer to Manual Align Config on page 149.

L-Check

L-Check function checks if the distance between pairs of point features are within specifications, see more information about L-Check on page 152.

Camera and Lights for Alignment

Camera and lights page, located under **Alignment** category, is used to configure cameras and lights for feature finders.



This page consists of two sub pages: Global Settings and Exposure Settings. Global Settings page is for camera hardware selection and global parameters setting. Exposure Settings page is for individual parameters setting for each features finder when the corresponding global parameters are disabled.

Global Settings

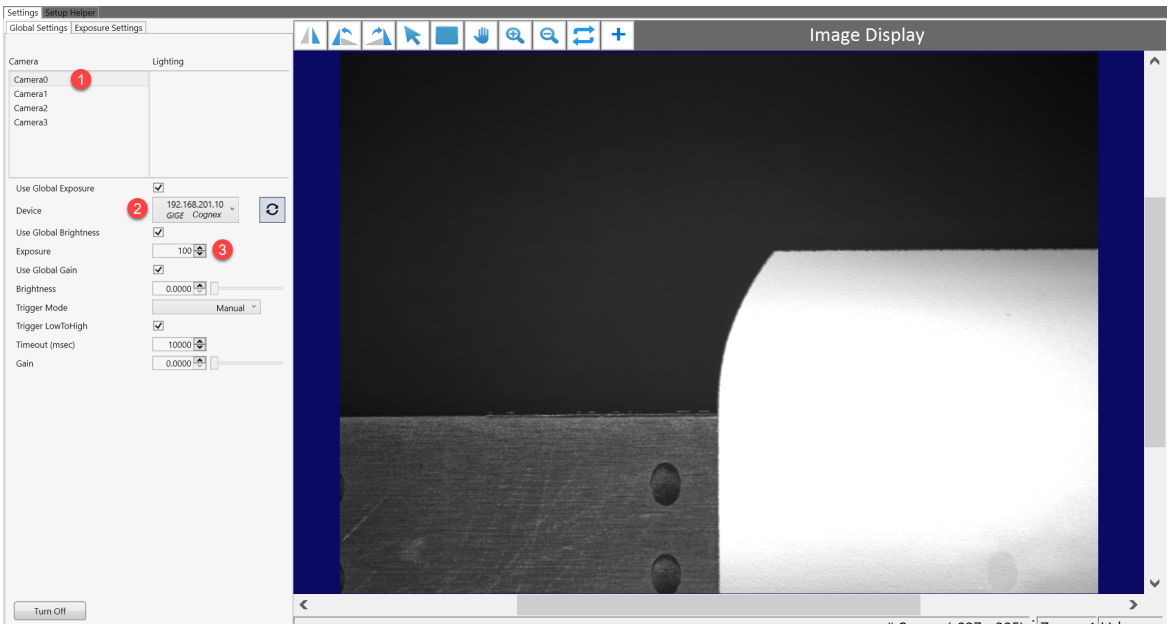
Global Settings include camera settings and lighting settings.

Camera Settings

Camera Settings allow users to select camera hardware devices for feature finding, and set their global exposures/brightnesses/gains.

The steps are:

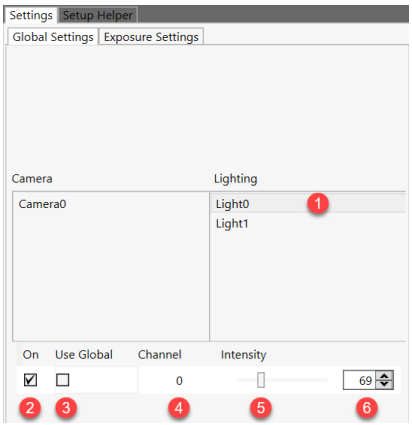
1. Select a camera device from camera list (These are camera devices configured in the Configuration Wizard).
2. Select the corresponding camera hardware device from **Device** list.
3. Set proper exposure/brightness/gain for each camera to obtain images of appropriate contrast in live mode.



Item	Content/Type	Description
Use Global Exposure	Boolean	True: use the same exposure for this camera whenever it is used for a feature finder False: use local exposures for different feature finders when this camera is used
Device	/	List of all connected camera hardware devices with information of their IP addresses (if they are GIGE Cameras) or serial numbers (if they are USB3 cameras), and brand names. Click button to update if a connected camera is not listed
Use Global Brightness	Boolean	True: use the same brightness for this camera whenever it is used for a feature finder False: use local brightnesses for different feature finders when this camera is used
Exposure	Double(in ms)	Set exposure time for the selected camera
Use Global Gain	Boolean	True: use the same gain for this camera whenever it is used for a feature finder. False: use local gains for different feature finders when this camera is used.
Brightness	Double	Set the brightness value for each image acquisition.
Trigger Mode	Manual/Auto/Semi	Manual: Software triggering. Auto: Acquires when it detects a transition on an external line. Semi: Acquires when the software is running and the external line detects a transition.
Trigger LowToHigh	Boolean	Image acquisition is triggered on the rising edge of the trigger signal when Trigger Mode is set as auto or semi-auto.
Timeout (msec)	Integer	Set a timeout period that determines how long the acquisition device waits for an image to become available before the application generates a timeout error.
Gain	Double	Increasing brightness of image by amplifying both signal and noise received in each pixel. Not recommended to increase this value when there are other options to improve the image brightness.

Lighting Settings

A light controller added in the Configuration Wizard can be configured on this page for its global parameters.



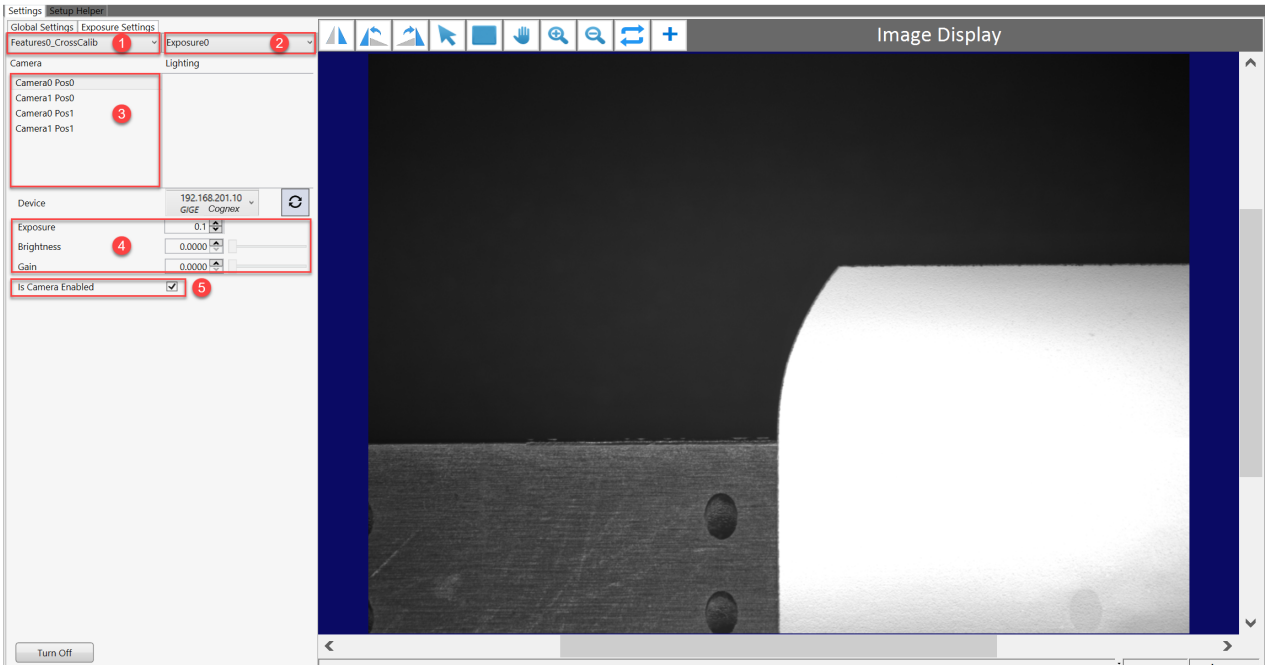
Item	Description
Lighting	Available light controllers
On/Off	Turn the selected channel's light on/off for image acquisition of the selected camera
Use Global	True: use the same intensity for this channel of light whenever it is used for a feature finder. False: use local intensities for different feature finders when this light is used
Channel	The channel index of selected light controller (the total number of channels for each controller is configured in the Configuration Wizard).
Intensity slide bar	Increase or decrease intensity of this channel by sliding the bar right or left (this function is convenient to monitor intensity changes on live image).
Intensity edit box	Input the intensity of current channel manually, the minimum value is 0, maximum is 255.

Exposure Settings

When a global exposure/brightness/gain of a camera or a global intensity of a light channel is disabled, the corresponding local values for different feature finders should be configured individually on Exposure Settings page. Otherwise, this page can be skipped.

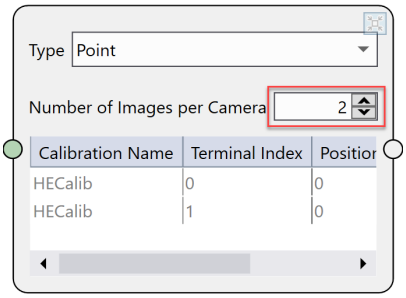
Camera Setting

The setting steps are:



1. Choose a feature finder from the feature finder list
2. Choose an acquisition

   By default, there is only one acquisition (Exposure0) in the option list. However, if the **Number of Images per Camera** is configured as more than 1 in the Configuration Wizard (such as shown blow), then there will be more acquisitions (such as Exposure0, Exposure1) that require respective camera settings.



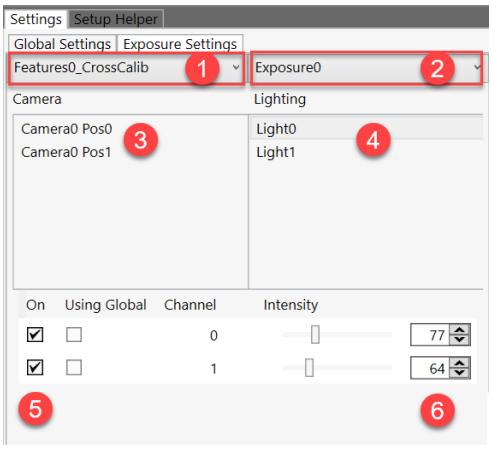
3. Choose a camera position
4. Set exposure/brightness/gain for current position, current acquisition
5. Check whether acquisition should be disabled under current position

   True: camera will acquire image at current position, current acquisition for selected feature finder

   False: camera will not acquire image at current position, current acquisition for selected feature finder
6. Walk through all feature finders, all acquisitions, and all positions to set exposures/brightnesses/gains respectively.

Lighting Setting

The lighting setting steps are:

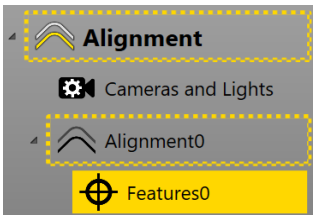


1. Choose a feature finder from the feature finder list
2. Select an acquisition
3. Select a camera position
4. Select a light controller, configure the on/off status and intensity for each channel
5. Choose another light controller, configure the on/off status and intensity for each channel
6. Walk through all feature finders and all acquisitions to set up their lighting respectively.

Tip: In the middle of lighting setting for different positions of a feature finder, if you would like to turn off all lights before moving on to the next position, you can click the "Turn Off" button at the lower left corner of this page to turn all lights off.

Configure Features Finder

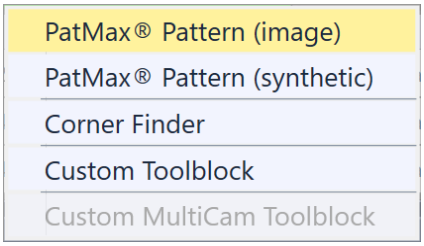
A Features Finder needs to be configured by user manually on HMI and saved in alignment recipe during machine setup. It is located under **Alignment** category. The alignment names and features finder names shown on the Navigation Tree are identical with the names of the corresponding components configured in the Configuration Wizard, such as "Alignment0" and "Features0" in the screenshot below:



A features finder is an AlignPlus toolblock that receives images from the multiple cameras that capture the image of a part, locates the features using the finders that are added by the user, and outputs a list of found features. The feature type could be point, line, or generic feature depending on the type selected in the corresponding finder component in the Configuration Wizard. Features finders therefore are classified into three types: Point Features Finder, Line Features Finder, and Generic Features Finder.

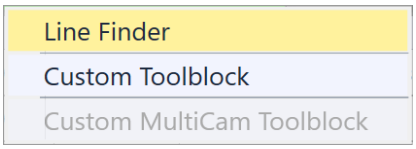
- Point Features Finder

  There are four types of point finders that users can choose for setup: PatMax Pattern(image), PatMax Pattern (synthetic), Corner Finder, and Custom Toolblock.



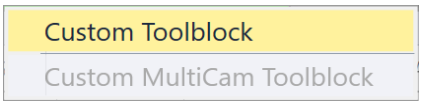
- Line Features Finder

  There are two types of line finders that users can choose for setup: Line Finder and Custom Toolblock.



- Generic Features Finder

  Generic feature finder only has one type: Custom Toolblock.



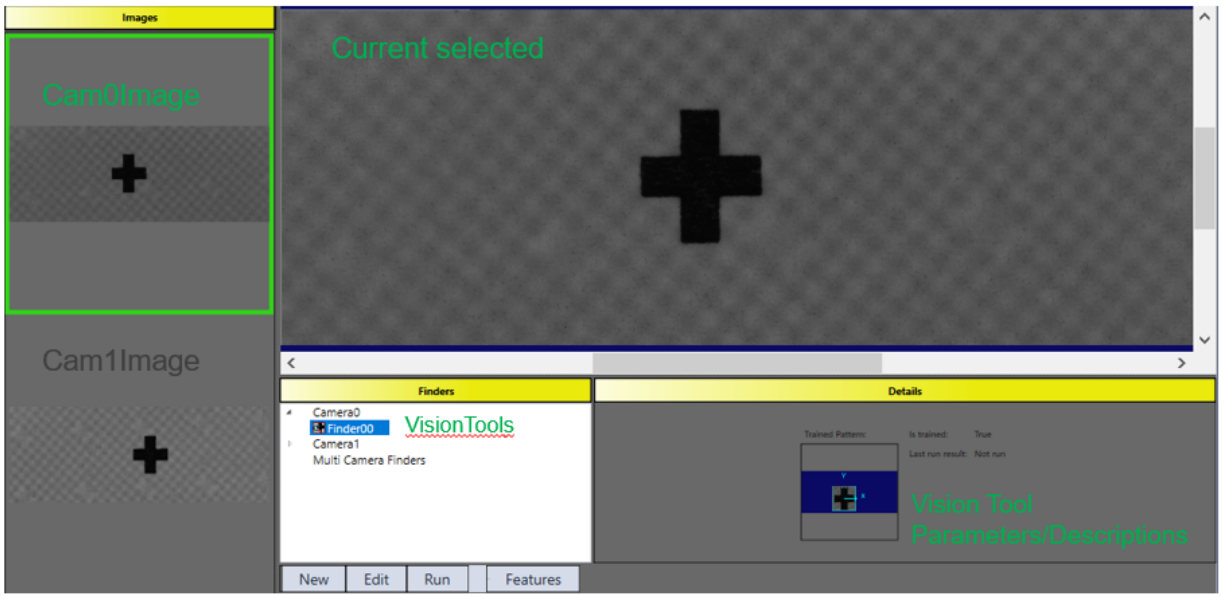
The following chapters will walk through step-by-step settings for all features finders and their finders within.

Point Features Finder

This UI controller allows the configuration of multiple point feature finders (for example, corner finders). When executed in a sequence, the block behind runs the configured finders on its input images and outputs a list of the found points and graphics.

Settings

Point Features Finder UI consists of four areas: Images, Main Display, Finders, and Details.



Images

This area lists out all available images for feature finding. The number of images is the same with the image input pins of bonded Point Feature Finder block inside the program. Images are visible on this UI only when there are valid images fed into the features finder block. If not, this area appears as blank and no further operations can be done.

Main Display

By clicking one of the images, the main display area will show the selected image in larger scale.

Finders

The finder area maintains all finders that are used to find features. If a finder locates feature using only one input image, that finder should be added under the corresponding camera icon. Otherwise, it should be added under “Multi Camera Finders” category.

There are several buttons allow user to create/edit/modify finders.

- **New**: Create a new finder such as PatMax Finder on page 113(image or synthetic), Corner Finder on page 118, or Custom Toolblock Point Finder on page 121.
- **Edit**: Edit the selected finder(rename, copy, delete, or edit)
- **Run**: Run the selected finder and show the result.

  If a finder runs successfully with a valid output feature, there will be a small green light ' ' shown at its right side. Otherwise, the light icon will be ' ', which indicates the feature is invalid or not found and will be ignored for pose computing. In this case, one need to resolve the error inside and make the finder run successfully.
- **Features**: Rename features found by finders. Feature names are the same with their finder names by default. However, users can rename to make them more meaningful. This function can also be used to pair features. For example, in assembly applications where Paired Features mode is used, features between two parts need to be paired using the same feature names.

Details

Details area is used to set parameters for selected finder.

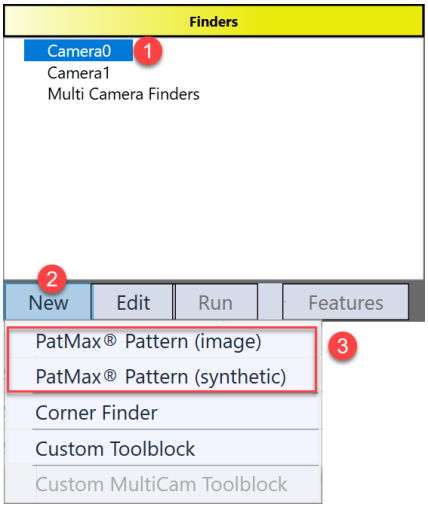
PatMax Finder

PatMax Finder finds a point feature using PatMax tool on input image.

Add

PatMax Finder is available in **Point Features Finder** HMI control. The steps to add it are:

1. Select one image
2. Click "New" button under the Finders panel
3. Choose "PatMax Pattern (image)" or "PatMax Pattern (synthetic)", then a PatMax finder will be available under the selected image for user to edit.

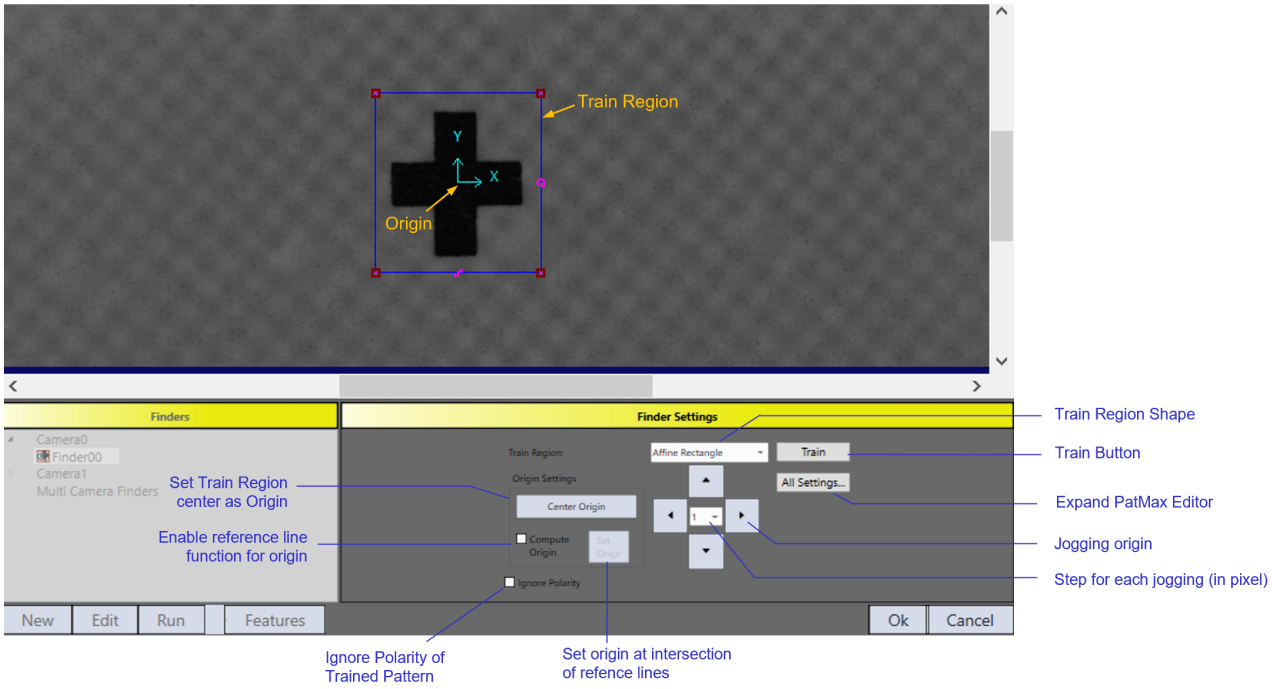


PatMax Pattern(Image) Finder

PatMax Pattern (Image) finder is a simplified PatMax tool which makes user focus on train time and run time parameter settings.

Train Time Settings

After PatMax Finder is added, the UI will automatically enter train time settings.

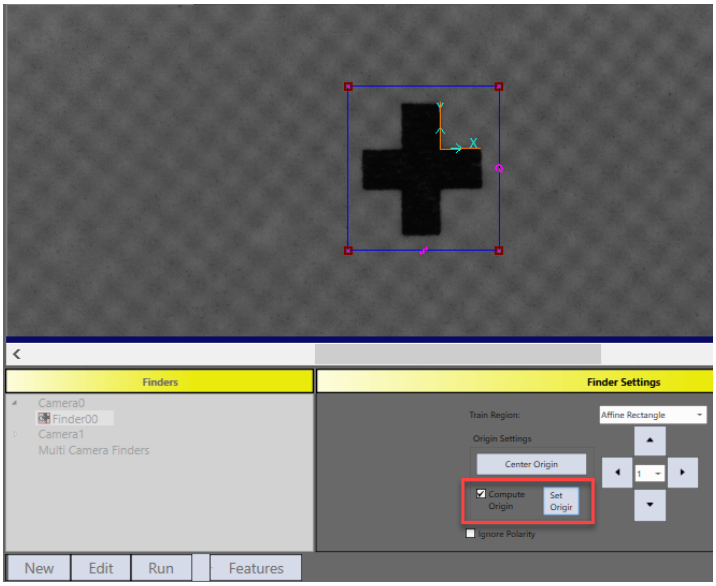


Category	Parameters	Description
Train Region	Train Region	Editable region on image.
	Train Region Shape	Select different options in the list for train region shape. The default option is affine rectangle.

Category	Parameters	Description
Origin	Origin	Origin of train pattern. It can be either manually moved on image display or jogged using four arrow buttons on **Finder Setting** panel. **Note**: The output feature's LocationX and LocationY are determined by the origin of found pattern on run time image, theta is determined by the angle difference from found pattern to trained pattern.
	Jogging origin	Four arrow-shape buttons for user to jog the origin on image display.
	Step for each jogging	Configure the number of pixels a jog moves
	Center Origin	Set the center point of train region as pattern origin.
	Compute Origin	Enable using two reference lines to compute their intersection as pattern origin.
	Set origin	After reference lines are manually placed and adjusted on image display, click this button to compute two lines' intersection and set it as pattern origin. It is enabled only when **Compute Origin** is checked.
Other Parameters	Ignore Polarity	Ignore the polarities of train time features.
	All Settings	Open PatMax tool editor for user to get access to all parameters in PatMax tool.
Train	Train	After all parameters are set, click this button to train pattern.

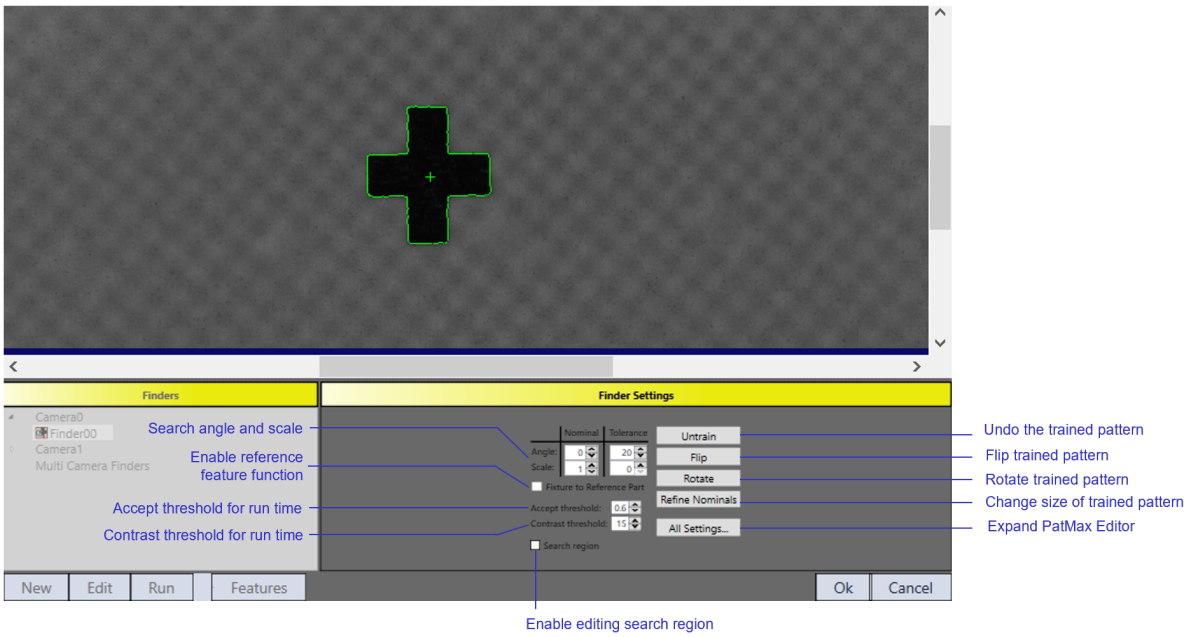
Compute Origin

In applications where PatMax Finder's output feature should report location of a specific point on a fiducial mark, Compute Origin can be used to precisely locate the point. Here is an example: after **Compute Origin** is checked, two orange colored reference lines will be available on image display. User then can manually align these two lines to two perpendicular edges of the cross mark, thereafter click **Set Origin** button to let the finder compute the intersection of these two lines and set the result point as pattern origin.



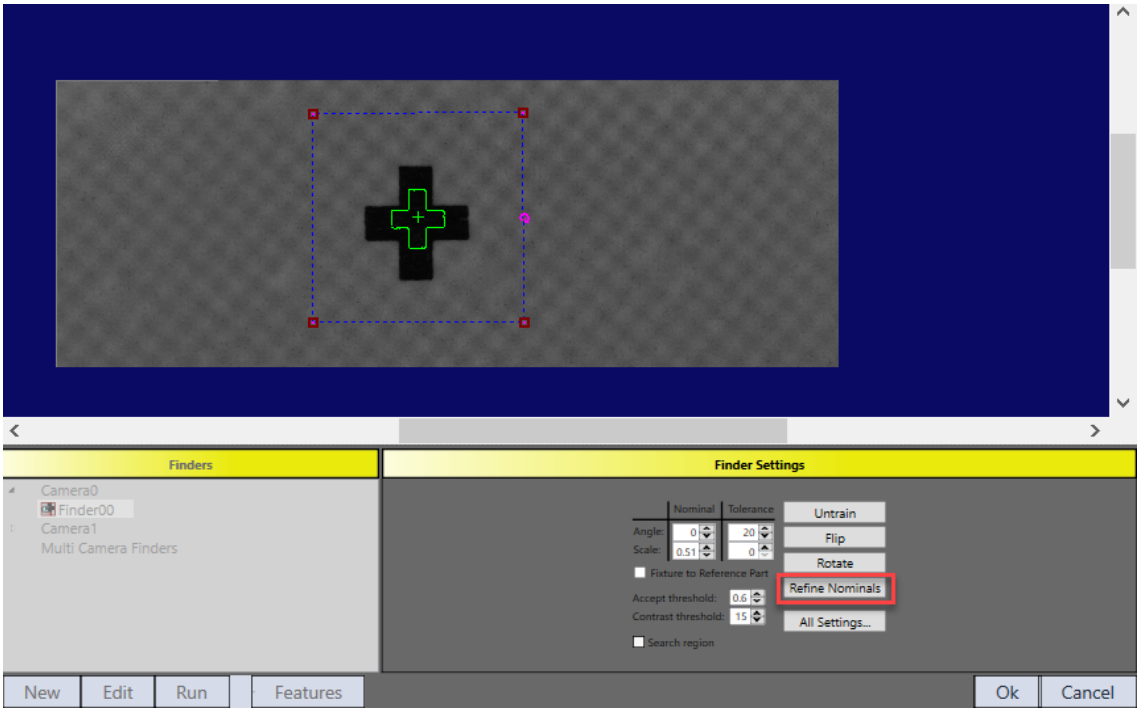
Run Time Settings

After pattern training, the **Finder Setting** panel moves next to run time settings.

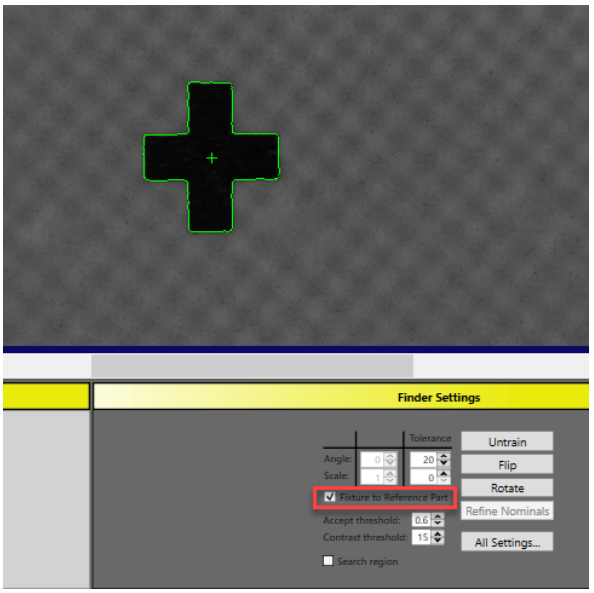


Category	Item	Description
Trained Pattern Origin	Flip	Flip trained pattern
	Rotate	Rotate trained pattern
	Refine Nominals	Change the size of trained pattern
	Untrain	Untrain the pattern, by clicking this button the HMI will roll back to train time settings
Run Time Parameters	Angle	Specifies the rotation range for pattern searching, the default angle values are from -20 to 20 degrees
	Scale	Specifies the scale range for pattern searching, the default value is 1 without tolerance
	Accept threshold	Coarse finding accept threshold. PatMax uses two steps for pattern finding: coarse finding and fine finding. Patterns found during coarse finding will be selected as candidates for fine finding only when their coarse scores exceeds this threshold.
	Contrast Threshold	Minimum acceptable contrast for a pattern instance. Only pattern instances where the average difference in pixel values across all feature boundaries exceeds the contrast threshold are considered by PMAlign
Search region	Search region	Enable/disable editing search region. When it's disabled, PatMax finder uses the entire image as search region.
Other Parameters	Enable a Reference Part	Disabled: Current PatMax Finder will be used for both golden pose training and run time feature finding Enabled: Current PatMax Finder will only be used for golden pose training(reference mode). Run time feature finding requires another finder.
	Expand PatMax Editor	Open PatMax tool editor for user to get access to all parameters in the PatMax tool

Refine Nominals button enables users to edit the scale of trained pattern. When the trained pattern and run time patterns have a constant scale difference, rescaling the trained pattern to make its size the same as run time patterns will reduce searching time for PatMax tool.



Enable **Fixture to Reference Part** option means current finder is only used for golden pose training, not for run time feature finding.

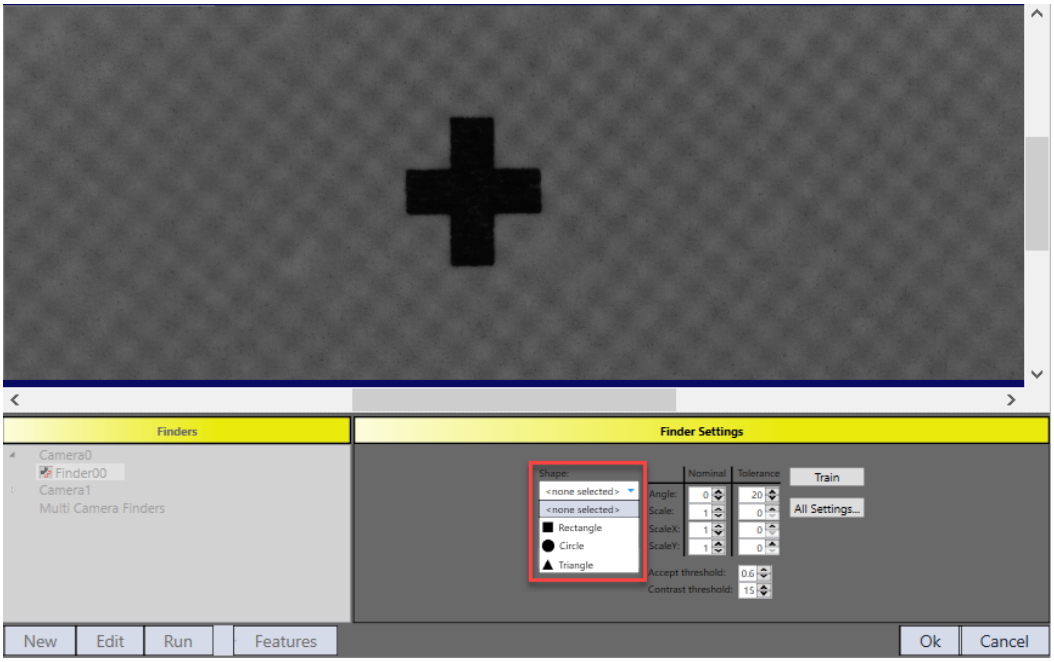


For more information about reference part, please refer to **Use reference part's feature position** in Golden Pose topic.

PatMax Pattern(Synthetic)

PatMax Pattern(Synthetic) is to train patterns which has regular geometrical shapes.

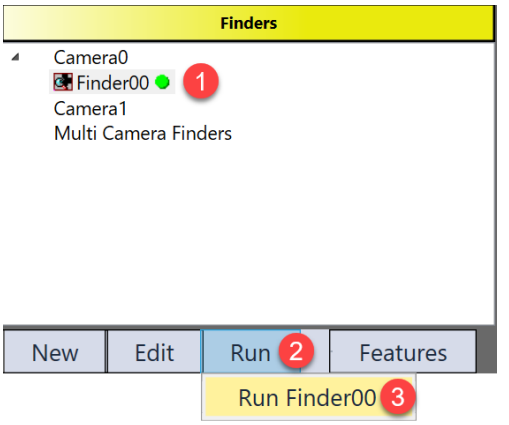
This finder offers rectangle, circle and triangle shapes for user's immediate choice. As for other shapes such as cross, manual drawing in Synthetic mode in advanced options of PatMax is needed.



For more details about PatMax tool, please refer to **PMAlign Edit Control** in VisonPro documentation.

Run

After configuring all parameters, click "OK" button, and run PatMax Finder to check the result.



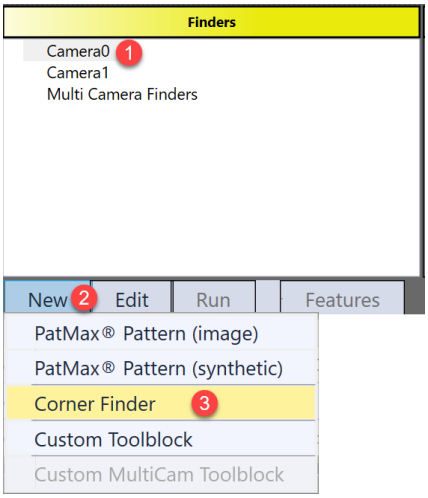
Corner Finder

Corner finder finds two edge lines around a corner and uses their intersection as the output point feature.

Add

Corner Finder is available in **Point Features Finder** HMI control. The steps to add it are:

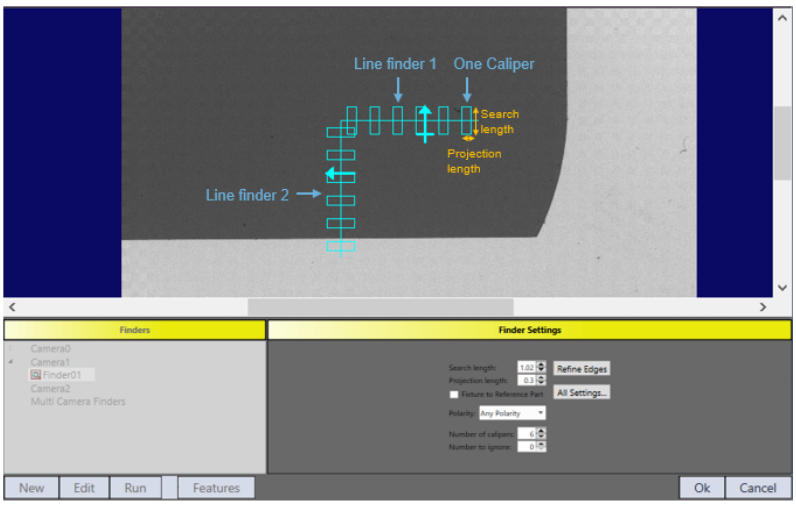
1. Select one image
2. Click "New" button under **Finders** panel
3. Choose "Corner Finder". After this, a corner finder will be available for user to edit



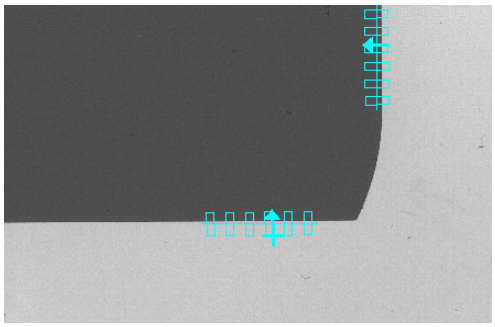
Setting

The setting steps of a Corner Finder are very similar with those of a Line Finder on page 127. The only difference is that Corner Finder has two lines to find:

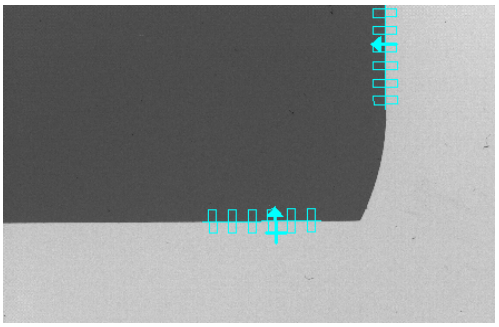
1. Align the two references lines to the two corner edges on image(s).



Click **Refine Edges** button to fine tune reference lines to real edges.



before refining edges



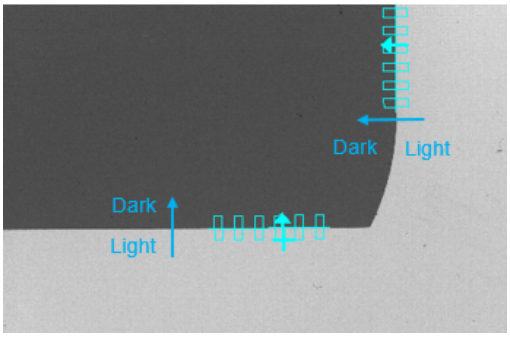
after refining edges

2. Adjust search length and projection length to find edges properly.

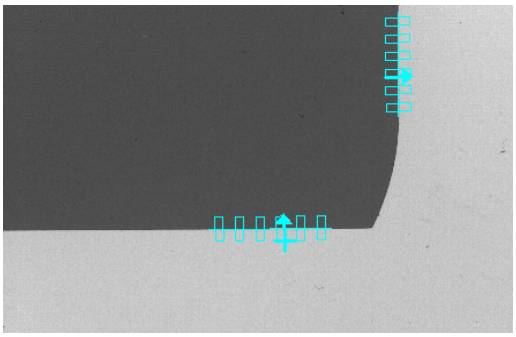
The two edge find tools share the same search length and projection length.

3. Choose polarity

Choose whether the edges are indicated by a dark to light transition, or light to dark transition, or any polarity along the arrow directions. The two edge find tools share the same polarity parameter, therefore the two arrow directions should be coordinated.

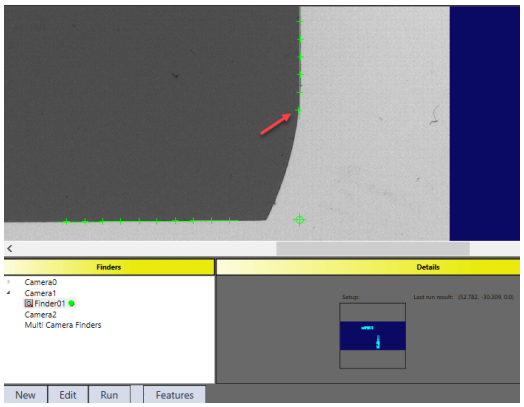


The following arrow directions will lead to an error unless “Any polarity” is chosen.

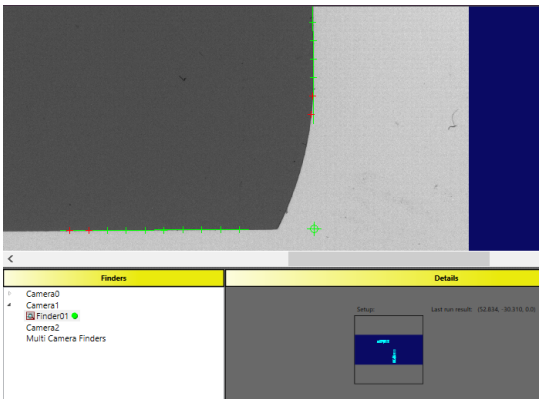


4. Adjust caliper number and ignore number

Number to ignore is the number of points that will be ignored in the fitting operation. Corner finder automatically filters out the given number of outliers that are farthest from fitting lines. Setting this number as non-zero value is important when pseudo edge points could impact line fitting results.



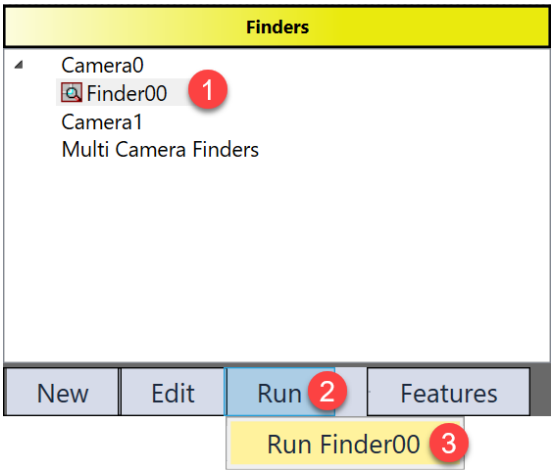
Number to ignore = 0



Number to ignore = 2

Run

After the settings, click "Ok" button, and then run the added corner finder to check the result following the steps shown below.



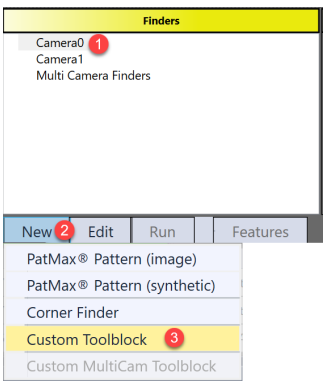
Custom Toolblock Point Finder

Custom Toolblock Point Finder is point feature finder that allows user to customize how the point should be found.Line Finder on page 127

Add

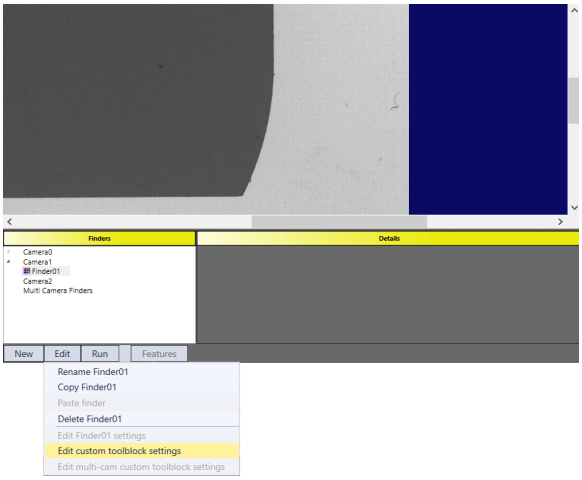
Custom Toolblock Point Finder is available in **Point Features Finder** HMI control. The steps to add it are:

1. Select one image
2. Click "New" button under the Finders panel
3. Choose "Custom Toolblock", then a custom toolblock point finder will be available for user to edit.

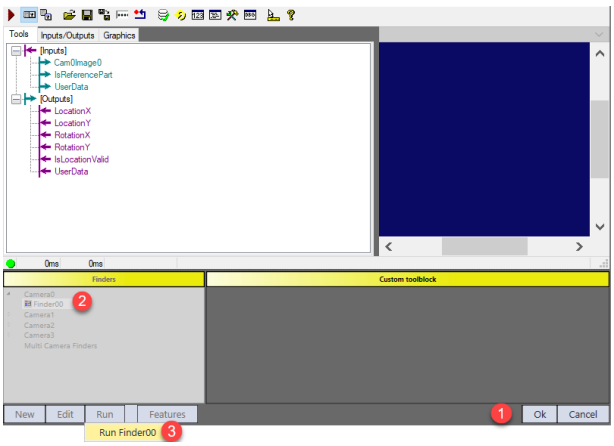


Setting

Settings of this finder are all contained in its custom toolblock. Follow the steps to open the toolblock: 1) select the finder, 2) click “Edit” button, 3) choose “Edit custom toolblock settings”.



However, the input image of this toolblock is initially null because the finder has not been run yet. To get input image, click “Ok” button, run the current finder once in Finder panel, and open the toolblock again. Then, the toolblock is ready for user to customize.



Inputs

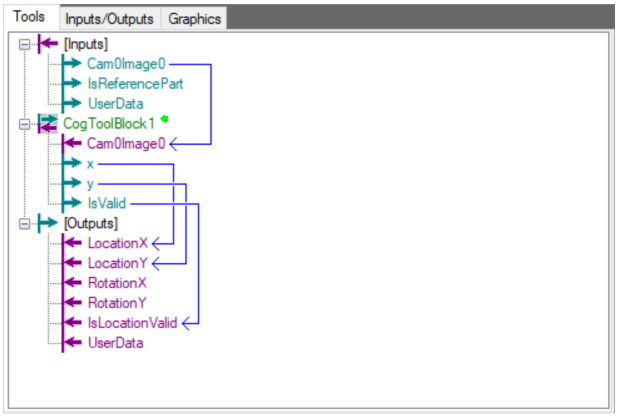
Parameters	Type	Description
Cam0Image0/Cam1Image1	Cognex.VisionPro.ICogImage	Input image(s) from corresponding camera(s)
InRefererecePart	Boolean	A input pin for user to decide whether and how to use it for golden pose training using reference part
User Data	Cognex.VisionPro.CogDictionary	Contains a CogDictionary with user specified data.

Outputs

Name	Type	Description
LocationX	Double	The X coordinate of the found feature. This value is undefined if IsFound = false.
LocationY	Double	The Y coordinate of the found feature. This value is undefined if IsFound = false.
RotationX	Double	The angle of the X Axis of the found feature(in radian). This value is undefined if IsFound = false and is valid only if found by a PatMax tool.
RotationY	Double	The angle of the Y Axis of the found feature(in radian). This value is undefined if IsFound = false and is valid only if found by a PatMax tool.
IsLocationValid	Boolean	The flag to show whether a feature is found or not. Later will be used to decide whether or not to use this feature to compute pose in A+ tasks.
UserData	CogDictionary	More data can be stored inside UserData as an output, this UserData is only accessible in scripting.

Editing

It is recommended to create an inner toolblock that contains all custom vision tools to keep the finder's toolblock neat.

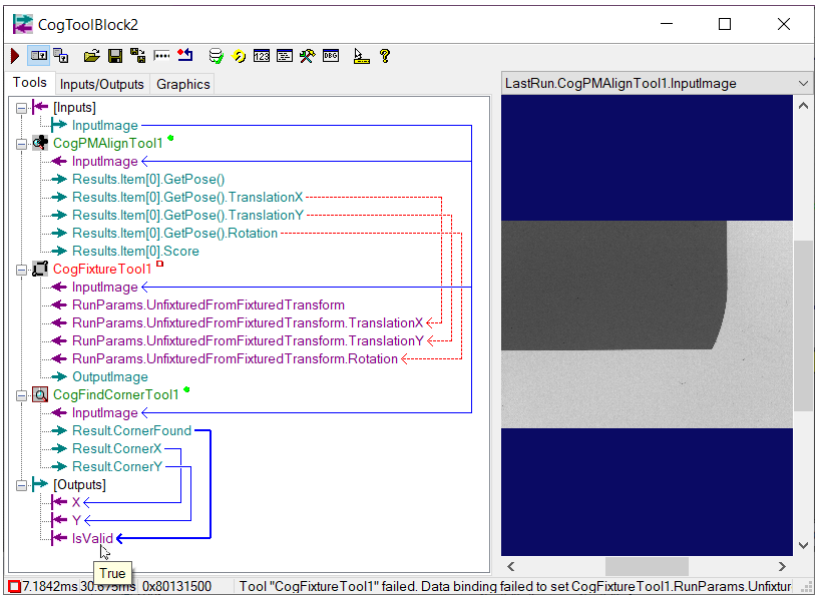


Note: If it is one point alignment, then the part's rotation must be computed in the inner toolblock and passed to RotationX and RotationY, so that the output feature will have x, y, and theta information for pose computation. The units of RotationX and RotationY within the finder's toolblock are radians, the finder will later transfer them into degrees and save them in a CogAlpsPointFeature object.

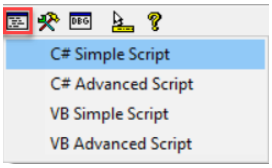
User has the flexibility to decide how the point should be found within the inner toolblock. However, few standard procedure should be considered:

1. Run time error check

   When toolblock has any run time error, its output pins keep their previous data. Which means if **IsValid** was previously true, it remains true even the inner toolblock has exceptions. Following that, AlignPlus pose computer will take current point as a valid feature and use it for pose computing without knowing feature finding is actually failed.

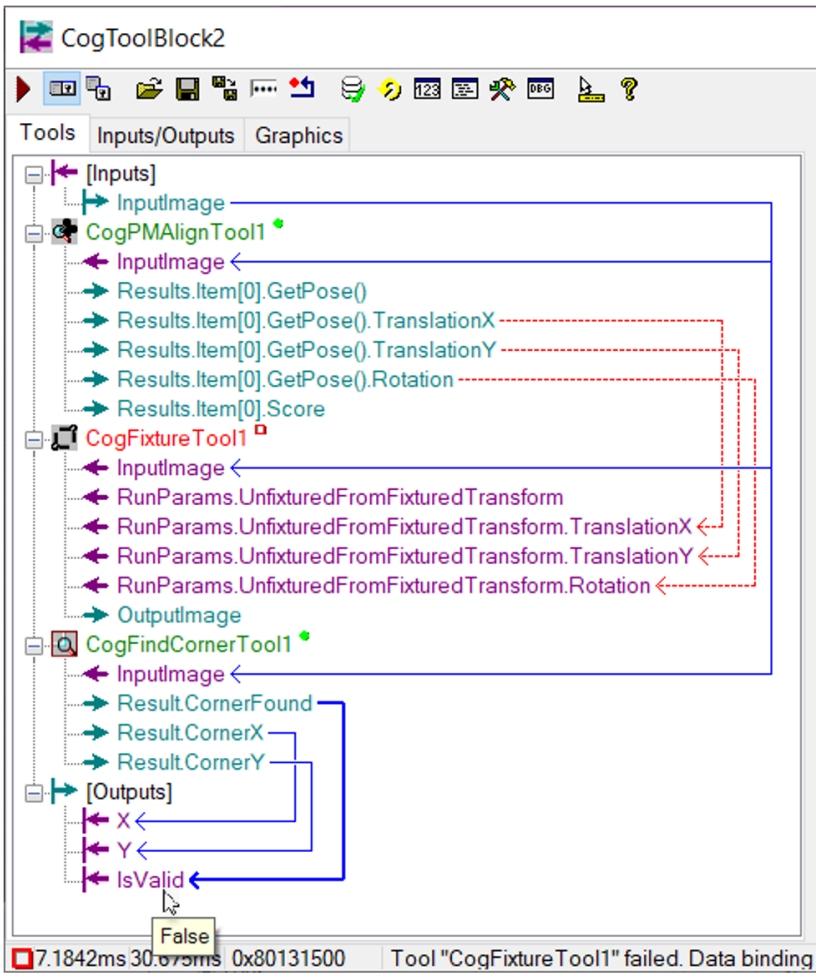


   To avoid this problem, some scripts need to be added in inner toolblock to check run time error and update IsValid output pin.




```
public override bool GroupRun(ref string message, ref CogToolResultConstants result)
{
  // To let the execution stop in this script when a debugger is attached, uncomment the following lines.
  // #if DEBUG
  // if (System.Diagnostics.Debugger.IsAttached) System.Diagnostics.Debugger.Break();
  // #endif

  Outputs.IsValid = true;
  // Run each tool using the RunTool function
  try
  {
    foreach(ICogTool tool in Tools)
      RunTool(tool, ref message, ref result);

    //Check if PatMax has successfully found feature
    if(this.Tools.CogPMAlignTool1.Results == null)
      Outputs.IsValid = false;
    else if(this.Tools.CogPMAlignTool1.Results.Count==0)
      Outputs.IsValid = false;
  }
  catch(SystemException ex)
  {
    Outputs.IsValid = false;
  }

  return false;
}
```

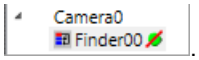
After applying the script, the IsValid pin will be false if there is any run time errors within the inner toolblock.



2. Space selection

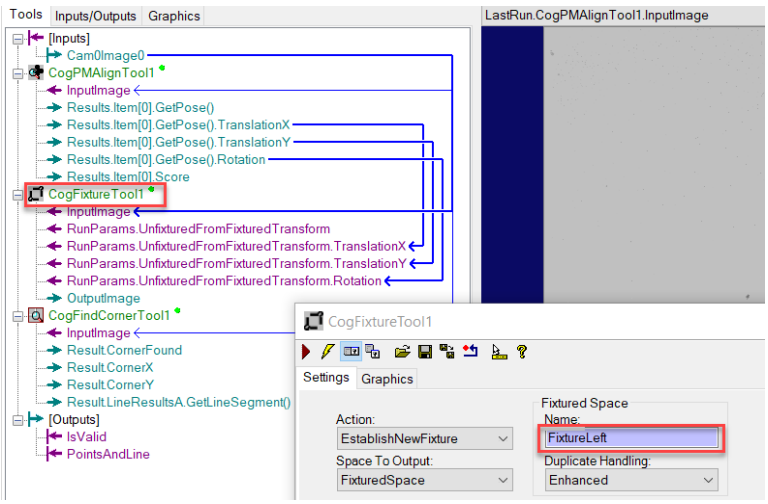
   In the example above, CogFindCornerTool's output results are based on image's selected space, but its search region follows CogFixtureTool's Fixture space. For more information on how and why to configure in this way, please refer to **Use Different Space for ROI and Returned Result** in Space Selection on page 326.

3. Use different fixture names for each finder

   Under one Point Features Finder, there might be multiple Custom Toolblock finders. If all of them are using CogFixtureTool whose default output space names are all "Fixture", there will be a name space confliction, and result in Finder failure 

.

   To solve this confliction, user can give different fixture space names for every CogFixtureTools used in current Point Features Finder.

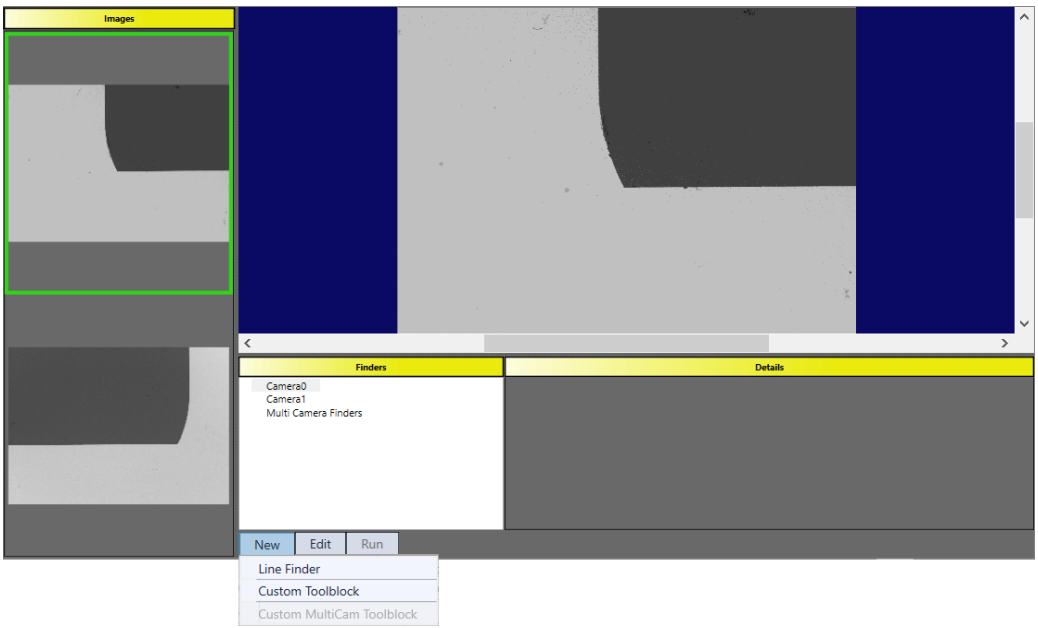


Run

After finish customizing the tool block, click "Ok" button and run the finder to check the result.

Line Features Finder

This UI controller allows the configuration of multiple line feature finders (for example, line finders). When executed in a sequence, this block behind runs the configured finders on its input images and outputs a list of the found lines and graphics.



Line Features Finder UI consists of four areas: Images, Main Display, Finders, and Details.

Images

This area lists out all available images for feature finding. The number of images is the same with the image input pins of bonded Line Feature Finder block inside the program. Images are visible on this UI only when there are valid images fed into that block. If not, this area appears as blank and no further operations can be done.

Main Display

By clicking one of the images, the main display area will show the selected image in larger scale.

Finders

The finder area maintains all finders that are used to find features. If a finder locates feature using only one input image, that finder should be added under the corresponding camera icon. Otherwise, it should be added under “Multi Camera Finders” category.

There are several buttons allow user to create/edit/modify finders.

- **New**: Create a new finder such as Line Finder on page 127, or Custom ToolBlock Line Finder on page 130
- **Edit**: Edit the selected finder(rename, copy, delete or edit)
- **Run**: Run the selected finder for user to check the result.
- **Features**: Rename features found by finders. This function can also be used to pair features. For example, in assembly applications where Paired Features mode is used, features between two parts need to be paired using the same feature names.

Details

Details area is used to set parameters for selected finder.

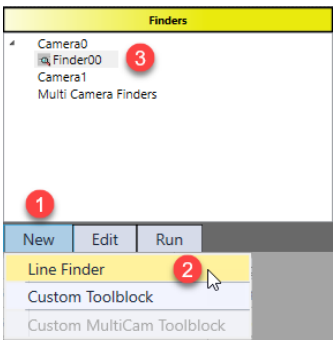
Line Finder

A line finder, represented by a VisionPro CogFindLineTool tool. This tool is set up to find a line feature on a single selected camera image.

Add

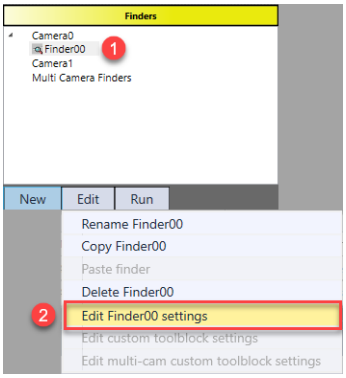
Line Finder is available in **Line Features Finder** HMI control. The steps to add it are:

1. Select one image
2. Click "New" button under the Finders panel
3. Choose "Line Finder", then a line finder will be available under the selected image for user to edit.



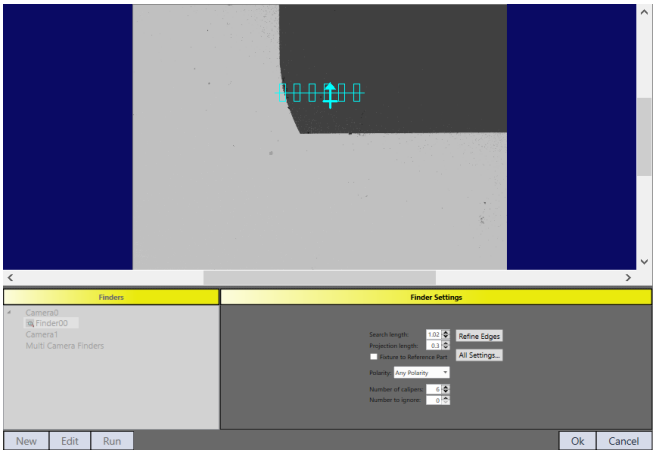
Setting

To enter edit mode, first select the finder, then choose **Edit finder settings** under "Edit" button.

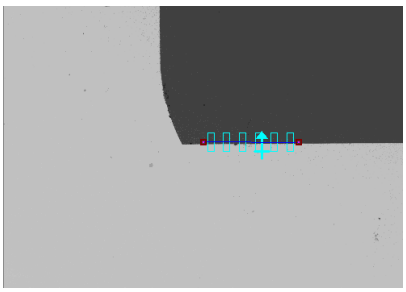


Here are steps to set it up:

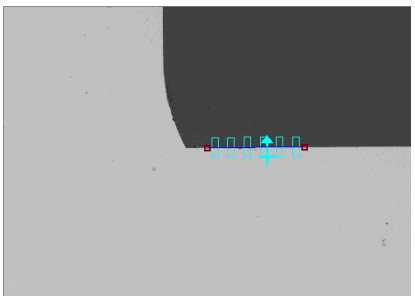
1. Align the reference line to the target edge.



Click **Refine Edges** button to automatically fine tune the reference line to real edge.



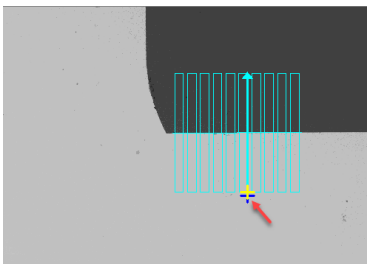
before refining edges



After refining edges

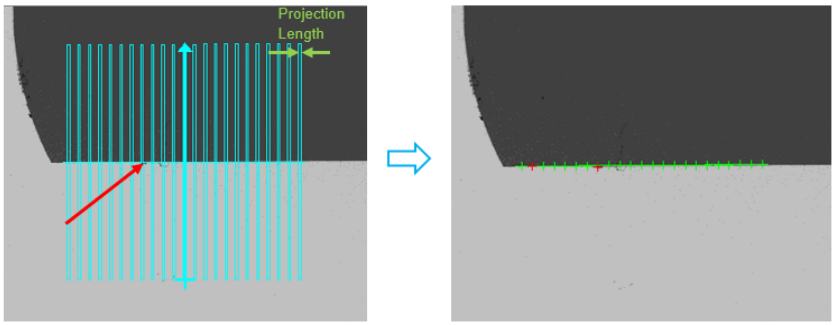
2. Adjust search length and projection length.

Line finer only finds edge points within its search length. Therefore, to cover run time part location variances, it is recommended to make the search length long enough during setup time. A easy way to adjust search length is to drag and drop the cross mark on CogFindLineTool interactive graphic.

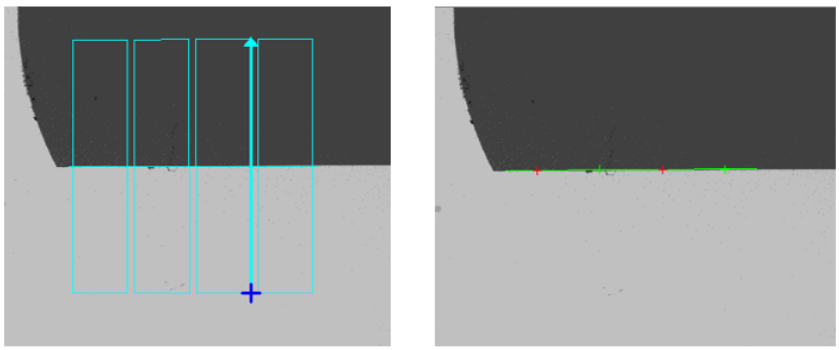


Projection Length is the width of calipers. The selection of proper width is based on the length of expected edge and the evenness if it. Here are two extreme cases:

- Calipers are too thin to resist noise



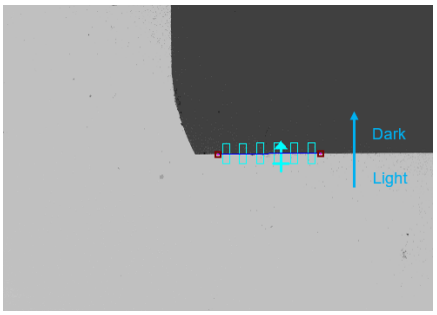
- Calipers are too wide to have enough number of calipers



Projection Length can also be adjusted by dragging and dropping the cross mark on CogFindLineTool interactive graphic.

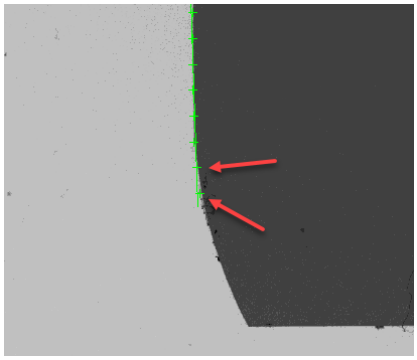
3. Choose polarity

Choose whether the edge is indicated by a dark to light transition, or light to dark transition, or any polarity along the arrow direction. In the image below, polarity can be set as "Light to Dark" or "Any Polarity" . However, "Light to Dark" is better than "Any Polarity" because it resists more noises.

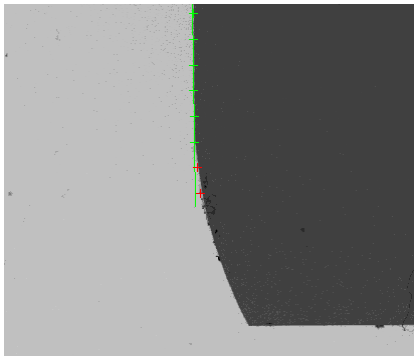


4. Adjust caliper number and ignore number

Number to ignore is the number of points that will be ignored in the fitting operation. Line finder automatically filters out the given number of outliers that are farthest from the fitting line. Setting this number as non-zero value is important when pseudo edge points could impact line fitting result.



Number to ignore = 0

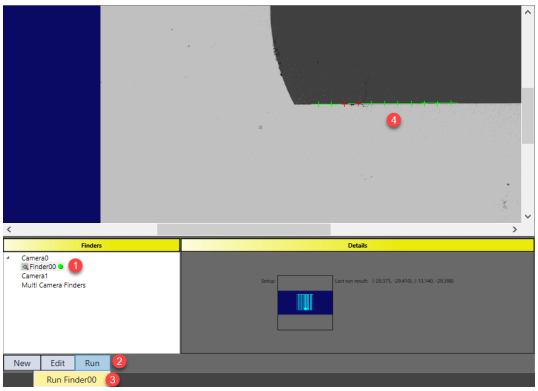


Number to ignore = 2

Note: Even the current setup image does not have any noise, it's still recommended to set ignore number to 2-3 in case there could be noises in run time.

Run

After the settings, click "Ok" button, and then run the added line finder to check the result following the steps shown below.

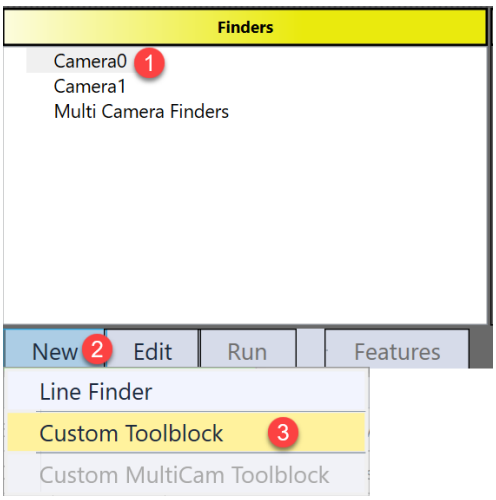


Custom ToolBlock Line Finder

Custom Toolblock Line Finder is line feature finder that allows user to customize how the line should be found through Line Finder on page 127

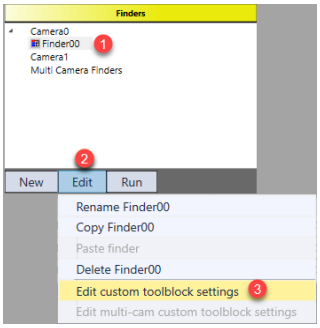
Add

Custom Toolblock Line Finder is available in **Line Features Finder** HMI control. The way to add it is first select a image; second, click the "New" button under **Finders** panel; and last, choose "Custom Toolblock".

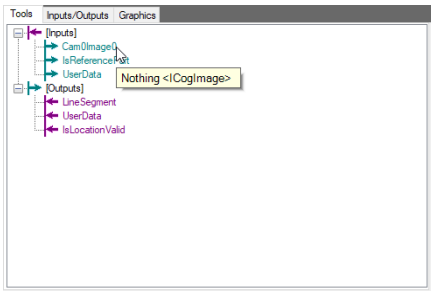


Setting

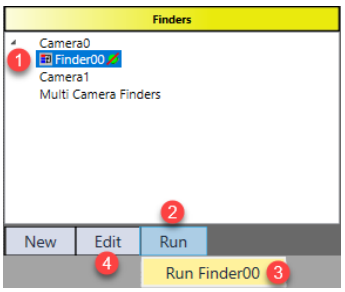
All settings of this finder are contained in the custom toolblock. Follow the steps to open the toolblock: 1) select the finder, 2) click “Edit” button, 3) choose “Edit custom toolblock settings”.

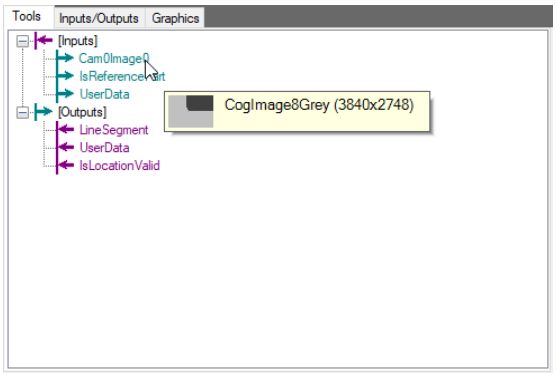


However, the input image of the toolblock is initially null because this Finder has not been run yet.



To get input image, click “Ok” button, run the current finder once in Finder panel, and open the toolblock again. Then, the toolblock is ready for user to customize.





Inputs

Parameters	Type	Description
Cam0Image0/Cam1Image1	Cognex.VisionPro.ICogImage	Input image(s) from corresponding camera(s)

Parameters	Type	Description
InRefererecePart	Boolean	A input pin for user to decide whether and how to use it for golden pose training using reference part
User Data	Cognex.VisionPro.CogDictionary	Contains a CogDictionary with user specified data.

Outputs

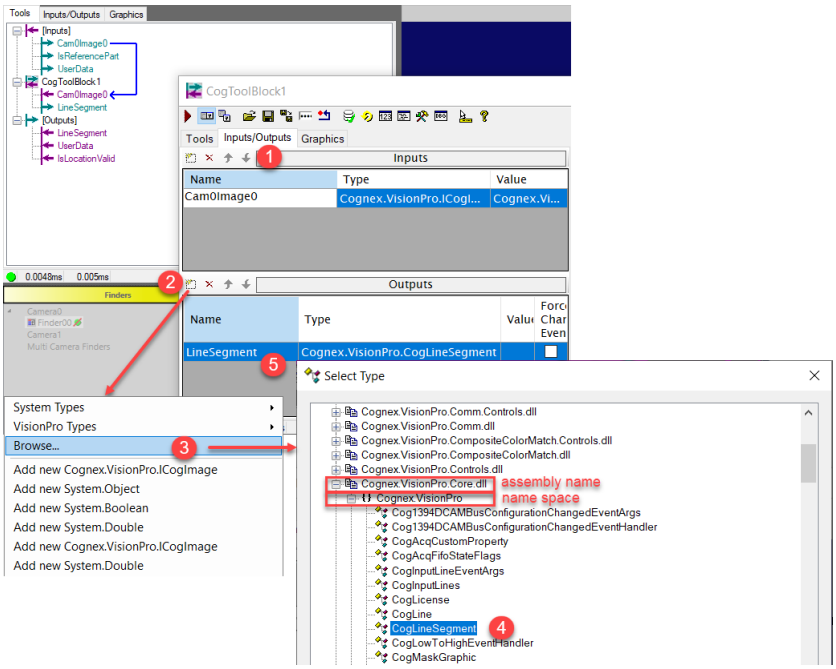
Parameter	Type	Description
LineSegment	Cognex.VisionPro.CogLineSegment	The found line segment.
UserData	Cognex.VisionPro.CogDictionary	Specifies extra data that user wants to output, can be left as null
IsLocationValid	Boolean	Specifies whether the line feature is successfully found.

Editing

Custom Tool Block Line Finder does nothing unless user configure vision tasks within it. User can directly add tools here or add an inner tool block first and then edit tools within it. The latter is recommended as it keeps the outside toolblock neat.

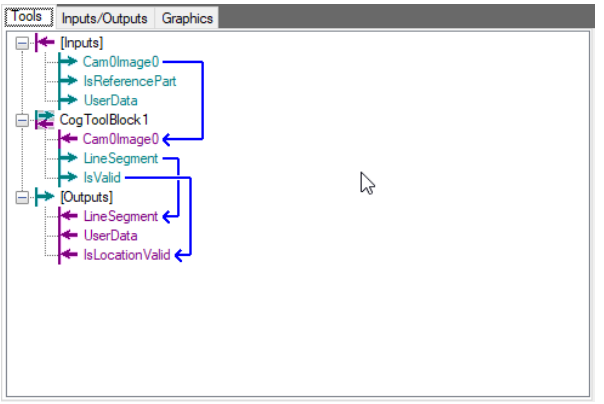
1. Within the inner tool block, add inputs and outputs

CogLineSegment type is under "Cognex.VisionPro" namespace within "Cognex.VisionPro.Core" assembly. Therefore, users can follow the following steps to add "LineSegment" ouput.



Add another output pin "IsValid" as a Boolean.

2. Link inner and out tool block input and output pins.



3. Editing inner tool block

User can design inside how the target line should be found. Here is one example of it.

1) Add CogPMAlignTool tool to locate the part

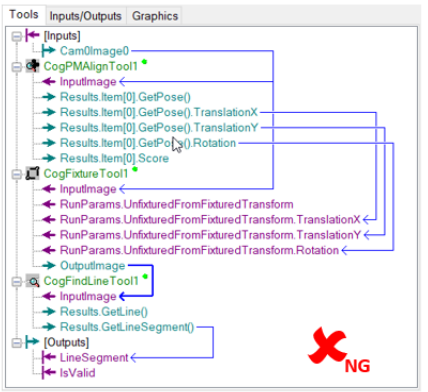
PatMax tool here helps locate the part and guide CogFindLineTool to search specific ROI on the part in run time.

2) Add CogFixtureTool to create a fixture space

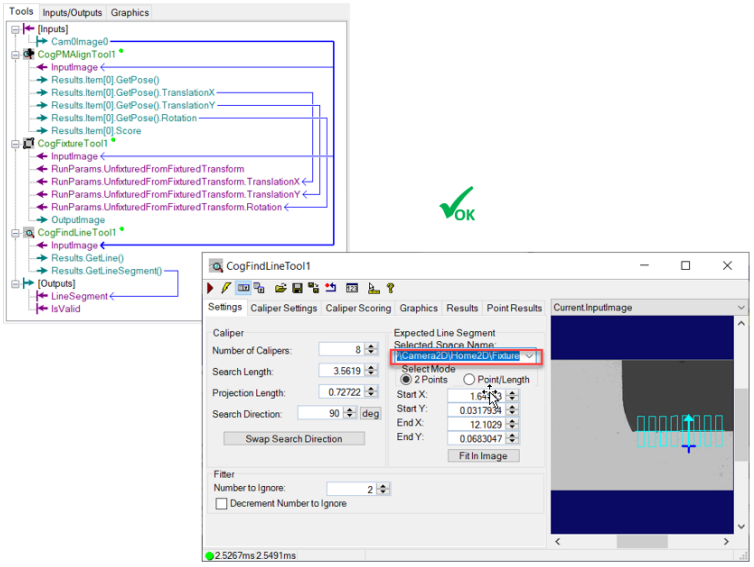
CogFixtureTool creates a new fixture space based on part's current location.

3) Add CogLineFinderTool to find a line

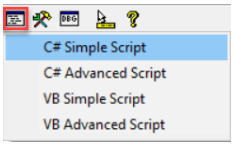
There is one detail to be considered: If CogLineFinderTool directly use the output image from CogFixtureTool, then the LineSegment result will be based on fixture space, not Home2D space.



To output Home2D space result as expected, CogLineFinderTool needs to get input image from the source image (whose selected space is Home2D) of the toolblock, and change its ROI region to use the fixture space generated by the CogFixtureTool. This operation will make sure CogLineFinderTool track the run time part and output result in Home2D.



4. Add script code inside inner tool block to check whether inner tool block runs successfully.




```
public override bool GroupRun(ref string message, ref CogToolResultConstants result)
{
  // To let the execution stop in this script when a debugger is attached, uncomment the following lines.
  // #if DEBUG
  // if (System.Diagnostics.Debugger.IsAttached) System.Diagnostics.Debugger.Break();
  // #endif

  Outputs.IsValid = true;
  // Run each tool using the RunTool function
  try
  {
    foreach(ICogTool tool in Tools)
      RunTool(tool, ref message, ref result);
  }
  catch(System.Exception ex)
  {
    Outputs.IsValid = false;
  }

  return false;
}
```

5. Test inside inner tool block to see if "IsValid" output reflects the ok/ng result correctly.

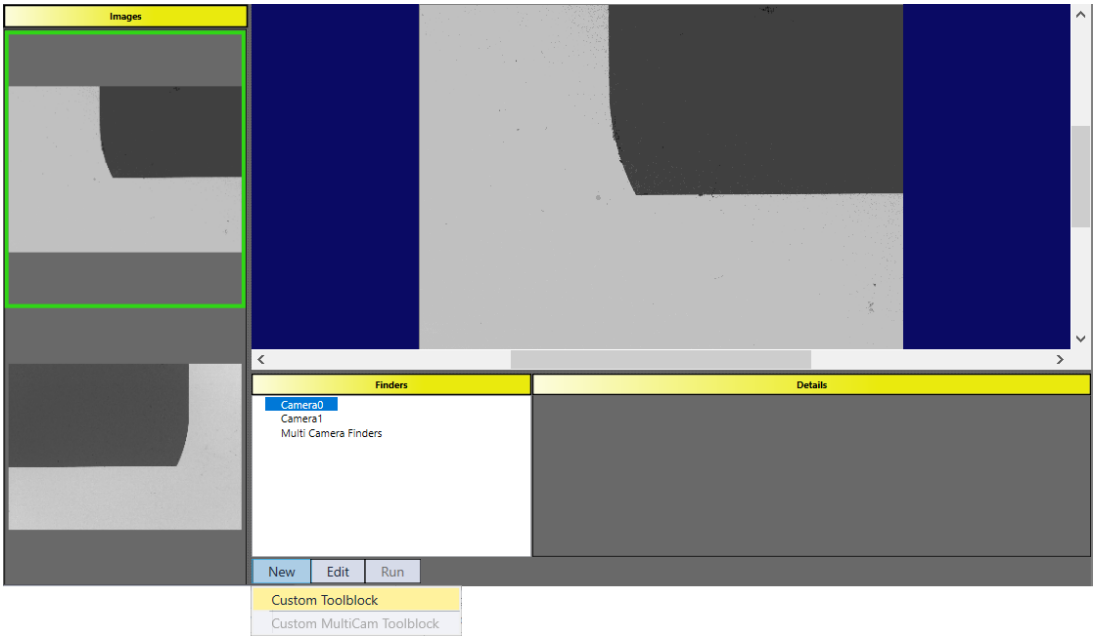
Note: When there are more than one CogFixtureTools used in a Line Features finder, these CogFixtureTools' output spaces should be renamed differently to avoid space confliction. See more information about it in **Editing** in Custom Toolblock Point Finder on page 121.

Run

After setting up the tool block, click "Ok" button and run finder to check the result.

Generic Features Finder

Generic Features Finder UI is the interactive HMI element of Generic Features Finder tool block. It allows user to configure multiple generic feature finders. When executed in a sequence, the block behind runs the configured finders on its input images and outputs a list of the generic features and graphics.



Settings

Point Features Finder UI consist of four areas: Images, Main Display, Finders, and Details.

Images

This area lists out all available images for feature finding. The number of images is the same with the image input pins of bonded Generic Feature Finder block inside the program. Images are visible on this UI only when there are valid images fed into the feature finder block. If not, this area appears as blank and no further operations can be done.

Main Display

By clicking one of the images, the main display area will show the selected image in larger scale.

Finders

The finder area maintains all finders that are used to find features. If a finder locates feature using only one input image, that finder should be added under the corresponding camera icon. Otherwise, it should be added under “Multi Camera Finders” category.

There are several buttons allow user to create/edit/modify finders.

- **New**: Create a new Custom ToolBlock Generic Finder on page 135
- **Edit**: Edit the selected finder(rename, copy, delete or edit)
- **Run**: Run the selected finder for user to check the result.
- **Features**: Rename features found by finders. Feature names are the same with their finder names by default. However, users can rename to make them more meaningful.

Details

Details area is used to set parameters for selected finder.

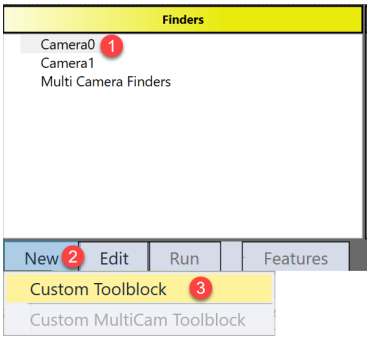
Custom ToolBlock Generic Finder

Custom ToolBlock Generic Finder is a finder that allows user to find a generic feature using custom tool block. A generic features is a System.Object which can be of any .NET type. It provides the user the flexibility to locate any type of feature such as lines, angles, fiducials etc. Here is an example to show how to set a point and a line segment as one generic feature in custom toolblock.

Add

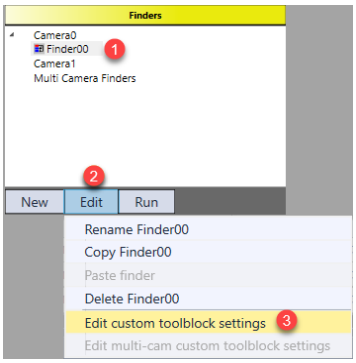
Custom Toolblock Line Finder is available in **Generic Features Finder** HMI control. The steps to add it are:

1. Select one image
2. Click "New" button under the Finders panel
3. Choose "Custom Toolblock", then a custom toolblock generic finder will be available for user to edit.

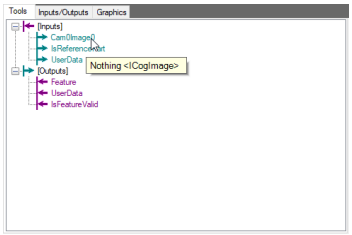


Setting

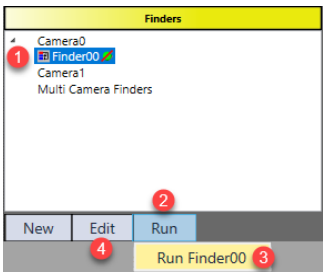
Settings of this finder are all contained in its custom toolblock. Follow the steps to open the toolblock: 1) select the finder, 2) click “Edit” button, 3) choose “Edit custom toolblock settings”.

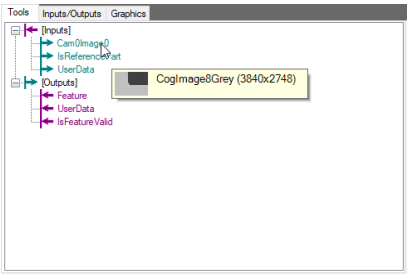


However, the input image of this toolblock is initially null because the finder has not been run yet.



To get input image, click “Ok” button, run the current finder once in Finder panel, and open the toolblock again. Then, the toolblock is ready for user to customize.





Inputs

Parameters	Type	Description
Cam0Image0 /Cam1Image1	Cognex.VisionPro.ICogImage	Input image(s) from corresponding camera(s)
InRefererecePart	Boolean	A input pin for user to decide whether and how to use it for golden pose training using reference part
User Data	Cognex.VisionPro.CogDictionary	Contains a CogDictionary with user specified data.

Outputs

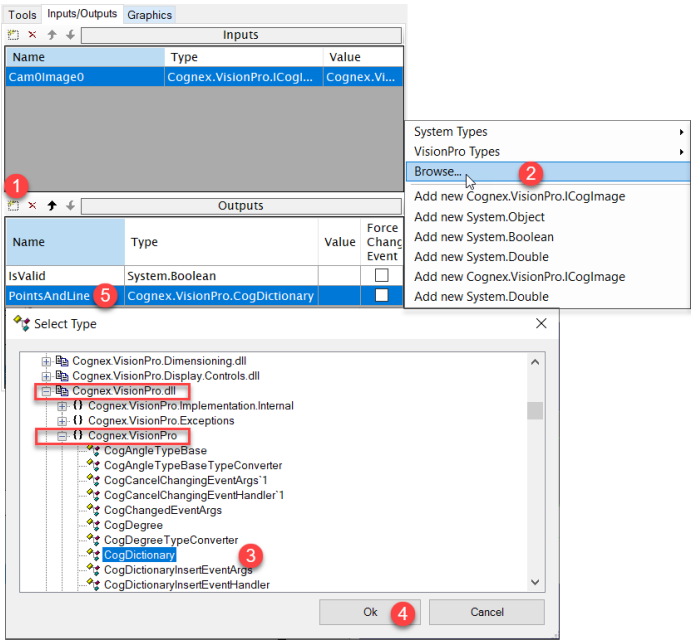
Parameters	Type	Description
Feature	Object	The result generic feature.
IsLocationValid	Boolean	The flag to show whether a feature is found or not. Later will be used to decide whether or not to use this feature to compute pose.
UserData	CogDictionary	More data can be stored inside UserData as an output, this UserData is only accessible in scripting.

Editing

Custom Tool Block Generic Finder does nothing unless user configure vision tasks inside it. Users can directly add tools here or add an inner tool block and edit tools within it. The latter is recommended as it will keep finder neat.

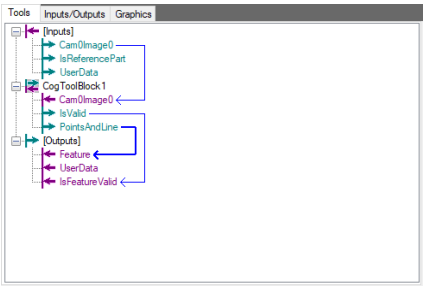
1. Within the internal tool block, add inputs and outputs

To contain a corner point and a line segment into one generic feature, user can use CogDictionary as the output type.



Add another output pin "IsValid" as a Boolean.

2. Link inner and out tool block input and output pins



3. Editing inner tool block

User have the flexibility to design how the target line and point should be found, how the output generic feature should be composed in this toolblock. Here is an example for reference:

1) Add CogPMAlignTool tool to locate the part.

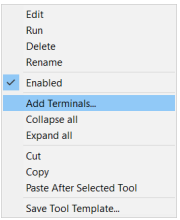
The PatMax tool here helps locate the part and guide the following CogCornerFindTool to target specific ROI on part in run time.

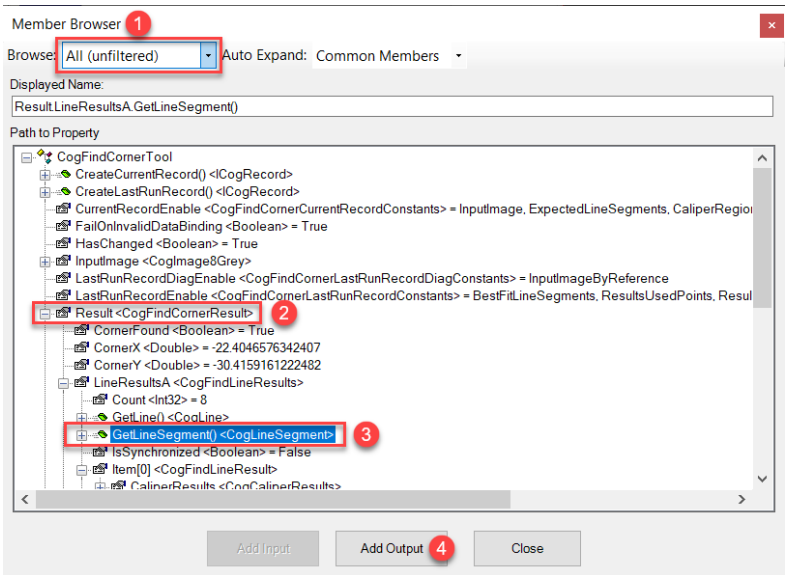
2) Add CogFixtureTool to create a fixture space.

CogFixtureTool creates a new fixture space based on part's current pose.

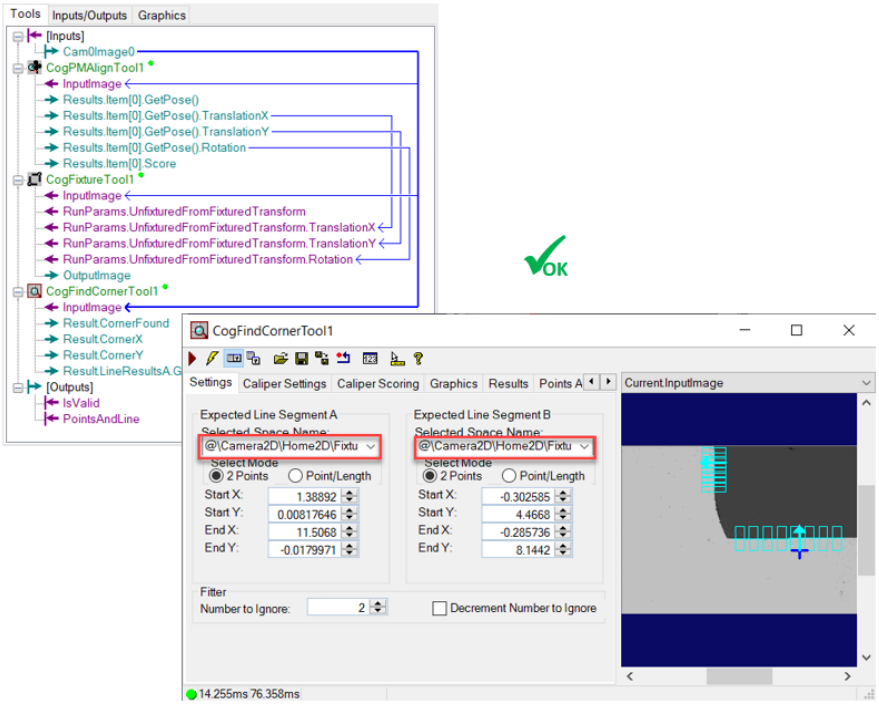
3) Add CogFindCornerTool to find two lines and one corner point.

CogFindCornerTool outputs only the corner point by default. To add an extra line segment output, right click the CogFindCornerTool and select "add terminal", in the opened dialog select the target line segment and then click "Add Output".



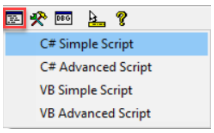


Link CogCornerFinderTool's input image to the source image (whose selected space is Home2D) of this toolblock, and change CogCornerFinderTool's ROI regions to use the fixture space generated by the CogFixtureTool. This will make sure CogCornerFinderTool track the run time part and output results in Home2D.



I

4. Add Scripting inside inner tool block to contain the point and the line segment into a CogDictionary output and check whether tool block runs successfully



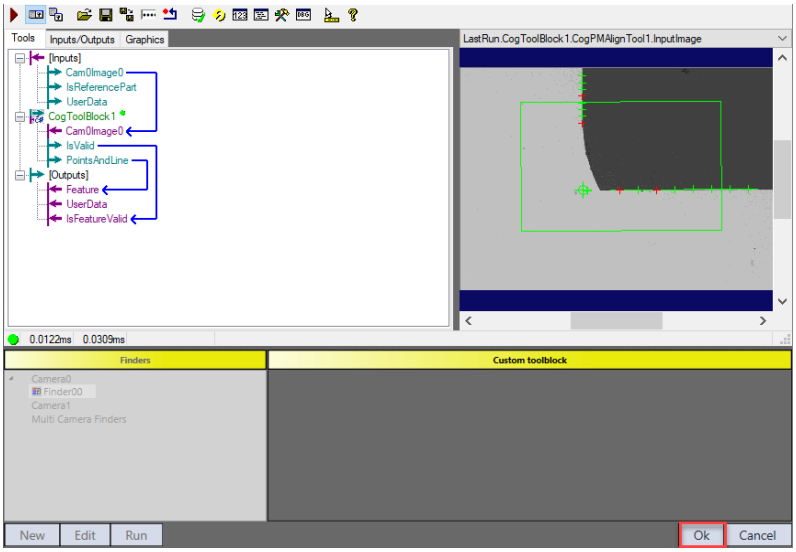

```
public override bool GroupRun(ref string message, ref CogToolResultConstants result)
{
  // To let the execution stop in this script when a debugger is attached, uncomment the following lines.
  // #if DEBUG
  // if (System.Diagnostics.Debugger.IsAttached) System.Diagnostics.Debugger.Break();
  // #endif

  Outputs.IsValid = true;
  if(Outputs.PointsAndLine==null)
    Outputs.PointsAndLine = new CogDictionary();
  // Run each tool using the RunTool function
  try
  {
    foreach(ICogTool tool in Tools)
      RunTool(tool, ref message, ref result);
    if(Tools.CogFindCornerTool1.Result.CornerFound)
    {
        Outputs.PointsAndLine.Clear();
        Outputs.PointsAndLine.Add("CornerX", Tools.CogFindCornerTool1.Result.CornerX);
        Outputs.PointsAndLine.Add("CornerY", Tools.CogFindCornerTool1.Result.CornerY);
        Outputs.PointsAndLine.Add("LeftLineSeg",
                        Tools.CogFindCornerTool1.Result.LineResultsA.GetLineSegment());
    }
    else
    {
      Outputs.IsValid = false;
    }
  }
  catch(Exception ex)
  {
    Outputs.IsValid = false;
  }

  return false;
}
```

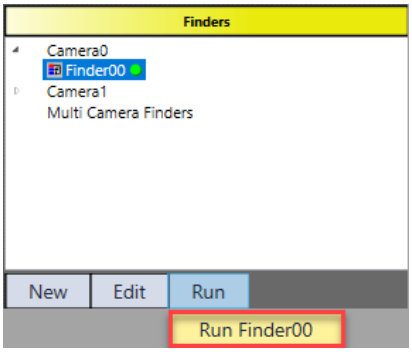
5. Test inside inner tool block to see if "IsValid" output reflects the ok/ng result correctly.

Note: When there are more than one CogFixtureTools used in a Generic Features finder, these CogFixtureTools' output spaces should be renamed differently to avoid space confliction. See more information about it in **Editing** in Custom Toolblock Point Finder on page 121.

Run

After setting up the tool block, click "Ok" button and run the finder to see result.



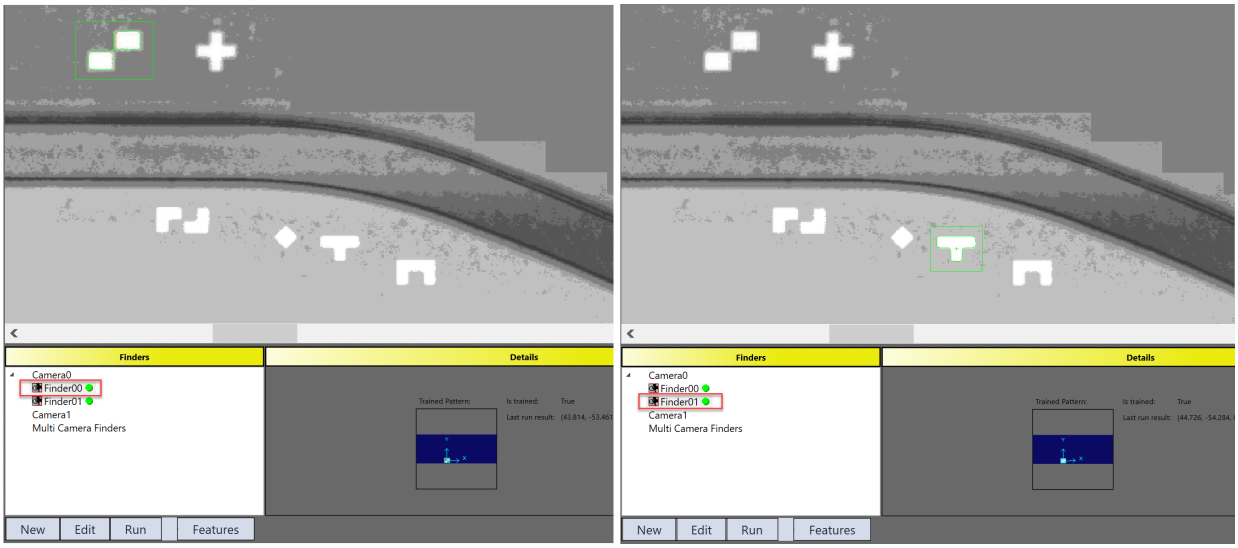


Multiple Features

Features finder allows user to find multiple features under one image, also allows to find one feature using multiple finders.

Multiple Features Under One Image

If there are multiple features to be found in one image, user can add multiple finders under that image, each finder finds one feature.



By default, feature's name is the same with its finder's name, such as "Finder00", "Finder01". Since their names are unique under a features finder, these features are considered as different features.

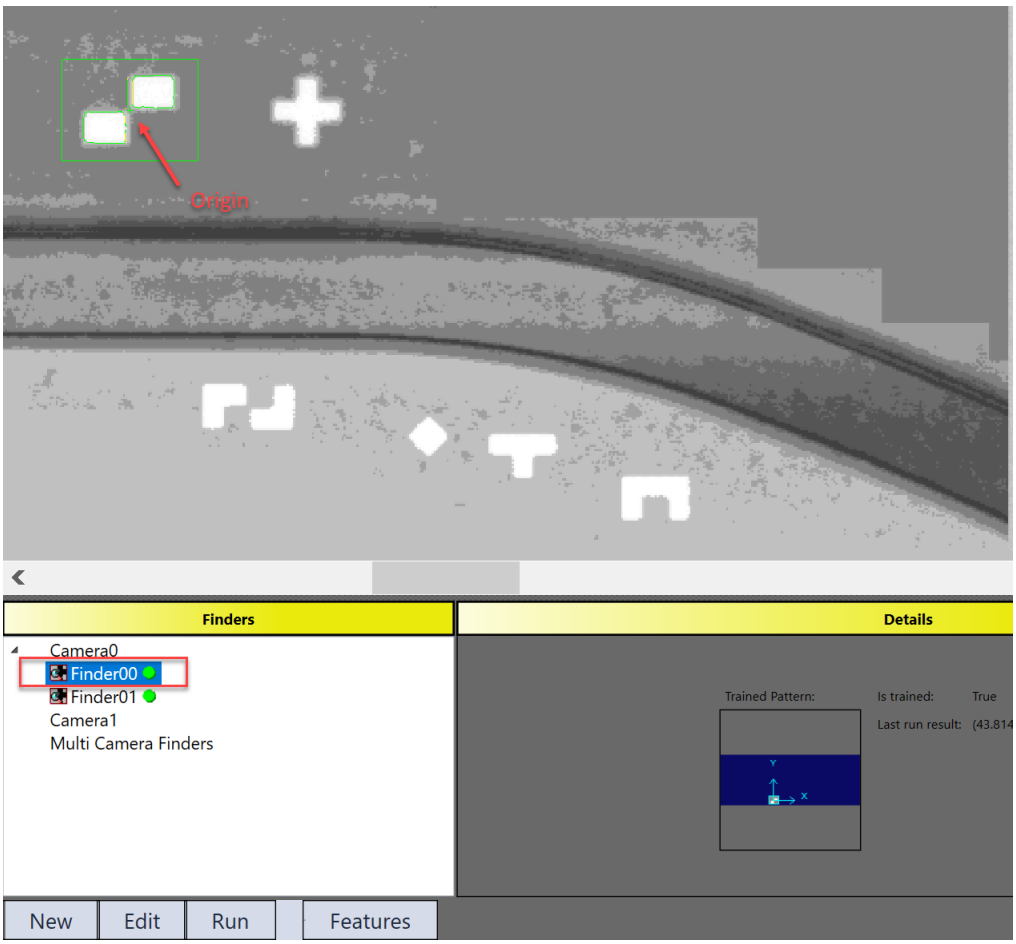
All the features found in a features finder, regardless of their sources of camera, will be used for pose computing.

One Feature with Multiple Finder

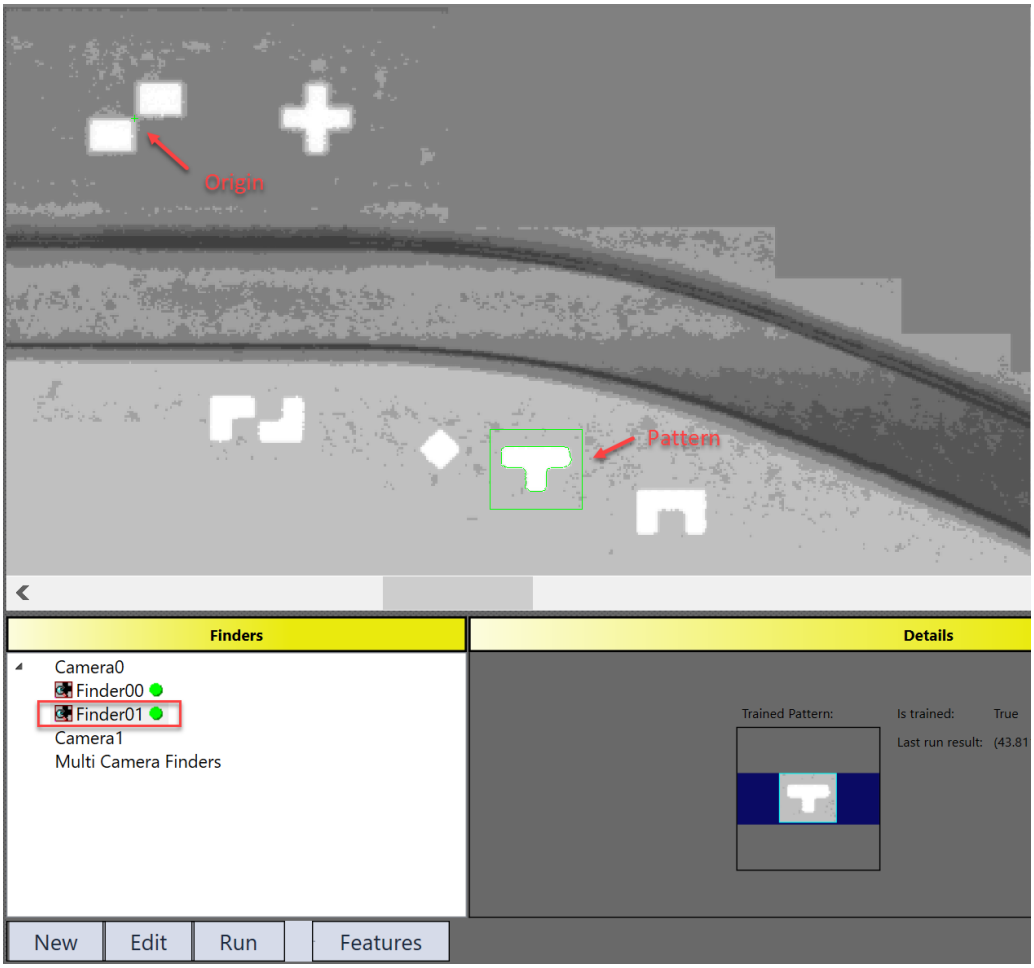
In applications where the main fiducial mark is contaminated and could not be found by finder, customer usually wants to find another fiducial mark to locate the part. However, to make no difference to pose computation, the second finder should report the same location point of the part as of the first finder. That is to say, to find the same feature point using different finders.

Here is an example of using two finders to find one feature:

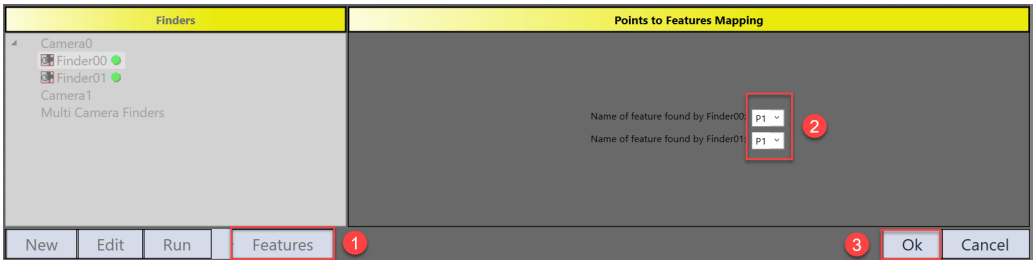
1. Add the first finder to find the first fiducial mark, put its output feature location at some obvious place on part such as the intersection of two edge lines on mark.



2. Add the second finder to find the second fiducial mark. Manually move its output feature location to the same place as configured in the first finder.



3. Click **Features** button, input the same name for **Finder00** and **Finder01**'s output features, and click **OK**.



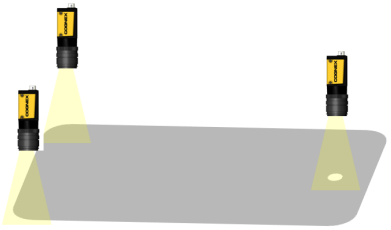
In AlignPlus, features with the same name are considered as the same feature, only the feature with highest score will be used for pose computing. In this way, whether only one feature is found, or both are found will not affect pose computation result. It's the same case if there are multiple finders.

⚠ **CAUTION**: Moving output origin far away from its found pattern/features may bring down the accuracy of feature finding. User need to consider whether this impact to alignment/assembly result is acceptable or not.

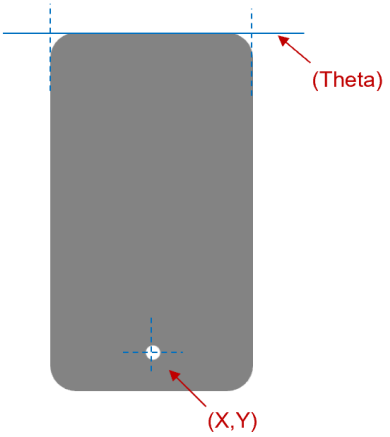
Custom Multi Camera ToolBlock Finder

Custom Mulit Camera Toolblock finder can be used when the result feature should be computed based on images from multiple cameras. This topic introduces how to use Custom Multi Camera Toolblock finer using an example of computing a point feature based on three images.

In this example, three cameras are looking at one part: two are focusing at corners on one side, and one focusing on a circle on the other side.



Based on customer's requirement, the feature result is a single point feature whose x and y are same with the circle center's, rotation is the angle of fitted line of the two corner points.

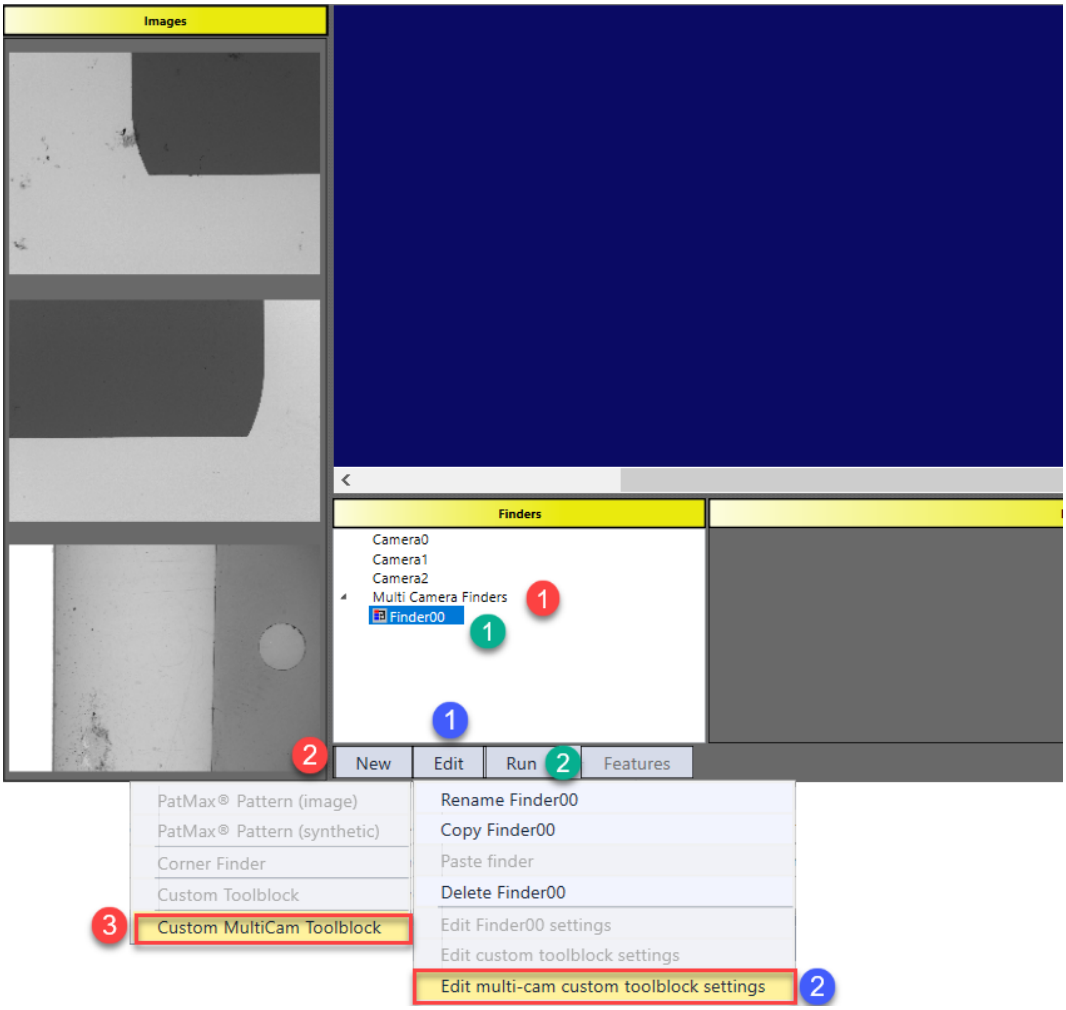


Since the final feature is a point feature, the feature type of finder component should be set as "Point" in the Configuration Wizard.

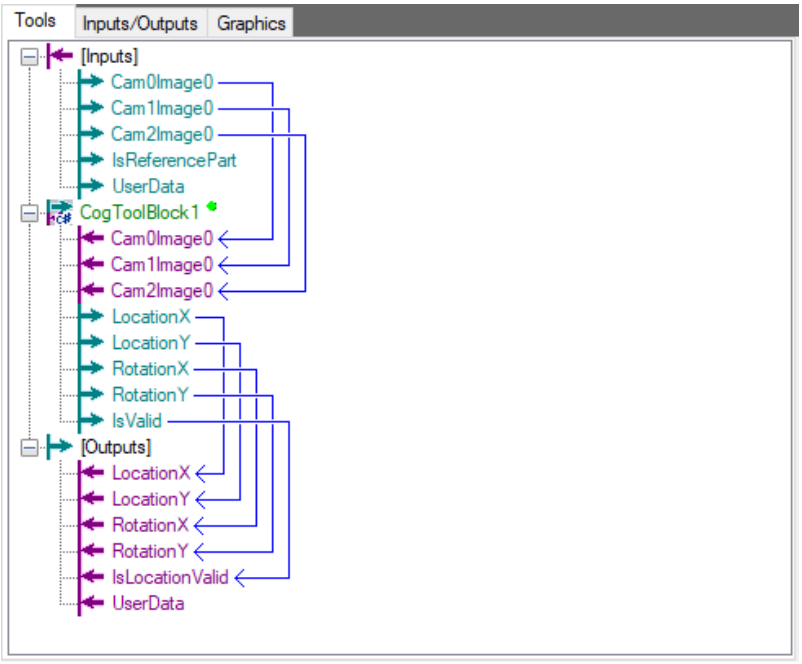
In the generated program, after calibration and acquiring three run time images of the part in run mode, users can start configuring Custom Multi Camera ToolBlock Finder:

1. In the "Finders" panel, one can find "Multi Camera Finders" category where custom multi camera toolblock should be added.
    - New: Select "Multi Camera Finders" and then click "New" button, in the opened drop-down list, choose "Custom MulitCam Toolblock"
    - Run: Select the added tool block, and then click "Run" button to run it once to get the input images

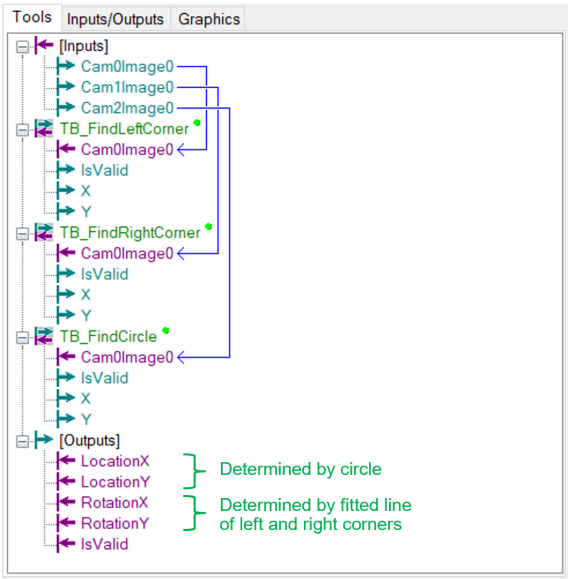
- Edit: Click "Edit" button and select "Edit multi-cam custom toolblock settings" to enter edit mode



2. In the setting page of the Multi Camera Custom Toolblock, add an inner ToolBlock, create its inputs and outputs, and link them as shown below.



3. Within the inner toolblock, add three sub toolblocks to find two corner points and one circle center respectively. For more information about how to find corners or circle in toolblock, please refer to Custom Toolblock Point Finder on page 121.

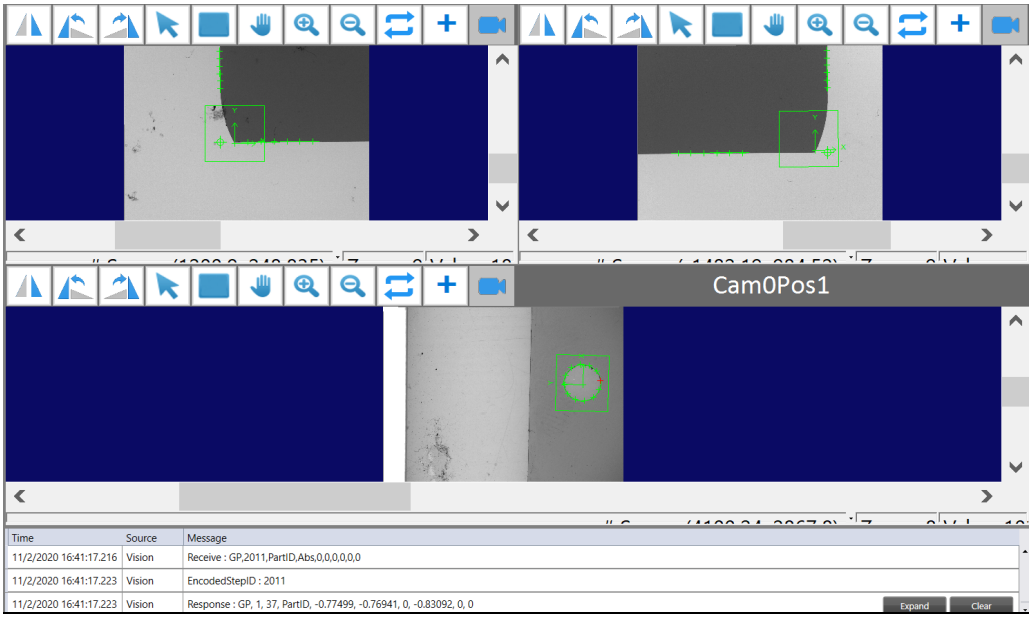


4. In added inner toolblock's scripts (C# simple script), input the following code to compute the result feature's x, y, and theta.

```
public override bool GroupRun(ref string message, ref CogToolResultConstants result)
{
  // To let the execution stop in this script when a debugger is attached, uncomment the following lines.
   #if DEBUG
   if (System.Diagnostics.Debugger.IsAttached) System.Diagnostics.Debugger.Break();
   #endif

  Outputs.IsValid = true;

  try
  {
    // Run each tool using the RunTool function
    foreach(ICogTool tool in Tools)
      RunTool(tool, ref message, ref result);

    double x1, y1, x2, y2, x3, y3;
    if(Convert.ToBoolean(this.Tools.TB_FindLeftCorner.Outputs["IsValid"].Value)
      && Convert.ToBoolean(this.Tools.TB_FindRightCorner.Outputs["IsValid"].Value)
      && Convert.ToBoolean(this.Tools.TB_FindCircle.Outputs["IsValid"].Value))
    {
      x1 = Convert.ToDouble(this.Tools.TB_FindLeftCorner.Outputs["X"].Value);
      y1 = Convert.ToDouble(this.Tools.TB_FindLeftCorner.Outputs["Y"].Value);
      x2 = Convert.ToDouble(this.Tools.TB_FindRightCorner.Outputs["X"].Value);
      y2 = Convert.ToDouble(this.Tools.TB_FindRightCorner.Outputs["Y"].Value);
      x3 = Convert.ToDouble(this.Tools.TB_FindCircle.Outputs["X"].Value);
      y3 = Convert.ToDouble(this.Tools.TB_FindCircle.Outputs["Y"].Value);
      double angle = Math.Atan2((y2 - y1),(x2 - x1));
      Outputs.LocationX = x3;
      Outputs.LocationY = y3;
      Outputs.RotationX = angle;
      Outputs.RotationY = angle;
    }
    else
      Outputs.IsValid = false;
  }
  catch(SystemException ex)
  {
    Outputs.IsValid = false;
  }

  return false;
}
```

Note: Similar with custom toolblock point finder, the units of RotationX and RotationY here are radian, AlignPlus program will later transfer them into degrees and save them in a CogAlpsPointFeature object.

Here is the result of feature finding and align result using the result from the Custom Multi Camera Toolblock.

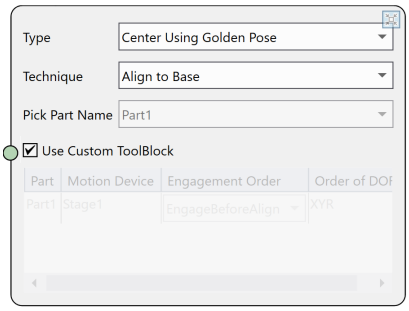


Custom Pose Computation

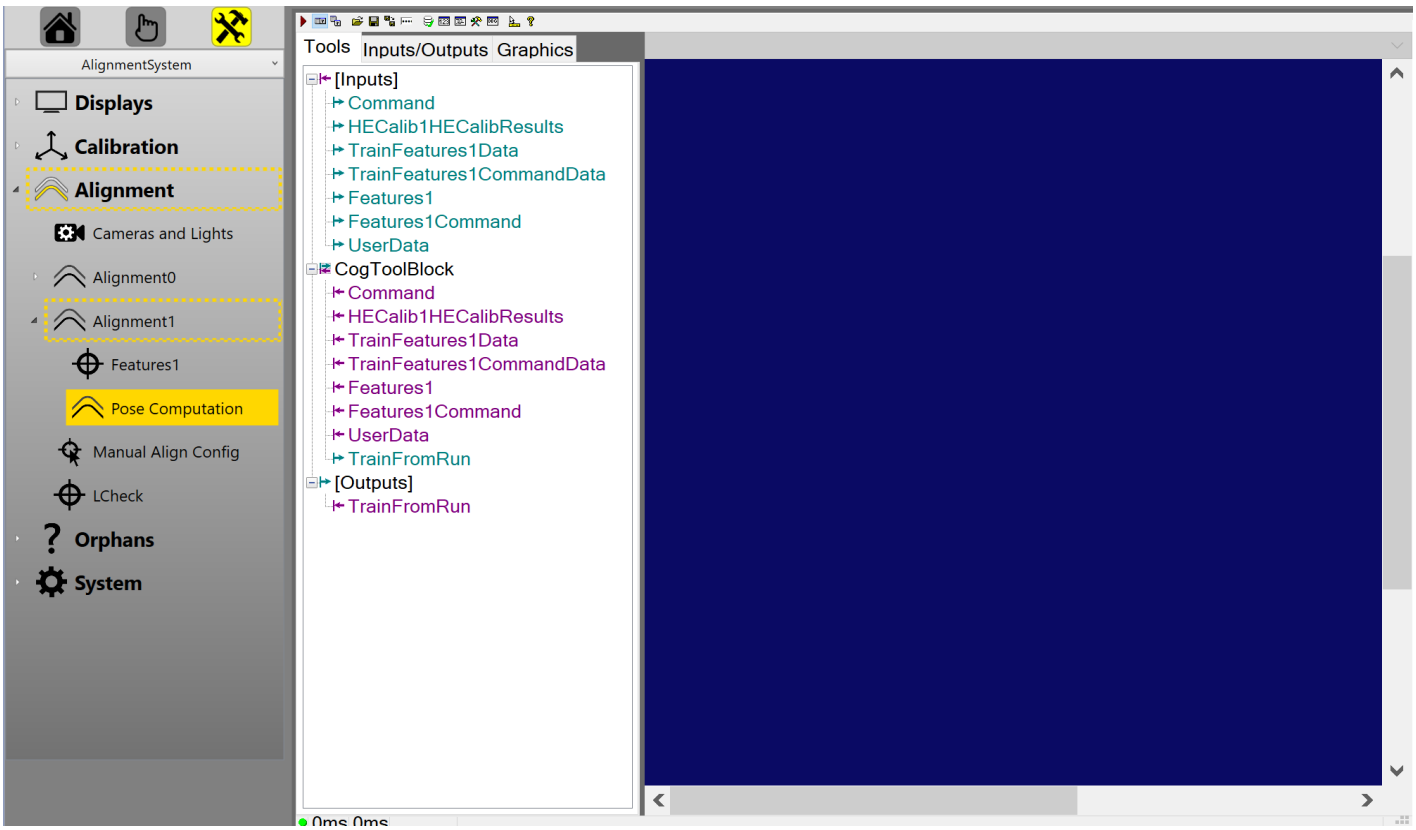
Each alignment component in wizard configuration has one pose computation page on HMI for user to customize pose computation. This page is configurable only when "Use Custom ToolBlock" is checked in alignment component during wizard configuration. Custom pose computation allows user to custom pose compute method based on specific requirements which cannot be achieved by other options.

Alignment Custom Pose Computation

Take AlignToBase application for example, check on "Use Custom ToolBlock" in alignment component to enable custom pose computation.



The corresponding pose computation page is as below.



The inputs include command of the alignment task, hand-eye calibration results, features from run time and train time and their commands in which run time and train time stage poses are included, and user data if there is any user defined data.

Input	Type	Description
Command	CommandArgs	Current command for pose computer task
HECalibResults	CogHandEyeCalibrationResults	Contains a list of CogHandEyeCalibrationResult objects, one for each camera.
TrainFeaturesData	List<CogAlpsPointFeature> / List<CogAlpsLineFeature>/ List<CogGenericFeature>	The list of train time features from connected feature finder

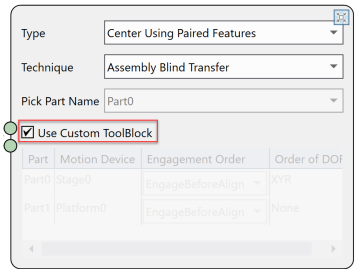
Input	Type	Description
TrainFeaturesCommandData		The last train time command used for connected feature finder's image acquisition and feature extraction
Features	List<CogAlpsPointFeature> / List<CogAlpsLineFeature>/ List<CogGenericFeature>	The list of run time features from connected feature finder
FeaturesCommand	CommandArgs	The last run time command to trigger connected feature finder to run image acquisition and feature extraction
UserData	Dictionary	User defined data

The output is a linear transform from run time features to train time features in current selected space. And the implementation of the pose computing should be done within **CogToolBlock** using tools or scripting.

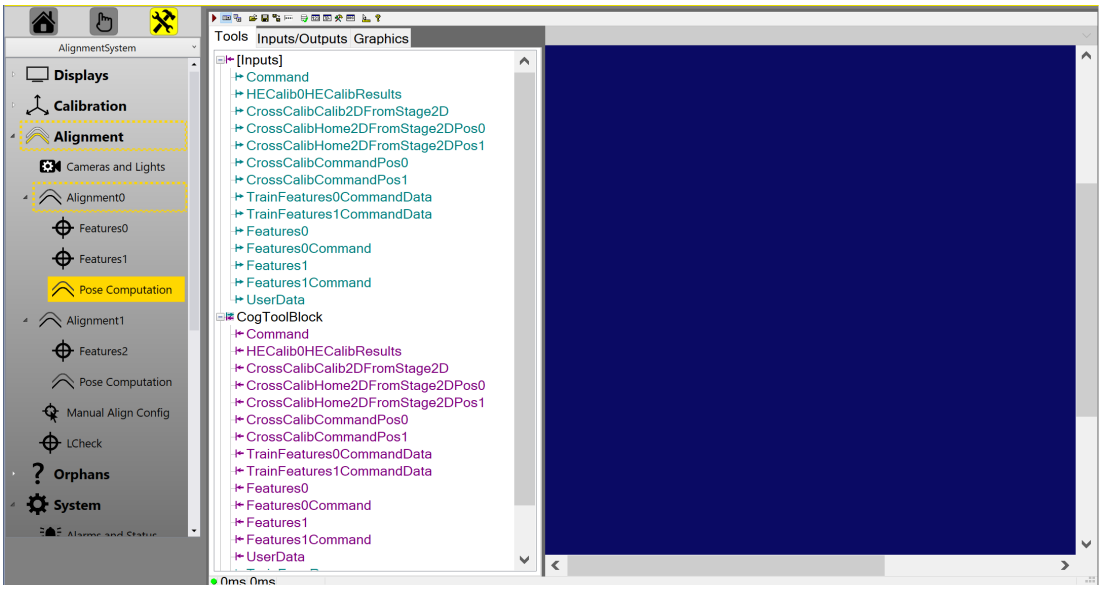
Output	Type	Description
TrainFromRun	CogTransform2DLinear	The transform from run time features to train time features

Assembly Custom Pose Computation

Here is one example of assembly application using custom pose computation and its corresponding pose computation page.

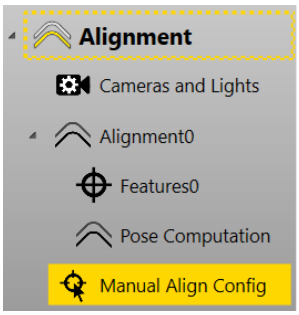


Since assembly uses pairs of features for pose computing, it has more inputs and outputs.



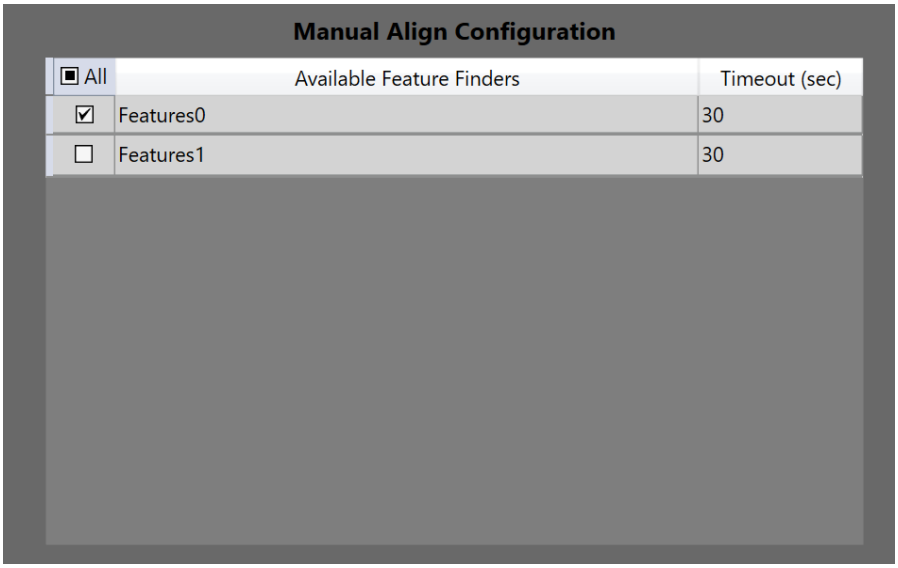
Manual Align Config

Manual Align Config allows operators to manually select feature locations on images in a pop-up window when run time feature finding fails. It is located under **Alignment** category in setup mode. This function only supports point features and only when there are more than two features.



Settings

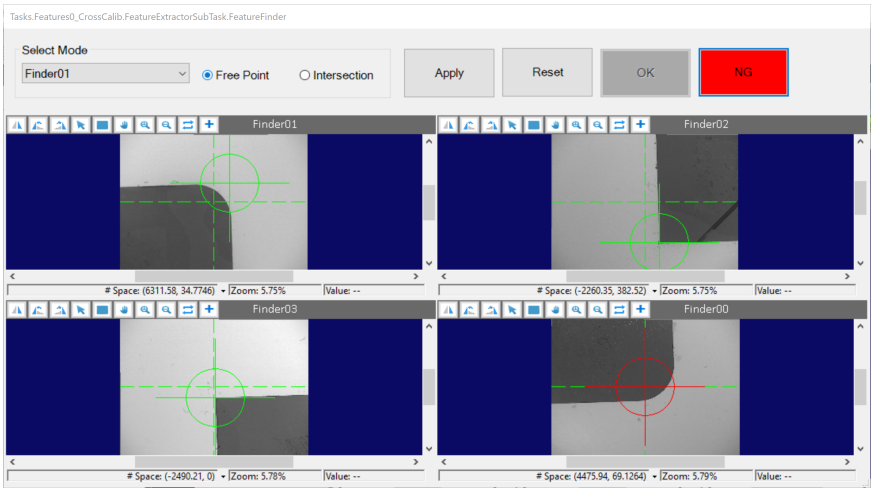
The **Manual Align Configuration** window shows all the point feature finders in the program. One can enable or disable manual align function for specific finder, and set timeout value (in seconds) for user's operation. When the **Run Time Edit Window on page 149** is idled more than the timeout value, the window will be closed automatically with result of NG. Zero means no timeout for the operation, the window will stay active until user take further action.



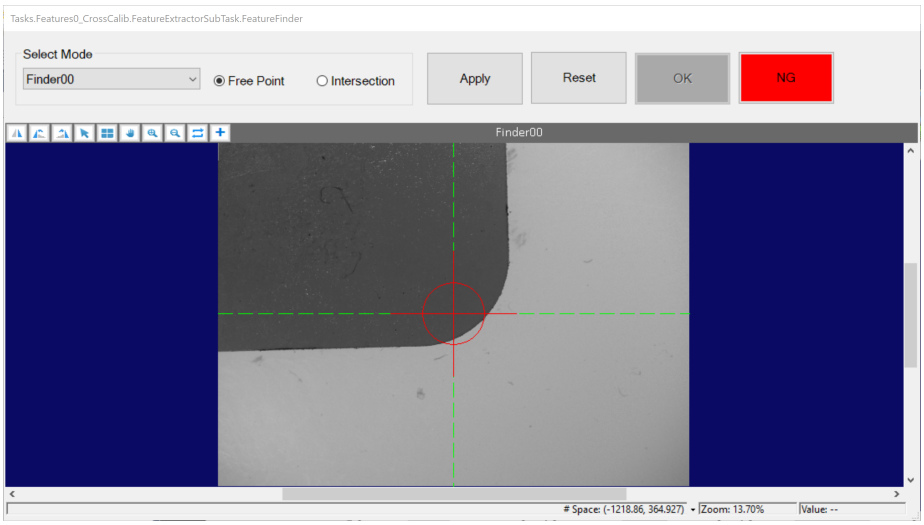
Run Time Edit Window

If the point feature finder is failed at run time and the its manual align function is enabled, the **Feature Point Edit Window** will pop up for user to select feature location on image manually.

The picture below shown the feature finder has four point features. If the feature is found, a point edit tool in green will be shown. Otherwise the point edit tool will be shown in red.



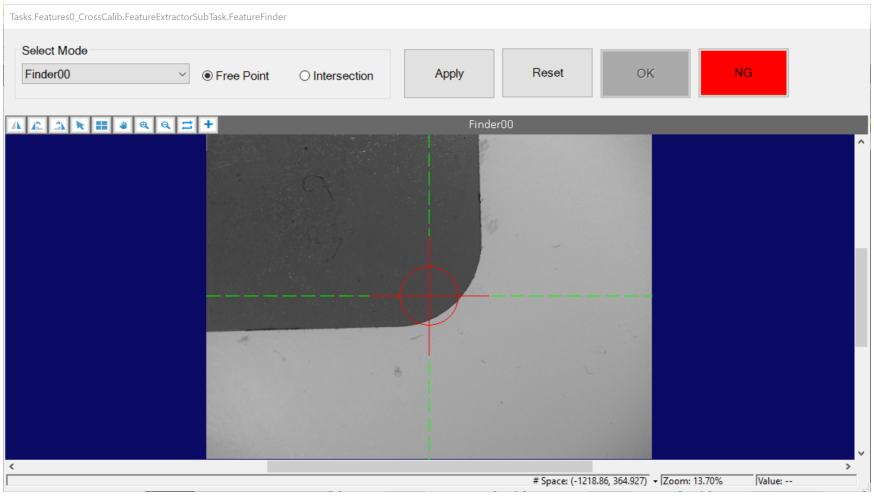
Double click one of the display to switch to single display mode:



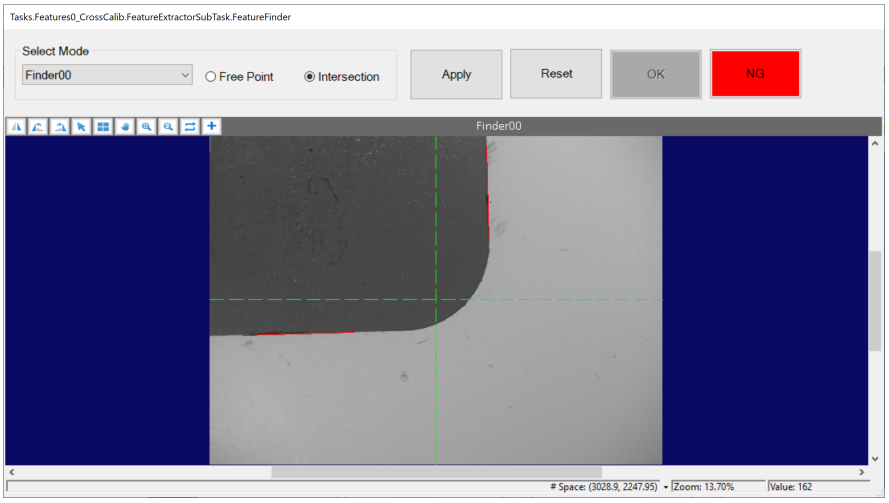
Function	Description
Select Mode	Select feature which needs manual align. Change of selected feature updates image display in single display mode.
Free Point	Manually align feature location using point tool for selected feature
Intersection	Manually align two line segments to part's edges and use their intersection as feature location for selected feature
Apply	Apply manual align results to run time feature locations. Once the button is clicked, points are not editable.
Reset	Reset the operations and switch back to edit mode.
OK	Set all features as valid and found with current locations, exit the window to finish manual align. This button is available only after changes are applied.
NG	Keep failed features as failed and exit manual align. Select this option when the part is defective.

Here is an example of using intersection as result point:

1. Double click the failed feature.

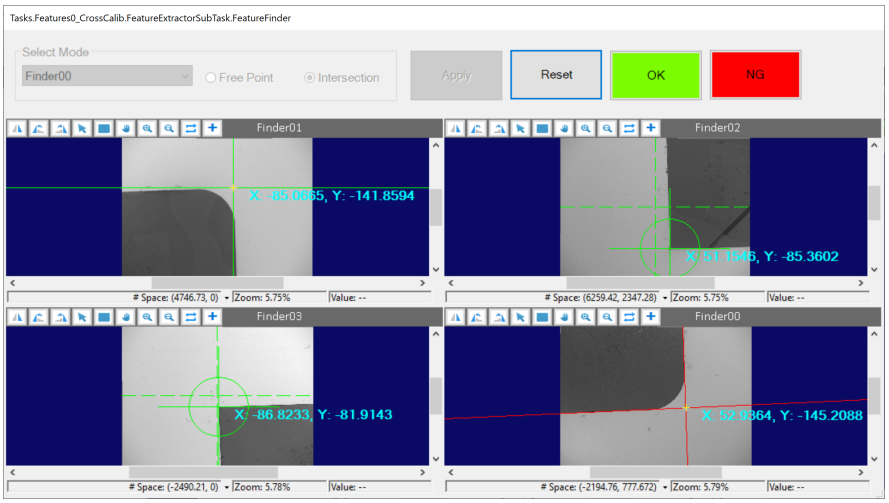


2. Choose **Intersection** and align the two line segments to two edges of the part.



Note: The intersection tool assumes manual align of line segments are accurate and uses no vision tool to fine tune them.

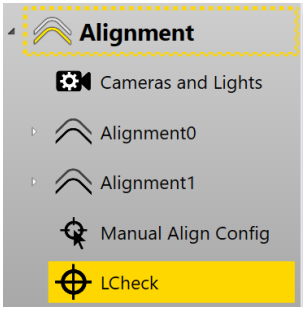
3. Click **Apply** button to calculate the intersection of two line segments and set it as the feature's location. Double click the single image display to switch back to multiple image display to check if there is any remain missing features.



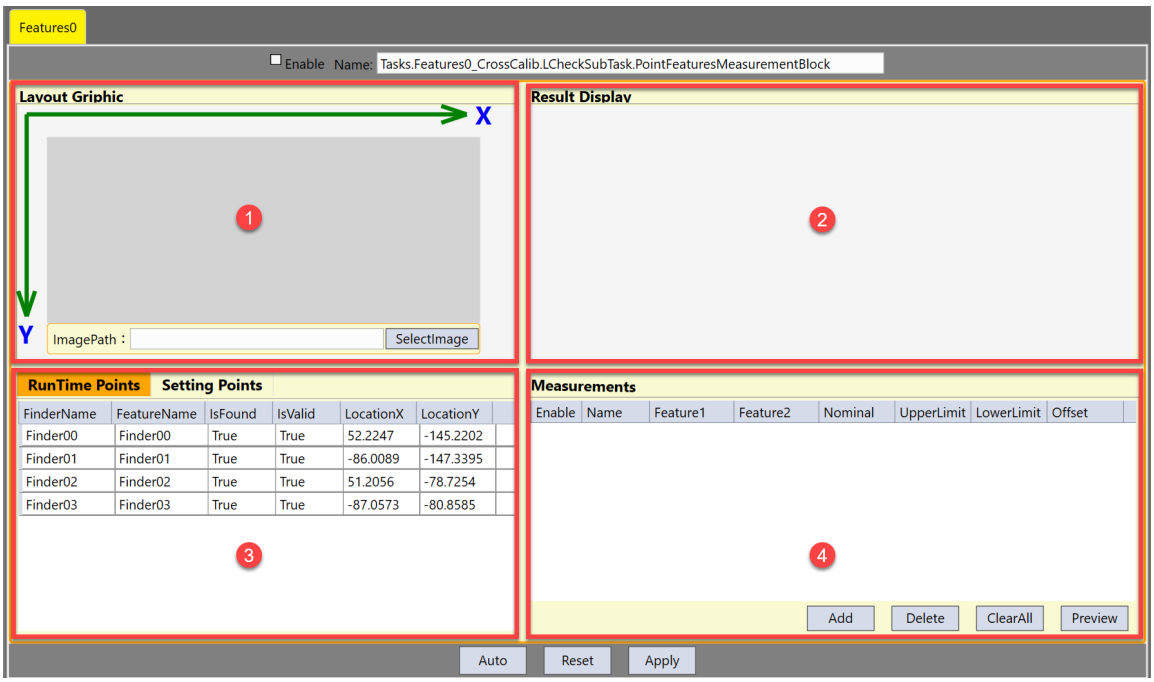
4. Click **OK** to confirm the manual feature finding.

L-Check

L-Check subtask checks if the distance between pairs of features are within specification. It is located under **Alignment** category in setup mode. L-Check function only supports distance check between point features, line features and generic features are not supported.

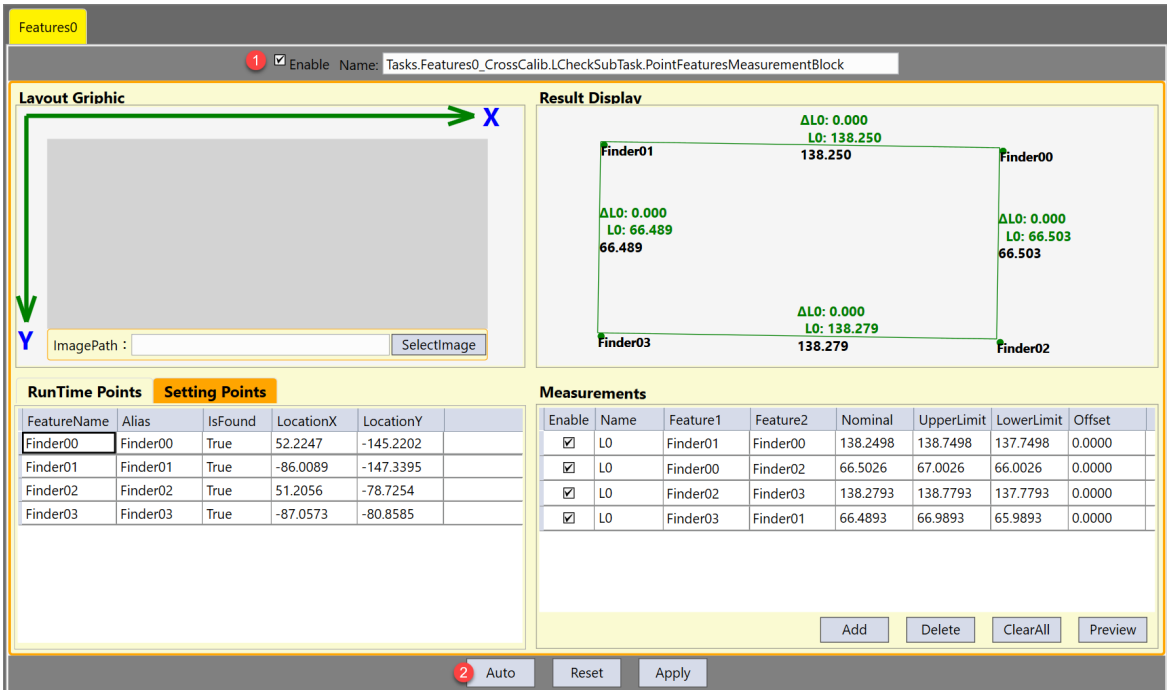


Every features finders have an independent L-Check functions, they're arranged in different tab at the top of L-Check setting page. Select one features finder first, then check **Enable** on/off to enable/disable its L-Check function. The **Name** besides **Enable** check box indicates specific sub task in feature finding task which will be enabled or disabled.



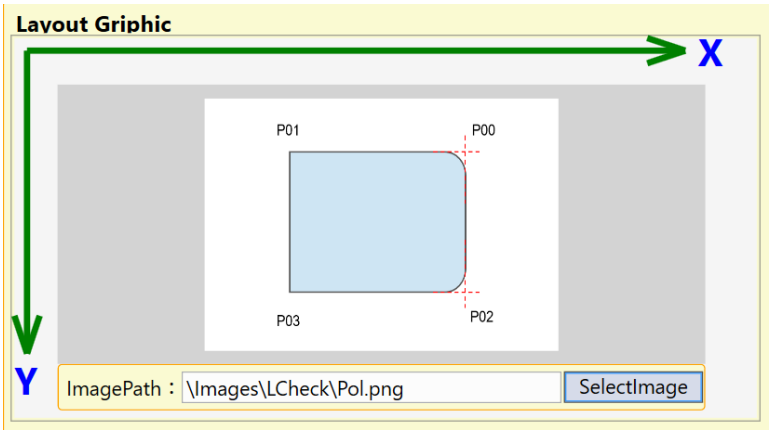
L-Check page mainly has four areas: **Layout Graphic**, **Feature Points**, **Result Display** and **Measurements**. Layout graphic shows all run time features in current features finder and setting points which will be used for length check. Result Display shows the graphics of L-Check in shared coordinate space. Measurements is the panel for user to edit length check points.

To start L-Check setting, click "Auto" button at the bottom of this page to let L-Check automatically generate setting points and measurement points, and update the graphics.



Layout Graphic

Layout graphic window allows user to load a layout image of the part in format of JPG, JPEG, BMP, TIF or PNG with size under 10 megabyte. This image helps user understand the layout of each feature point on part for convenience of measurement points setting. This image should be drawn by user based on specific part type but not mandatory.



Feature Points

Feature points displays and manages two sets of points: **RunTime Points** or **Setting Points**.

RunTime Points are current run time features of selected features finder. To make them available, the features finder needs to run at least once.

RunTime Points | Setting Points

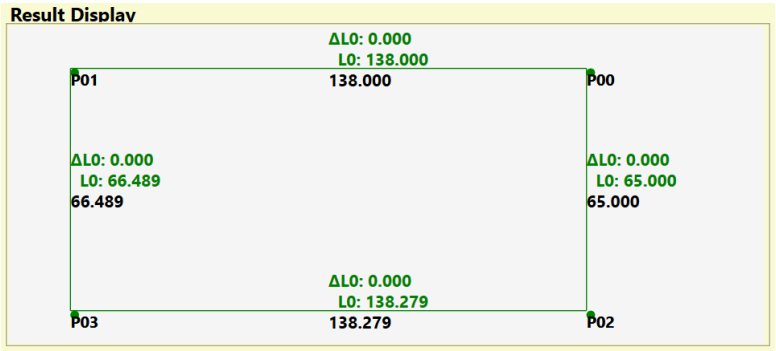
FinderName	FeatureName	IsFound	IsValid	LocationX	LocationY
Finder00	Finder00	True	True	52.2247	-145.2202
Finder01	Finder01	True	True	-86.0089	-147.3395
Finder02	Finder02	True	True	51.2056	-78.7254
Finder03	Finder03	True	True	-87.0573	-80.8585

Setting points are the candidate points for measurement. L-Check auto function will make a copy of all run time points and set them as setting points. However, one can change the aliases of the setting points to make them more meaningful.

RunTime Points | Setting Points

FeatureName	Alias	IsFound	LocationX	LocationY
Finder00	P00	True	52.2322	-145.2165
Finder01	P01	True	-86.0127	-147.3376
Finder02	P02	True	51.2044	-78.7298
Finder03	P03	True	-87.0655	-80.8595

To preview the changes, click "Preview" button at the bottom right corner under Measurements panel to update the graphics in result display.

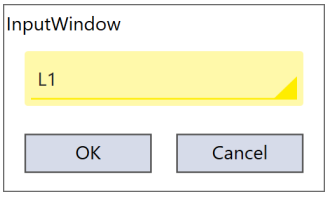


Measurements

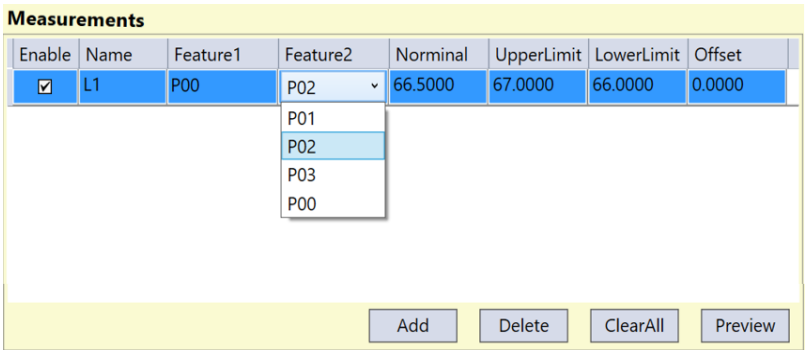
Measurements allows user to add, enable/disable, or delete length check between point pairs.

Add

Click **Add** button to add one length check. Input its name in the pop-up dialog and click "OK".



Select feature points, input length nominal, upper and lower limits, and offset.



Parameter	Description
Enable	Enable or disable current length check.
Name	Name of current length check.
Feature1/Feature2	The starting/ending feature points of current length check. Select them from drop list. The order of two points doesn't matter as long as they're not identical.
Nominal	Nominal value of current length
UpperLimit	Upper limit of current length
LowerLimit	Lower limit of current length
Offset	Compensation to measured value

The criteria to judge a length check is:

- Pass: if *LowerLimit* ≤ *(measured distance + offset)* ≤ *UpperLimit*
- Fail: if *(measured distance + offset)* < *LowerLimit* or *(measured distance + offset)* > *Upper Limit*

Enable

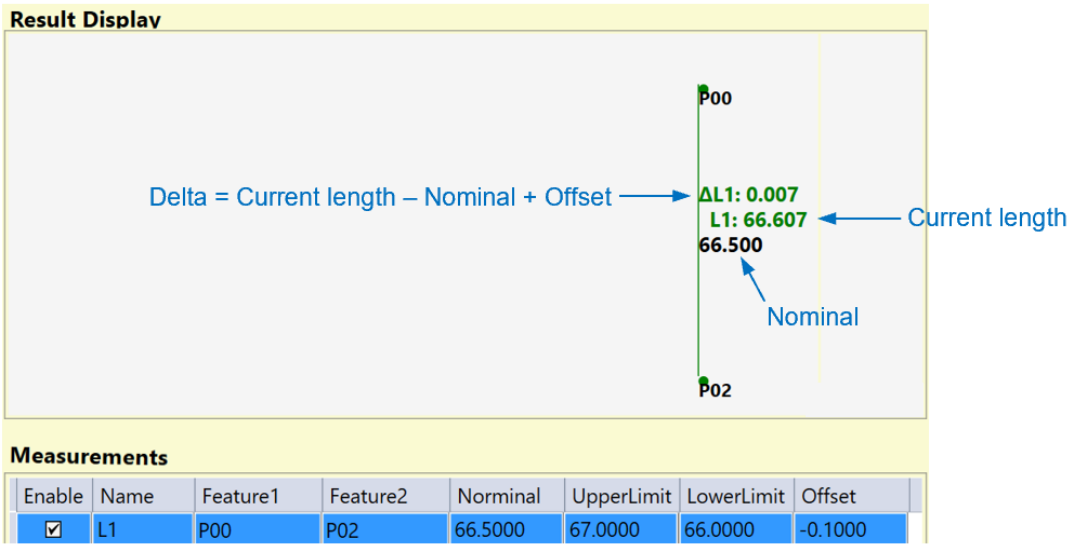
Disable one length check by checking off its **Enable** option.

Delete

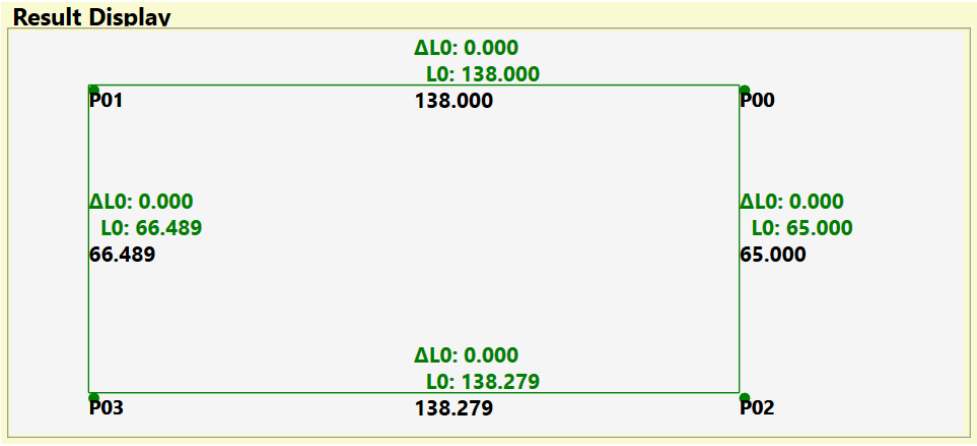
Select one length check, and click **Delete** button.

Result Display

Result display shows all length check's feature points, nominal values, current values, and differences. Difference indicates how the compensated length (measured result + offset) is different to nominal. If compensated length is within range, then the graphics will be green, otherwise will be red.



When all measurement points are set, click "Preview" button to update the result graphics.



Reset

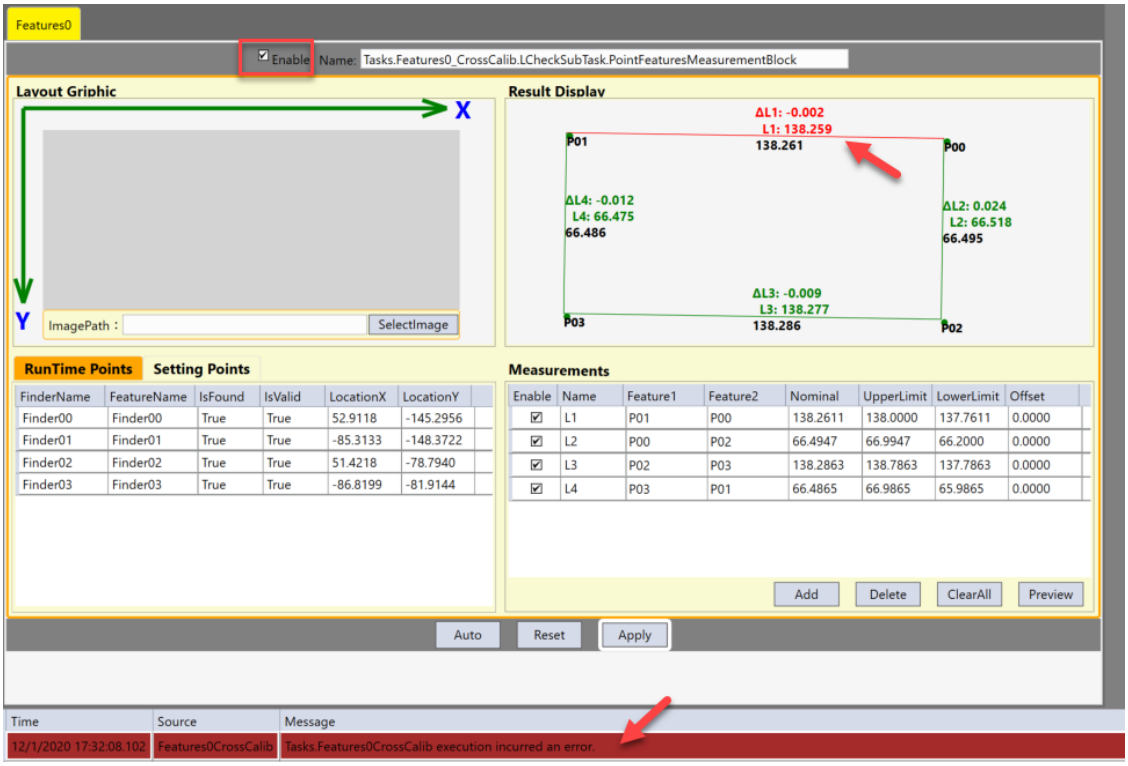
To clear setting points and measurement points, click "Reset" button.

Apply and Save Recipe

When there are any unsaved changes, the "Apply" button is highlighted as "Apply". Click it and select "Yes" to save the changes. After this, the "Apply" button will turn back to "Apply". Go to **System/Product Recipe** page to save current **Alignment** sub recipe. Otherwise all the changes will be lost after the program is closed. For more information about recipe, please refer to Product Recipe on page 160.

Run

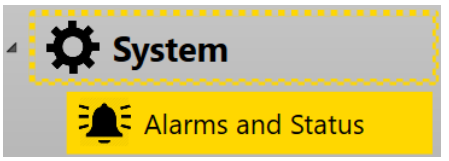
If a feature finding task's L-Check function is enabled and configured, during run time, after the features are extracted, L-Check will check the configured lengths of the part. If a length is out of specification, the Result Display will show it in red, and the feature finding task will throw an exception; otherwise, If all length checks are passed, then their graphics will be displayed in green, and no exception will be thrown.


Features0
Enable
Name: Tasks.Features0_CrossCalib.LCheckSubTask.PointFeaturesMeasurementBlock
Layout Graphic
X
Y
ImagePath :
SelectImage
Result Display
ΔL1: -0.002
L1: 138.259
138.261
P01
P00
ΔL4: -0.012
L4: 66.475
66.486
ΔL2: 0.024
L2: 66.518
66.495
ΔL3: -0.009
L3: 138.277
138.286
P03
P02
RunTime Points
Setting Points
FinderName | FeatureName | IsFound | IsValid | LocationX | LocationY
Finder00 | Finder00 | True | True | 52.9118 | -145.2956
Finder01 | Finder01 | True | True | -85.3133 | -148.3722
Finder02 | Finder02 | True | True | 51.4218 | -78.7940
Finder03 | Finder03 | True | True | -86.8199 | -81.9144
Measurements
Enable | Name | Feature1 | Feature2 | Nominal | UpperLimit | LowerLimit | Offset
L1 | P01 | P00 | 138.2611 | 138.0000 | 137.7611 | 0.0000
L2 | P00 | P02 | 66.4947 | 66.9947 | 66.2000 | 0.0000
L3 | P02 | P03 | 138.2863 | 138.7863 | 137.7863 | 0.0000
L4 | P03 | P01 | 66.4865 | 66.9865 | 65.9865 | 0.0000
Add
Delete
ClearAll
Preview
Auto
Reset
Apply
Time
Source
Message
12/1/2020 17:32:08.102
Features0CrossCalib
Tasks.Features0CrossCalib execution incurred an error.

System

Alarms and Status

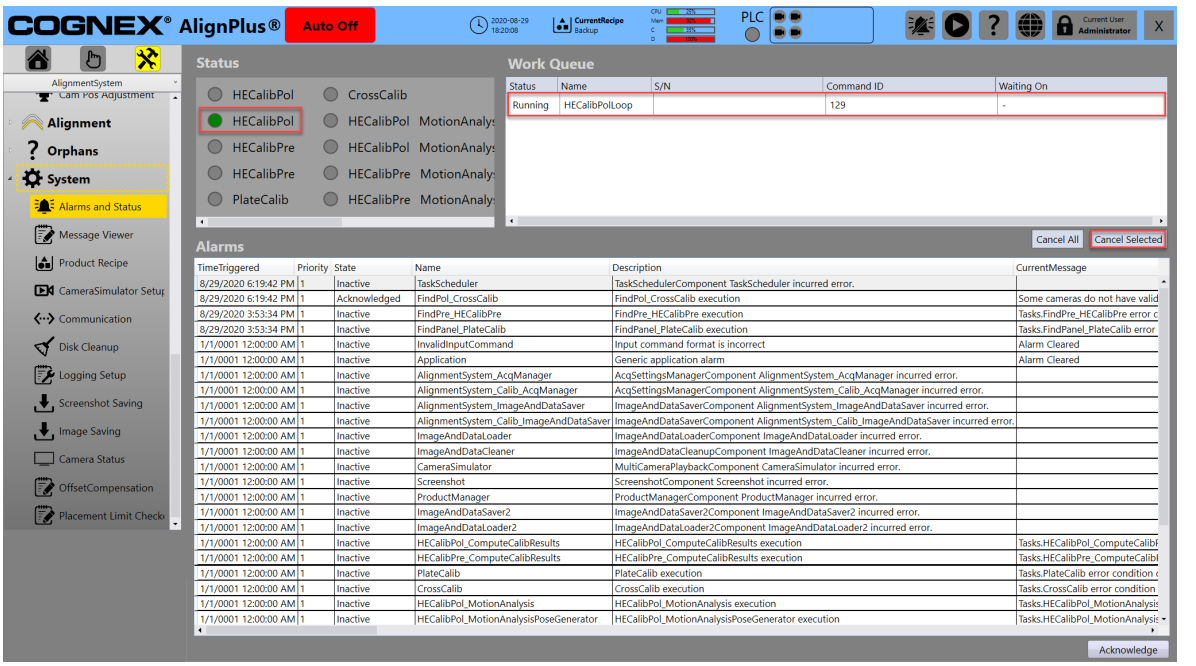
Alarms and status page shows tasks' running status and alarms. It can be found under **System** category in setup mode.



Status

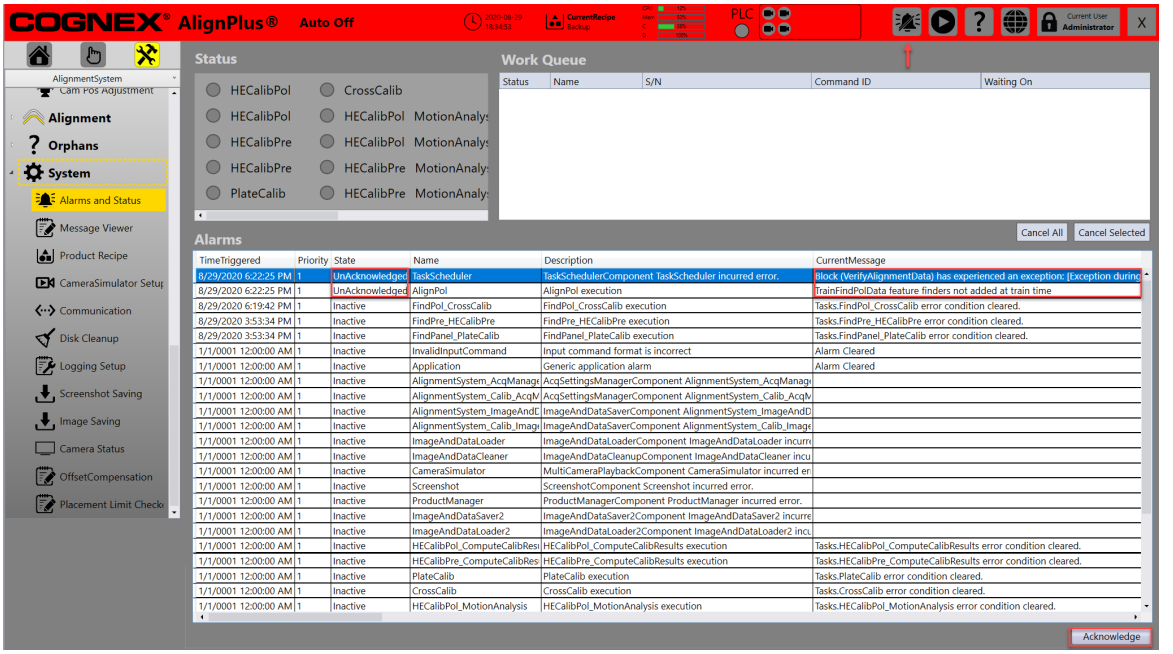
When a certain task is running, its status light will be in green, otherwise will be in gray.

If you want to cancel the current running task, just select it from **Work Queue** and then click **Cancel Selected** button at the bottom. Or click **Cancel All** to cancel all running tasks.



Alarms

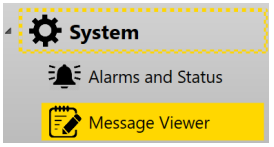
When there is any exception happening in the program, the title bar will flash between red and gray to remind user to check the alarm. Check the **Unacknowledged** alarms listed at the top of alarms table to see what has caused the exception and take actions accordingly. Once the exception is checked, click **Acknowledge** button at the button of alarms table to reset the title bar status. This flash could also be reset by clicking "  " icon on the title bar.



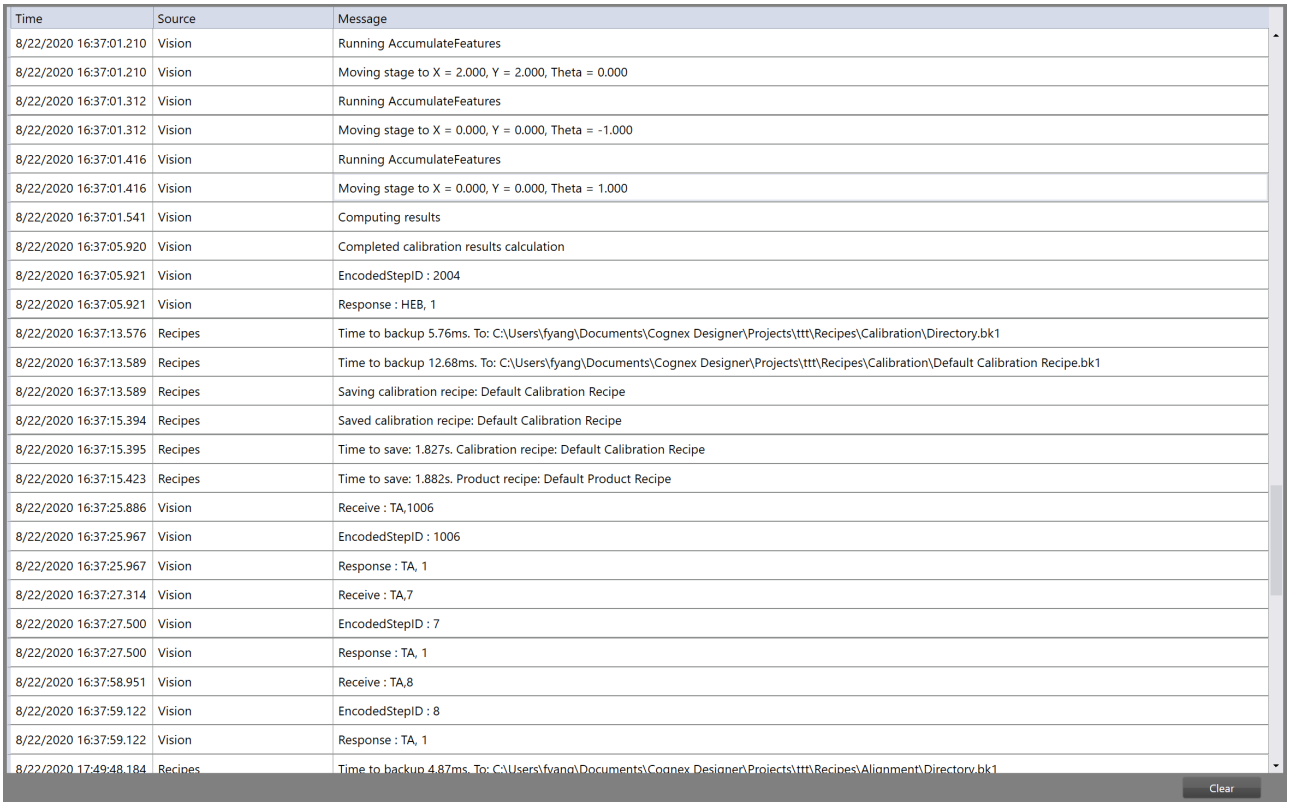
After the alarm is acknowledged, its status will turn to **Inactive** and stays in the alarm list as one record.

Message Viewer

Message viewer shows the system log in the main window. It can be found in **System** category in setup mode.



Message viewer helps user glance at all items and analyze the sequences in a larger window.



Time	Source	Message
8/22/2020 16:37:01.210	Vision	Running AccumulateFeatures
8/22/2020 16:37:01.210	Vision	Moving stage to X = 2.000, Y = 2.000, Theta = 0.000
8/22/2020 16:37:01.312	Vision	Running AccumulateFeatures
8/22/2020 16:37:01.312	Vision	Moving stage to X = 0.000, Y = 0.000, Theta = -1.000
8/22/2020 16:37:01.416	Vision	Running AccumulateFeatures
8/22/2020 16:37:01.416	Vision	Moving stage to X = 0.000, Y = 0.000, Theta = 1.000
8/22/2020 16:37:01.541	Vision	Computing results
8/22/2020 16:37:05.920	Vision	Completed calibration results calculation
8/22/2020 16:37:05.921	Vision	EncodedStepID : 2004
8/22/2020 16:37:05.921	Vision	Response : HEB, 1
8/22/2020 16:37:13.576	Recipes	Time to backup 5.76ms. To: C:\Users\fyang\Documents\Cognex Designer\Projects\ttt\Recipes\Calibration\Directory.bk1
8/22/2020 16:37:13.589	Recipes	Time to backup 12.68ms. To: C:\Users\fyang\Documents\Cognex Designer\Projects\ttt\Recipes\Calibration\Default Calibration Recipe.bk1
8/22/2020 16:37:13.589	Recipes	Saving calibration recipe: Default Calibration Recipe
8/22/2020 16:37:15.394	Recipes	Saved calibration recipe: Default Calibration Recipe
8/22/2020 16:37:15.395	Recipes	Time to save: 1.827s. Calibration recipe: Default Calibration Recipe
8/22/2020 16:37:15.423	Recipes	Time to save: 1.882s. Product recipe: Default Product Recipe
8/22/2020 16:37:25.886	Vision	Receive : TA,1006
8/22/2020 16:37:25.967	Vision	EncodedStepID : 1006
8/22/2020 16:37:25.967	Vision	Response : TA, 1
8/22/2020 16:37:27.314	Vision	Receive : TA,7
8/22/2020 16:37:27.500	Vision	EncodedStepID : 7
8/22/2020 16:37:27.500	Vision	Response : TA, 1
8/22/2020 16:37:58.951	Vision	Receive : TA,8
8/22/2020 16:37:59.122	Vision	EncodedStepID : 8
8/22/2020 16:37:59.122	Vision	Response : TA, 1
8/22/2020 17:49:48.184	Recipes	Time to backup 4.87ms. To: C:\Users\fyang\Documents\Cognex Designer\Projects\ttt\Recipes\Alignment\Directory.bk1

Clear

Click **Clear** button if you want to clear log in view table.

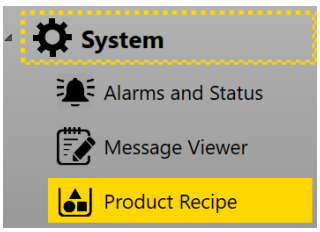
If there is any other extra message to be logged, you can use **$LogData** function in script inside the program to add it into the system log.

```
$LogData(string source, string message)
```

The system log will be saved into a CSV file under the directory specified in Logging Setup on page 172.

Product Recipe

Product recipe manages all recipes in the master and sub recipe structure. It can be found in **System** category in setup mode.



Master Recipe and Sub Recipe

Product recipe is the master recipe which maintains the name list of sub recipes. Product recipe does not save any other specific data. Under each product, there are three sub recipes: acquisition recipe, calibration recipe and alignment recipe. Here is the summary of the three sub recipes:

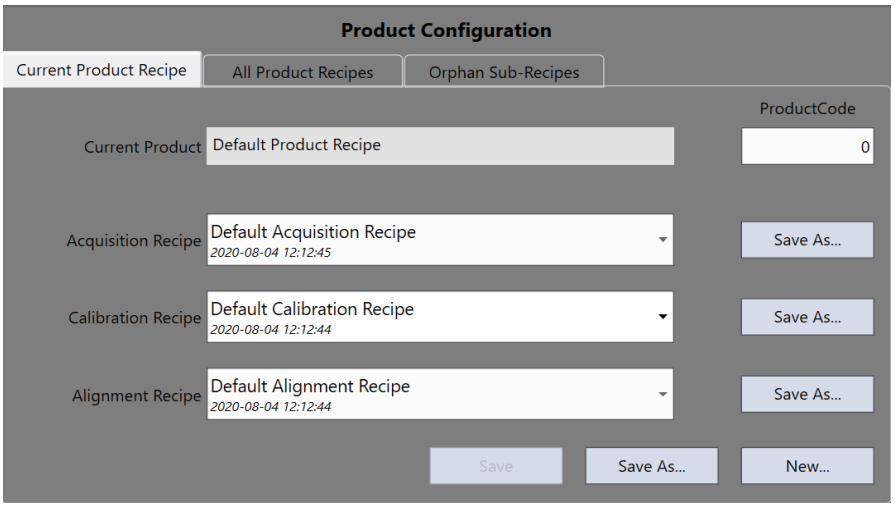
Sub Recipe	Data Saved	When to Save
Acquisition recipe	All camera and lights settings, and acquisition settings for each task	Every time camera or lights settings, or acquisition settings such as camera IP address, exposure time, light intensity have been changed, acquisition recipe needs to be saved again.
Calibration recipe	All calibration settings and calibration results.	Every time calibration settings(which may lead to some calibration results invalid) are changed, or any station is recalibrated, calibration recipe needs to be saved again.
Alignment recipe	All feature finders and their trained data, camera enable/disable status, offsets setting, LCheck setting, Placement Limit setting	Every time feature finder vision tools are reconfigured, feature finder is retrained, camera changes its disable/enable status, offsets/LCheck/Placement Limit setting changes, alignment recipe needs to be saved again.

The benefit of the master-sub recipe structure is that once the product changes, you only need to change the product recipe as all its sub recipes will be automatically changed altogether and you do not need to take time to distinguish which sub recipe belongs to which product. Also, product recipes are indexed with product codes; these codes could be called by external devices through communication which makes automatic recipe change possible, which saves model change time and avoids manual mistakes in the factory.

The product recipe setting has three pages: **Current Product Recipe**, **All Product Recipes** and **Orphan Sub-Recipes**.

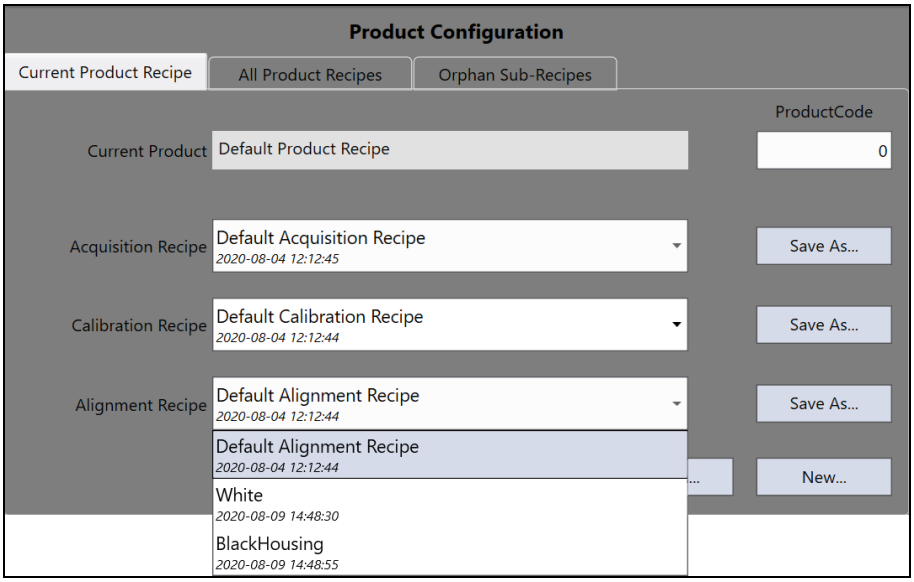
Current Product Recipe

This page is for the operations on the current product recipe.



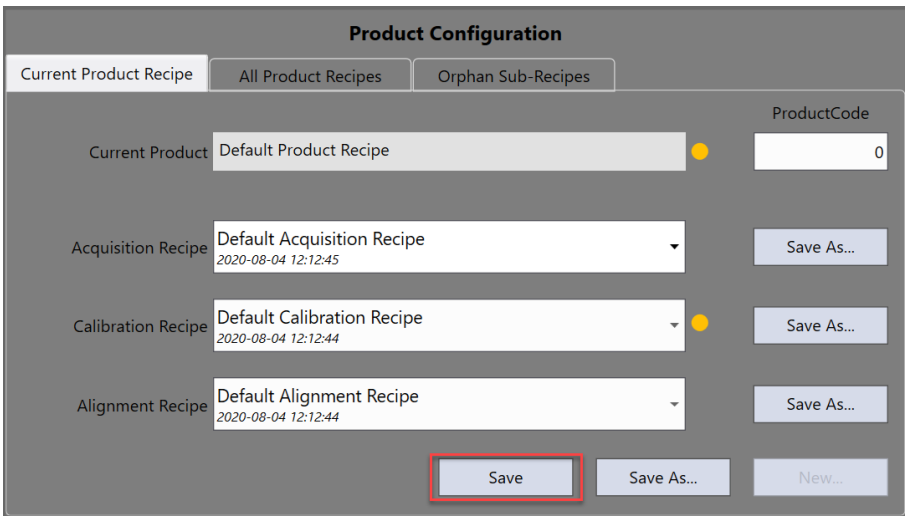
Item	Description
Current Product	The name of the currently loaded product(master recipe)
Product Code	The code number for the current product that can be used for automatic recipe change
Acquisition Recipe	The name of acquisition sub recipe of the current product
Calibration Recipe	The name of calibration sub recipe of the current product
Alignment Recipe	The name of alignment sub recipe of the current product

Each sub recipe category has a drop-down list that shows all the available sub recipes and allows the current product to have different sub recipes. All sub recipes have time stamp automatically attached when they were created so that it is more convenient for you to identify them.

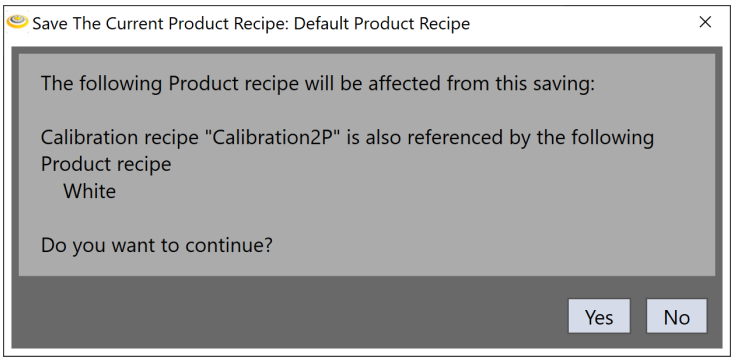


Save

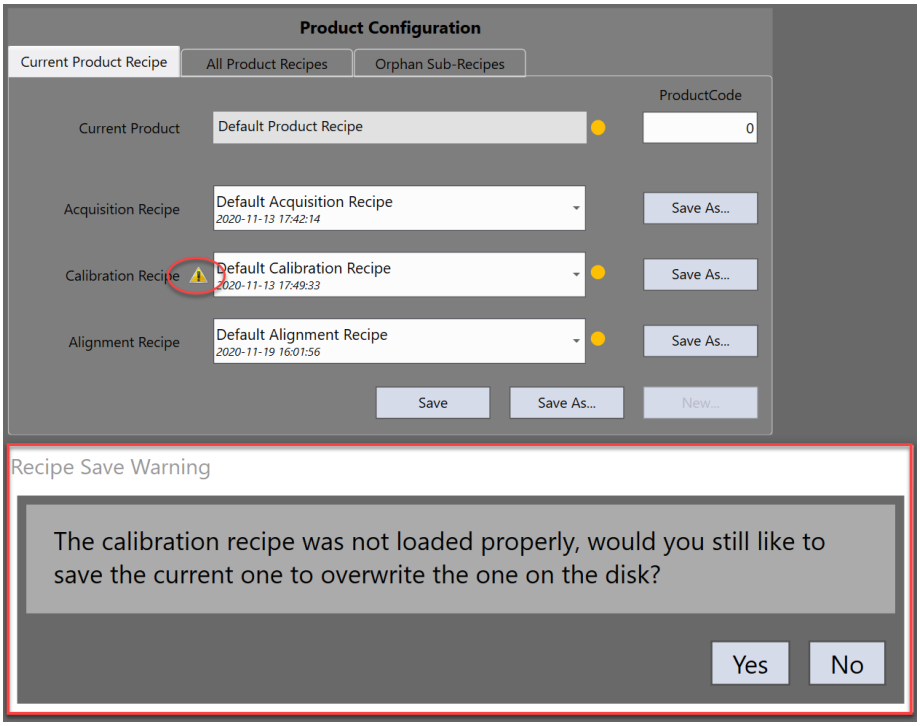
When there is any change happening to any sub recipe, that sub recipe as well as the product recipe will have a yellow dot appearing next to them. Clicking the "Save" button will save the change to its corresponding sub recipe and the current product recipe.



If any of the sub-recipes are also shared by any other master recipe, a message window will pop up to inform what will be impacted.

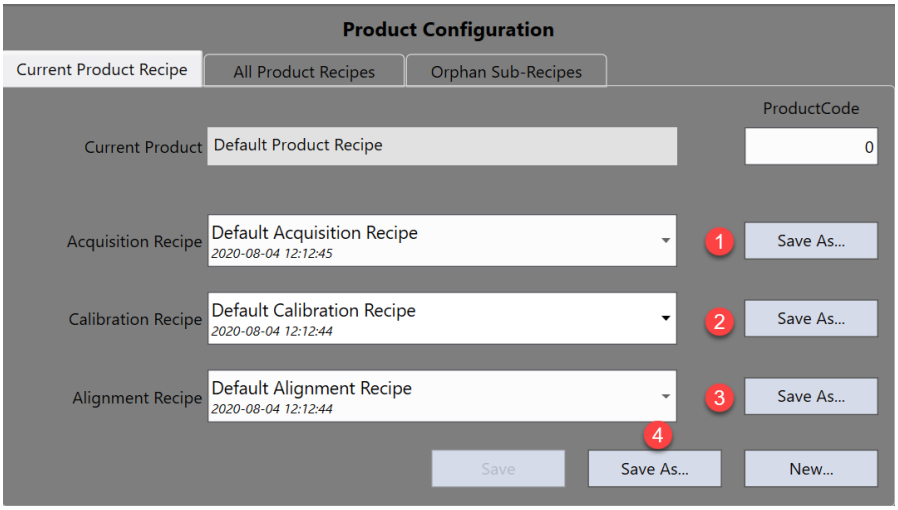


If the current recipe was not loaded properly, clicking the "Save" button will pop up a message window to confirm with you whether you want to overwrite the one on the disk.

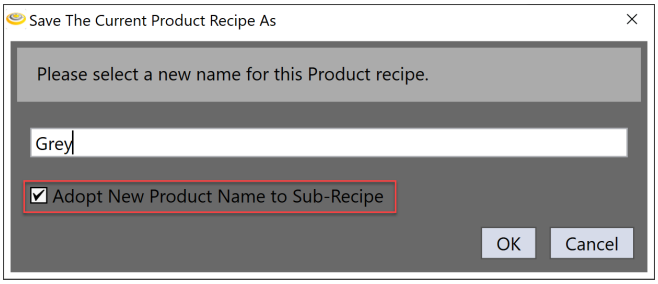


Save As

For each individual sub recipe, the **Save As** button allows the current state of the sub-recipe (Acquisition, Calibration or Alignment) to be saved under a different name. This sub recipe's new name will be selected for the current product recipe. This function can be used to backup recipes.



The fourth **Save As** at the bottom is to save the current product under a different name specified in the pop-up dialog, and this new product name will be selected as the current product.

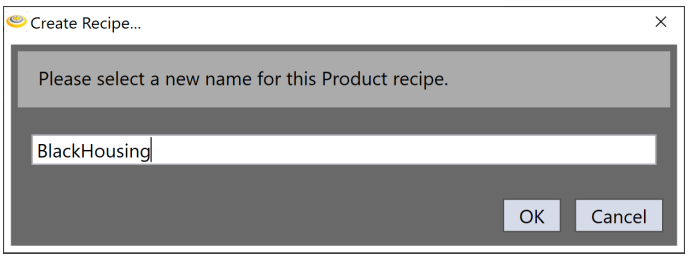


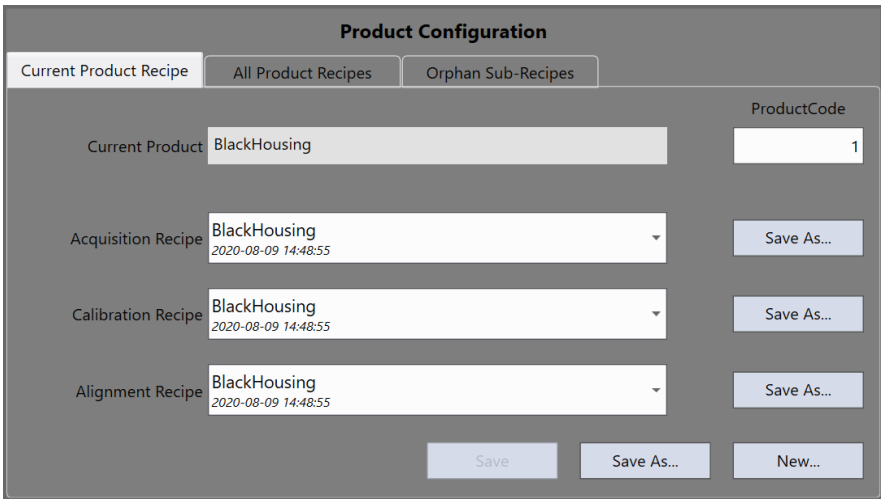
If **Adopt New Product Name to Sub-Recipe** is unchecked, the new product will share the sub-recipes with the current product. If it is checked, the new product will create its own sub recipes that have the same names with the new product and save all changes to them respectively.

⚠ **CAUTION**: When the Configuration Wizard reruns, the contents in "Default Product Recipe" will be cleared out. Please save it as a different name before rerun the wizard.

New

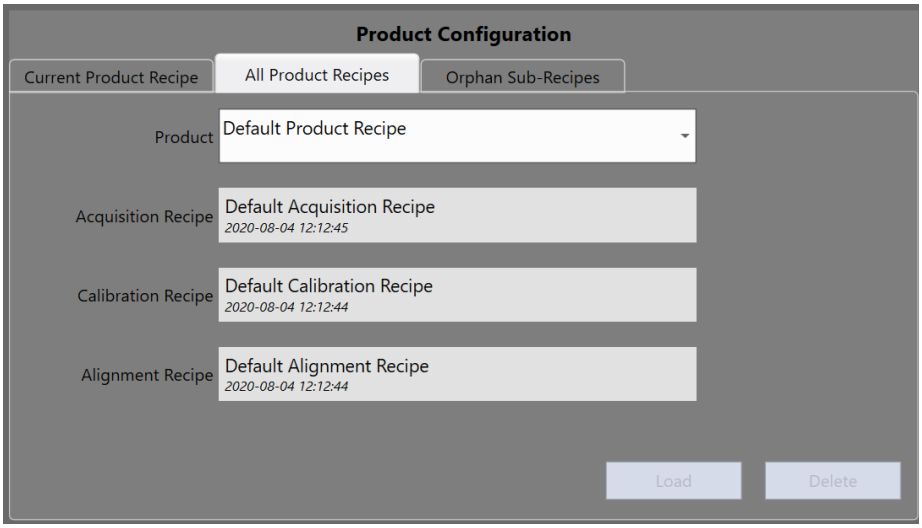
Click **New** button to create a new product recipe as well as the sub recipes. This new product name will be selected as current product. Note if any sub recipe with the given name already exists, the existing sub recipe will be used instead of creating a brand new sub recipe.





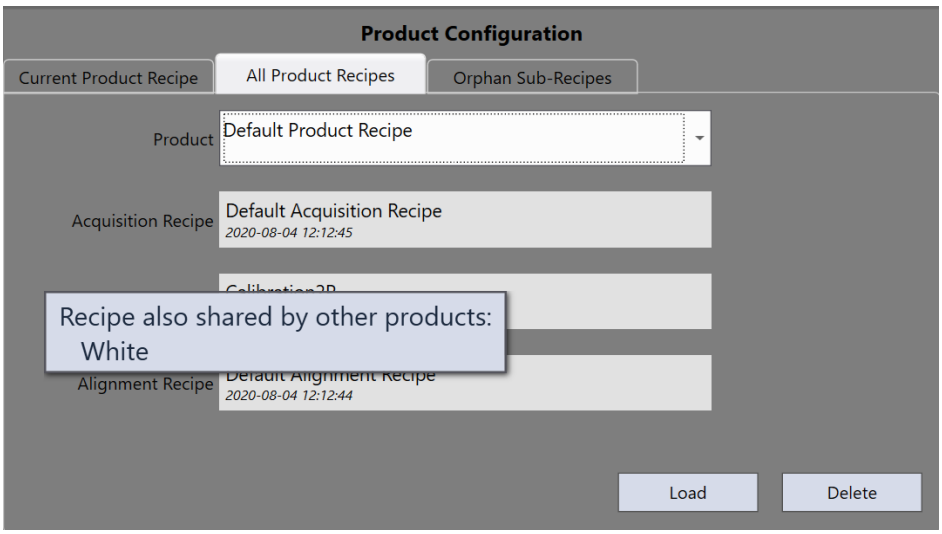
All Product Recipes

This tab is for loading or deleting operation on the existing products.



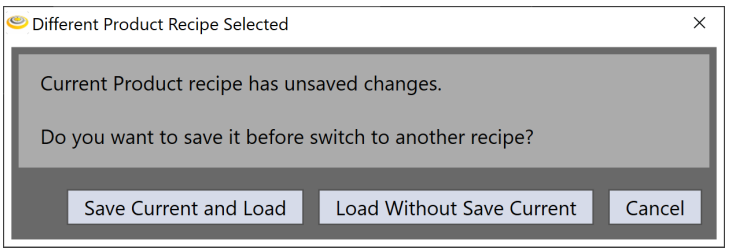
Item	Description
Product	The drop down list shows all the available product recipes.
Acquisition Recipe	The name of acquisition sub recipe of the selected product
Calibration Recipe	The name of calibration sub recipe of the selected product
Alignment Recipe	The name of alignment sub recipe of the selected product
Product Code	The code number of the current product

If a sub recipe is shared between products, the information will be shown in a pop up control when mousing over the display.



Load

Loads the selected product as current product. If the current recipe or the sub recipes have unsaved changes, a message window will pop up to allow the user to perform desired action.

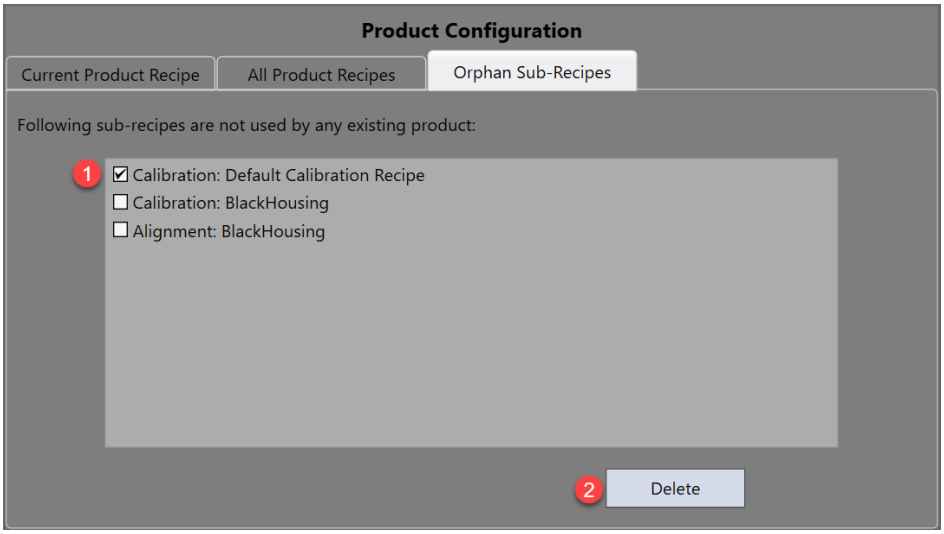


Delete

Delete the selected product. Note that the corresponding sub recipes associated with the deleting product will not be deleted. Use the "Orphan Sub-Recipes" tab to delete unwanted sub-recipes.

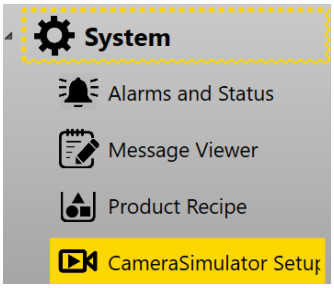
Orphan Sub-Recipes

This tab is for deleting operations on those orphan sub-recipes (the recipes are not referenced by any existing product). If you want to delete them, just select them, and then click the "Delete" button.



Camera Simulator

Camera Simulator is a function that allows AlignPlus program to acquire images by reading image files from designated local folders instead of acquiring from physical cameras when cameras are not available. This function can be found under **System** category in setup mode.



To make camera simulator work, you need to assign specific folders and put images into these folders. There are two ways to do it: simple mode and advanced mode.

Simple Mode

In simple mode, AlignPlus will automatically create all sub folders for each camera at each acquisition position once the root directory is appointed. However, after that, it takes time to put correct images into each sub folder.

Here are the steps:

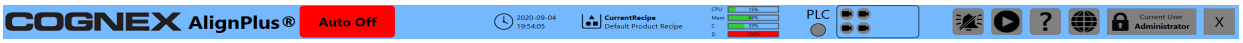
1. Enable camera simulator

Enable camera simulator by checking on **Simulator Cameras** at the upper left corner of this page.

Once the camera simulator is enabled, the background color of title bar will change from yellow to blue to remind user that real cameras are not used for image acquisition. When is disabled, the background color will change back to yellow and the application will acquire images from real cameras.

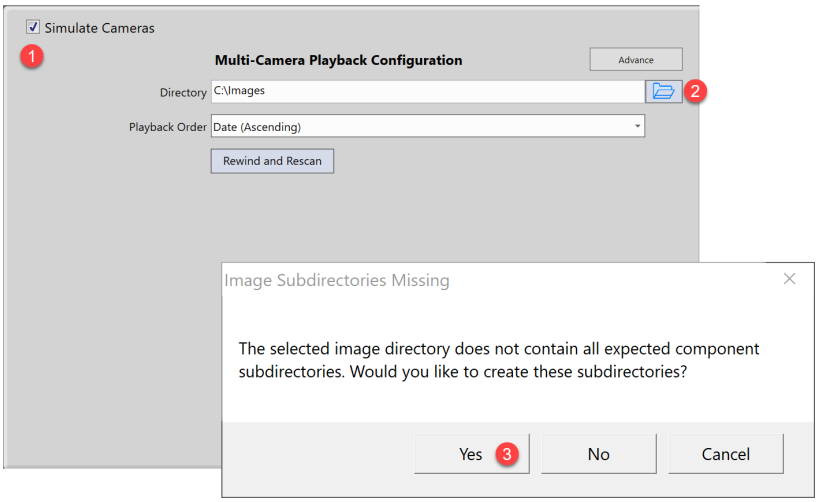
- Using cameras:

- Using camera simulator:

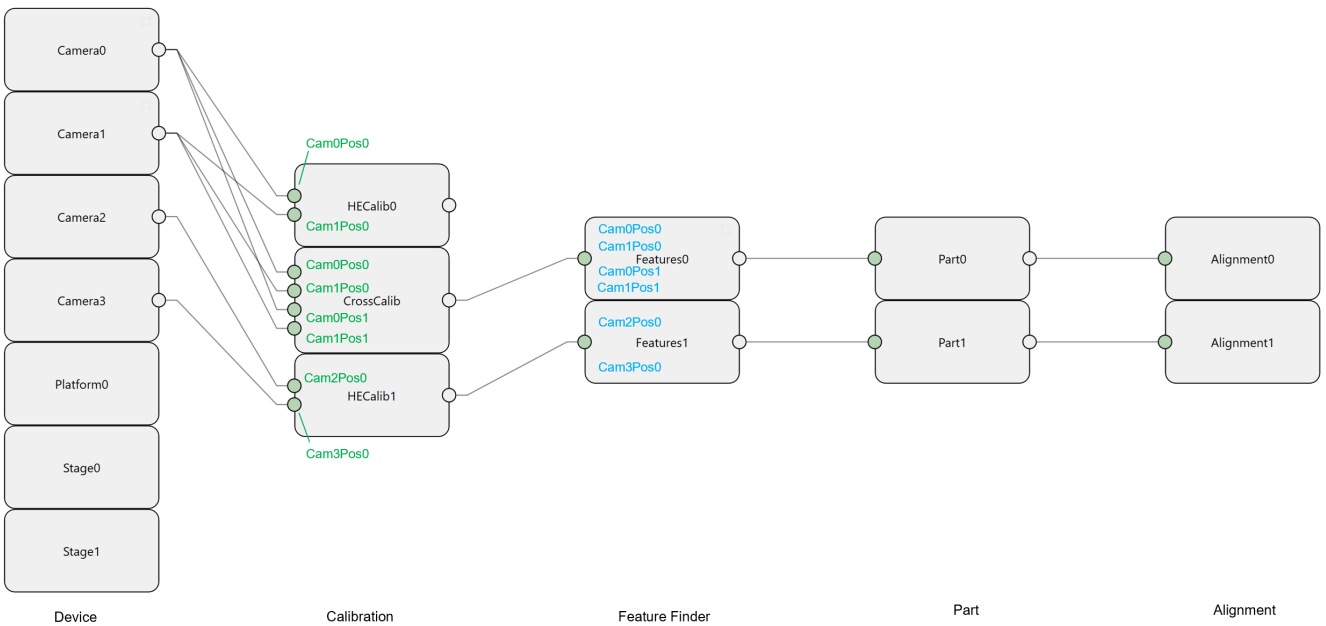


2. Select directory

Click "  " icon and select one existing empty folder for **Directory**. If the selected directory is empty, the application will pop up a dialog asking if you want sub directories to be created automatically. If click **Yes**, then the program will automatically generate empty sub directories for all image acquisitions needed in this application.

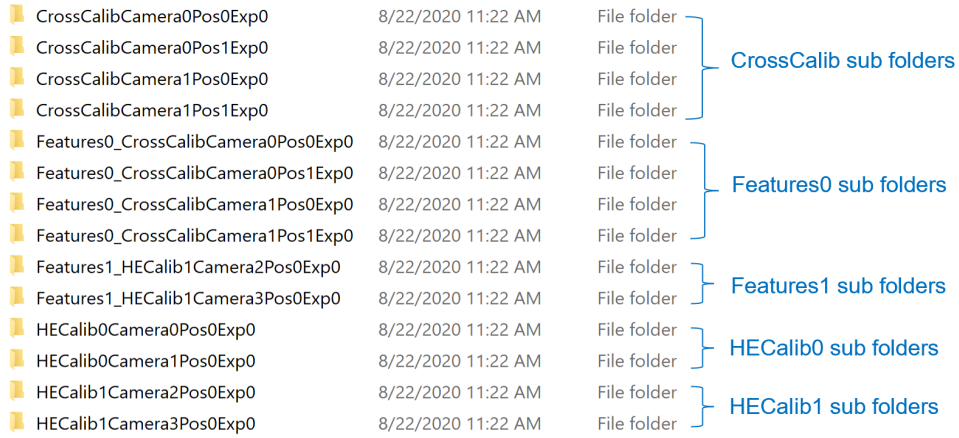


The number of sub directories and their names are based on the wizard configuration. One sub directory is for one camera at one acquisition position. In the example below, **HECalib0** task will have two sub directories, one for **Camera0Pos0**, another for **Camera1Pos0**. It is the same case with **HECalib1**: one for **Camera2Pos0**, another for **Camera3Pos0**.

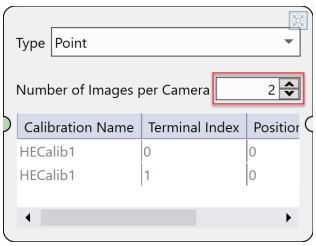


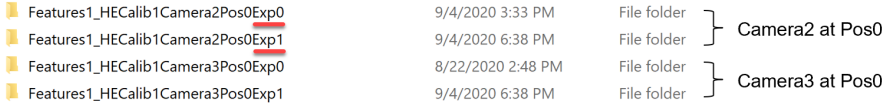
As for **CrossCalib**, since it has four pins connected(two cameras at two positions), it will have four sub directories: **Camera0Pos0**, **Camera1Pos0**, **Camera0Pos1**, **Camera1Pos1**. For **Features0**, it also requires four sub directories since it also acquires images from two cameras at two positions. **Features1** will have two sub directories similar as **HECalib1**.

Here are all the generated sub directories:



In most cases, one camera will only acquire one image at each position in feature finder task. However, if it is configured as acquiring two images in the wizard configuration, then the corresponding sub directories will be doubled with different exposure indexes, such as below:





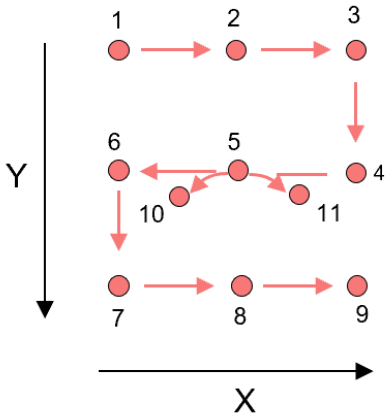
3. Copy simulation Images into each sub directory

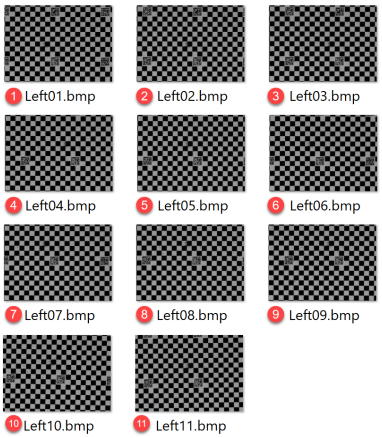
All these automatically generated sub directories are empty inside, we need to manually put images in to make the program be able to read images from these directories. The supported image formats are: BMP, JPG, PNG and TIF.

These images should come from real cameras on a real machine in a real alignment or assembly application so that when they are used in this new project, the calibration and feature finding will make sense. Otherwise, the calibration may fail using some random calibration images. Or even if the calibration passes, the feature finding may still fail as the corrected images out of the wrong calibration results are so distorted.

For hand-eye calibration sub directories, the image quantity and order for each camera should match with the loop settings in the hand-eye calibration parameter.

Calibration Loop	Corresponding Images

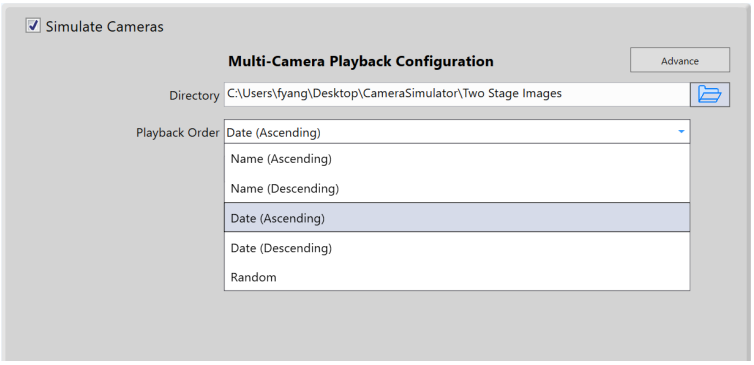




For other directories, images should be put in an order that every time when the application reads images from each cameras' sub directories, the images are loaded in a way as if they were acquired of a real part from real cameras.

4. Confirm playback order

After putting all necessary images under each sub directory, come back to camera simulator page, and choose the playback order from its drop-down list. The default option is **Name(Ascending)**. You can also use **Data** if it describes the order better. **Random** is not recommended as it will break the order how those images were acquired.



5. Rewind and Rescan

Click **Rewind and Rescan** button to let the application scan once for all images under each sub directory to register their names and numbers so that when it is time to play back, the program knows which image to read.

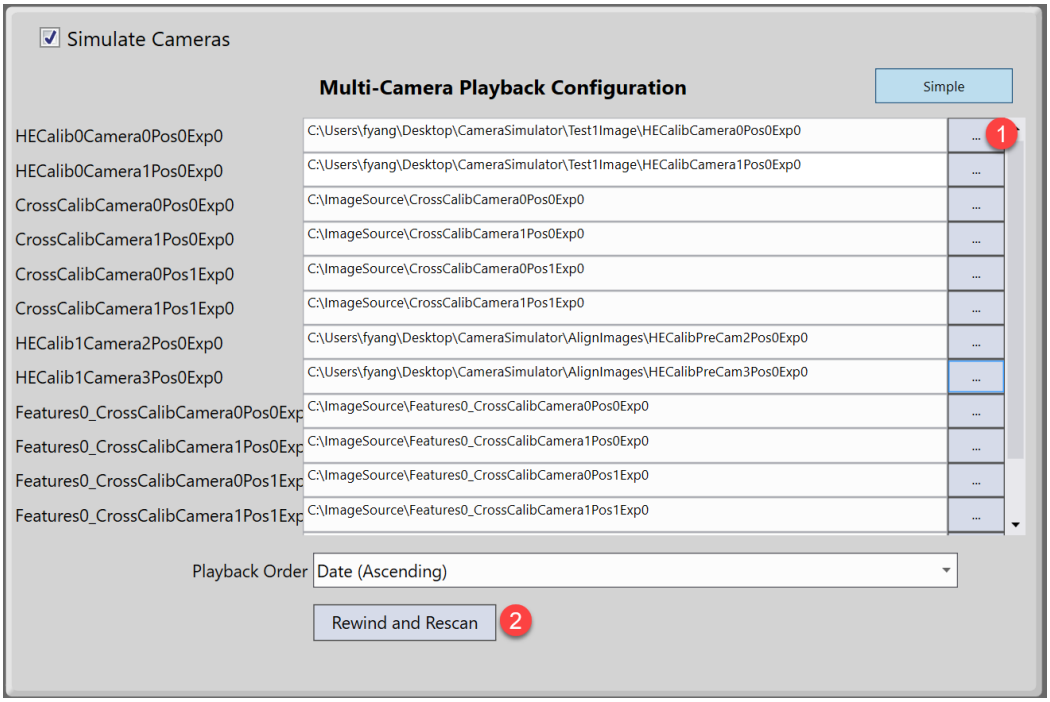
Rewind and Rescan should be clicked every time there is any image quantity or name changes under any sub directory. Otherwise, the program will assume the original images are still there and would continue to load them, which typically results in an acquisition exception.

During hand-eye calibration, if any exception(such as no valid data matrix code is found) occurs in the middle which caused the cancellation of the calibration, **Rewind and Rescan** should be clicked again after the exception is handled and before a new hand-eye calibration process starts to make sure all images will be loaded from the beginning. Otherwise, the application will continue to load the next images from the point where last hand-eye calibration is canceled, which will result in calibration failure in the end.

Advanced Mode

When you have already created a bunch of sub folders and images for other projects, you may feel reluctant to copy those images again one by one into new project's sub directories just because they have different sub directory names. Advanced mode can help you on this.

In advanced mode, you re-navigate the path for each sub directory instead of copying images inside from directory to directory. This may make sub directory names quite colorful, however it does not affect the new program to load images correctly from each directory since it maintains a mapping between new directory names and default sub directory names.



Click the **Advance** button at the upper right corner to enter advanced mode. The setting page will change to as above: each default sub directory will have one directory browser that you can assign a new path. In the example above, all hand-eye calibration sub directories have been re-directed to some already existed image folders to avoid copying images around.

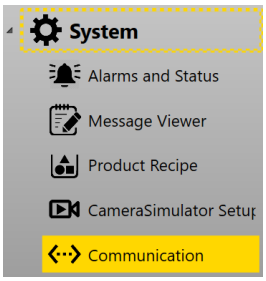
After all folders have been configured, double check playback order and then click **Rewind and Rescan** for the program to register all the changes.

Run

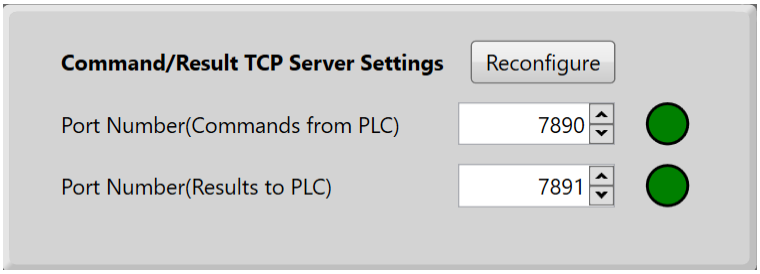
After all settings are done either in simple mode or advanced mode, you are ready to run the program as if cameras were all set.

Communication

Communication page provides the settings for TCP/IP configuration. It can be found in System category in Setup mode.

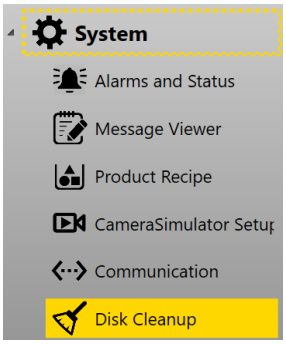


Change the port numbers for CommandsFromPLC device or ResultsToPLC device and click **Reconfigure** button. When the ports are available, the corresponding status light on the right will turn green, otherwise will be red.

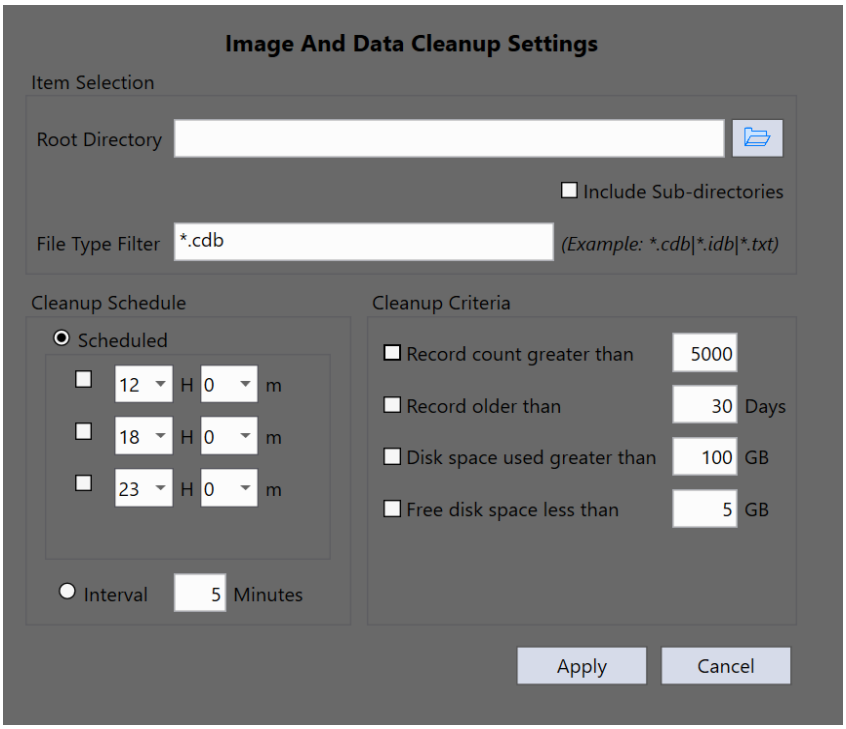


Disk Cleanup

Disk cleanup function helps clean up the files under a specific folder when the size of that folder is getting too large. It can be found in **System** category in setup mode



Disk cleanup has many convenient settings that allow you to decide under what condition to clean and when to clean. It even allows you to set timepoints to run the clean-up task according to your convenience, one common instance of which is when the machine is not busy.



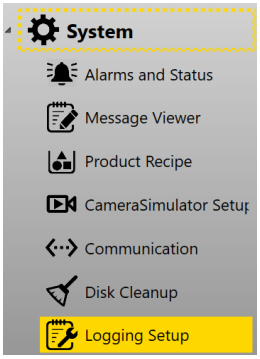
Item	Description
Root Directory	The root directory under which files will be cleaned up.

Item	Description
Include Sub-directories	• Unchecked: only files under the root directory will be cleaned up. • Checked: all files under the root directory and its sub folders will be cleaned up.
File Type Filter	The file types to be cleaned up: it could be images, text files, or both.
Cleanup Schedule	Schedule specific timepoints of a day to run the clean-up task.
Interval	Run clean-up at certain intervals. **Note**: Clean-up will slow system down therefore it's not recommended to run it at a very short intervals.
Cleanup Criteria	• Record count greater than: run clean-up task when the root folder size is greater than the given threshold. • Record older than: run clean-up task when the files in it is older than given threshold. • Disk space used greater than: run clean-up task when the disk space used is over the given threshold. • Free disk space less than: run clean-up task when the remaining disk space is less than the given threshold.

After you set up all parameters, click **Apply** to save them.

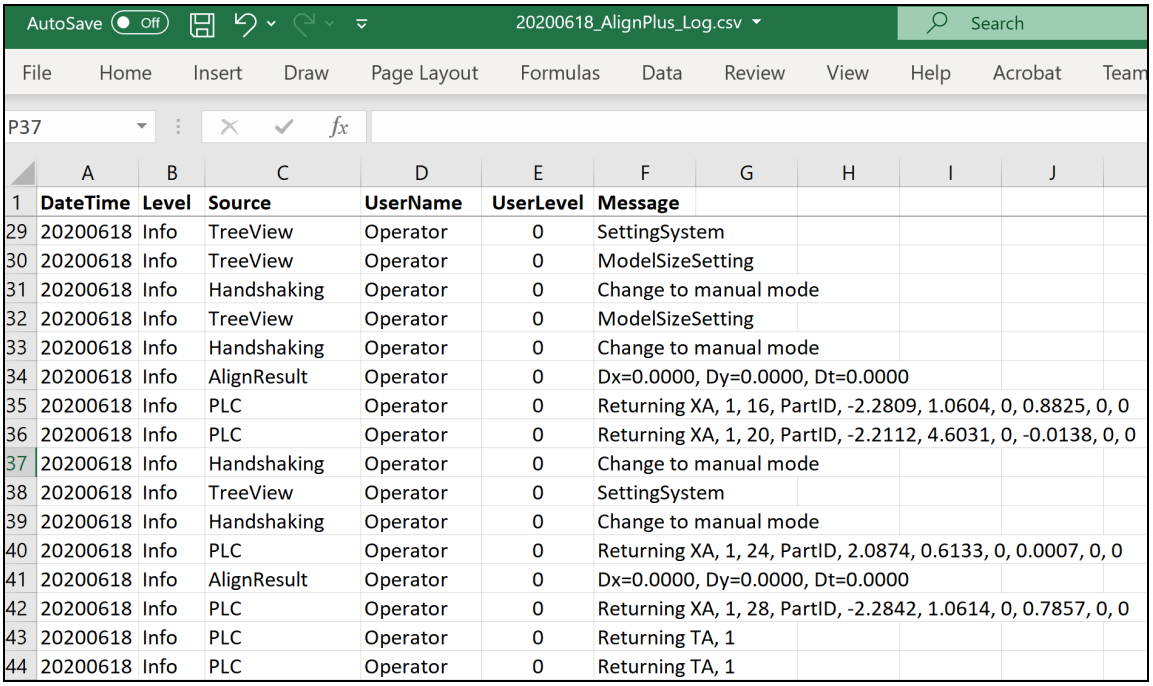
Logging Setup

Logging setup pages specifies two types of log settings: System Logging and Alignment Logging. It can be found in **System** category in setup mode.



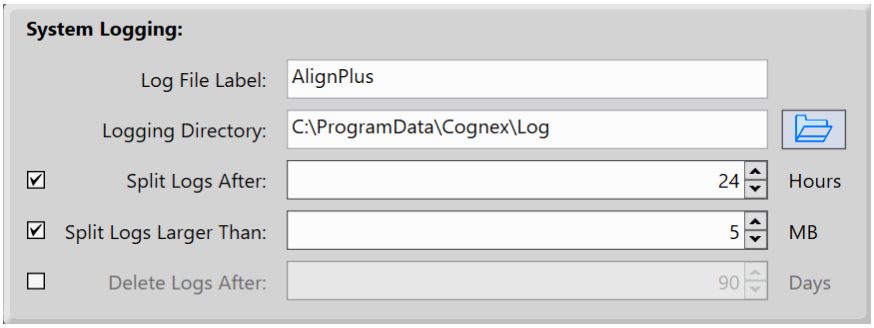
System Logging

System Logging saves system logs that are shown at the button bar into .csv files. It covers commands received from external devices, results sent out by vision, operator's actions, error message when exception happens, etc. A system log includes DateTime, user name and control level, message type, message source and message content. The default directory of the saved .csv file is "C:\ProgramData\Cognex\Log". The default file name is "<Date>_AlignPlus_Log".



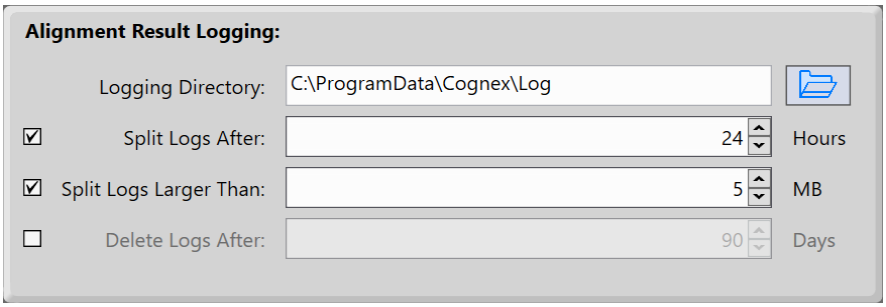
	A	B	C	D	E	F
1	DateTime	Level	Source	UserName	UserLevel	Message
29	20200618	Info	TreeView	Operator	0	SettingSystem
30	20200618	Info	TreeView	Operator	0	ModelSizeSetting
31	20200618	Info	Handshaking	Operator	0	Change to manual mode
32	20200618	Info	TreeView	Operator	0	ModelSizeSetting
33	20200618	Info	Handshaking	Operator	0	Change to manual mode
34	20200618	Info	AlignResult	Operator	0	Dx=0.0000, Dy=0.0000, Dt=0.0000
35	20200618	Info	PLC	Operator	0	Returning XA, 1, 16, PartID, -2.2809, 1.0604, 0, 0.8825, 0, 0
36	20200618	Info	PLC	Operator	0	Returning XA, 1, 20, PartID, -2.2112, 4.6031, 0, -0.0138, 0, 0
37	20200618	Info	Handshaking	Operator	0	Change to manual mode
38	20200618	Info	TreeView	Operator	0	SettingSystem
39	20200618	Info	Handshaking	Operator	0	Change to manual mode
40	20200618	Info	PLC	Operator	0	Returning XA, 1, 24, PartID, 2.0874, 0.6133, 0, 0.0007, 0, 0
41	20200618	Info	AlignResult	Operator	0	Dx=0.0000, Dy=0.0000, Dt=0.0000
42	20200618	Info	PLC	Operator	0	Returning XA, 1, 28, PartID, -2.2842, 1.0614, 0, 0.7857, 0, 0
43	20200618	Info	PLC	Operator	0	Returning TA, 1
44	20200618	Info	PLC	Operator	0	Returning TA, 1

It will split into different .csv files based on time span or file size to keep the files feasible to operate. The default time span is 24 hours, default size limit is 5 MB and you can also change them to meet your needs.



Alignment Result Logging

Alignment result log is also saved under the same directory of system log file with a different name: "<Date>_<Alignment Name>_Log", for example "20200616_Alignment0_Log". **Alignment Name** here is the alignment task name, each alignment task will have its own alignment log file. Those files will also be divided into different files based on time span and file size.



Alignment result log content has two parts. The first part is the result x, y, theta values that vision sent back to motion device including absolute and relative positions; the second part is result data of all features in run time and train time.

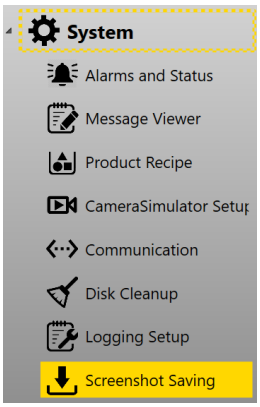
Name	Description
TimestampUTC	Time stamp when the log is written down
ErrorMsg	Error message, empty when there is no error
AbsoluteStageMotionX	X value of target position that motion device should move to
AbsoluteStageMotionY	Y value of target position that motion device should move to
AbsoluteStageMotionThetaDegrees	Theta value of target position that motion device should move to
RelativeStageMotionX	X value of relative position that motion device should move
RelativeStageMotionY	Y value of relative position that motion device should move
RelativeStageMotionThetaDegrees	Theta value of relative position that motion device should move

The result data of each feature contains the following data:

Item	Description
IsFound	Indicates if the feature is found.
IsValid	Indicates if the result of the finder is valid. If this is true then the X, Y and Score properties are valid. If MultipleFindersPerFeatureMode is true, IsValid can be true, but IsFound can be false.
FeatureLocationX	The X coordinate of the found feature. This value is undefined if IsFound = false.
FeatureLocationY	The Y coordinate of the found feature. This value is undefined if IsFound = false.
FeatureRotationXInDegrees	The angle of the X Axis of the found feature. This value is undefined if IsFound = false and is valid only if found by a PatMax tool.
FeatureRotationYInDegrees	The angle of the Y Axis of the found feature. This value is undefined if IsFound = false and is valid only if found by a PatMax tool.
CameraIndex	The index of the camera containing the point feature finder and the feature location. The value is -1 for multi camera custom feature finder.
Score	Score of found feature. This value is only valid when PatMax Finder is used to find the feature

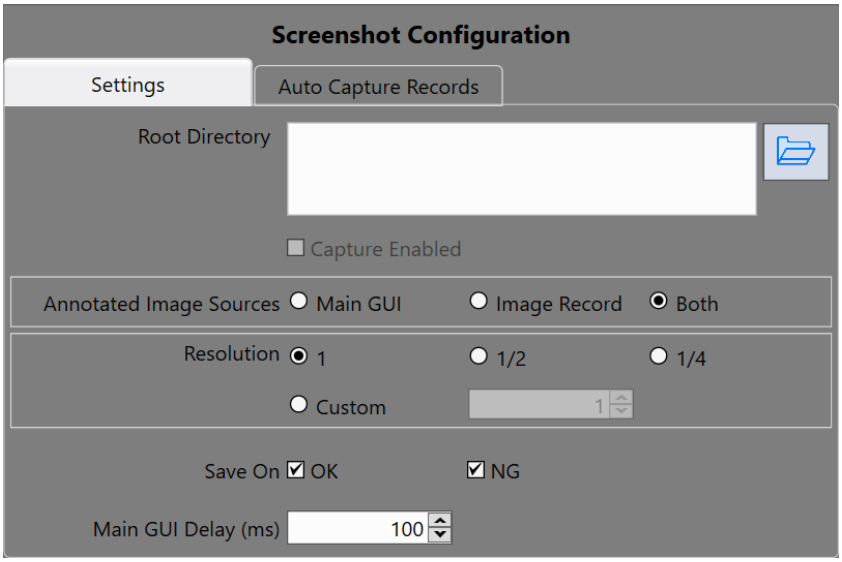
Screenshot Saving

Screenshot saving page configures the settings for screenshot images. It can be found in **System** category in setup mode.



Settings

The Settings tab configures the resolution of screen shot images, whether to save GUI or not, etc.



Item	Description
Root Directory	The root directory for all screenshot images to save
Capture enabled	Obsolete
Annotated Image Sources	Main GUI: The screen shot of main GUI will be saved Image Record: The run time images together with result graphics will be saved Both: Both screen shot of main GUI and images with result graphics will be saved
Resolution	1: Save images at its original resolution 1/2: Save images at its half resolution 1/4: Save one quarter the size of original resolution Custom: Use customized size to save image
Save On	OK: Save OK images NG: Save NG images When they are both checked on, the program will save both OK and NG images
Main GUI Delay(ms)	Add delay time for main GUI screen shot to make sure all image displays have been updated before being captured

Auto Capture Records

This tab configures which result images to save. For most cases, only feature finder tasks' FeatureExtractorSubTasks need to be saved as they have the run time image results. The list below may be a bit complicated for you to choose, however, you can set it in Image Saving on page 176 instead as **Image Saving** has clearer description about what screenshot images to save.

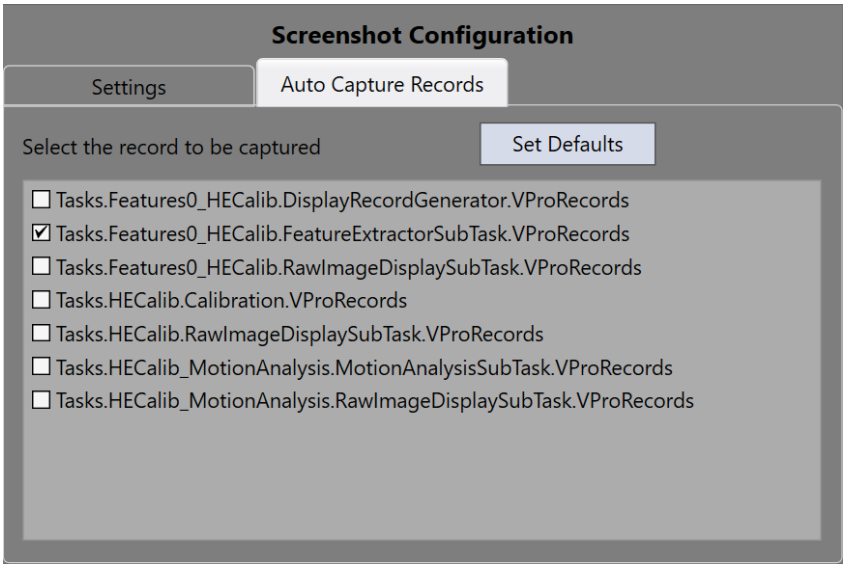
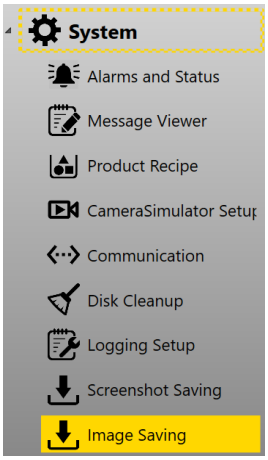


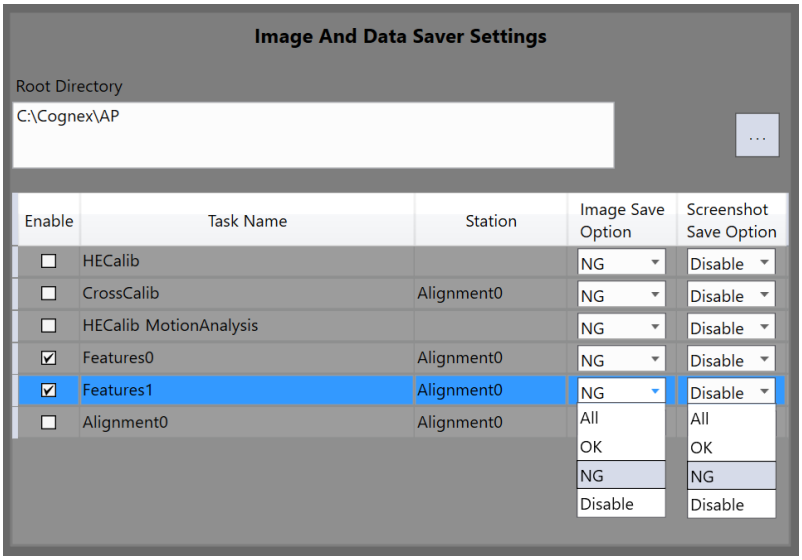
Image Saving

Image saving function disables or enables raw image or screen shot saving. It can be found in **System** category.



Settings

The setting page is as below:



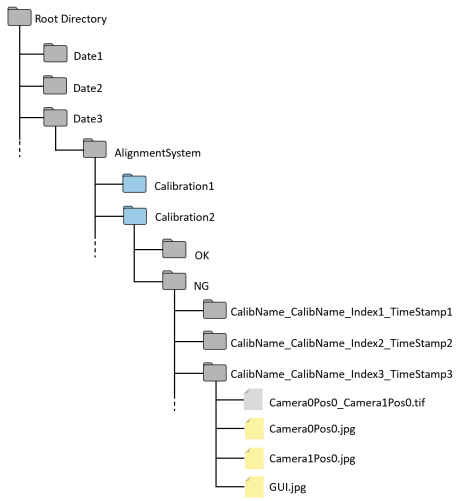
Item	Description
Root Directory	Root directory for raw image saving and screen shot saving.
Enable	Disable or enable raw image saving and screens shot saving functions in selected task.
Task Name	The task whose images are to be saved: every calibration, feature finder, motion analysis, and alignment tasks can save images independently
Station	Alignment component name in configuration wizard
Image Save Option	• OK: only save OK images; • NG: only save NG images: • All: save both OK and NG images; • Disable: not save any images (this function allows raw image saving and screen shot saving to be disabled or enabled independently).
Screenshot Save Option	• OK: only save OK images; • NG: only save NG images: • All: save both OK and NG images; • Disable: not save any images (this function allows raw image saving and screen shot saving to be disabled or enabled independently).

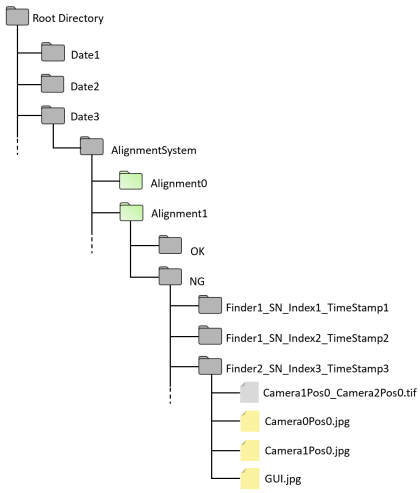
Sub Directories

All images would be saved under root directory but will be distributed into different sub directories depending on date, alignment system, station name, etc. Here is the directory structure for calibrations and feature finders.

Calibration sub directories

Feature finder sub directories





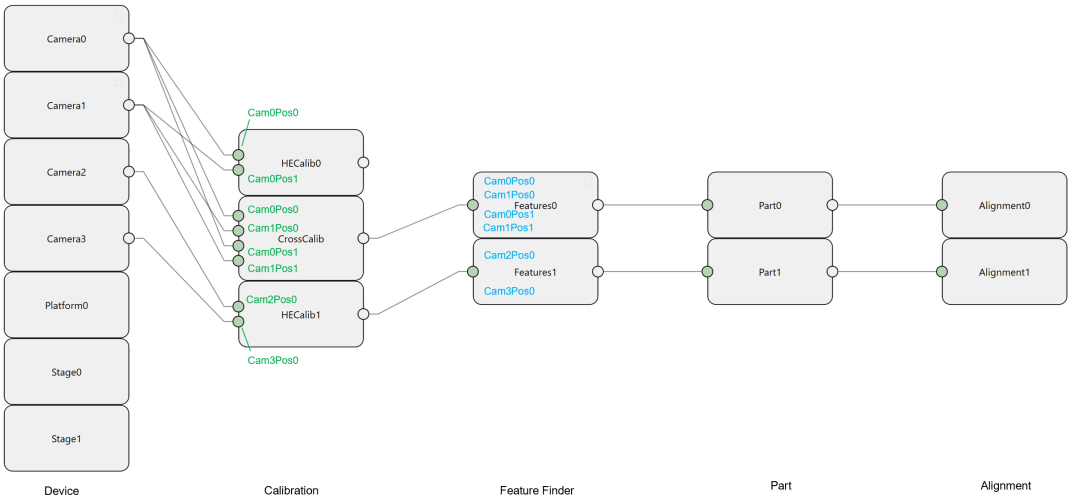
Here are the keywords used for sub directories:

Keyword	Description
Root Directory	Root directory specified in **Image Saving** HMI
Date	Date in the format of "yyyyMMdd" when images are saved
AlignmentSytem	Current alignment system
Calibration	The calibration component name specified in configuration wizard, such as "HECalib0", "CrossCalib"
Alignment	The alignment component name specified in configuration wizard, such as "Alignment0", "Alignment1"
OK	The folder to save images into when calibration or feature finding result is OK
NG	The folder to save images into when calibration or feature finding result is NG
Finder	The feature finder component name specified in configuration wizard, such as "Features0", "Features1"
SN	Serial number of the part whose images are saved
Index	Index which increases by 1 automatically every time there is one new raw image from one camera is saved
TimeStamp	Time in the format of "HHmmss.fff" when images are saved

Note that raw images and screen shots are saved under the same sub directory if they are both enabled to be saved.

Image Files

Images files are named using **AcqName** which is composed of two parts: camera name and position name, such as **Camera0Pos0**, or **Camera1Pos0**. In the example below, all AcqNames are marked for calibrations and feature finders in configuration wizard:



Raw Images

Raw images are saved in TIF files in the unit of acquisition position. For all cameras that acquire images at the same position in one calibration or feature finding task, their raw images will be saved together into one TIF file whose name is a combination of all AcqNames, such as "Camera0Pos0_Camera1Pos0.tif" or ""Camera0Pos1_Camera1Pos1.tif". If there are four cameras acquiring images at the same time at one position, then the image file will be like "Camera0Pos0_ Camera1Pos0_Camera2Pos0_Camera3Pos0.tif". The TIF file will contain four images inside then.

Image Records

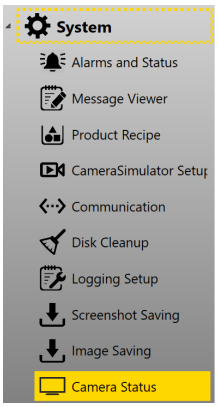
Image records are annotated images with result graphics. It is one of the screen shot image types. Image records are saved in JPG file in the unit of **AcqName**. such as "Camera0Pos0.jpg", "Camera1Pos0.jpg". See more settings about image records in Screenshot Saving on page 174.

Main GUI Screenshot

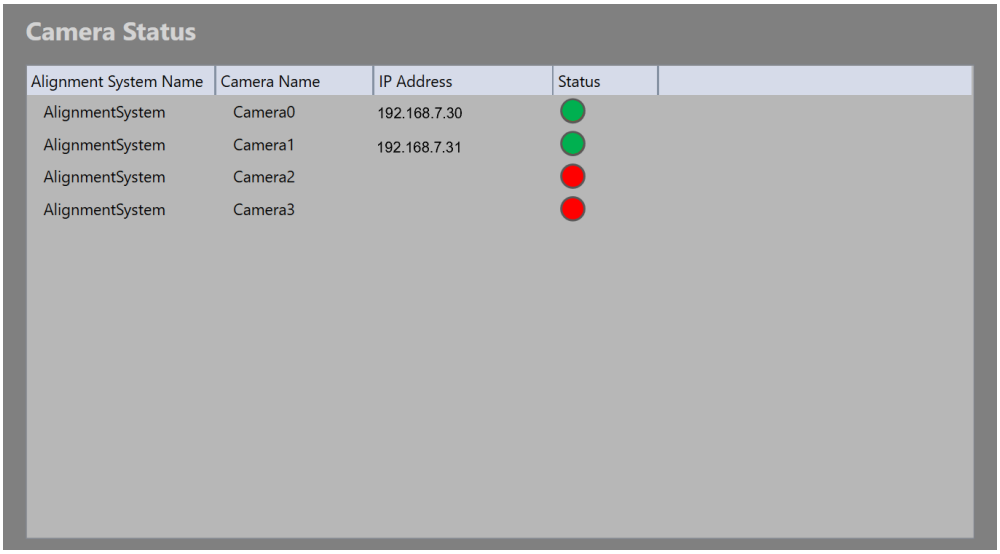
Main GUI saves screen shot of the main user interface. It is always named as "GUI.jpg". See more settings about main GUI saving in Screenshot Saving on page 174.

Camera Status

Camera Status page shows whether a certain camera is connected or not. It can be found under **System** category.



When a camera is connected, its status light will be in green. Otherwise, in red.



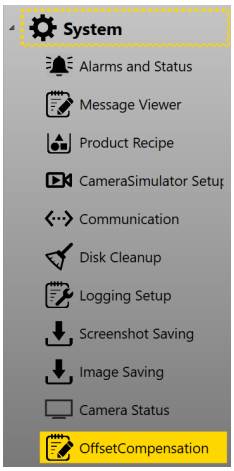
Offset Compensation

In alignment applications, run time part is aligned to its trained golden pose. However, the trained golden pose may not be at the ideal pose for further mechanical operation such as dispensing, gluing. Therefore, even run time part is perfectly aligned to its golden pose, the constant discrepancy between golden pose and the ideal target pose caused by mechanical positioning should be compensated to achieve alignment goals.

Likewise, in assembly applications, even if features from both parts are precisely located, and motion device moves accurately to target pose provided by the vision system, the gaps between them may not meet the specifications when the two parts are actually assembled. These differences between actual gaps and ideal gaps are caused by other mechanical process such as part transferring, or attachment. However, these differences are typically constant so that they can be compensated.

Offset compensation function allows user to manually add offset values to run time part features, so that when motion device moves to target pose provided by vision system, the desired alignment or assembly characteristics can be obtained. It currently only support point features.

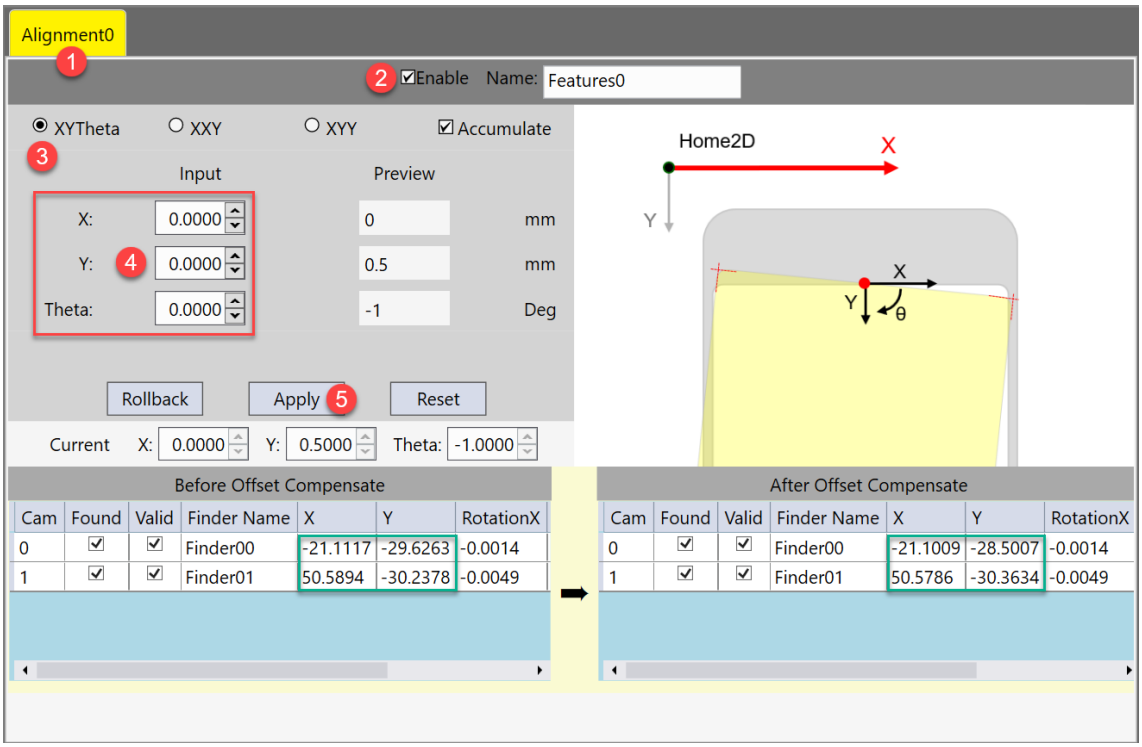
Offset compensation function can be found under **System** category in setup mode.



It provides three methods of compensation: **XYTheta** mode which compensates part feature locations based on X, Y, Theta offsets, **XXY** or **XYY** mode which compensates based on three gaps measured between real part and its target position.

XYTheta Mode

In XYTheta mode, x offset is offset along stage's x axis, y offset is offset along stage's y axis, theta offset(Θ)'s direction is from stage's x axis to its y axis. Those offsets could be calculated based on inspection results from inspection machine or calculated by operator to bring run time part to its target pose. After receiving these offsets, this function will first apply the theta offset by rotating all features around features' mean center. Second, compute all features' location changes caused by theta offset. Third, add the x, y offsets to features' changed locations computed in the second step.



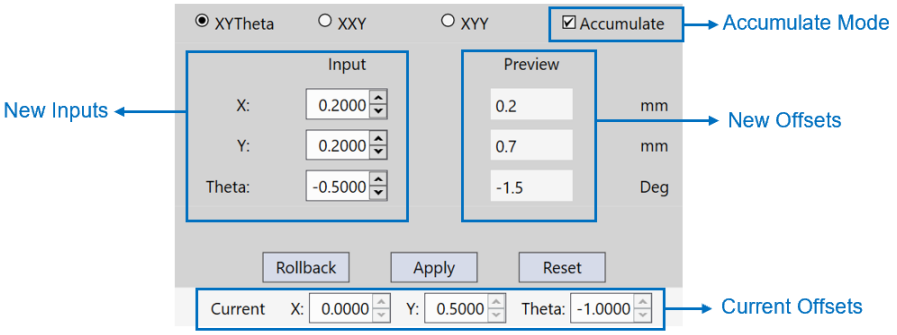
Here are the steps to add x, y, theta offsets:

1. Select one station
2. Enable that station's compensation function
3. Choose XYTheta mode
4. Input manually estimated X, Y, Theta offsets.

   X, Y, Theta offsets should be based on current part coordinate. Theta is in degree.
5. Run feature finding and pose computer one time.
6. Click **Apply** button to apply the offsets.

   After applying, one can find the current part's feature coordinates before and after compensation.
7. Save current recipe so that these offsets will be saved in the alignment sub recipe.

After the offsets are applied, you may find alignment or assembly result improved, but still a bit away from specifications. In this case you may want to add some extra small offsets to the current ones. This can be done checking on **Accumulate**.



As marked above, **New offsets** = **New Inputs** + **Current Offsets**. Click **Apply** button to overwrite current offsets with new offsets.

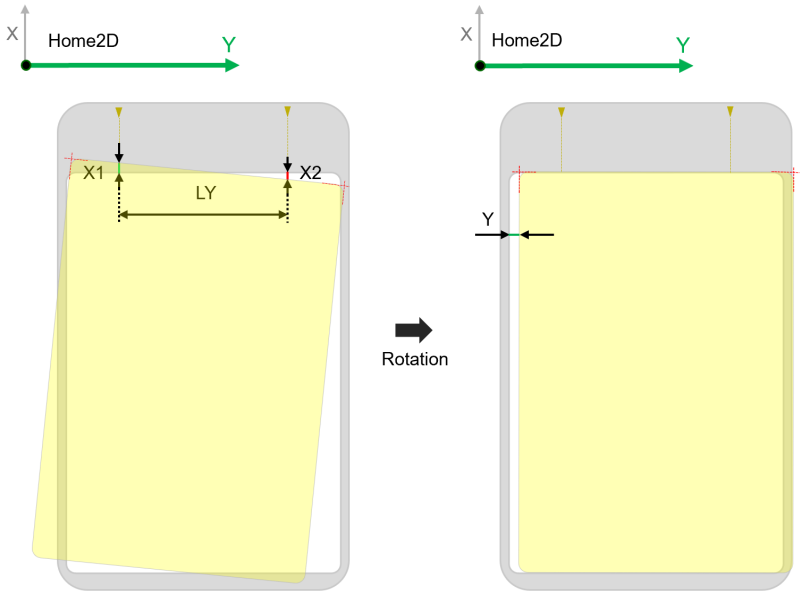
If by chance the newly added offsets make result drift further away, you can revert to the last offsets by clicking **Rollback** button, or clear all offsets by clicking **Reset** button.

Compensate Based on Gaps

Another way to compensate theta of run time part is by inputting two symmetric gaps between run time features locations and their corresponding target locations along one side of the part. If the gaps are equal, it indicates no theta compensation. Otherwise, an angle should be computed based on the two gaps' difference and the distance between where the two gaps are measured, and then compensated it back. When this side of part is along y axis of the stage, please choose XXY Mode. Otherwise, choose XYY Mode.

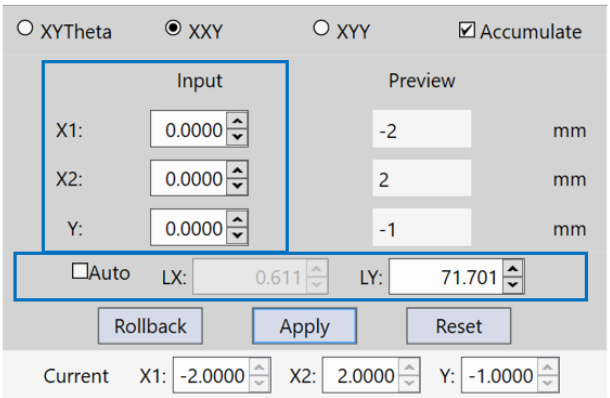
XXY Mode

XXY Mode uses two gaps along x axis of the stage, one gap along y axis of the stage to compensate run time part feature locations.



Item	Data Type	Description
X1	Signed Double	First measurement point's x gap from its target position, or how much the first point should move along x axis to its target x position. **Note**: The first measurement point is the point which is closer to stage's origin compared with the second point
X2	Signed Double	Second measurement point's x gap from its target position, or how much the second point should move along x axis to its target x position. **Note**: The second measurement point is symmetric to the first point on the part and is farther away from stage's origin
Y	Signed Double	The estimated Y offset of run time part after its theta has been compensated
LY	Unsigned Double	Absolute value of Y direction difference between two points where X1 and X2 are measured

The steps of adding XXY offsets are the same as those of adding XYTheta offsets. Only the inputs have been changed to X1, X2, Y, and LY.

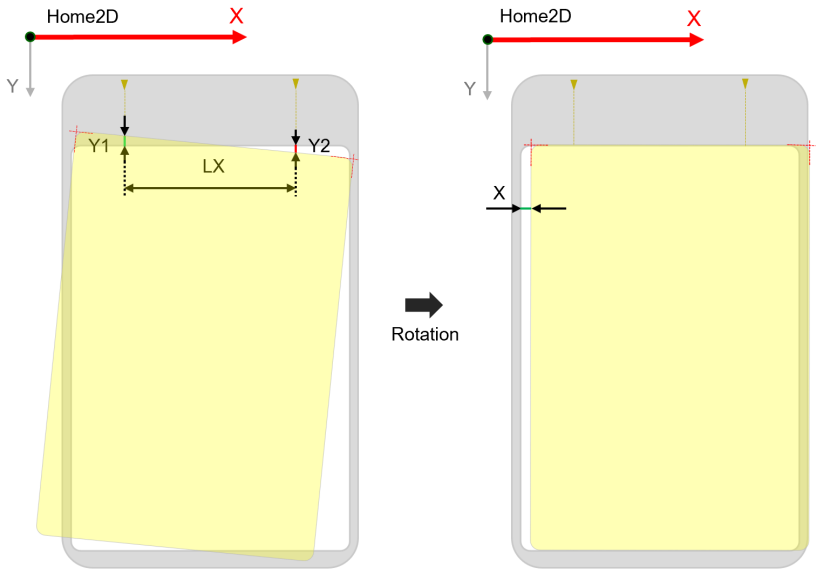


LY can either been manually inputted or automatically calculated by offset compensation function based on found run time feature locations. Check on "Auto" to let the program calculate LY and LX(x difference between two measurement points, should be 0 when there is no rotation issue, for reference only) automatically. The auto-calculated LY may be slightly different with the y direction distance between two measurement points as the measurement points may not be feature points. However, their difference is negligible because the mechanism of offset compensation is to compensate the part to be one-step closer to its ideal pose each time for several times to avoid over-compensation.

After receiving these values, offset compensation will first compute theta offset based on X1, X2 and LY. Second, compute all features' translation(x, y) changes resulted from rotating the part by theta around all features' mean center. Third, compute x offset for each feature based on X1 or X2 depending on which side the feature is and its x changes caused by rotation. Fourth, compute a common y offset for all features based on input Y and average y changes of all features caused by rotation.

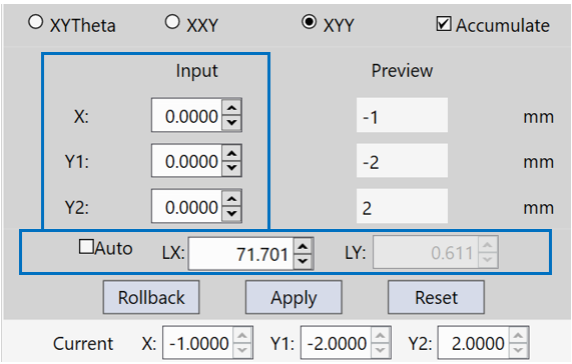
XYY Mode

XYY Mode uses one gap along x axis and two gaps along y axis of the stage to compensate run time part feature locations.



Input	Data Type	Description
X	Signed Double	The estimated X offset of run time part after its theta has been compensated
Y1	Signed Double	First measurement point's y gap from its target position, that is to say, how much the first point should move along y axis to its target y position. **Note**: The first measurement point is the point which is closer to stage's origin compared with the second point
Y2	Signed Double	Second measurement point's y gap from its target position, that is to say, how much the second point should move along y axis to its target y position. **Note**: The second measurement point is symmetric to the first point on the part and is farther away from stage's origin
LX	Unsigned Double	Absolute value of Y direction difference between two points where Y1 and Y2 are measured

The steps of adding XYY offsets are the same with adding XXY offsets. Only the inputs have been changed to Y1, Y2, X, and LX.

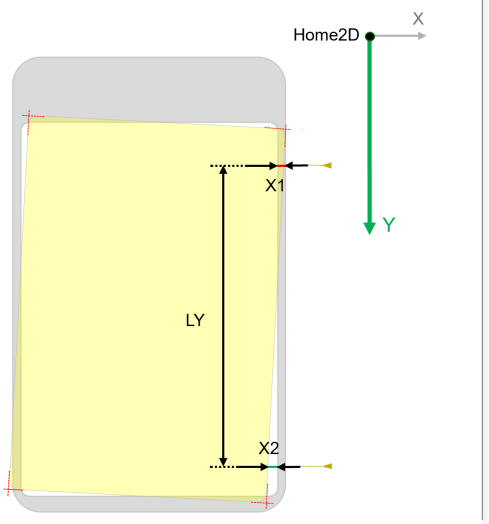


LX can also either be inputted manually or auto-calculated by this function using x difference of found feature locations. The way to compute run time feature location changes is the same with the XXY mode, only replacing X1, X2 with Y1, Y2, replacing LY with LX.

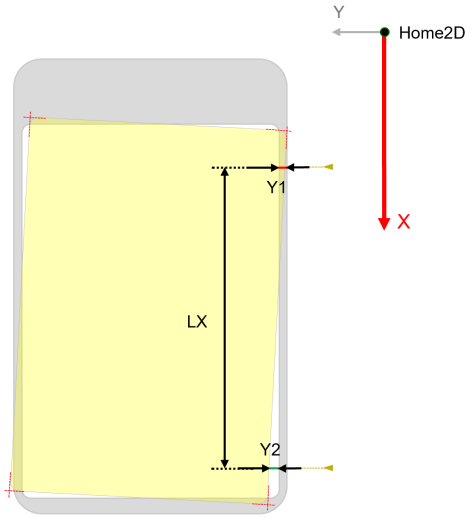
How to compensate multiple features

When there are more than 3 point features, it is recommended to measure the gaps along the long side of the part if XXY or XYY mode is used.

- **XXY Mode**

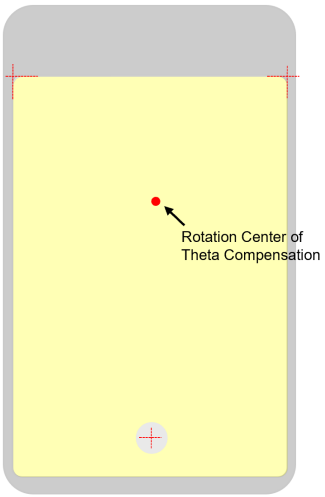


- **XYY Mode**



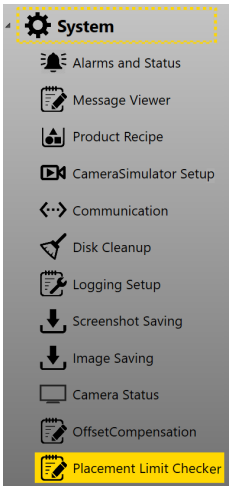
When "Auto" is checked on, offset compensation under XYY/XXY mode will use x/y difference between two features along the long side of the part as LX/LY.

However, if the mean center of all features is not expected to be rotation center of theta compensation, it is recommended to use XYTheta mode only.

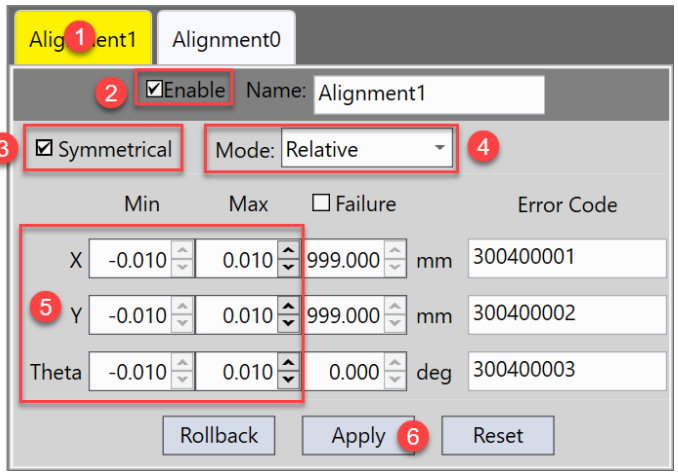


Placement Limit Checker

Alignment displacement limit check is a function to check whether output x, y and theta values provided by the pose computer are within certain limits. If they are, then alignment check will be OK. Otherwise, it will be NG. This function can be found in **System** category in setup mode.



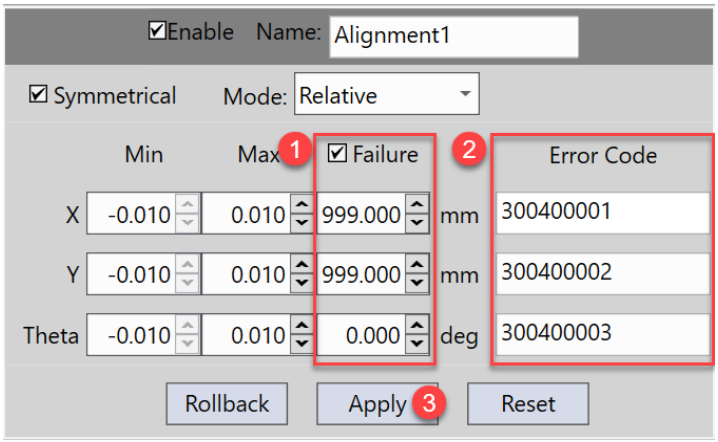
Here are the steps to set placement limit:



1. Select the station that needs placement limit check.
2. Check on **Enable** option.
3. Select if upper and lower limits should be symmetrical or not.
4. Choose to input relative limits or absolute limits.
5. Input upper(**Max**) and lower(**Min**) limits for X, Y and Theta values.

   If **Symmetrical** is checked on, then you only needs to input upper limits as lower limits will be generated automatically.
6. Click **Apply** to save those limits.
7. Save the current alignment recipe.

Besides upper and lower limits, you can also specify default values when check result is NG so that external devices can easily identify it is a NG case. The default NG values for X, Y, and Theta are 999.000, 999.000, and 0 respectively. If these default values are what you need, check on **Failure** option first, and then you can edit them.



Also, specific error codes are assigned when limit check is NG. The error code will be forwarded to external device so that it knows which specific value has failed the placement limit check. You can also edit these error codes here according to your own needs.

After all changes are done, click **Apply** and save the alignment recipe again.

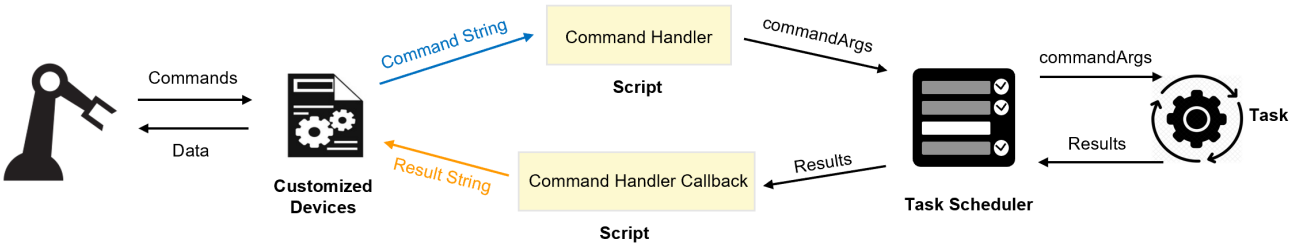
Program Workflow

In order to execute the various vision tasks in the vision application, command strings, whose format is defined in the Communication Protocol document, can be sent over a TCP/IP channel to the vision application by the customer PLC. Alternatively, other forms of communication can be used. However, the communication data should be converted into the equivalent AlignPlus command string. The command string is sent to the script **CommandHandler**.

The **CommandHandler** script, which handles these commands, converts the command string into an equivalent **CommandArgs** object that contains all the information and data needed to execute the target vision task. Execution of the vision task and gathering the results of the execution is performed by a **TaskScheduler** component. Following execution, the TaskScheduler executes its callback script which in turn calls the **CommandHandlerCallback** script. The CommandHandlerCallback script converts the results of the vision task execution into an equivalent result string, as documented in the Communication Protocol document. If an alternate form of communication is used, this string should be converted into the equivalent PLC response and sent back to the PLC.

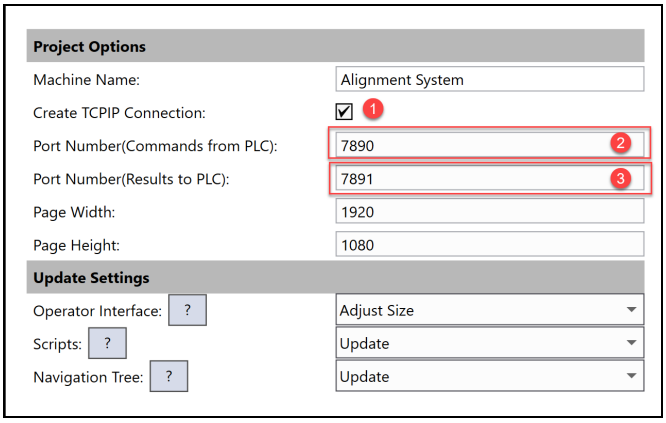
- For TCP/IP communication, the overall work flow is as below:

- For customized communication, the overall work flow is as below:



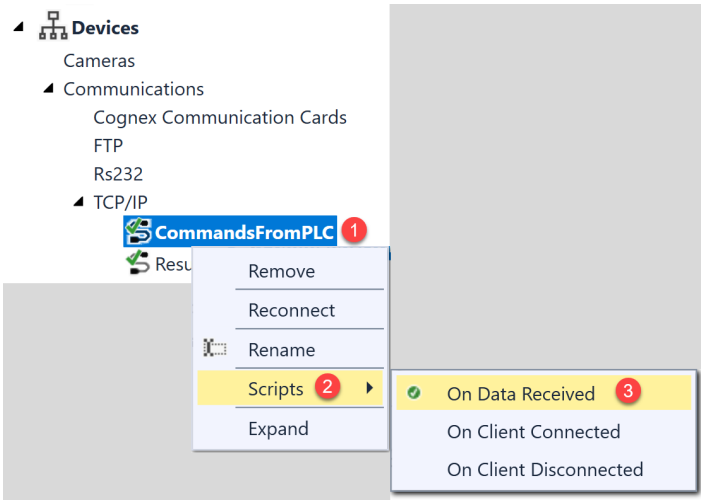
TCP/IP Communication Devices

To have the TCP/IP Devices automatically generated and functions implemented, enable the **Create TCPIP Connection** option and set the proper port numbers for TCP/IP server and client in the configuration wizard during application generation.

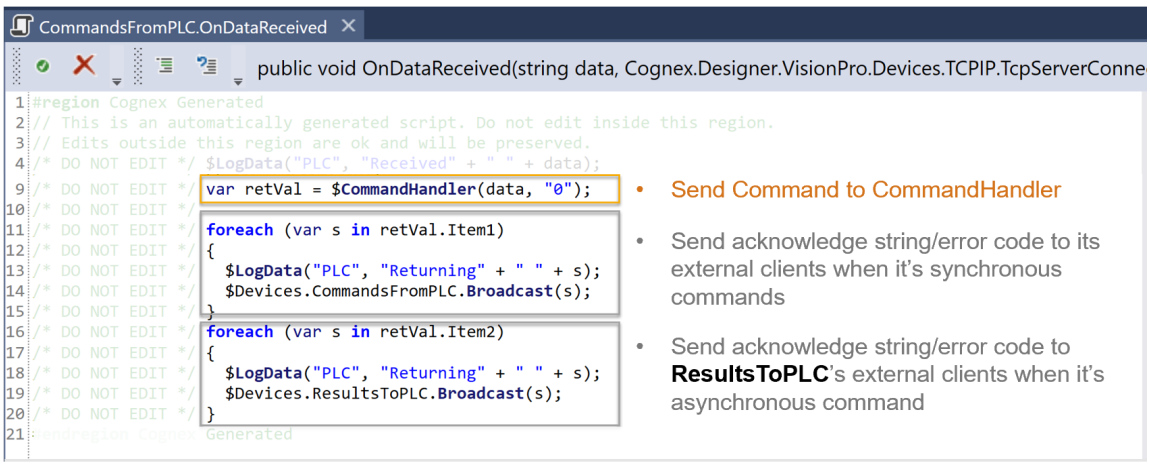


Commands From PLC

CommandsFromPLC is an TCP/IP server device that can receive command string from external TCP clients and sends the result string back to them after vision task finish running if task runs in synchronous mode. Once it receives a command string, it will call its **On Data Received** callback script. You can find On Data Received script by right click the device, and then choose it under Scripts.



The content of the On Data Received script is implemented by configuration wizard during application generation. It performs the following functions:



1. Send command string to **CommandHandler**

CommandHandler will first verify the format of command string, then it will generate a suitable CommandArgs object that is used by the task scheduler to pass data to the task that has to be executed.

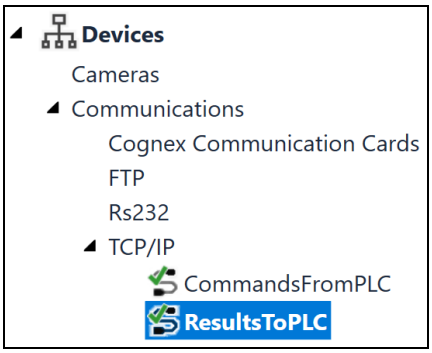
2. Send acknowledge string back to external devices.

When the task is to be run in synchronous mode, the acknowledge string which stored in `reVal.Item1` will be sent back via CommandsFromPLC chanenel.

When the task is to be run in asynchronous mode, the acknowledge string which stored in `reVal.Item2` will be sent back via ResultsToPLC channel.

Results To PLC

ResultsToPLC is also an automatically generated TCP/IP server device. It is responsible for sending acknowledge string and result string back to external device when the vision task is run in asynchronous mode.



Methods implemented by the ResultsToPLC device is called at two places in application:

1. OnDataReceived script of CommandsFromPLC device before command is run.

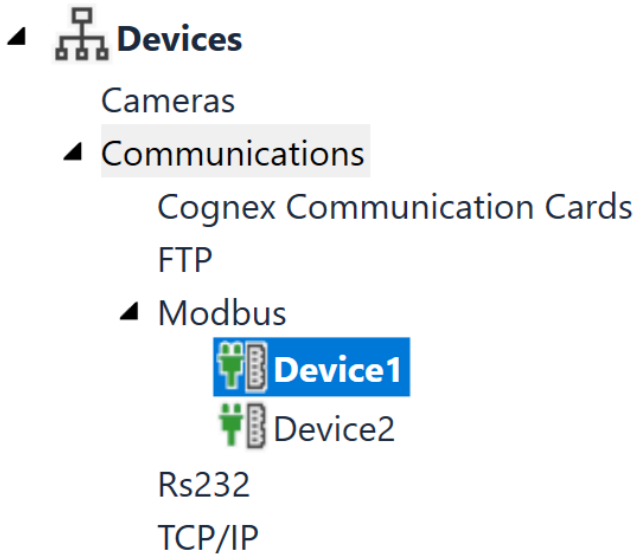
Acknowledge string will be send back to external device when asynchronous command is called.

2. AcknowledgeTaskResults callback script of TaskScheduler after the task execution is completed.

Result string will be sent back to external device when asynchronous command is called.

Customized Communication Devices

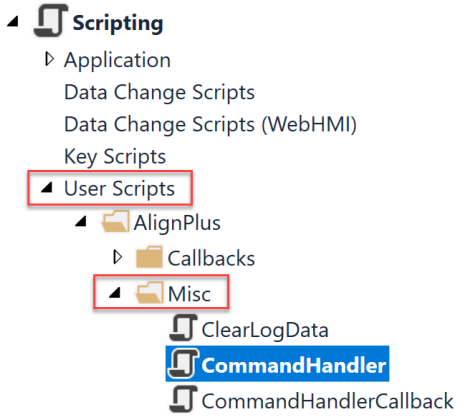
Custom communication devices can be employed using the scripting functionality available in Designer, or by creating a Designer plugins that generates Designer components. Here is an example of a ModBus device implemented as a Designer device:



For more information about designer plugin development, please refer to **Cognex Designer User Manual \ How To... \ How To... Plugins**.

CommandHandler

CommandHandler is an automatically generated script that can be found in the folder Scripting\User Scripts\Misc. It is the command call center in program. For almost all applications, it is expected that this file would not modified, It has been documented for completeness.



CommandHandler performs the following functions:

1. Check the command format and send back a suitable acknowledgement string to either CommandsFromPLC or ResultsToPLC device.

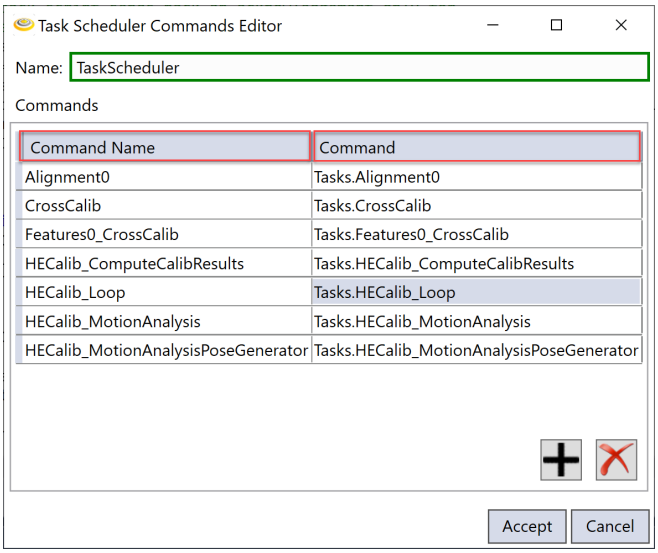
The CommandHandler script attempts to decode the command. If the format of the command is incorrect, it returns a suitable error code back to the PLC.

2. When the format is correct, it generates a suitably initialized CommandArgs object and send it to Task Scheduler.

In CommandHandler, information like task's StepID, task name, the behavior of task(Acquire, Process or both), as well as other information such as position index, x, y, theta of stage are extracted and decoded from command string and stored in one or more CommandArgs object(s) as a list. The list is sent to Task Scheduler that executes the appropriate task.

Task Scheduler

Task Scheduler is a Designer Component which is automatically generated by configuration wizard. This component maintains a list of task names and the corresponding system paths.



Task Scheduler is responsible for passing all the relevant data to a task and execute it. It is also responsible for gathering the results generated by the tasks.

Task Scheduler implements a set of methods. These methods allow the execution of tasks in at least three different ways:

- Synchronously

  When a task is executed synchronously, the task scheduler method which executes the task does not return until the task has finished execution. Typically the task executes in the same thread as the one that initiated the execution.

- Asynchronously

  When a task is executed asynchronously, the task scheduler starts the execution of the task in a different thread and returns. The scheduler does not wait for the execution of the task to complete.

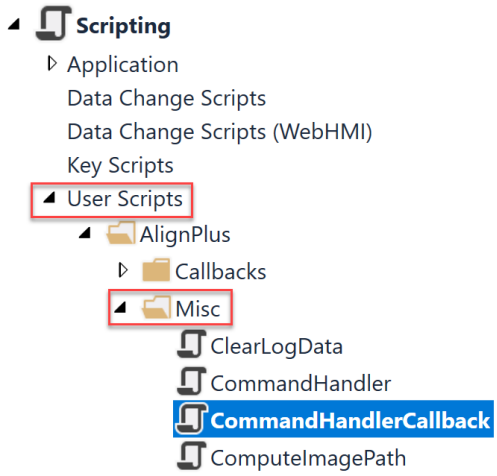
- Sequentially

  When a task scheduler runs sequentially, it runs more than one task.

After the requested task or tasks finish running, Task Scheduler's callback function AcknowledgeTaskResults will be triggered, wherein CommandHandlerCallback script will be called. The callback is called even if the task execution was unsuccessful.

CommandHandlerCallback

CommandHandlerCallback script can be found in the folder Scripting\User Scripts\Misc.



It is responsible for the following actions:

1. Converting results into a result string.

   For example, if a command to align parts is sent, this script extracts result data such as target x, y, theta for motion device and converts them to a result string.

   Some of the data that the result string can contain are:

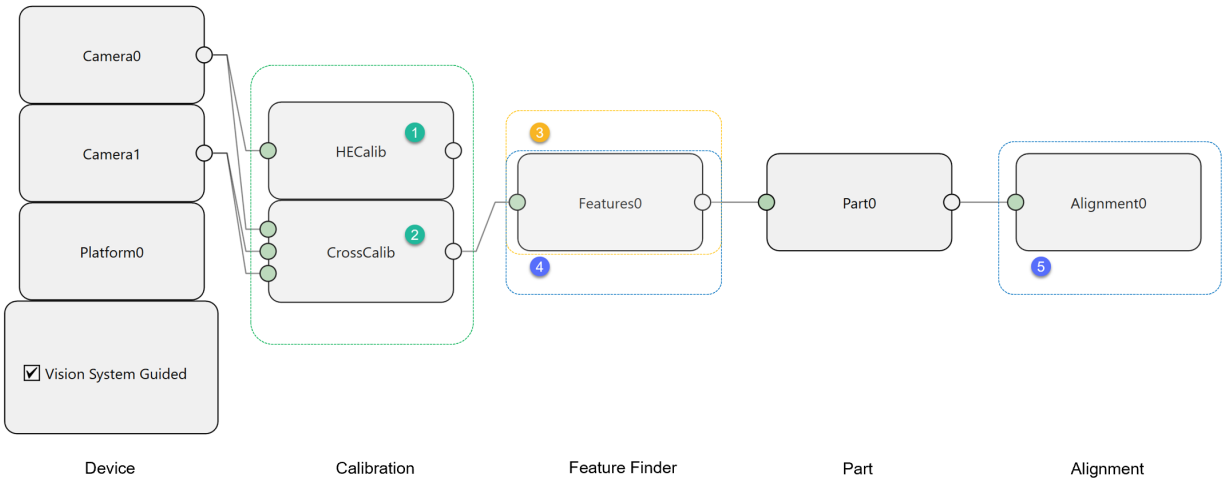
Key	Type	Description
X	Object	X value of current position of motion device, only used for vision-guided hand-eye calibration or ultracalibration
Y	Object	Y value of current position of motion device, only used for vision-guided hand-eye calibration or ultracalibration
ThetaInDegrees	Object	Theta value of current position of motion device, only used for vision-guided hand-eye calibration or ultracalibration
AbsoluteStageMotionX	Object	X value of target position of motion device should move to
AbsoluteStageMotionY	Object	Y value of target position of motion device should move to
AbsoluteStageMotionThetaDegrees	Object	Theta value of target position of motion device should move to
RelativeStageMotionX	Object	X value of relative position of motion device should move
RelativeStageMotionY	Object	Y value of relative position of motion device should move
RelativeStageMotionThetaDegrees	Object	Theta value of relative position of motion device should move
IsOK	Object	Specifies whether alignment result is within limit
PoseValid	Object	Whether current position is valid or not, used only for vision guided hand-eye calibration

2. Send the result string either to CommandsFromPLC or ResultsToPLC depending on whether the task run in synchronous mode or not.

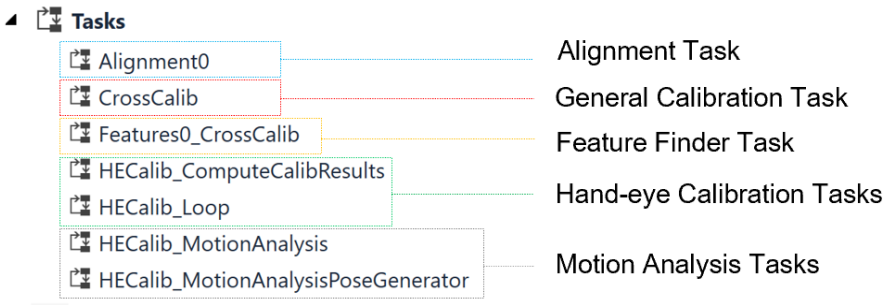
Tasks

AlignPlus tasks are automatically generated by configuration wizard based on configurations of calibration, feature finder, and alignment components. Here is an example of configuration wizard and its generated tasks.

- configuration wizard Configuration



- Tasks generated by configuration above



Each component in configuration wizard will result in the generation of one or more tasks after configuration wizard finishes running. The names of those tasks have certain relationships with their corresponding component names.

Component	Generated Task	Task Example	Task Function	Task Trigger Mode
hand-eye calibration	<Calibration Name>_Loop	HECalib_Loop	The purpose of this task is to generate various motion device poses for the purpose of hand-eye calibration, and at each pose to call the <Calibration Name>_ ComputeCalibResults task. This task is only generated when **Vision System Guided** option is checked on in the stage device in configuration wizard	Manually or by external command
	<Calibration Name>_ ComptueCalibResults	HECalib_ ComptueCalibResults	Move the motion device if necessary, capture images, extract features and accumulate features of calibration target at each stage pose and finally run calibration calculation.	Manually, by external command, or by hand-eye calibration loop task.

Component	Generated Task	Task Example	Task Function	Task Trigger Mode
General Calibration(Manual Calibration, Checkerboard Calibration , or Cross Calibration)	<Calibration Name>	CrossCalib	Capture images at each acquisition position, and then extract features and run calibration.	Manually, by external command
Finder	<Finder Name>_<Connected Calibration Name>	Features0_CrossCalib	This task captures all the images needed to find the features on a part and locates all the features by running feature finders. This task can run during train time or run time.	Manually, by external command
Alignment	<Alignment Name>	Alignment0	Compute the alignment pose for the motion device to achieve the desired assembly characteristics. This task is executed after the tasks that locate the features on the part are executed.	Manually, by external command
Hand-eye Calibration	<Calibration Name>_ MotionAnalysisPoseGenerator	HECalib_ MotionAnalysisPoseGenerator	Motion analysis tasks are used to conduct various tests of the motion device. The purpose of this task is to generate the various motion device poses for the purpose of testing, and at each pose to call the <Calibration Name>_ MotionAnalysis task.	Manually
	<Calibration Name>_ MotionAnalysis	HECalib_MotionAnalysis	This task moves the motion devices, acquires the images, extracts and accumulates features and stage poses, and computes the results using the accumulated data.	Called by Motion Analysis Pose Generator task

Note: When an application that controls the motion device during calibration is generated by selecting the Vision System Guided option, during hand-eye calibration, the PLC has to send commands that execute the tasks that perform "looping". This task calls the <Calibration Name>_ComputeCalibResults task. If the Vision System Guided option is not selected, the PLC has to send commands that execute the <Calibration Name>_ComputeCalibResults task. This condition also applies to tasks that perform motion analysis.

Task Execution Mode

The various vision tasks in the application serve various purposes. Vision tasks can be generated to perform hand-eye calibration, cross-calibration, extract features on parts, to compute alignment parameters, etc. Each task can be executed to perform specific type of actions. Tasks can be executed in various modes. For example, in application that employ a single camera to capture two images of non-overlapping image regions on a single part, the task can be executed to acquire images of a part at a first position, and then it can be executed to acquire images of the part at a second position and to use both the images to locate features. Largely each vision task can be executed in three modes.

- Acquire Only
- Acquire and Process
- Process Image.

The execution mode is defined in the command string that external device sends to the vision system.

StepID

StepIDs are automatically defined when the application is generated by the configuration wizard. They can be found at the top of CommandHandler script.

```
/* Communication Info Begin
StepID = 0                    HECalib_Loop
StepID = 1                    HECalib_ComputeCalibResults, Camera position: 0
StepID = 2                    CrossCalib, Camera position: 0
StepID = 3                    CrossCalib, Camera position: 1
StepID = 4                    CrossCalib, Camera position: 2
StepID = 5                    Features0, Calibration: CrossCalib, camera position: 1
StepID = 6                    Features0, Calibration: CrossCalib, camera position: 2
StepID = 7                    Alignment0
StepID = 8                    HECalib_MotionAnalysis
StepID = 9                    HECalib_MotionAnalysisPoseGenerator, Camera position: 0
   Communication Info End */
```

For sample application above, the automatically generated StepIDs are as follow:

- Hand-eye calibration loop task's step ID is 0
- Hand-eye calibration ComputeCalibResult task's step ID is 1
- Cross calibration task has three StepIDs since it has three different image acquisition positions. Accordingly, external devices should call this task three times with different StepIDs and different execution modes to finish cross calibration process.
- Features0 has two StepIDs since it acquires images at two different positions. It should be called two times with different StepIDs during train-time and run-time.
- Alignment0 has one StepID, 7
- Motion analysis tasks have stepIDs 8 and 9.

EncodedID

EncodedID is part of command string that external device sends to the vision system to run a task. It contains StepID of requested task, as well as the task execution mode:

1. Acquire Only

In this mode, EncodedID = StepID + 1000.

2. Acquire and Process

In this mode, EncodedID = StepID

3. Process Only

In this mode, EncodedID = StepID + 2000.

This table summarizes the various vision tasks that can be performed by the sample application:

Category	Task	StepID	Action	EncodedID
Calibration	HECalib_ Loop	0	Only one command with **Process Image** action to trigger hand-eye calibration loop, then the loop will take control the rest of process	2000
	CrossCalib	2	First command to **Acquire Image** at position 0	1002
		3	Second command to **Acquire Image** at position 1	1003
		4	Third command to **Acquire Image and Process** at position 2	4
Train Time	Features0_ CrossCalib	5	First command to **Acquire Image** at position 0	1005
		6	Second command to **Acquire Image and Process** at position 1	6
Run Time	Features0_ CrossCalib	5	First command to **Acquire Image** at position 0	1005
		6	Second command to **Acquire Image and Process** at position 1	6
	Alignment0	7	One command to **Process Image**	2007

Command String

Command String is a string which external devices send to vision application requesting one task or multiple tasks be executed, it is composed of three parts: Command Key, EncodedID and optional parameters.

Command Key

Command key indicates the type of action that a vision task has to perform. For example, the hand-eye calibration task can be started, continued or completed by sending a suitable command key. Similarly a task that finds the features on a part can be executed to perform train-time or run-time actions through a suitable command key.

Here are the most frequently used command keys in AlignPlus:

CommandKey	Description	When to use
ACB	Auto Calibration Begins	Vision Guided Hand-eye Calibration
AC	Auto Calibration	
HEB	Hand-eye Calibration Begins	Motion Guided Hand-eye Calibration
HE	Hand-eye Calibration	
HEE	Hand-eye Calibration Ends	
IC	Intrinsic Calibration	Checkerboard Calibration, Cross Calibration or Manual Calibration
TA	Train Alignment. Registers the golden pose of the target.	Train Time Commands
TT	Train VGR step (register target, single or multiply shots)	
TTR	Train VGR step (register robot pick/place position for a target)	
LF	Locate Features	Feature Finding in run time.
GP	Get Pose	Pose Computing
LFA	Locate Features Asynchronously	Feature Finding in run time in asynchronous mode
GPA	Get Pose Asynchronously	Pose Computing in asynchronous mode
LFGP	First run Locate Feature, second run Get Pose	Feature Finding and Pose Computer in run time
MEA	Measure	Run inspection
MEAA	Measure Asynchronously	Run inspection asynchronously
LFMEA	Locate Features, then Measure	Run feature finding first, then run inspection

CommandKey	Description	When to use
MGP	Multi-part Get Pose	Multi-part pose computation
MGPA	Multi-part Get Pose Asynchronously	Run multi-part pose computation
LFMGP	Locate Featues, then run Multi-part Get Pose	Run feature finding first, then run multi-part pose computation synchronously

Parameters

Some commands require additional parameters from the PLC. For example, applications such as LF, or GP require the current pose of the alignment device to execute correctly. These additional parameters are encoded in the command string.

CommandArgs

When the application receives a command string, an object of type CommandArgs is created in the CommandHandler script. This object contains all the data needed to execute the task correctly. Some of the data that is encoded includes the task execution mode, current stage position received in the command string etc. Additional information can be added to the object using the CommandArgs.SetCustomArgs method. Every task has a Command subtask which is used to receive CommandArgs settings from Task Scheduler.

The table below shows main properties of the CommandArgs class and their equivalent with Command String.

Name	Description	Command String
CommandFromPLC	Command received from PLC/Robot	Command String itself
StepID	StepID associated with the command	StepID decoded from EncodedID
CommandName	The name of requested task	Decoded from StepID
PositionIndex	Acquisition position index. This parameter will be used when a camera or set of cameras, captures images of a part at different physical locations	Decoded from StepID
ExecutionMode	• Acquire: Perform acquisition only • AcquireAndProcessImage: Acquire and process image • ProcessImage: Do not acquire and process only	Decoded from EncodedID
AlignmentOperation	Specify if the pose computer is receiving training data or data to be used for alignment, or if the feature finder is extracting training data or run-time data for alignment • Train: Pose computer is receiving training data, or feature finder is extracting training data • Align: Pose computer is receiving data for aligning parts	Decoded based on CommandKey: TT or TA command are for Train, LF, LFA or LFGP commands are for Align

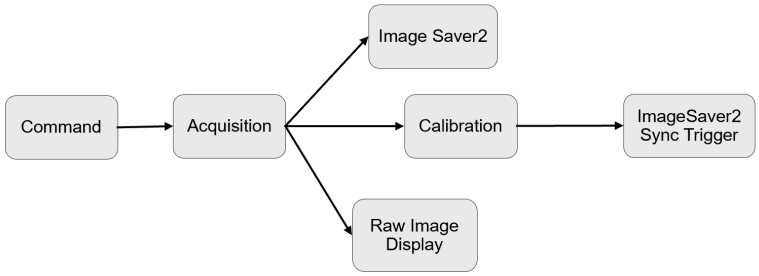
Name	Description	Command String
HECalibrationOperation	Specify hand-eye calibration operations • ClearAndAccumulateFeatures: Accumulate features after clearing accumulator • AccumulateFeatures: Accumulate features • ComputeResults: Compute hand-eye calibration results • AccumulateFeaturesAndComputeResults: Compute hand-eye calibration results	Decoded based on CommandKey: HEB command will set operation as ClearAndAccumulateFeatures, HE command will set as AccumulateFeatures; HEE command will set as ComputeResults. When Vision Guided mode is used for hand-eye calibration, those parameters will be set by hand-eye calibration loop task automatically.
ErrorCode	Error code associated with the command	ErrorCode is generated by the vision system, and then stored here and at last forwarded to external device when there is any error happens.
SerialNumber	Serial number of part	Equals to PartID specified in LF, LFA, GP, GPA, LFGP, MEA, MEAA, LFMEA, MGP, MGPA or LFMGP command. For other commands, it's an empty string.
OtherArgs	Other Info, like current x, y, theta value of motion or any other user defined content set using the method CommandArgs.SetOtherArgs	Saves current pose of motion device that specified in parameters of command string
Token	A unique identifier for the command argument	An ID generated by vision system for any input Command String

Task Workflow

Hand-eye Calibration Task

Hand-eye Calibration ComputeCalibResults task extracts and accumulates features of the calibration target at each stage pose, and finally computes hand-eye calibration results. The task can be executed by three types of command keys: HEB command to initialize the hand-eye calibration, HE command at each stage pose to extract and accumulate features and poses, HEE command to compute the calibration results using the accumulated data. Henceforth, we refer hand-eye calibration ComputeCalibResults task as hand-eye calibration task.

The hand-eye calibration task has 6 subtasks: Command, Acquisition, Image Saver2, Raw Image Display, Calibration, and ImageSaver2 Sync Trigger.



Command

When a command string is received by the CommandHandler script, an equivalent CommandArgs object is created. This object contains all the data that is needed to execute the target task. The task scheduler passes the CommandArgs object to

the target task and the object is made available to the task by the Command subtask. The Command subtask makes the object available in its output pin, and also publishes it as a tag. Various components within the task either use the tag or the output available at the pin to act appropriately. For example, a downstream subtask may use the tag to implement a Condition expression that conditionally executes the task. Or, the customer can intercept the object that is available at the output of the subtask to add any additional data that are required by their customizations.

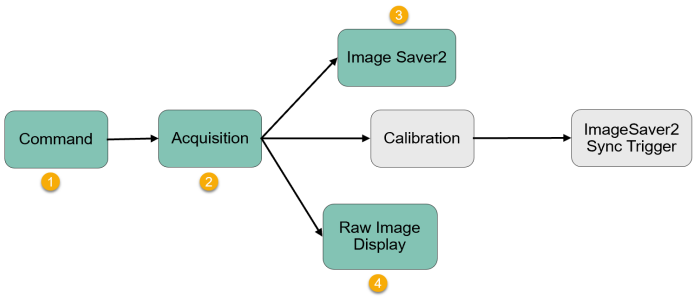
Execution Mode

The hand-eye calibration task uses three execution modes: Acquire, AcquireAndProcessImage, and ProcessImage.

- ExecutionMode: **Acquire**

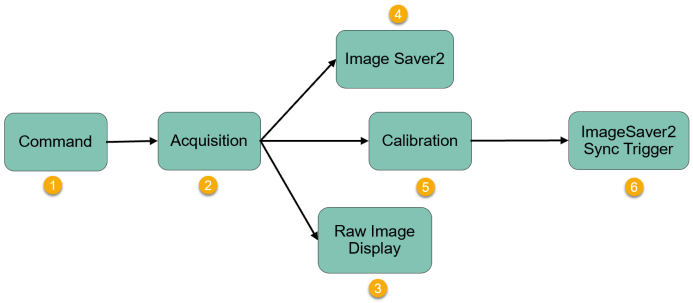
  AlignPlus supports applications that use a single camera to acquire multiple images of various non-overlapping regions of a part, also known as shuttling camera applications. During the hand-eye calibration process, for every pose of the alignment motion device, the task acquires the images at all positions before locating the calibration target features, The command string that executes the task should have the execution mode field set to Acquire in the encodedID when only acquiring images at the various positions. For applications that do not use shuttling cameras, the *ExecutionMode* is never set to *Acquire*.only.

  When ExecutionMode is Acquire, the task will only runs Acquisition subtask, Raw Image Display subtask, and Image Saver2 task.



- ExecutionMode: **AcquireAndProcessImage**

  When cameras are at last acquisition position for a given stage pose during calibration, the execution mode in the encodedID in the command string has to be set to AcquireAndProcessImage. When the task is executed, the image at the last position is captured and calibration target features are located in all images captured at all camera positions. In this case, *ExecutionMode* should be set as *AcquireAndProcessImage*. The workflow below shows all its sub tasks and the execution sequence.

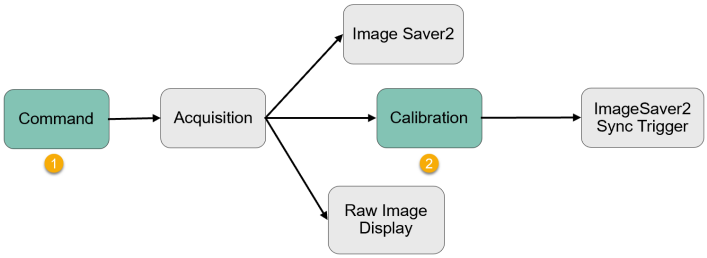


- ExecutionMode: **ProcessImage**

  At the beginning of hand-eye calibration process, the accumulated features and stage poses inside calibration subtask has to be cleared. The task has to be executed by creating a command string with command key set to HEB and the *ExecutionMode* in the encodedID set to *ProcessImage*.

  When all the features need to calibrate have been accumulated, the task is executed using a command string with command key set to HE and execution mode in the encodedID set to ProcessImage. This results in the computation of the hand-eye calibratino results.

  The work flow below shows all its sub tasks and their execution sequence.



Acquisition

Running condition:*[ExecutionMode = Acquire || AcquireAndProcessImage]*

Acquisition subtask triggers cameras to acquire images of the calibration target at a position encoded in the encodedID in the command string that executes the task.

For a shuttling camera application that uses two cameras that shuttle to two positions, the hand-eye calibration task is executed twice. The first using a command string with an encodedID that encodes the first position and with execution mode set to Acquire. The second using a command string with an encodedID that encodes the second position and with execution mode set to AcquireAndProcessImage. Both the executions execute the Acquisition subtask.

Command	ExecutionMode	Position Index	Image Item	Before Acquisition	Acquisition at Pos0	Acquisition at Pos1
1	Acquire	0	Cam0Pos0	Null		/
		0	Cam1Pos0	Null		/
2	AcquireAndProcessImage	1	Cam0Pos1	Null	Null	
		1	Cam1Pos1	Null	Null	

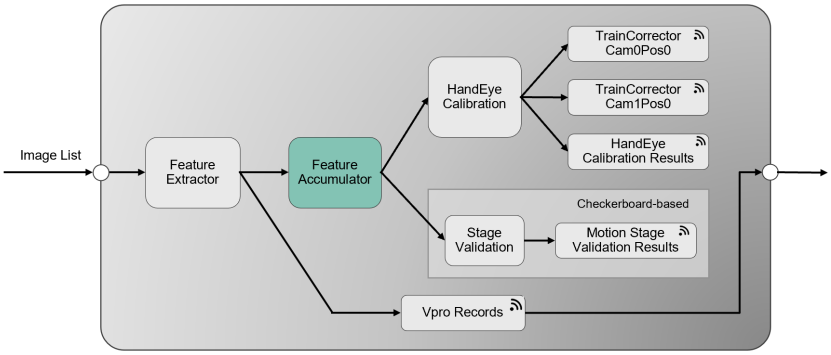
For non-shuttling camera applications, a single command string with an encodedID that encodes the execution mode to AcquireAndProcessImage is used to acquire the image.

Command	ExecutionMode	Position Index	Image Item	Before Acquisition	Acquisition at Pos0
1	AcquireAndProcessImage	0	Cam0Pos0	Null	
		0	Cam1Pos0	Null	

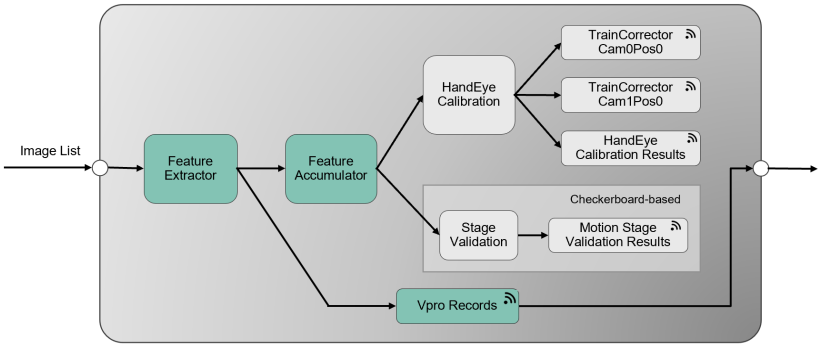
Calibration

Running condition:*[ExecutionMode = ProcessImage || AcquireAndProcessImage]*

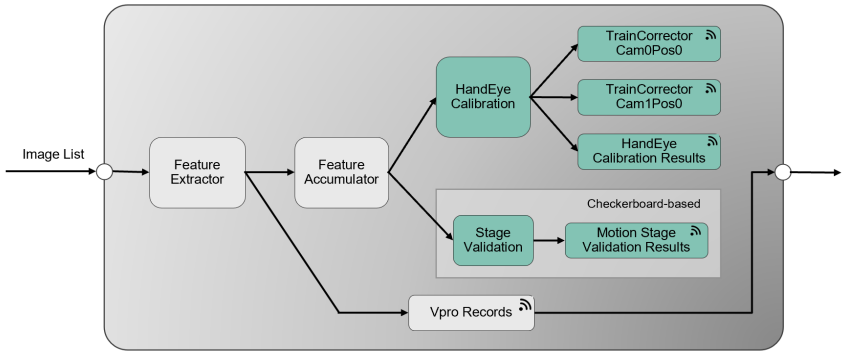
The calibration subtask is executed during various phases of the hand-eye calibration process. When the task is executed because of a command string with command key set to HEB, the accumulators that gather calibration target features and stage poses are cleared and readied to perform a new hand-eye calibration.



IDuring the feature accumulation phase, the task is executed by a command string with command key set to HE. During this phase the features on the calibration target are extracted and accumulated in the Calibration subtask. Image and graphic data that are computed during feature extraction and published by the VPro Records publisher. Image display HMI subscribe the published data and provide a visual feedback of the calibration process to the user.



After all the required features are extracted, the task is executed by a command string with the command key set to HEE. During this phase, the accumulated features are used to compute and publish the hand-eye calibration results in the Calibration subtask. Image correctors that compensate for imaging system non-linearities to generate a distortion free image are trained. Data that provides feedback about the quality of the motion device is publishedAfter all the required features are extracted, the task is executed by a command string with the command key set to HEE. During this phase, the accumulated features are used to compute and publish the hand-eye calibration results in the Calibration subtask. Image correctors that compensate for imaging system non-linearities to generate a distortion free image are trained. Data that provides feedback about the quality of the motion device is published.



Subtasks for Saving Images

The hand-eye calibration task saves all acquired images in TIF format along with the command data that executed the task. These images can be used to generate hand-eye calibration results in an off-line mode. It also supports screenshots saving in JPG files for user to manually review feature extraction results. There are two subtasks that are used to save images.

Image Saver2

Running condition: Always run.

Image Saver2 subtask accumulates raw images and their corresponding commands every time when hand-eye calibration task acquires images. The accumulated images and commands would get cleared when Image Saver2 Sync Trigger subtask is called at the end of the task.

Image Saver2 Sync Trigger

Running condition:[ExecutionMode = ProcessImage || AcquireAndProcessImage]

Image Saver2 Sync Trigger subtask saves all accumulated raw images and their commands from Image Saver2 subtask into TIF files if current task's raw image saving function is enabled. It also saves image records into JPG files when screenshot saving function is enabled. See how to enable raw image saving and screenshot saving in Image Saving on page 176Image Saving on page 176

Raw Image Display

Running condition:[ExecutionMode = ProcessImage || AcquireAndProcessImage]

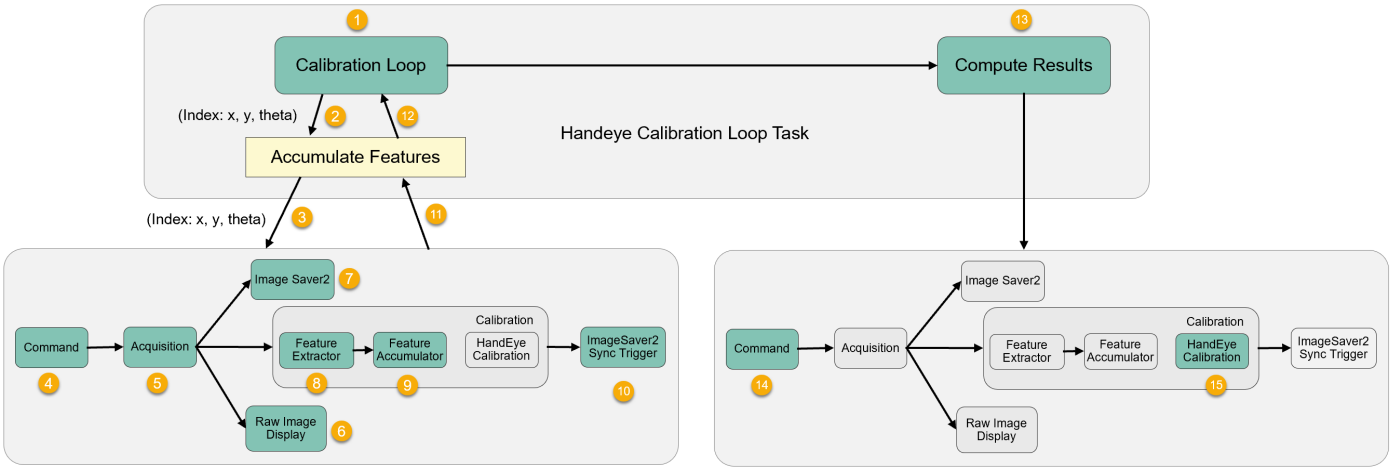
Raw Image Display subtask publishes raw images for image display. The published images are the raw images captured by the cameras and do not include any graphics. Those records will be shown on task's image display page if "Display Raw Image" is selected in functional panel below navigation tree. See more information at **Show Graphics** in Multiple Display on page 62.

Hand-eye Calibration Loop Task

The purpose of the hand-eye calibration loop task is to move a motion device to various poses and at each pose to execute the hand-eye calibration task to extract, accumulate and finally compute the hand-eye calibration results. The task is executed when the application receives a command string with command key HEB or ACB, with a suitably EncodeID.

Calibration Loop

The hand-eye calibration loop task has only one subtask called CalibrationLoop. The subtask has two blocks of significance. The first block, the CalibrationLoopBlock, is responsible for initializing the hand-eye calibration process and for locating and accumulating all calibration features and stage poses. A user sets the properties for the block that include the range of motion for each degree of freedom of the motion device, and parameters that control how this range is sampled. The block first initializes the hand-eye calibration task by executing it with the HEB command. It extracts and accumulates calibration target features by executing the hand-eye calibration task with the HE command. Once all the features are accumulated, the second block in the CalibrationLoop subtask, called the Compute Results block, computes the hand-eye calibration results by executing the hand-eye calibration task with the HEE command.

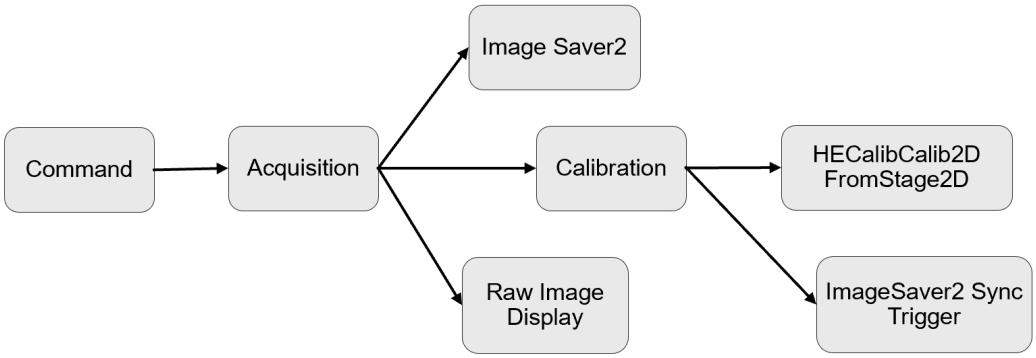


Stage Move

The stage is moved before any image is acquired during the hand-eye calibration process. A script <StageName>Move is called by the Acquisition subtask in the hand-eye calibration task before the images are acquired. See more information about this in How to Command Stage to Move.

Cross Calibration Task

Cross calibration task uses the result of hand-eye calibration that has been performed on one of the substations to compute the relationship between Raw2D and Home2D for the cameras at the other substations. The task has 7 subtasks: Command, Acquisition, Calibration, HECalibCalib2DFromStage2D, Image Saver2, ImageSaver2 Sync Trigger, and Raw Image Display. To trigger a cross calibration task, you can use IC command.



Command

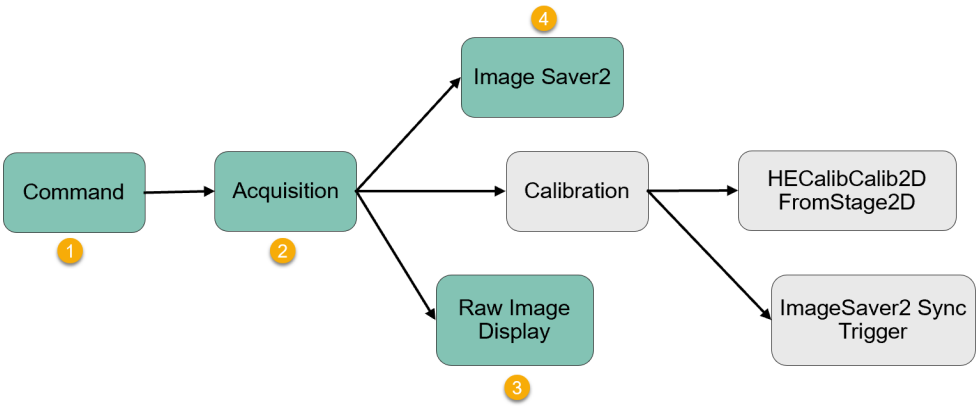
When a command string is received by the CommandHandler script, an equivalent CommandArgs object is created. This object contains all the data that is needed to execute the target task. Command subtask is the interface to receive CommandArgs object from task scheduler which makes the object available to current task as an output pin and a public tag.

There are two properties in CommandArgs object that control the work flow of cross calibration task: **Execution Mode** and **PositionIndex**. ExecutionMode decides whether current task should acquire image and/or process image. PositionIndex specifies which position cameras should acquire images at.

Execution Mode

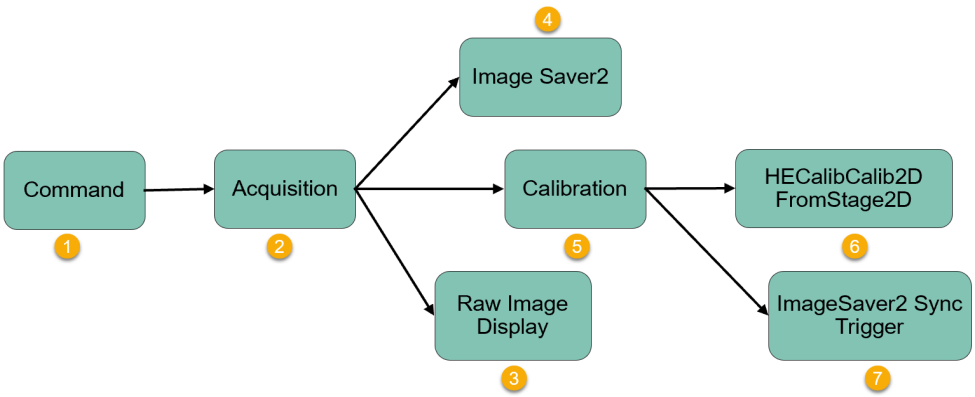
- ExecutionMode: **Acquire**

  When cameras are not at their last acquisition position, cross calibration task shall only acquire images when it's called. Therefore, the *ExecutionMode* should be set as *Acquire* in this case, and accordingly, the task will only run Acquisition, Image Saver2 and Raw Image Display subtasks to acquire, accumulate, and display raw images at current position.



- ExecutionMode: **AcquireAndProcessImage**

  When cameras are at their last acquisition position, cross calibration task should acquire images and run calibration when it's called. *ExecutionMode* here should be set as *AcquireAndProcessImage* so that the task will run Acquisition, Calibration, and HECalib Calib2D From Stage2D subtasks sequentially to finish image acquisition, run checkerboard calibration, and map Home2D from hand-eye calibrated station to Raw2D for cameras at current station. It also save images to image files if image saving function is enabled and generates raw image records in case user wants to show raw images on image display. The work flow below shows all its sub tasks and their execution sequence.



Acquisition

Running condition:*[ExecutionMode = Acquire || AcquireAndProcessImage]*

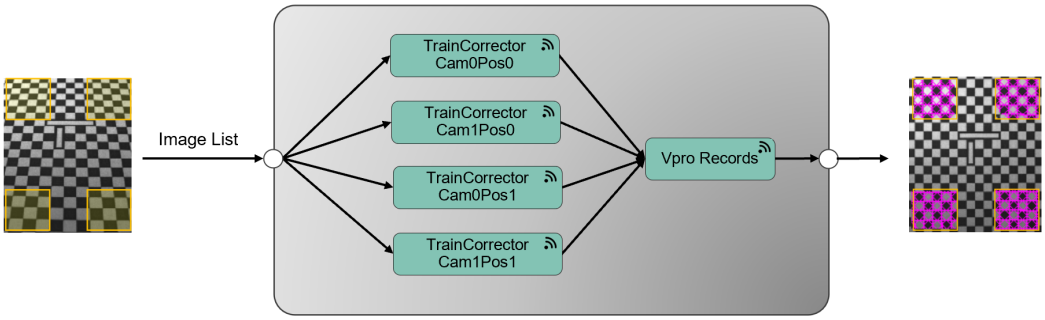
For a cross-calibration task where cameras at a second substation are cross-calibrated to a set of hand-eye calibrated cameras at the first station the cross-calibration task is executed twice. The first using a command string with an encodedID that encodes the one of the positions and with execution mode set to Acquire. The second using a command string with an encodedID that encodes the other position and with execution mode set to AcquireAndProcessImage. Both the executions execute the Acquisition subtask.

Command	ExecutionMode	Position Index	Image Item	Before Acquisition	Acquisition at Pos0	Acquisition at Pos1
1	Acquire	0	Cam0Pos0	Null		/
		0	Cam1Pos0	Null		/
2	AcquireAndProcessImage	1	Cam0Pos1	Null	Null	
		1	Cam1Pos1	Null	Null	

Calibration

Running condition:*[ExecutionMode = ProcessImage || AcquireAndProcessImage]*

Calibration subtask is only available in checkerboard-based or hybrid cross calibration, it runs checkerboard calibration for each input raw images, and publish their train correctors for later use in feature finding task's image correction subtask. The output is a dictionary of VisionPro records with corrected checkerboard images and their extracted features.



HECalib Calib2D FromStage2D

Running condition: *[ExecutionMode = ProcessImage || AcquireAndProcessImage]*

This subtask establishes transforms between Home2D from hand-eye calibrated stations with Raw2D from cameras in current station. These transforms will be used for run time feature finding.

Image Saving

Cross calibration task can save all raw images in TIF files for user to playback cross calibration process. It also supports screenshots saving in JPG files for user to manually review feature extraction results.

Image Saver2

Running condition: Always run.

Image Saver2 subtask accumulates raw images and their corresponding commands every time when current task acquires images. The accumulates images and commands would get cleared when Image Saver2 Sync Trigger subtask is called at the end of the task.

Image Saver2 Sync Trigger

Running condition:[ExecutionMode = ProcessImage || AcquireAndProcessImage]

Image Saver2 Sync Trigger subtask saves all accumulated raw images and their commands from Image Saver2 subtask into TIF files if current task's raw image saving function is enabled. It also saves image records into JPG files when screenshot saving function is enabled. See how to enable raw image saving and screenshot saving in Image Saving on page 176

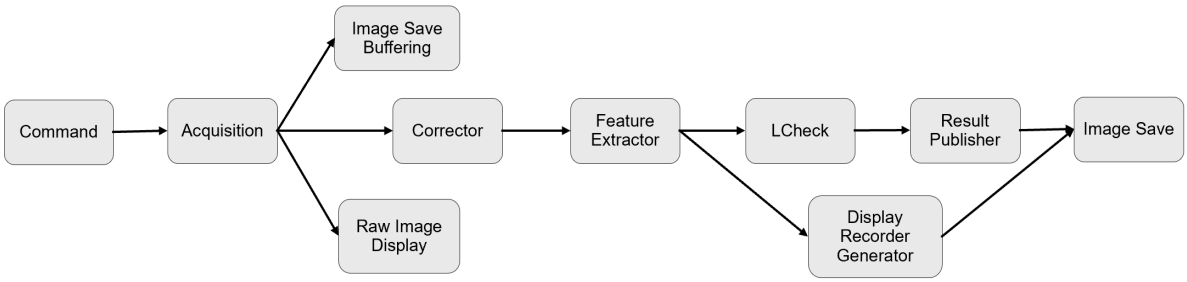
Raw Image Display

Running condition:[ExecutionMode = ProcessImage || AcquireAndProcessImage]

Raw Image Display subtask generates raw image records(images without graphics) for image display. Those records will be shown on task's image display page if "Display Raw Image" is selected in functional panel below navigation tree. See more information at **Show Graphics** in Multiple Display on page 62.

Feature Finding Task

The Feature Finding task acquires images of a part and locates the features on the part. It has the subtasks shown in the image below. This task could be executed during train-time or run-time. During train-time the task could be executed to locate part feature locations at train-time and/or set train-time gripper positions. During run-time the task is executed to locate run-time part feature locations. Some of the command keys that can be encoded in the command string are TA/TT/TTA commands, used during train-time, or LF/LFA used during run-time.



The following are the commands that the user can send to the feature finder task for an application that acquires images of a single part at two positions. The application is one where the gripper of the robot is to be aligned to the part:

Command Index	Command Key	Position Index	Description
1	TA	0	Acquire images at position 0 at train-time
2	TA	1	Acquire image and process image at position 1 at train-time
3	TTR	1	Save gripper's train time pose
4	LF/LFA	0	Acquire images at position 0 at run-time
5	LF/LFA	1	Acquire image and process image at position 1 at run-time

Command

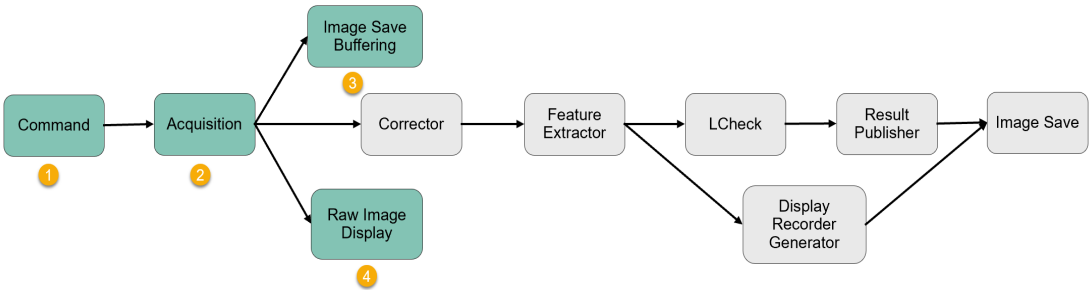
When a command string is received by the CommandHandler script, an equivalent CommandArgs object is created. This object contains all the data that is needed to execute the target task. Command subtask is the interface to receive CommandArgs object from task scheduler which makes the object available to current task as an output pin and a public tag.

There are four properties in the CommandArgs object that controls the work flow of feature finding task: **ExecutionMode**, **PositionIndex**, **AlignmentOperation** and **SavePoseDuringTraining**. ExecutionMode decides which sub tasks should got executed. PositionIndex specifies which position cameras should acquire images at. AlignmentOperation specifies whether it's train time or run time fro feature finding. SavePoseDuringTraining indicates whether motion device's train time pose should be save or not.

Execution Mode

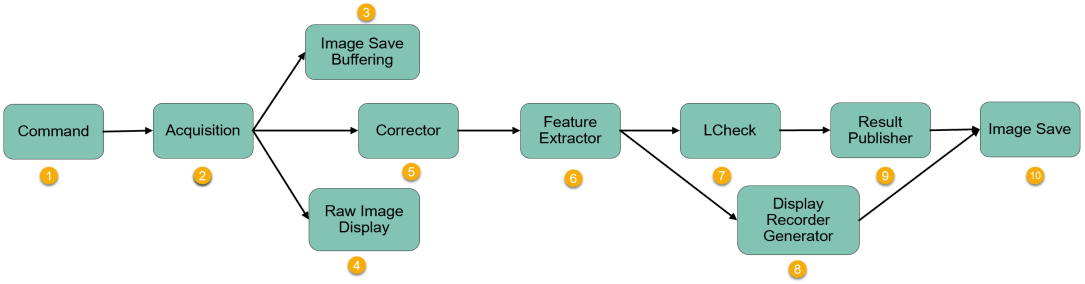
- ExecutionMode: **Acquire**

  When cameras are not at their last acquisition position, feature finding task shall only acquire images when it's called. Therefore, the *ExecutionMode* should be set as *Acquire* in this case, and accordingly, the task will only run Acquisition, Image Saver2 and Raw Image Display subtasks to acquire, accumulate, and display raw images at current position.



- ExecutionMode: **AcquireAndProcessImage**

  When cameras are at their last acquisition position, feature finding task should acquire images and run feature finding when it's called. In this case, *ExecutionMode* here should be set as *AcquireAndProcessImage*, and the task will run Acquisition, Corrector, Feature Extractor, L-Check, Result Publisher subtasks sequentially to finish image acquisition, image correction, feature extraction, feature length check and publish results. It also save images to image files if image saving function is enabled and generates raw image records in case user wants to show raw images on image display. The work flow below shows all its subtasks and their execution sequence.

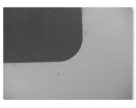


Acquisition

Running condition: *[ExecutionMode = Acquire || AcquireAndProcessImage]*

Acquisition subtask acquires images of the part at the position index encoded in the encodedID.

For a shuttling camera application that uses two cameras that shuttle to two positions, the feature finder task is executed twice. The first using a command string with an encodedID that encodes the first position and with execution mode set to Acquire. The second using a command string with an encodedID that encodes the second position and with execution mode set to AcquireAndProcessImage. Both the executions execute the Acquisition subtask.

Command	ExecutionMode	Position Index	Image Item	Before Acquisition	Acquisition at Pos0	Acquisition at Pos1
1	Acquire	0	Cam0Pos0	Null		/
		0	Cam1Pos0	Null		/

Command	ExecutionMode	Position Index	Image Item	Before Acquisition	Acquisition at Pos0	Acquisition at Pos1
2	AcquireAndProcessImage	1	Cam0Pos1	Null	Null	
		1	Cam1Pos1	Null	Null	

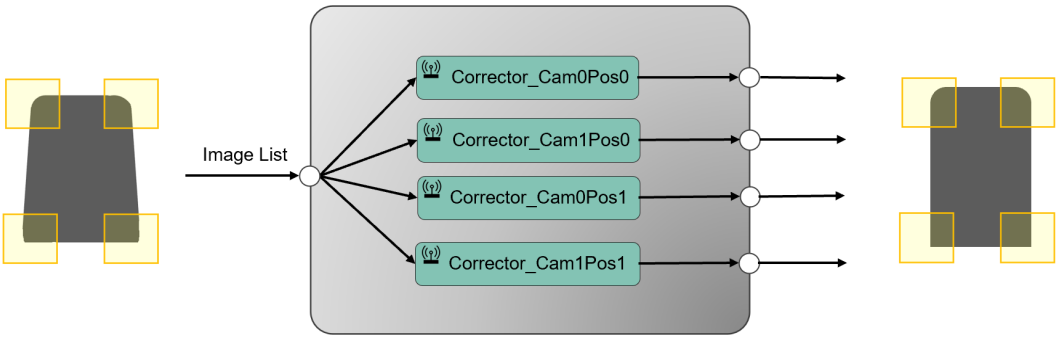
For non-shuttling camera applications, a single command string with an encodedID that encodes the execution mode to AcquireAndProcessImage is used to acquire the image.

Command	ExecutionMode	Position Index	Image Item	Before Acquisition	Acquisition at Pos0
1	AcquireAndProcessImage	0	Cam0Pos0	Null	
		0	Cam1Pos0	Null	

Corrector

Running condition: *[ExecutionMode = ProcessImage || AcquireAndProcessImage]*

The corrector subtask uses image correctors for removing non-linear imaging distortion. All Cognex vision tools assume an affine relationship between the image and the world. By removing non-linear distortion image correctors ensure the vision tools work accurately. The output of this task is a list of corrected images.



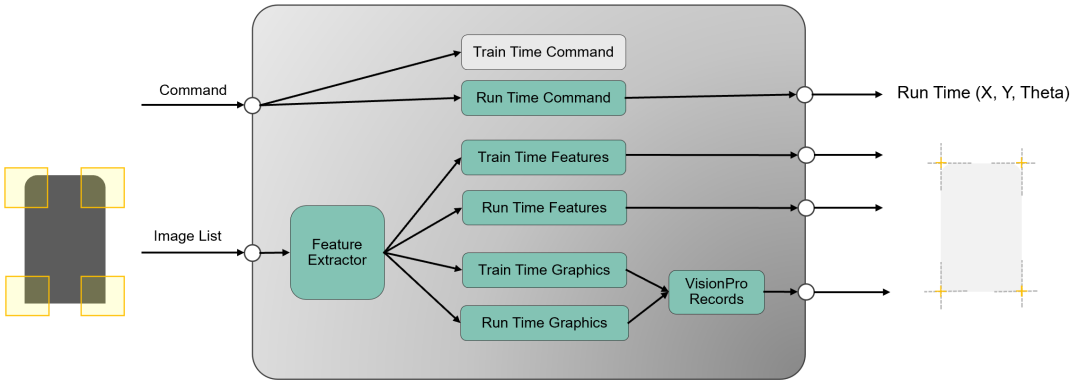
Feature Extractor

Running condition: *[ExecutionMode = ProcessImage || AcquireAndProcessImage]*

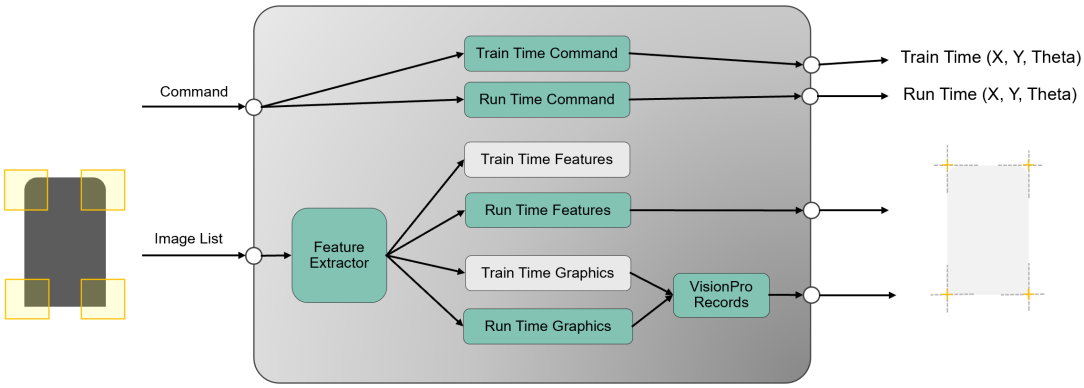
Feature extractor sub task extract features from corrected images using feature finders set by user in Alignment setup interface for locating train-time and run-time features. The output of this sub task are train time and/or run time command, train time and/or run time features, and their graphics.

- Train Time

  At train time, when TA/TT command string is sent to vision system, CommandHandler will set *AlignmentOperation* of current command as *Train*. Within Feature Extractor subtask, it will first run feature extraction, then update train time features in its publisher. It will also update train time graphics.

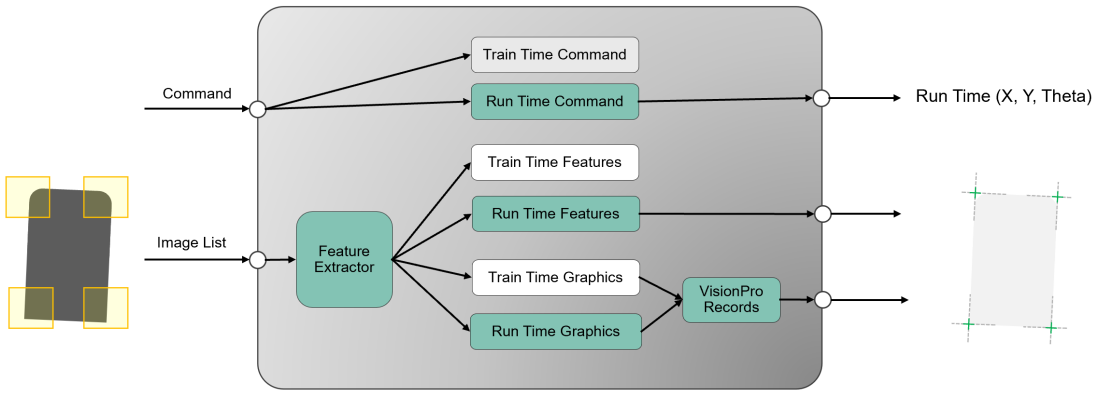


When TTR command string is sent to vision system, CommandHandler will set as *True* , and the Feature Extractor subtask will save train time command(motion device's train time X, Y, Theta are included) into its publisher.



- Run Time

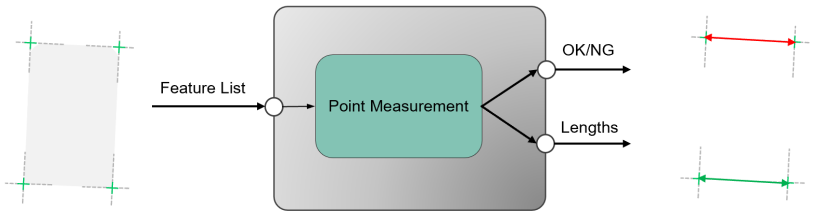
  At train time, when LF/LFA command string is sent to vision system, CommandHandler will set *AlignmentOperation* of current command as *Align*. Within Feature Extractor subtask, it will first run feature extraction, then update run time command(motion device's current X, Y, Theta are included) and run time features so that they will be save in their publishers. It will also update run time features to VproRecords so that corresponding image display will be updated.



L-Check

Running condition: *[ExecutionMode = ProcessImage || AcquireAndProcessImage]*

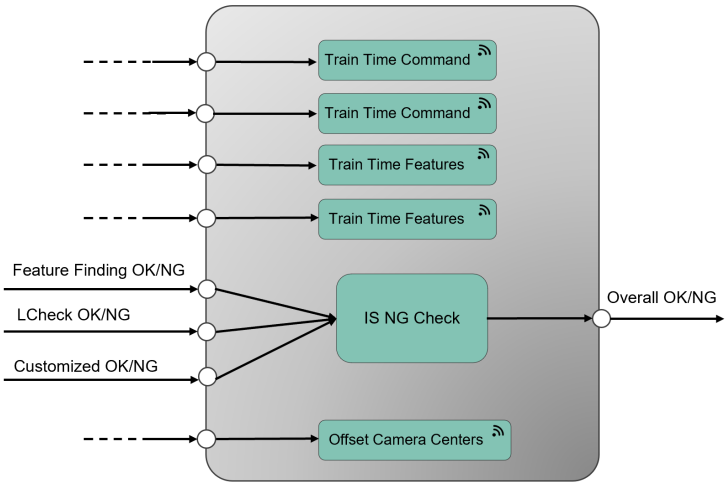
L-Check subtask checks if the distance between pairs of features are within specification. The outputs of this sub task are measured lengths and if the lenghts meet the specification.



Result Publisher

Running condition: *[ExecutionMode = ProcessImage || AcquireAndProcessImage]*

The Result Publisher subtask publishes all train-time, offset camera centers, and run-time feature locations for other tasks in the applications to consume. It also publishes the CommandArgs object that was received at train-time. Other tasks consume the object to get the train-time gripper position. Result publisher subtask also outputs overall OK/NG signal based on feature finding OK/NG signal, L-Check OK/NG signal and customized OK/NG signal. The customized OK/NG signal is from on open input pin, it will be ignored for overall OK/NG judgement if it's not implemented.



Display Recorder Generator

Running condition: *[ExecutionMode = ProcessImage || AcquireAndProcessImage]*

Display Recorder Generator sub task merges VisionPro records from feature extractor subtask with customized VisionPro records for image displays.

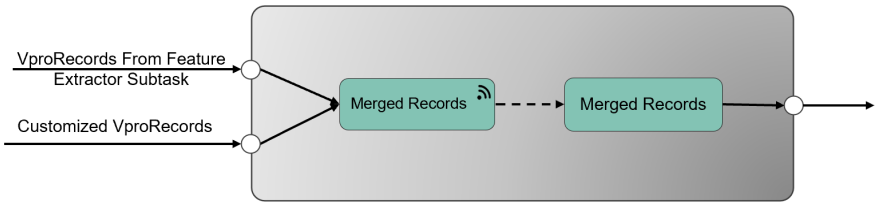


Image Saving

Feature finding task can save all raw images in TIF files for user to playback feature finding process. It also supports screenshots saving in JPG files for user to manually review feature extraction results.

Image Saver2

Running condition: Always run.

Image Saver2 subtask accumulates raw images and their corresponding commands every time when current task acquires images. The accumulates images and commands would get cleared when Image Saver2 Sync Trigger subtask is called at the end of the task.

Image Saver2 Sync Trigger

Running condition:[ExecutionMode = ProcessImage || AcquireAndProcessImage]

Image Saver2 Sync Trigger subtask saves all accumulated raw images and their commands from Image Saver2 subtask into TIF files if current task's raw image saving function is enabled. It also saves image records into JPG files when screenshot saving function is enabled. See how to enable raw image saving and screenshot saving in Image Saving on page 176

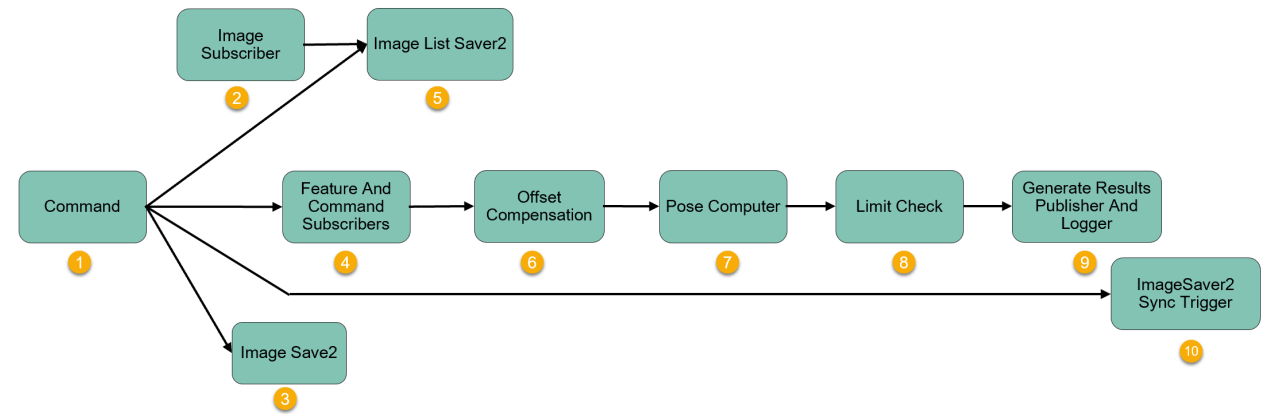
Raw Image Display

Running condition:[ExecutionMode = ProcessImage || AcquireAndProcessImage]

Raw Image Display subtask generates raw image records(images without graphics) for image display. Those records will be shown on task's image display page if "Display Raw Image" is selected in functional panel below navigation tree. See more information at **Show Graphics** in Multiple Display on page 62.

Alignment Task

Alignment task calculates how much motion device should move to bring the parts to target positions. It also has other functions such as allowing user to add offsets, checking whether alignment data is with spec and saving all images from connected feature finders to image files. Alignment task is triggered by GP/GPA command, or LFGP command if user want to run alignment right after run time feature finding.



Command

When a command string is received by the CommandHandler script, an equivalent CommandArgs object is created. This object contains all the data that is needed to execute the target task. Command subtask is the interface to receive CommandArgs object from task scheduler which makes the object available to current task as an output pin and a public tag.

There are two properties in the CommandArgs object that controls the workflow of alignment task: **ExecutionMode** and **AlignmentOperation**. In alignment task, *ExecutionMode* should always be set as *ProcessImage* and *AlignmentOperation* is always set as *Align*.

Feature And Command Subscribers

Running condition: Always run.

This subtask subscribes train time and run time features and their commands from corresponding feature finders' tasks. For assembly application when assembly is performed using paired-features mode, this subtask will subscribe two corresponding feature finder's run time features and their commands. For assembly application where assembly parameters are computed using golden pose mode, the two corresponding feature finders' both train time and run time features as well as their commands will be subscribed. For alignment applications, the subtask will subscribe to only one corresponding feature finder's train time and run time features and their commands, it will also subscribe that feature finder's offset camera

centers, and use either the train time features or offset camera centers as output trained features depending on user's choice. See more information about offset camera centers at Camera Center Parameters on page 1.

Offset Compensation

Running condition: Always run.

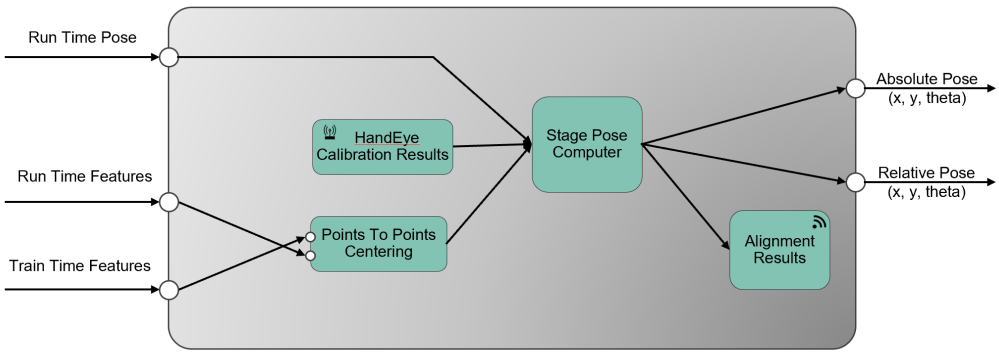
Offset compensation allows user to manually input offsets to run time features to compensate the final alignment or assembly result based on feedback from inspection results. See more information at Offset Compensation on page 180.

Pose Computer

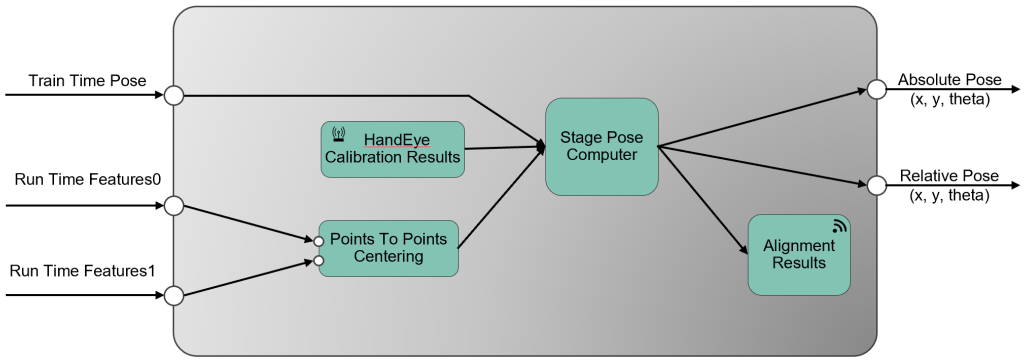
Running condition: *[ExecutionMode = ProcessImage]*

Pose computer subtask computes transforms between target position and current position using Points To Points Centering Block on page 240, and then use Stage Pose Computer on page 241 to calculate the desired stage pose based on hand-eye calibration results, stage's current pose or trained pose as well as the transform computed by Points To Points Centering Block. The output of this block are absolute target pose and the relative change in stage pose.

Here is an example of Pose Computer subtask for Align To Base application where a transform from run time features to train time features are computed first, and then forwarded to stage pose computer that computes target pose.



Here is an example for Assembly Guided Pick application using paired features mode.



Limit Check

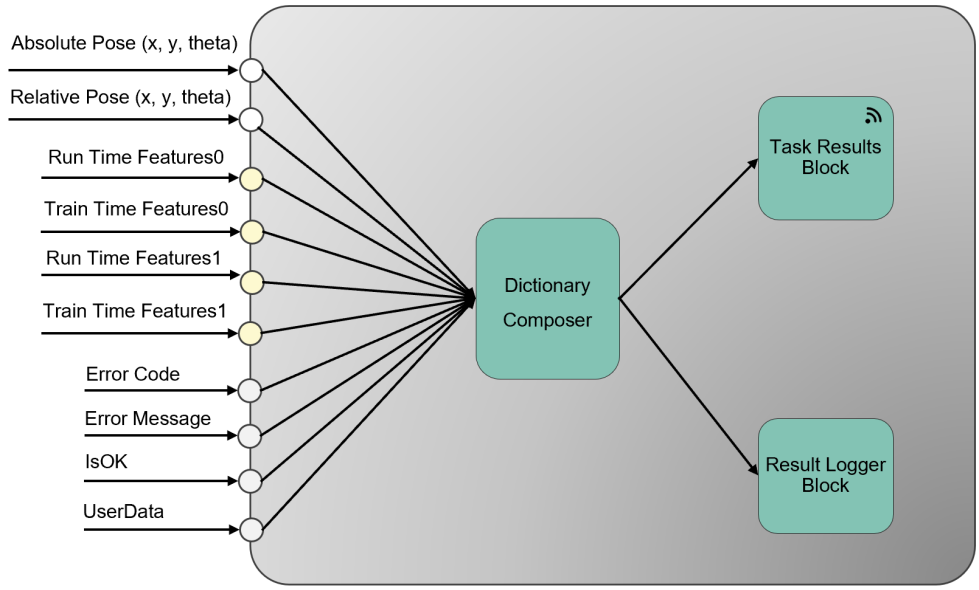
Running condition: Always run.

Limit check use pose computer's output x, y, theta as input, to check if they're within certain alignment limit specs defined on by user. See more information at Placement Limit Checker on page 185.

Generate Results Publisher And Logger

Running condition: Always run.

This subtask saves all alignment or assembly result data into a dictionary and publishes the dictionary data in Task Results Block. It also saves all the result data using the Result Logger Block into an alignment log file whose stored directory is specified by user in Logging Setup on page 172.



The result data includes: connected feature finder's train time(if golden pose is used) and run time features information, relative and absolute pose that motion device should move to, Error Code, Error Message, IsOK from LCheck, and user data from user's customization.

Subtasks for Image Saving

When alignment task's image saving function is enabled, it save the following images to corresponding sub folders:

1. Save each feature finder's run time raw images and their commands into TIF files, save those images with their graphics into JPG files under that feature finder's sub folder;

2. Save an 1x1 dummy TIF file with alignment command into alignment sub folder.

Here is an example of auto generated sub folders for alignment image saving:

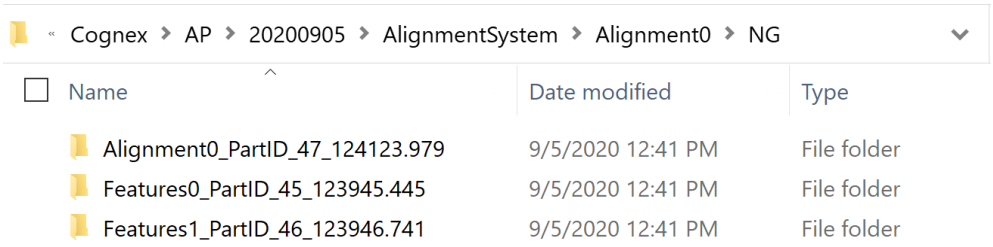


Image Subscriber

Running condition: Always run.

This subtask subscribes all images lists and their commands from connected feature finders. Those images later will can be used for image saving or image display.

Image List Saver2

Running condition: Always run.

Image List Saver2 subtask split image lists based on different feature finders so that afterward images from different feature finders will be saved into different sub folders.

Image Saver2

Running condition: Always run.

Image Saver2 subtask combines alignment command with an 1x1 image which would be saved into a TIF file in Image Saver2 Sync Trigger subtask.

Image Saver2 Sync Trigger

Running condition: Always run.

Image Saver2 Sync Trigger subtask saves all images and their commands saved in Image List Saver2 subtask(s) and Image Saver2 subtask into TIF files if current feature finder's raw image saving function is enabled. It also saves image records in Image List Saver2 subtasks(s) into JPG images when screenshot is enabled. See how to enable raw image saving and screenshot saving in Image Saving on page 176

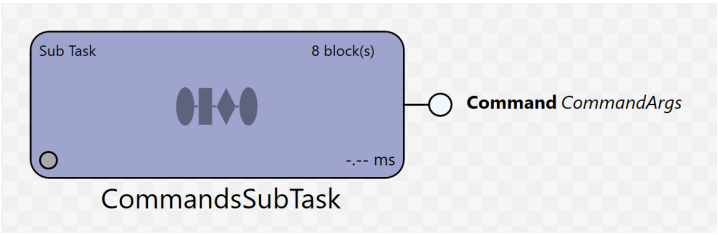
Subtasks

Common Sub Tasks

The following subtasks are common subtasks that are used in many different tasks.

Commands Subtask

When a command string is received by the CommandHandler script, an equivalent CommandArgs object is created. This object contains all the data that is needed to execute the target task. The task scheduler passes the CommandArgs object to the target task and the object is made available to the task by the Command subtask. The Command subtask makes the object available in its output pin, and also publishes it as a tag. Various components within the task either use the tag or the output available at the pin to act appropriately. For example, a downstream subtask may use the tag to implement a Condition expression that conditionally executes the task. Or, the customer can intercept the object that is available at the output of the subtask to add any additional data that are required by their customizations.

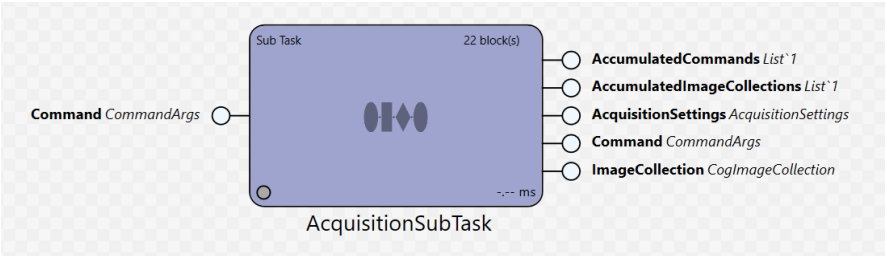


The output of this subtask is the received command.

Output	Type	Description
Command	CommandArgs	Received command from CommandHandler for current task

Acquisition Subtask

Acquisition subtask acquires images in calibration tasks and feature finding tasks at given acquisition position encoded in the EncodedID of a command string and consequently in the CommandArgs object that is output by the Command subtask.



The input is current command which comes from Command subtask. Acquisition subtask only acquires images at given position index specified by input command.

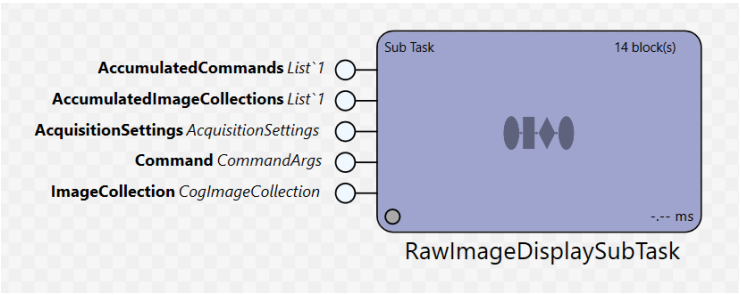
Input	Type	Description
Command	CommandArgs	Current input command of the task

The outputs are acquisition settings of current task, current command and image collection, as well as accumulated commands and images collections at all already acquired positions.

Output	Type	Description
AccumlatedCommands	List<CommandArgs>	Accumulated commands that have been sent to Acquisition subtask to acquire a whole set of images at all acquisition positions.
AccumulatedImageCollection	List<CogImageCollection>	Accumulated images acquired at different acquisition positions.
AcquisitionSettings	AcquisitionSettings	The acquisition settings for current task, which includes camera indexes, camera enable/disable status, exposure time and position indexes.
Command	CommandArgs	Current command of the task, it's the same with the input Command.
ImageCollection	CogImageCollection	A collection of images acquired at current position index.

Raw Image Display Subtask

Raw Image Display subtask creates raw image records(images captured by the cameras, without graphics) that will be shown on feature finder's display if user choose to display raw images.

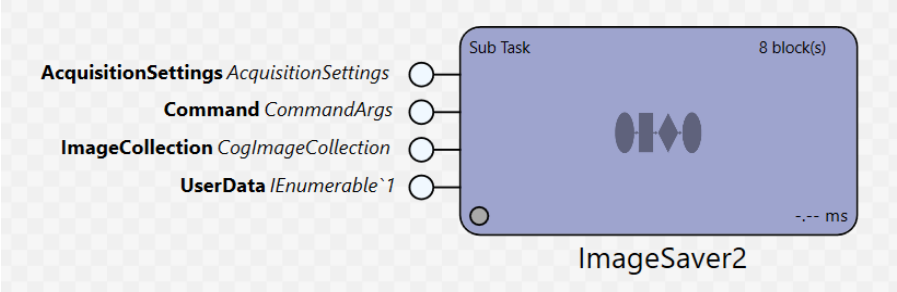


Its inputs are the same with Acquisition subtask's outputs.

Input	Type	Description
AccumlatedCommands	List<CommandArgs>	Accumulated commands that have been sent to Acquisition subtask to acquire a whole set of images at all acquisition positions.
AccumulatedImageCollection	List<CogImageCollection>	Accumulated images acquired at different acquisition positions.
AcquisitionSettings	AcquisitionSettings	The acquisition settings for current task, which includes camera indexes, camera enable/disable status, exposure time and position indexes.
Command	CommandArgs	Current command of the task, it's the same with the input Command.
ImageCollection	CogImageCollection	A collection of images acquired at current position index.

ImageSaver2 Subtask

Image Saver2 subtask accumulates raw images and their corresponding commands every time when images arrive. The accumulated data would get cleared when a subsequent Image Saver2 Sync Trigger subtask executes.

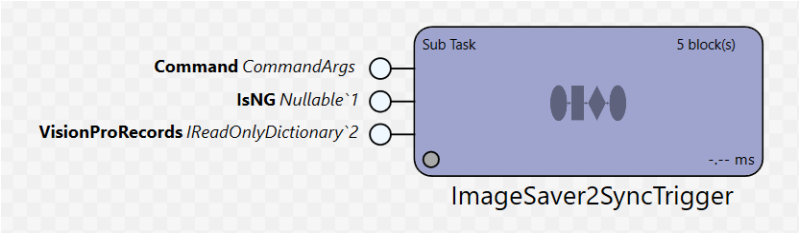


Its inputs include command, image collection and user data at current acquisition position, and acquisition settings of current task.

Input	Type	Description
AcquisitionSettings	AcquisitonSettings	The acquisition settings for current task, which includes camera indexes, camera enable/disable status, exposure time and position indexes.
Command	CommandArgs	Current command of the task
ImageCollection	CogImageCollection	A collect of images acquired at current position index.
UserData	IEnumerable<KeyValuePair<String,String>>	Metadata which will be saved into CDB files, not used in current version.

ImageSaver2 Sync Trigger Subtask

Image Saver2 Sync Trigger subtask saves all accumulated raw images and their commands, received by the Image Saver2 subtask, into TIF files if raw image saving function of current task is enabled. It also saves image records into JPG images when screenshot saving is enabled.



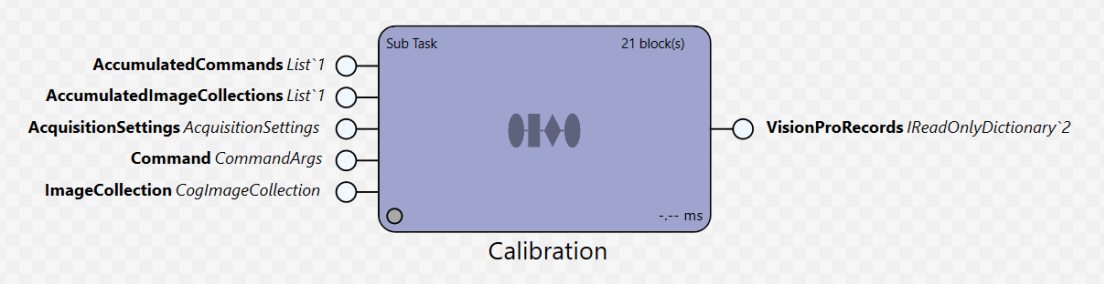
Its inputs include current command, overall OK/NG judgement for current part, and VisionPro records for each camera at each acquisition position.

Input	Type	Description
Command	CommandArgs	Current command of the task
IsNG	Nullable<Boolean>	Whether those images are NG images
VisionProRecords	IReadOnlyDictionary<String, VisionProRecord>	A dictionary of images and their graphics for each camera at each acquisition position

Calibration Sub Tasks

Calibration Subtask

Calibration subtask performs calibration related functions.



The inputs are acquisition settings of current task, current command and image collection, as well as accumulated commands and images collections at all already acquired positions.

Input	Type	Description
AccumlatedCommands	List<CommandArgs>	Accumulated commands that have been sent to Acquisition subtask to acquire a whole set of images at all acquisition positions.
AccumulatedImageCollection	List<CogImageCollection>	Accumulated images acquired at different acquisition positions.
AcquisitionSettings	AcquisitionSettings	The acquisition settings for current task, which includes camera indexes, camera enable/disable status, exposure time and position indexes.
Command	CommandArgs	Current command of the task, it's the same with the input Command.
ImageCollection	CogImageCollection	A collection of images acquired at current position index.

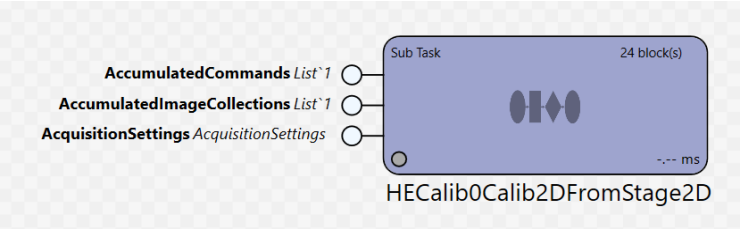
The output is a dictionary of VisionPro Records that could be used to shown on image display.

Output	Type	Description
VisionProRecords	IReadOnlyDictionary<String, vprorecord>	A dictionary of images and their graphics from current task

This subtask publishes the results of calibration that other tasks consume.

Calib2DFromStage2D Subtask

This subtask computes the Home2DFromRaw2D transform for the current station.

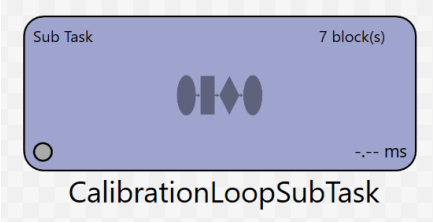


The inputs are acquisition settings of current task, accumulated commands, and images collections at all already acquired positions.

Input	Type	Description
AccumlatedCommands	List<CommandArgs>	Accumulated commands that have been sent to Acquisition subtask to acquire a whole set of images at all acquisition positions.
AccumulatedImageCollection	List<CogImageCollection>	Accumulated images acquired at different acquisition positions.
AcquisitionSettings	AcquisitionSettings	The acquisition settings for current task, which includes camera indexes, camera enable/disable status, exposure time and position indexes.

Calibration Loop Subtask

Calibration Loop subtask does not have any input or output pins. In applications where the vision system guides the motion device, it works with the hand-eye calibration task to move the motion device to various poses, gather calibration target data and compute calibration results.



Feature Finding Sub Tasks

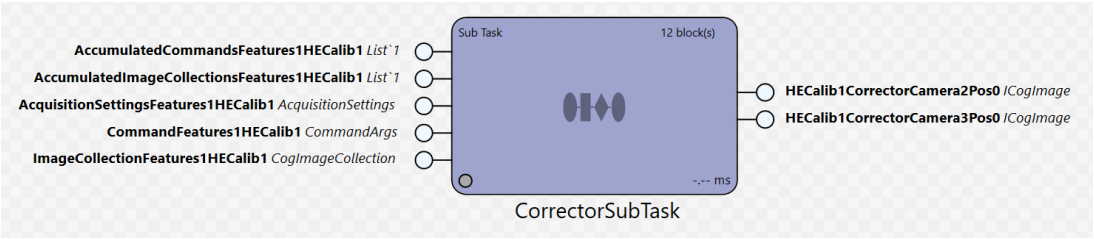
The following subtasks are used in feature finding tasks.

Corrector Subtask

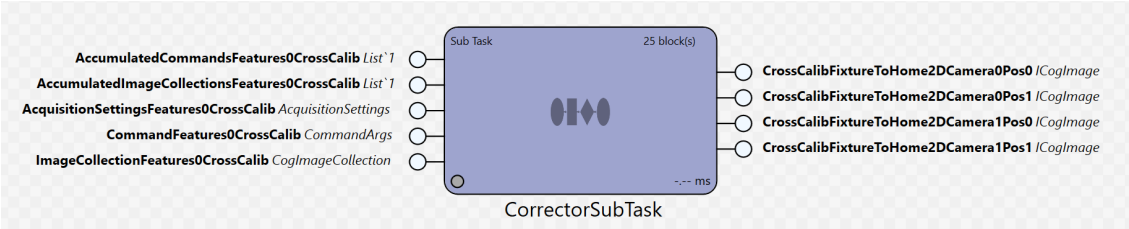
Images captured by the cameras can have a non-linear relationship between the image pixel locations and real world locations, due to non-linearities in the imaging system. The Corrector subtask generates images free from these non-linearities. The output images will be corrected images without lens distortion and perspective distortion.

The input and output pin names depend upon the finding task name. Output pin names depend upon the camera names and the acquisition positions.

- Hand-eye Calibration Corrector



- Cross Calibration Corrector



This subtask has five inputs generally named: Accumulated Commands, Accumulated Image Collection, Acquisition Settings, Command, and Image Collection.

Input	Type	Description
AccumulatedCommands	List<CommandArgs>	Accumulated commands that have been sent to Acquisition subtask to acquire a whole set of images at all acquisition positions.
AccumulatedImageCollection	List<CogImageCollection>	Accumulated images acquired at different acquisition positions.
AcquisitionSettings	AcquisitionSettings	The acquisition settings for current task, which includes camera indexes, camera enable/disable status, exposure time and position indexes.

Input	Type	Description
Command	CommandArgs	Current command of the mother task, it's the same with the input Command.
ImageCollection	CogImagesCollection	A list of images acquired at current position index.

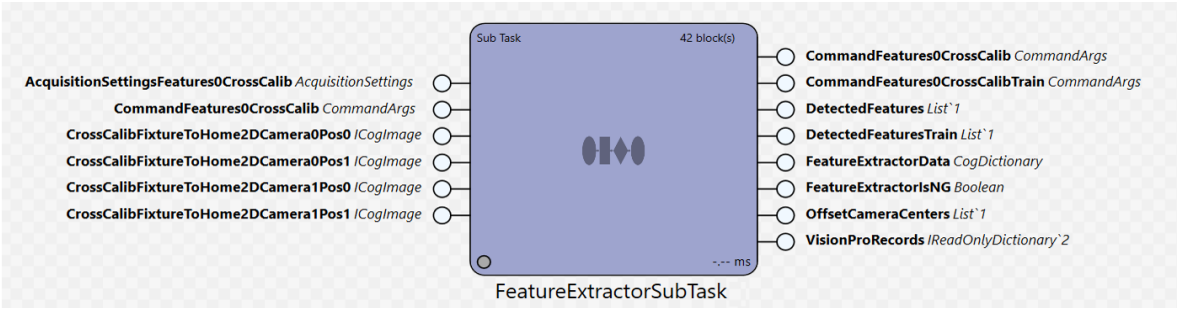
Outputs are corrected images for each camera's acquisition position.

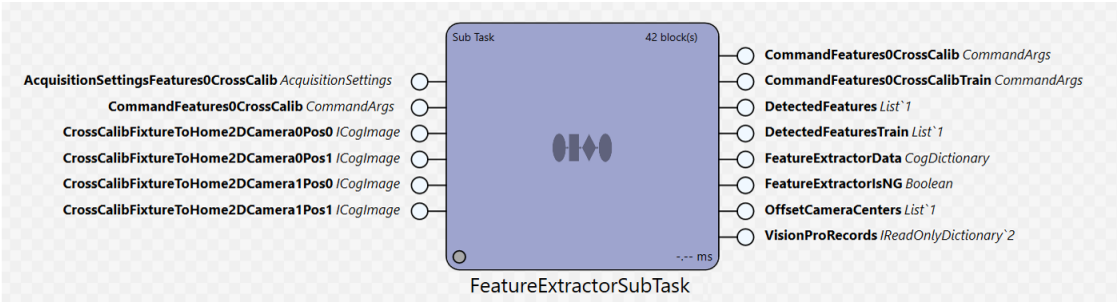
Output	Type	Description
Camera0Pos0	ICogImage	Corrected image of Camera0 at position0
Camera0Pos1	ICogImage	Corrected image of Camera0 at position1
Camera1Pos0	ICogImage	Corrected image of Camera1 at position0
Camera1Pos1	ICogImage	Corrected image of Camera1 at position1

Note: Input and output descriptor labels in the tables above are illustrative only, the actual input and output pin names will be based on project configuration.

Feature Extractor Subtask

Feature extractor subtask extracts features from corrected image. The feature type could be point, line, or generic features (see more information about generic feature in AlignPlus Concept\Feature Finding\Features) depending on which type is selected in configuration wizard. The input and output names are appended with feature finder name and/or its connected calibration name. For example, "AcquisitionSettingsFeatures0CrossCalib" means Acquisition Settings for "Features0" finder which connects "CrossCalib" calibration component.





The input image number would change according to the number of images a feature finding task acquires for feature extraction.

Input	Type	Description
AcquisitionSettings	AcquisitionSettings	The acquisition settings for current task, which includes camera indexes, camera enable/disable status, exposure time and position indexes.
Command	CommandArgs	Current command of the task, it is the same with the input pin.

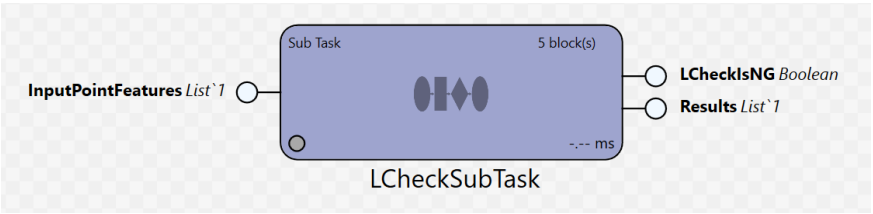
Input	Type	Description
Camera0Pos0	ICogImage	Corrected image of Camera0 at position0.
Camera0Pos1	ICogImage	Corrected image of Camera0 at position1.
Camera1Pos0	ICogImage	Corrected image of Camera1 at position0.
Camera1Pos1	ICogImage	Corrected image of Camera1 at position1.

The outputs include train time and run time features and their corresponding commands, feature extraction OK/NG signal, feature extraction data, and VisionPro Records.

Output	Type	Description
Command	CommandArgs	Current run time command, it is the same with input pin.
CommandTrain	CommandArgs	Command that is saved when stage pose is trained.
DetectedFeatures	List<CogAlpsPointFeature> / List<CogAlpsLineFeature> / List<CogAlpsGenericFeature>	Current run time feature list.
DetectedFeaturesTrain	List<CogAlpsPointFeature> / List<CogAlpsLineFeature> / List<CogAlpsGenericFeature>	Saved train time feature list.
FeatureExtractorData	CogDictionary	A dictionary which tracks the serial number of current part and whether its feature finding is OK or NG. It has two keys inside: "IsNG" and "TagFileName".
FeatureExtractorIsNG	Boolean	Whether current part's feature are all found and valid
OffsetCameraCenters	List<CogAlpsPointFeature>	A list of point features whose locations are provided by user's inputs on "Camera Center Parameters" page. This list of features will be used as trained golden pose for pose computer in alignment task if user choose "Align To Camera Center" on HMI
VisionProRecords	IReadOnlyDictionary<String, vprorecord>	A dictionary of VisionPro records which saves each camera acquisition position's image and graphics.

L-Check Subtask

L-Check subtask checks lengths between pairs of feature points. It only supports single part point features currently.



The input is a point feature list.

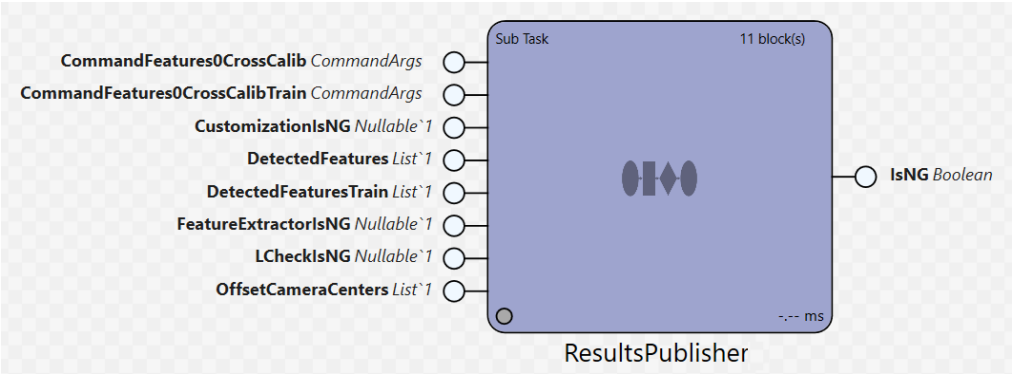
Input	Type	Description
InputPointFeatures	List<CogAlpsPointFeature>	A list of found point features

Outputs are each length and its OK/NG check result, and the overall L-Check result.

Output	Type	Description
L-CheckIsNG	Boolean	Whether input point features pass all length checks specified by user on user interface.
Results	List<Tuple<String, Double, Boolean>>	A dictionary of all length checks with length values and whether they pass specifications.

Results Publisher Subtask

Results publisher subtask publishes: run time and train time features, run time and train time commands. The published data is subscribed by other tasks in the application.



Input pins includes run time and train time features and their commands and three NG checks: Feature extraction OK/NG check, L-Check OK/NG check and customized OK/NG check. For the sake of generality, specific finder name and corresponding calibration name in input names are ignored.

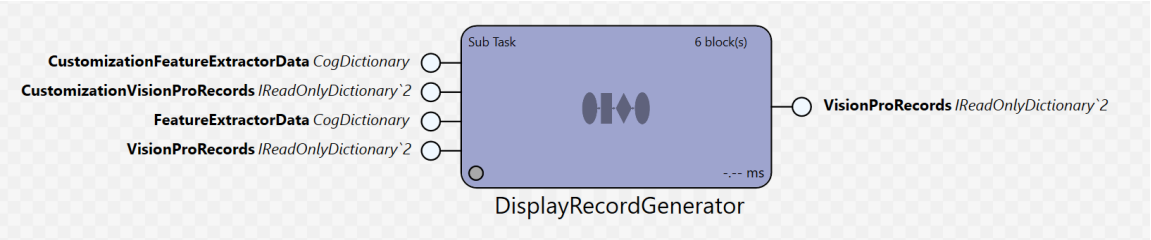
Input	Type	Description
Command	CommandArgs	Current command of the task
CommandTrain	CommandArgs	Save train time command
CustomizationIsNG	Nullable Boolean	Open pin for user to input customized OK/NG check
DetectedFeatures	List<CogAlpsPointFeature> / List<CogAlpsLineFeature> / List<CogAlpsGenericFeature>	Current run time feature list
DetectedFeaturesTrain	List<CogAlpsPointFeature> / List<CogAlpsLineFeature> / List<CogAlpsGenericFeature>	Train time feature list
FeatureExtractorIsNG	Nullable Boolean	Whether feature extraction is successful or not for current part
L-CheckIsNG	Nullable Boolean	Whether L-Check is passed or not for current part
OffsetCameraCenters	List<CogAlpsPointFeature>	A list of point features whose locations are provided by user's inputs on "Camera Center Parameters" page. This list of features will be used as trained golden pose for pose computer in alignment task if user choose "Align To Camera Center" on HMI

The output is the overall OK/NG judgement.

Output	Type	Description
IsNG	Boolean	Overall OK/NG check based on FeatureExtractorIsNG, L-CheckIsNG and CustomizationIsNG input pins

Display Record Generator Subtask

Display Recorder Generator subtask merges VisionPro records from feature extractor subtask with customized VisionPro records for image displays.



Input	Type	Description
CustomizationFeatureExtractorData	CogDictionary	Customized feature extraction data. It depends on user how to specify the content inside.
CustomizationVisionProRecords	IReadOnlyDictionary<String, vprorecord>	Customized VisionPro records
FeatureExtractorData	CogDictionary	A dictionary which tracks the serial number of current part and whether its feature finding is OK or NG. It has two keys inside: "IsNG" and "TagFileName".
VisionProRecords	IReadOnlyDictionary<String, vprorecord>	A dictionary of VisionPro records from feature extractor subtask which saves each camera acquisition position's image and graphics.

The output is a dictionary of merged VisionPro records for each camera at each acquisition position.

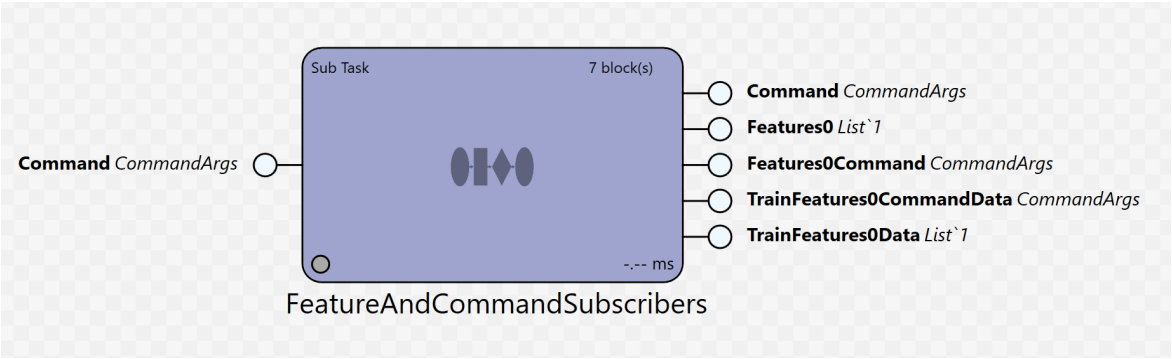
Input	Type	Description
VisionProRecords	IReadOnlyDictionary<String, vprorecord>	A dictionary of merged VisionPro records which saves each camera acquisition position's image and graphics.

Alignment Sub Tasks

The following subtasks are used in alignment task.

Feature And Command Subscribers Subtask

This subtask subscribes train time and run time features and their commands from feature finder tasks that compute features needed to align the part.



The input is current command of the task.

Input	Type	Description
Command	CommandArgs	Current command for alignment task

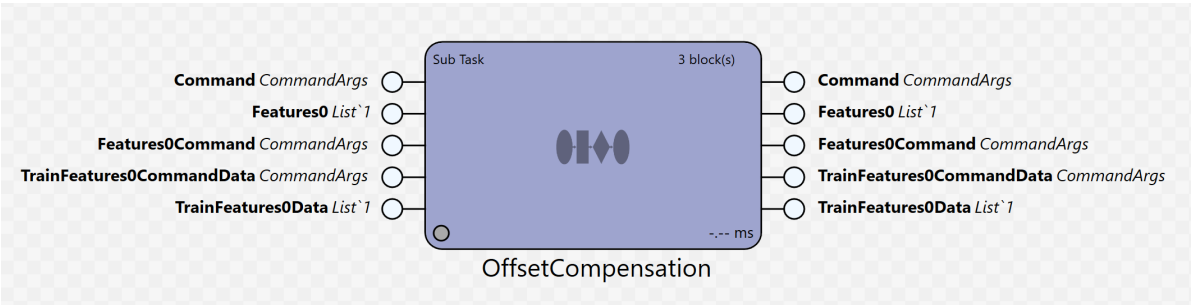
The outputs are subscribed train time and run time features and their commands, and current alignment task command.

Output	Type	Description
Command	CommandArgs	Current command for alignment task, the same with input command
Features0	List<CogAlpsPointFeature> / List<CogAlpsLineFeature>/ List<CogGenericFeature>	The list of run time features from connected feature finder
Features0Command	CommandArgs	The last run time command to trigger connected feature finder to run image acquisition and feature extraction

Output	Type	Description
MultiplePartFeatures	List<List<CogAlpsPointFeature>>	The list of run time parts' features from connected multiple part feature finder. Only available when alignment type is configured as "Multiple parts" in the Configuration Wizard
MultiplePartStatus	List<Integer>	The feature finding result status list for all sub regions on a tray from connected multiple part feature finder. For each staus: 1: sucess <=0: fail Only available when alignment type is configured as "Multiple parts" in the Configuration Wizard
TrainFeatures0CommandData	CommandArgs	The last train time command used for connected feature finder's image acquisition and feature extraction
TrainFeatures0Data	List<CogAlpsPointFeature> / List<CogAlpsLineFeature>/ List<CogGenericFeature>	The list of train time features from connected feature finder

Offset Compensation Subtask

Offset compensation allows user to manually add offsets to run time features to compensate the final alignment or assembly result based on inspection result feedback.



The inputs are subscribed train time and run time features and their commands, and current alignment task command.

Input	Type	Description
Command	CommandArgs	Current command for alignment task, the same with input command
Features0	List<CogAlpsPointFeature> / List<CogAlpsLineFeature>/ List<CogGenericFeature>	The list of run time features from connected feature finder
Features0Command	CommandArgs	The last run time command to trigger connected feature finder to run image acquisition and feature extraction
MultiplePartFeatures	List<List<CogAlpsPointFeature>>	The list of run time parts' features from connected multiple part feature finder. Only available when alignment type is configured as "Multiple parts" in the Configuration Wizard
MultiplePartStatus	List<Integer>	The feature finding result status list for all sub regions on a tray from connected multiple part feature finder. For each staus: 1: sucess <=0: fail Only available when alignment type is configured as "Multiple parts" in the Configuration Wizard

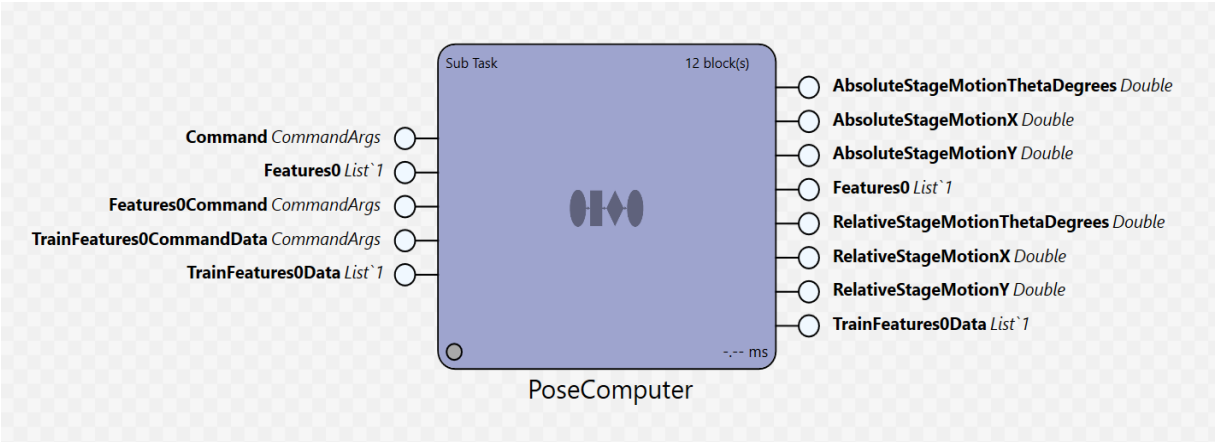
Input	Type	Description
TrainFeatures0CommandData	CommandArgs	The last train time command used for connected feature finder's image acquisition and feature extraction
TrainFeatures0Data	List<CogAlpsPointFeature> / List<CogAlpsLineFeature>/ List<CogGenericFeature>	The list of train time features from connected feature finder

The outputs are the same with inputs except that run time features/multiple parts features may have been compensated with offsets.

Output	Type	Description
Features0	List<CogAlpsPointFeature> / List<CogAlpsLineFeature>/ List<CogGenericFeature>	The compensated run time features from connected feature finder
MultiplePartFeatures	List<List<CogAlpsPointFeature>>	The list of run time parts' features from connected multiple part feature finder. Only available when alignment type is configured as "Multiple parts" in the Configuration Wizard

Pose Computer Subtask

Pose computer subtask computes transforms between target position and current position, and calculates alignment parameters that would result in the desired part alignment or assembly.



The inputs are the same with offset compensation subtask's.

Input	Type	Description
Command	CommandArgs	Current command for alignment task, the same with input command
Features0	List<CogAlpsPointFeature> / List<CogAlpsLineFeature>/ List<CogGenericFeature>	The list of run time features from connected feature finder
Features0Command	CommandArgs	The last run time command to trigger connected feature finder to run image acquisition and feature extraction
MultiplePartFeatures	List<List<CogAlpsPointFeature>>	The list of run time parts' features from connected multiple part feature finder. Only available when alignment type is configured as "Multiple parts" in the Configuration Wizard

Input	Type	Description
MultiplePartStatus	List<Integer>	The feature finding result status list for all sub regions on a tray from connected multiple part feature finder. For each staus: 1: sucess <=0: fail Only available when alignment type is configured as "Multiple parts" in the Configuration Wizard
TrainFeatures0CommandData	CommandArgs	The last train time command used for connected feature finder's image acquisition and feature extraction
TrainFeatures0Data	List<CogAlpsPointFeature> / List<CogAlpsLineFeature>/ List<CogGenericFeature>	The list of train time features from connected feature finder

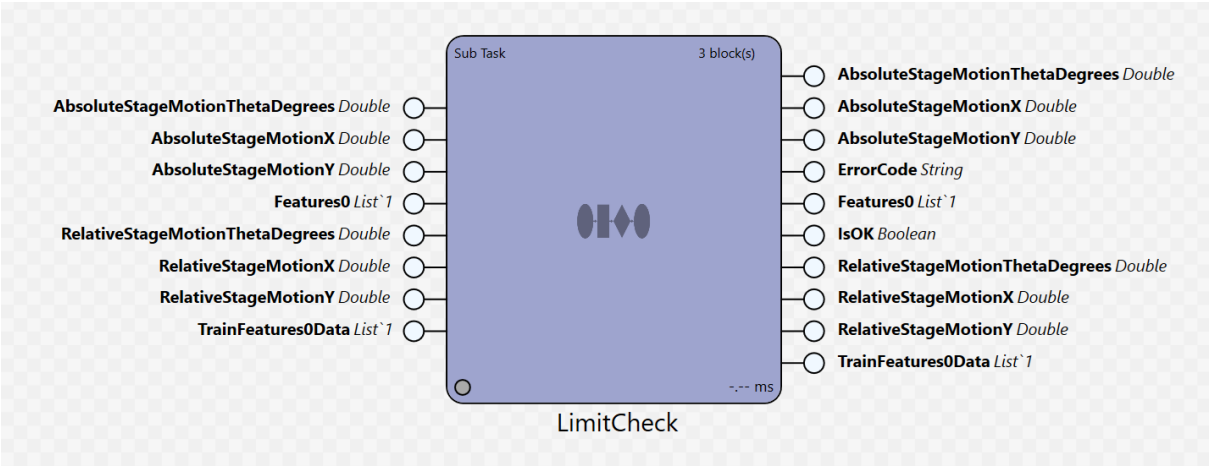
The outputs are train time and run time features, relative and absolute stage positions.

Output	Type	Description
AbsoluteStageMotionThetaDegrees	• Double: for single part alignment or two parts assembly • List<Double>: for multiple parts alignment	The Theta-value of absolute pose stage should move to from current pose or trained pose
AbsoluteStageMotionX	• Double: for single part alignment or two parts assembly • List<Double>: for multiple parts alignment	The Y-value of absolute pose stage should move to from current pose or trained pose
AbsoluteStageMotionY	• Double: for single part alignment or two parts assembly • List<Double>: for multiple parts alignment	The X-theta value of absolute pose stage should move to from current pose or trained pose
Features0	List<CogAlpsPointFeature> / List<CogAlpsLineFeature>/ List<CogGenericFeature>	The list of features from connected feature finder
MultiplePartFeatures	List<List<CogAlpsPointFeature>>	The list of run time parts' features from connected multiple part feature finder. Only available when alignment type is configured as "Multiple parts" in the Configuration Wizard
MultiplePartStatus	List<Integer>	The feature finding result status list for all sub regions on a tray from connected multiple part feature finder. For each staus: 1: sucess <=0: fail Only available when alignment type is configured as "Multiple parts" in the Configuration Wizard
RelativeStageMotionThetaDegrees	• Double: for single part alignment or two parts assembly • List<Double>: for multiple parts alignment	The Theta-value of relative pose stage should move from current pose or trained pose
RelativeStageMotionX	• Double: for single part alignment or two parts assembly • List<Double>: for multiple parts alignment	The Y-value of relative pose stage should move from current pose or trained pose

Output	Type	Description
RelativeStageMotionY	• Double: for single part alignment or two parts assembly • List<Double>: for multiple parts alignment	The X-theta value of relative pose stage should move from current pose or trained pose
TrainFeatures0Data	List<CogAlpsPointFeature> / List<CogAlpsLineFeature>/ List<CogGenericFeature>	The list of trained features from connected feature finder

Limit Check Subtask

Limit check use pose computer's output x, y, theta as input, to check if they're within certain alignment limit specs defined by user.



The inputs are the same with Pose Computer subtask's outputs.

Input	Type	Description
AbsoluteStageMotionThetaDegrees	• Double: for single part alignment or two parts assembly • List<Double>: for multiple parts alignment	The Theta-value of absolute pose stage should move to from current pose or trained pose
AbsoluteStageMotionX	• Double: for single part alignment or two parts assembly • List<Double>: for multiple parts alignment	The Y-value of absolute pose stage should move to from current pose or trained pose
AbsoluteStageMotionY	• Double: for single part alignment or two parts assembly • List<Double>: for multiple parts alignment	The X-theta value of absolute pose stage should move to from current pose or trained pose
Features0	List<CogAlpsPointFeature> / List<CogAlpsLineFeature>/ List<CogGenericFeature>	The list of features from connected feature finder
MultiplePartFeatures	List<List<CogAlpsPointFeature>>	The list of run time parts' features from connected multiple part feature finder. Only available when alignment type is configured as "Multiple parts" in the Configuration Wizard

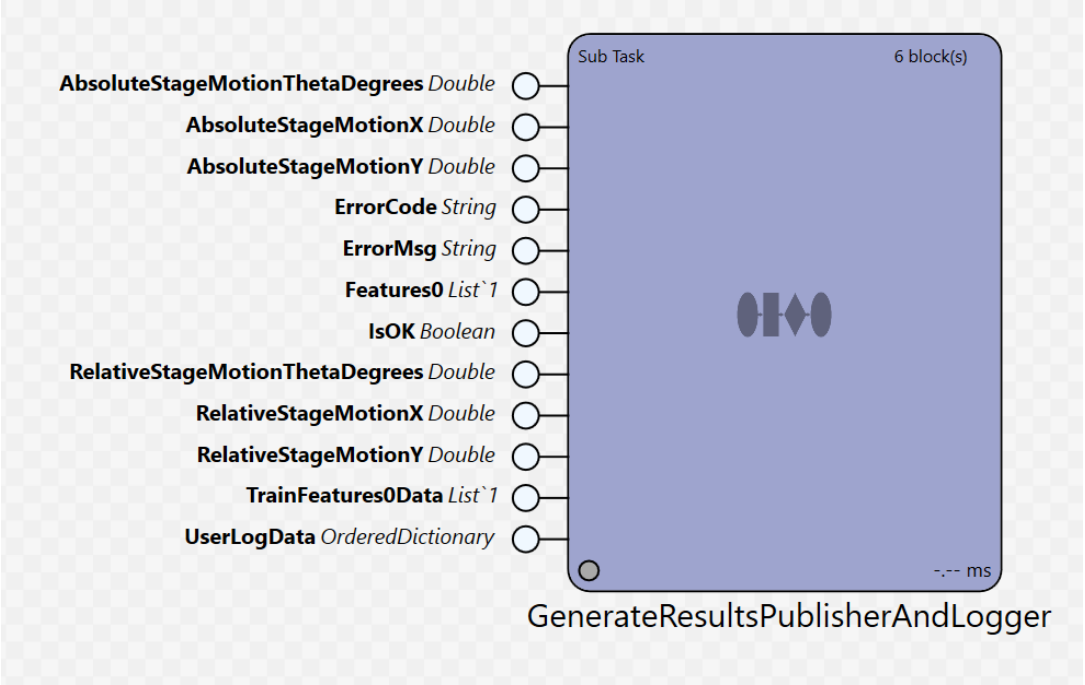
Input	Type	Description
MultiplePartStatus	List<Integer>	The feature finding result status list for all sub regions on a tray from connected multiple part feature finder. For each staus: 1: sucess <=0: fail Only available when alignment type is configured as "Multiple parts" in the Configuration Wizard
RelativeStageMotionThetaDegrees	• Double: for single part alignment or two parts assembly • List<Double>: for multiple parts alignment	The Theta-value of relative pose stage should move from current pose or trained pose
RelativeStageMotionX	• Double: for single part alignment or two parts assembly • List<Double>: for multiple parts alignment	The Y-value of relative pose stage should move from current pose or trained pose
RelativeStageMotionY	• Double: for single part alignment or two parts assembly • List<Double>: for multiple parts alignment	The X-theta value of relative pose stage should move from current pose or trained pose
TrainFeatures0Data	List<CogAlpsPointFeature> / List<CogAlpsLineFeature>/ List<CogGenericFeature>	The list of trained features from connected feature finder

Besides passing all the original inputs without any change down as ouputs, Limit Check subtask also outputs Limit Check OK/NG result, and ErrorCode.

Output	Type	Description
ErrorCode	• String for single part alignment or two parts assembly • List<String> for multiple parts alignment	ErrorCode or a list of ErrorCode when Limit Check is NG
IsOK	Boolean	Whether current pose computation result(s) pass Limit Check

GenerateResultsPublisherAndLogger Subtask

This subtask saves all alignment or assembly result data into a dictionary and publishes that dictionary in Task Results Block.



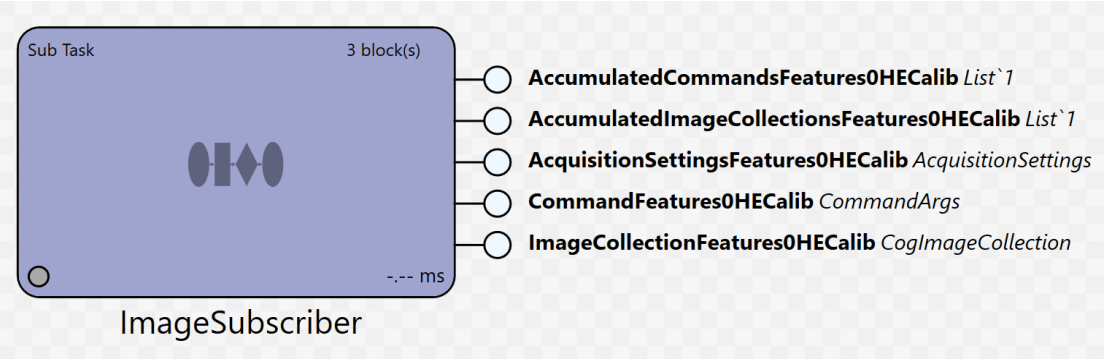
The inputs are train time and run time features, relative and absolute pose stage should move/move to, LCheck OK/NG result, LCheck ErrorCode, customized Error Message, and customized User Log Data.

Input	Type	Description
AbsoluteStageMotionThetaDegrees	• Double: for single part alignment or two parts assembly • List<Double>: for multiple parts alignment	The Theta-value of absolute pose stage should move to from current pose or trained pose
AbsoluteStageMotionX	• Double: for single part alignment or two parts assembly • List<Double>: for multiple parts alignment	The Y-value of absolute pose stage should move to from current pose or trained pose
AbsoluteStageMotionY	• Double: for single part alignment or two parts assembly • List<Double>: for multiple parts alignment	The X-theta value of absolute pose stage should move to from current pose or trained pose
ErrorCode	• String: for single part alignment or two parts assembly • List<String>: for multiple parts alignment	ErrorCode when LCheck is NG
ErrorMsg	String	Error message for user to customize
Features0	List<CogAlpsPointFeature> / List<CogAlpsLineFeature>/ List<CogGenericFeature>	The list of features from connected feature finder
MultiplePartFeatures	List<List<CogAlpsPointFeature>>	The list of run time parts' features from connected multiple part feature finder. Only available when alignment type is configured as "Multiple parts" in the Configuration Wizard

Input	Type	Description
MultiplePartStatus	List<Integer>	The feature finding result status list for all sub regions on a tray from connected multiple part feature finder. For each staus: 1: sucess <=0: fail Only available when alignment type is configured as "Multiple parts" in the Configuration Wizard
IsOK	Boolean	Whether LCheck is passed or not
RelativeStageMotionThetaDegrees	Double	The Theta-value of relative pose stage should move from current pose or trained pose
RelativeStageMotionX	• String: for single part alignment or two parts assembly • List<String>: for multiple parts alignment	The Y-value of relative pose stage should move from current pose or trained pose
RelativeStageMotionY	• String: for single part alignment or two parts assembly • List<String>: for multiple parts alignment	The X-theta value of relative pose stage should move from current pose or trained pose
TrainFeatures0Data	List<CogAlpsPointFeature> / List<CogAlpsLineFeature>/ List<CogGenericFeature>	The list of trained features from connected feature finder
UserLogData	OrderedDictionary	User data for user to customize data to be added at the end alignment log

Image Subscriber Subtask

This subtask subscribes images captured by the corresponding feature finder tasks that compute features that are used by this alignment task.



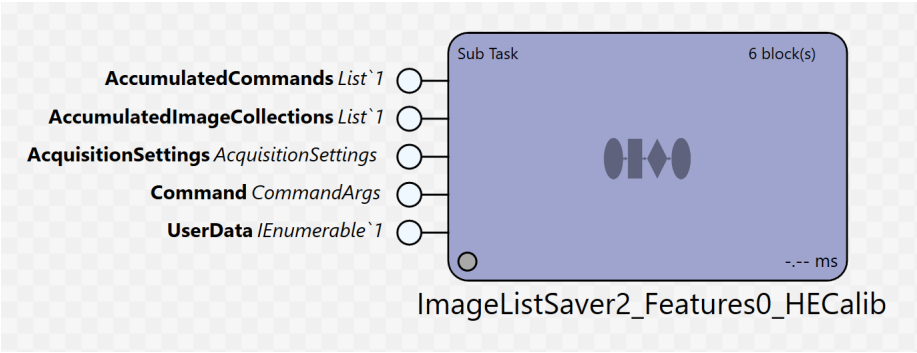
The output pin names are related to specific feature finding task names. For the same of generality, feature finding task names are ignored in decription. The outputs are acquisition settings of the feature finding task, command and image collection, as well as accumulated commands and images collections from feature finding task.

Output	Type	Description
AccumulatedCommands	List<CommandArgs>	Accumulated commands that have been sent to Acquisition subtask to acquire a whole set of images at all acquisition positions in connected feature finding task.
AccumulatedImageCollections	List<CogImageCollection>	Accumulated images acquired at different acquisition positions in connected feature finding task.

Output	Type	Description
AcquisitionSettings	AcquisitonSettings	The acquisition settings for connected feature finding task, which includes camera indexes, camera enable/disable status, exposure time and position indexes.
Command	CommandArgs	Last run time command of connected feature finding task
ImageCollection	CogImageCollection	A collection of images acquired at last position index in connected feature finding task

Image List Saver2 Subtask

Image List Saver2 subtask saves the images captured by corresponding feature finder task.



The inputs are acquisition settings of connected feature finding task, last run time command and accumulated commands and images collections from collected feature finding task, as well as UserData that could be used as metadata for CDB files.

Input	Type	Description
AccumulatedCommands	List<CommandArgs>	Accumulated commands that have been sent to Acquisition subtask to acquire a whole set of images at all acquisition positions in connected feature finding task.
AccumulatedImageCollection	List<CogImageCollection>	Accumulated images acquired at different acquisition positions in connected feature finding task.
AcquisitionSettings	AcquisitonSettings	The acquisition settings for connected feature finding task, which includes camera indexes, camera enable/disable status, exposure time and position indexes.
Command	CommandArgs	Last run time command of connected feature finding task
UserData	IEnumerable<KeyValuePair<String,String>>	User defined Metadata for image saving in CDB format. Not used.

General Tool Block

Alignment

Generic Features Finder

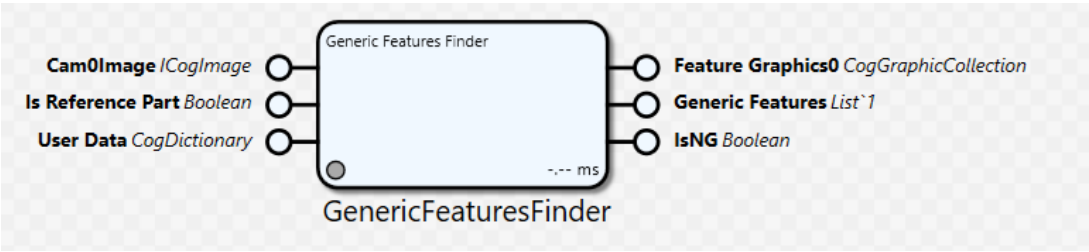
This block allows the configuration of multiple generic feature finders (for example, a feature finder). When executed in a sequence, this block runs the configured finders on its input images and outputs a list of the found features and graphics.

This block has three modes of operation:

- Setup Mode - An interactive editor is used to add, remove or modify the feature finders. A version of this interactive editor is available in the HMI Toolbox, under AlignPlus->Alignment->Generic Features Finder. Alternatively, the public methods provided by the block can be used for this purpose.
- Reference Part Mode - The block is executed and the feature finders will be run as they should be on a reference part.
- Runtime Mode - The block is executed and the feature finders will be run as they should be for a runtime part.

This block supports only custom feature finder:

- A custom feature finder, represented by a VisionPro CogToolBlock. The tool block contains input terminals to receive an image set from a single camera and output terminals to return the located generic feature results. The tool block can use all the images in the image set to locate a single feature.



When adding feature finders, a given input image can have multiple feature finders configured for it, and some input images need not have any feature finders.

The output result for the block is a list of elements of type CogAlpsGenericFeature, where each entry will have the feature location, a Boolean that indicates if the feature was found, a camera identifier and an image identifier. Additionally, the block will output a set of graphics of type CogGraphicCollections, one for each camera, which can be used to display the features location graphics. Each collection will contain the graphics generated by all the feature finders for the corresponding camera. It is expected that a collection will be combined with an appropriate image to generate a VisionPro record for display.

General Information

Class name: GenericFeaturesFinderBlock

Namespace: Cognex.Designer.AlignPlus.Alignment

Assembly: Cognex.Designer.AlignPlus.Alignment.dll

Inputs

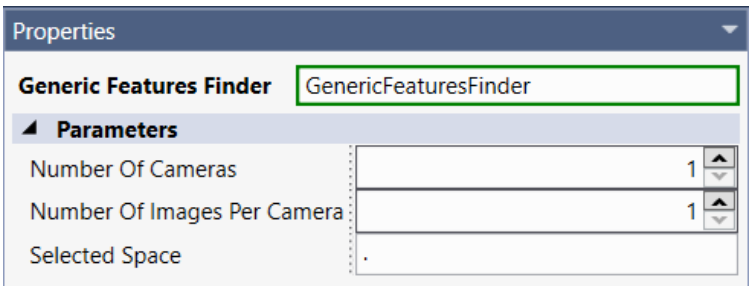
Name	Type	Description
Cam0Image	ICogImage	The image from camera 0. A corrected image should be supplied on this pin. Additional pins will be created based on the Number Of Cameras property. Pins of this type will be displayed only if the Number Of Images Per Camera property is equal to one.

Name	Type	Description
Is Reference Part	Boolean	If true, when executed the block computes features on a reference part. If false, when executed the block will behave as it should for a run time part. For custom tool block, it depends on user how to utilize this parameter
User Data	CogDictionary	Contains a CogDictionary with user specified data. This data will be forwarded to all finders When there is no data needed, still an empty user data need to link to this pin, otherwise, tool block will consider this pin as null and pop out an error when running

Outputs

Name	Type	Description
Feature Graphics0	CogGraphicCollection	The graphics generated by all the feature finders for camera 0. Additional pins of this type will be created based on the Number Of Cameras property. Each graphic will be in the Selected Space for this block. Custom feature finder tool blocks must generate all results and graphics in the Selected Space. Additional logic (such as the CogRecord Creator block) can be used to combine these graphics with an image, for display in a VisionPro Display.
Generic Features	List< CogAlpsGenericFeature on page 1>	Contains a list of generic features. Each item in the list corresponds to the results of a single feature finder. Each item will contain the feature location and a Boolean indicating if the feature was found. There is no particular order in which the generic features are output, however the order does not change until another feature finder has been added.
IsNG	Boolean	True: features are all found False: one or more features are not found

Properties



Parameters	Type	Description
Number of Cameras	Integer	Get/set the number of cameras supplying images to this block • The total number of CamXImage pins or is equal to the Number of Cameras property. The number of pins changes immediately upon a change to this property. • Throws System.ArgumentOutOfRangeException: If it is set to a value out of expected range
Number of Images Per Camera	Integer	Get/set the number of cameras supplying images to this block. If this property is greater than one, then the block will accept image sets (CogImageCollections) for its input pins. The purpose of accepting an image set from each camera is to allow feature location to be performed on images captured under different acquisition conditions from a single camera. • Throws System.ArgumentOutOfRangeException: If it is set to a value out of expected range
Selected Space	String	Get/set the selected space in which the finders run. The selected space is the coordinate system in which all feature finders return results (such as locations, distances and graphics) and in which the tools interpret input data (such as regions of interest). • Each input image should allow the mapping of image features in the corrected image to this selected space. • All the feature locations will be generated in the selected space. If the selected space is not present in the coordinate space tree of an image to be used for feature location, a suitable exception will be thrown when the sequence block is executed.

Line Features Finder

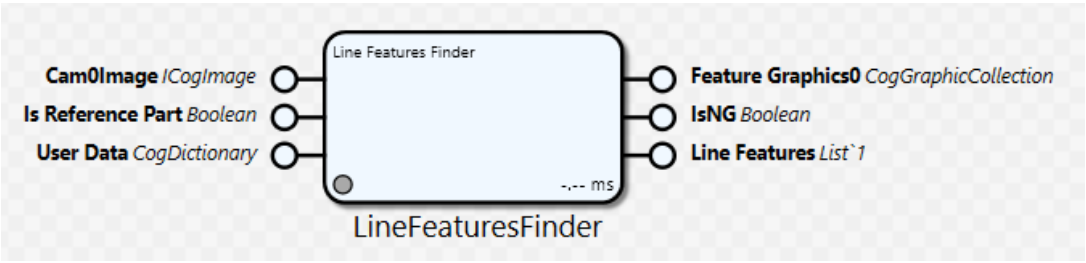
This block allows the configuration of multiple line feature finders (for example, a line finder). When executed in a sequence, this block runs the configured finders on its input images and outputs a list of the found lines and graphics.

This block has three modes of operation:

- Setup Mode - An interactive editor is used to add, remove or modify the feature finders. A version of this interactive editor is available in the HMI Toolbox, under AlignPlus->Alignment->Line Features Finder. Alternatively, the public methods provided by the block can be used for this purpose.
- Reference Part Mode - The block is executed and the feature finders will be run as they should be on a reference part.
- Runtime Mode - The block is executed and the feature finders will be run as they should be for a runtime part.

Two types of line feature finders are supported:

- A line finder, represented by a VisionPro CogFindLineTool tool. This tool is set up to run on a single selected camera image.
- A custom feature finder, represented by a VisionPro CogToolBlock. The tool block contains input terminals to receive an image set from a single camera and output terminals to return the located line feature results. The tool block can use all the images in the image set to locate a single feature.



When adding feature finders, a given input image can have multiple feature finders configured for it, and some input images need not have any feature finders.

The output result for the block is a list of elements of type CogAlpsLineFeature, where each entry will have the feature location, a Boolean that indicates if the feature was found, a camera identifier and an image identifier. Additionally, the block will output a set of graphics of type CogGraphicCollections, one for each camera, which can be used to display the features location graphics. Each collection will contain the graphics generated by all the feature finders for the corresponding camera. It is expected that a collection will be combined with an appropriate image to generate a VisionPro record for display.

General Information

Class name: LineFeaturesFinderBlock

Namespace: Cognex.Designer.AlignPlus.Alignment

Assembly: Cognex.Designer.AlignPlus.Alignment.dll

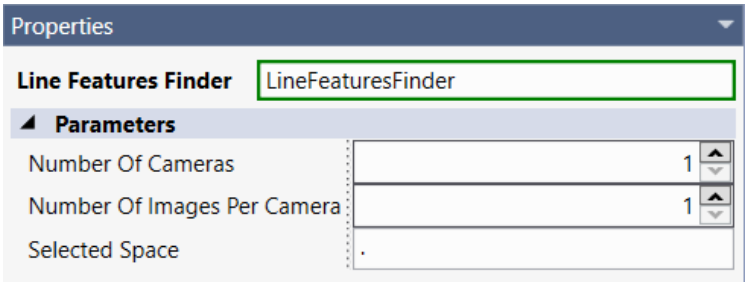
Inputs

Name	Type	Description
Cam0Image	ICogImage	Image from specific camera
Is Reference Part	Boolean	True: train time reference feature False: run time For custom tool block, it depends on user how to utilize this parameter
User Data	CogDictionary	Dictionary type of data available for user to feed more information into toolblock. When there is no data needed, still an empty user data need to link to this pin, otherwise, toolblock will consider this pin as null and pop out an error when running

Outputs

Name	Type	Description
Feature Graphics0	CogGraphicCollection	A collection of graphics on one image to show where the feature is. This can be configured/customized on UI.
IsNG	Boolean	True: features are all found False: one or more features are not found
Line Features	List<CogAlpsLineFeature on page 1>	A list of features in which feature's location, name, and many other properties are all included.

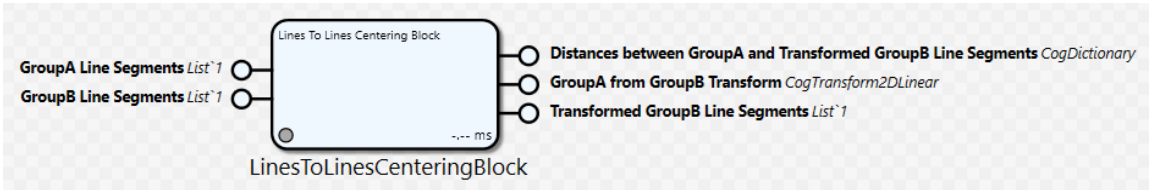
Properties



Name	Type	Description
Number of Cameras	Integer	Get/set the number of cameras supplying images to this block • The total number of CamXImage pins or is equal to the Number of Cameras property. The number of pins changes immediately upon a change to this property. • Throws System.ArgumentOutOfRangeException: If it is set to a value out of expected range
Number of Images Per Camera	Integer	The Number Of Images Per Camera property controls the number of images expected for each camera. If this property is greater than one, then the block will accept image sets (CogImageCollections) for its input pins. The purpose of accepting an image set from each camera is to allow feature location to be performed on images captured under different acquisition conditions from a single camera. • Throws System.ArgumentOutOfRangeException: If it is set to a value out of expected range
Selected Space	String	Get/set the selected space in which the finders run. The selected space is the coordinate system in which all feature finders return results (such as locations, distances and graphics) and in which the tools interpret input data (such as regions of interest). • Each input image should allow the mapping of image features in the corrected image to this selected space. • All the feature locations will be generated in the selected space. • The graphics generated by the first feature finders (line finder) will also be in this space. If the selected space is not present in the coordinate space tree of an image to be used for feature location, a suitable exception will be thrown when the sequence block is executed.

Lines To Lines Centering Block

The block computes the transform that centers the GroupB line segments to their corresponding GroupA line segments.



General Information

Class name: LinesToLinesCenteringBlock

Namespace: Cognex.Designer.AlignPlus.Alignment

Assembly: Cognex.Designer.AlignPlus.Alignment.dll

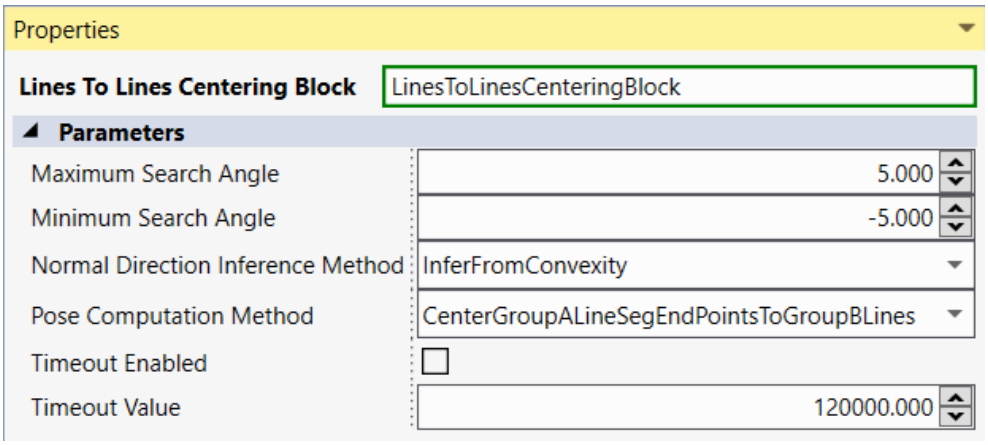
Inputs

Name	Type	Description
GroupA Line Segments	List<CogAlpsLineFeature on page 1>	Line features representing GroupA
GroupB Line Segments	List<CogAlpsLineFeature on page 1>	Line features representing GroupB

Outputs

Name	Type	Description
Distances between GroupA and Transformed GroupB Line Segments	CogDictionary	The gap distances between Transformed GroupB Line Segments and GroupA Line Segments. For mode CenterGroupALineSegEndPointsToGroupBLines, distance for each line segment is computed by averaging the point-line distance of its two end points. For mode CenterGroupAPointPairsToGroupBLinePairs, distance for each line segment is exactly the distance computed by the CogCenterPointsToLines tool.
GroupA from GroupB Transform	CogTransform2DLinear	The transform that centers the line segments of GroupB to line segments of GroupA
Transformed GroupB Line Segments	List< CogAlpsLineFeature on page 1>	GroupB's line features after applying transform, theoretically it should be very close to Group A line features

Properties



Name	Type	Description
Maximum Search Angle	double	This parameter sets the upper limit the rotation angle of the result transform. The value is specified in degrees
Minimum Search Angle	double	This parameter sets the lower limit the rotation angle of the result transform. The value is specified in degrees

Name	Type	Description
Normal Direction Inference Method	Constant	Property specifies how the line segment normal direction is inferred • **InferFromConvexity** Automatically infer the line segment normal direction for each input line segment. It requires the input line segments to form a convex shape. It selects the normal direction for each line segment such that the line normal points outwards. • **UseLineSegmentNormalDirection** Use the direction of the given line segments to determine the normal direction for each line segment. The direction of a line segment is implied by the ordering of its start and end points (end - start). The normal direction is defined to be +90 degrees from the line segment direction.
Pose Computation Method	Constant	There are two options: • **CenterGroupALineSegEndPointsToGroupBLines** For point to line fitting. The feature correspondences are between GroupA line segment end points(two end points for each line segment) and GrouB lines. This block will minimize the variance in the point to line signed distance across all point to line correspondences, as a result, it tries to make all of the point to line distances the same. • **CenterGroupAPointsPairsToGroupBLinePairs** For point pair to line pair fitting. The feature correspondences are between GroupA line segment middle points(one middle point for each line segment) and GrouB lines. This block will minimize the variation in the point to line signed distance differences, across all point pair line pair correspondences. The point to line distance difference for a given point pair line pair correspondence is the difference between the point to line distance for the first point and the point to line distance for the second point in the point pair. The class tries to make the two point to line signed distances in each point pair line pair correspondence the same. Signed distances are measured in the direction of the normal to the line. For more information, please refer to **AlignPlus Concpet\Pose Compute\Part Pose Change Computation**.
Timeout Enabled	Boolean	If true, the tool will time out after the time specified by the TimeoutValue property has elapsed
Timeout Value	Double	If TimeoutEnabled property is true, the tool will time out after the value of this property. The value is in milliseconds.

Point Feature Finder

This block allows the configuration of multiple point feature finders (for example, a PatMax pattern or a corner finder). When executed in a sequence, this block runs the configured finders on its input images and outputs a list of the found points and graphics.

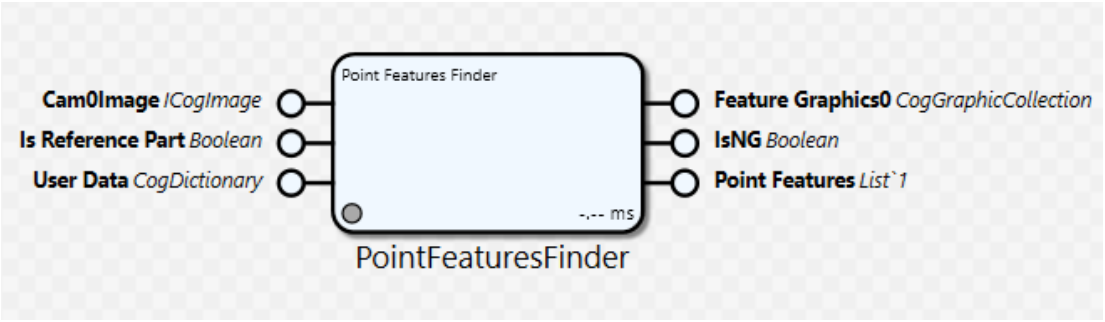
This block has three modes of operation:

- Setup Mode - An interactive editor is used to add, remove or modify the feature finders. A version of this interactive editor is available in the HMI Toolbox, under AlignPlus->Alignment->Point Features Finder. Alternatively, the public methods provided by the block can be used for this purpose.
- Reference Part Mode - The block is executed and the feature finders will be run as they should be on a reference part.
- Runtime Mode - The block is executed and the feature finders will be run as they should be for a runtime part.

Three types of point feature finders are supported:

- A pattern finder, represented by a VisionPro CogPMAlignTool tool. This tool is set up to run on a single selected camera image.
- A corner finder, represented by a VisionPro CogCornerFinderTool tool. This tool is set up to run on a single selected camera image.

- A custom feature finder, represented by a VisionPro CogToolBlock. The tool block contains input terminals to receive an image set from a single camera and output terminals to return the located point feature results. The tool block can use all the images in the image set to locate a single feature.



When adding feature finders, a given input image can have multiple feature finders configured for it, and some input images need not have any feature finders.

The output result for the block is a list of elements of type CogAlpsPointFeature, where each entry will have the feature location, a Boolean that indicates if the feature was found, a camera identifier and an image identifier. Additionally, the block will output a set of graphics of type CogGraphicCollections, one for each camera, which can be used to display the features location graphics. Each collection will contain the graphics generated by all the feature finders for the corresponding camera. It is expected that a collection will be combined with an appropriate image to generate a VisionPro record for display.

General Information

Class name: PointFeaturesFinderBlock

Namespace: Cognex.Designer.AlignPlus.Alignment

Assembly: Cognex.Designer.AlignPlus.Alignment.dll

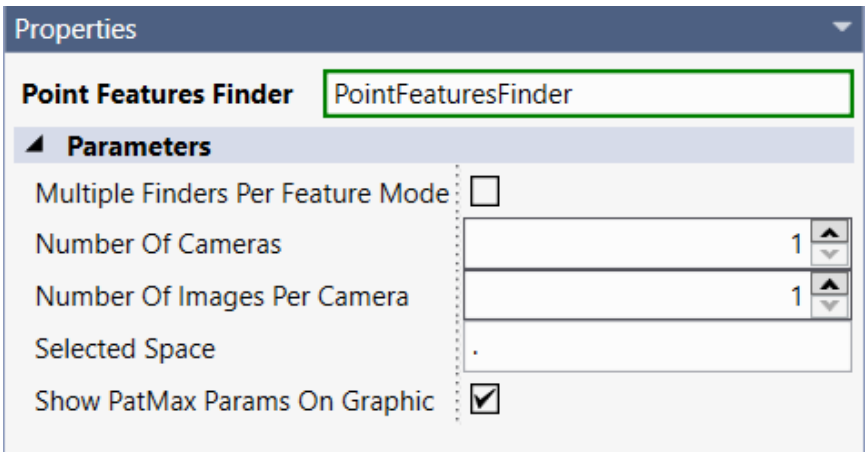
Inputs

Name	Type	Description
Cam0Image	ICogImage	The image from camera 0. A corrected image should be supplied on this pin. Additional pins will be created based on the Number Of Cameras property. Pins of this type will be displayed only if the Number Of Images Per Camera property is equal to one.
Is Reference Part	Boolean	If true, when executed the block computes features on a reference part. If false, when executed the block will behave as it should for a run time part. For custom tool block, it depends on user how to utilize this parameter
User Data	CogDictionary	Contains a CogDictionary with user specified data. This data will be forwarded to all finders. When there is no data needed, still an empty user data need to link to this pin, otherwise, toolblock will consider this pin as null and pop out an error when running

Outputs

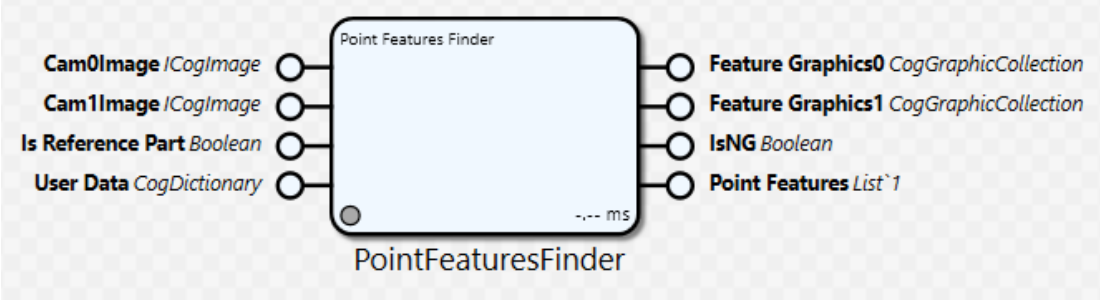
Name	Type	Description
Feature Graphics0	CogGraphicCollection	The graphics generated by all the feature finders for camera 0. Additional pins of this type will be created based on the Number Of Cameras property. Each graphic will be in the Selected Space for this block. Custom feature finder tool blocks must generate all results and graphics in the Selected Space. Additional logic (such as the CogRecord Creator block) can be used to combine these graphics with an image, for display in a VisionPro Display.
IsNG	Boolean	True: features are all found False: one or more features are not found
Point Features	List< CogAlpsPointFeature on page 1>	Contains a list of point features. Each item in the list corresponds to the results of a single feature finder. Each item will contain the feature location and a Boolean indicating if the feature was found. There is no particular order in which the point features are output, however the order does not change until another feature finder has been added.

Properties



Name	Type	Description
Multiple Finders Per Feature Mode	Bool	True: There could be more than one single feature finders outputs features with the same feature name. Those features with the same feature name will be considered as just one feature and only the one with the highest score will take part in pose computation. False: One single feature outputs one unique feature.
Number of Cameras	Integer	Get/set the number of cameras supplying images to this block • The total number of CamXImage(X stands for camera index) pins or is equal to the Number of Cameras property. The number of pins changes immediately upon a change to this property. • Throws System.ArgumentOutOfRangeException: If it is set to a value out of expected range
Number of Images Per Camera	Integer	Get/set the number of images supplied by each camera. If this property is greater than one, then the block will accept image sets (CogImageCollections) for its input pins. The purpose of accepting an image set from each camera is to allow feature location to be performed on images captured under different acquisition conditions from a single camera. • Throws System.ArgumentOutOfRangeException: If it is set to a value out of expected range
Selected Space	String	Get/set the selected space in which the finders run. The selected space is the coordinate system in which all feature finders return results (such as locations, distances and graphics) and in which the tools interpret input data (such as regions of interest). • Each input image should allow the mapping of image features in the corrected image to this selected space. • All the feature locations will be generated in the selected space. • The graphics generated by the first two feature finders (the pattern finder and corner finder) will also be in this space. If the selected space is not present in the coordinate space tree of an image to be used for feature location, a suitable exception will be thrown when the sequence block is executed.
Show PatMax Params On Graphic	Bool	True: show PatMax result scores on image display False: hide PatMax result scores on image display

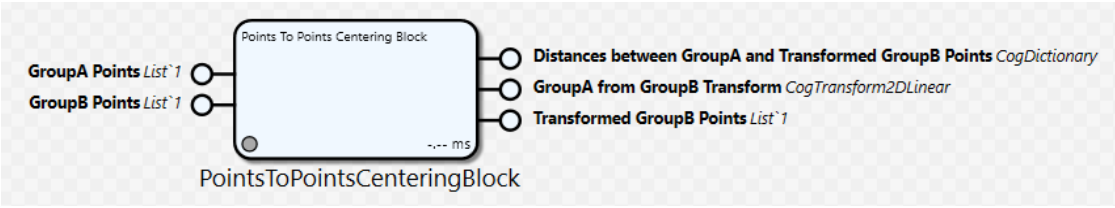
When camera number is 2, the block's appearance is as below:



Note: PointFeatureFinder block is optimized for minimizing vision task execution time: Each finder runs independently and asynchronously within Point Feature Finder tool block.

Points To Points Centering Block

The block computes the transform that maps the GroupB points to their corresponding GroupA points.



General Information

Class name: PointsToPointsCenteringBlock

Namespace: Cognex.Designer.AlignPlus.Alignment

Assembly: Cognex.Designer.AlignPlus.Alignment.dll

Inputs

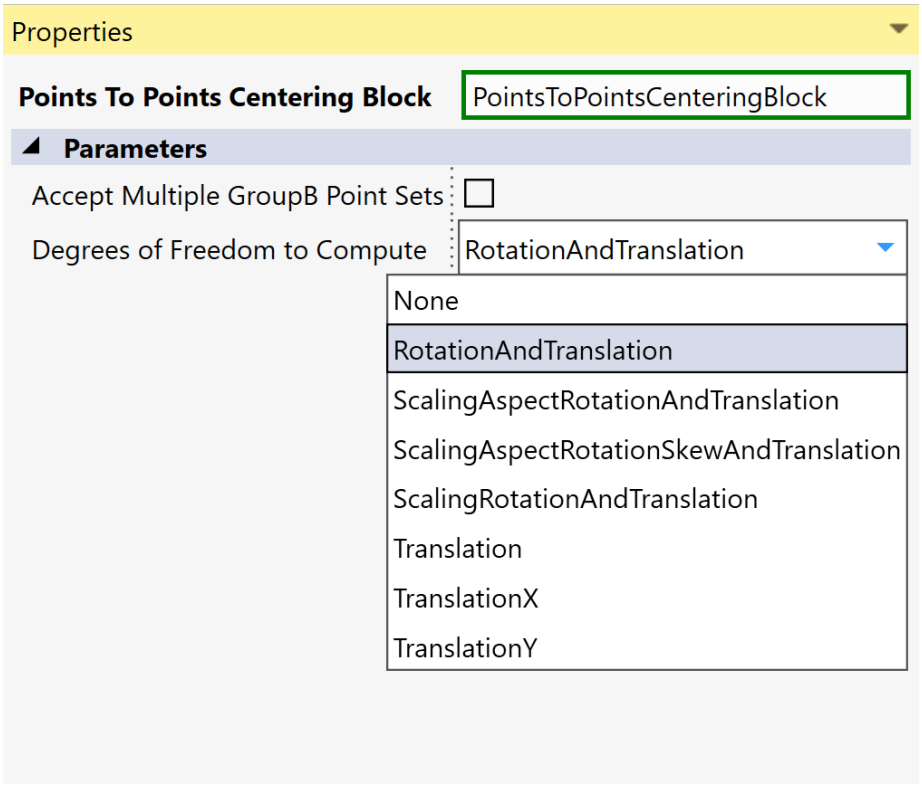
Name	Type	Description
GroupA Points	List<CogAlpsPointFeature>	Point features representing the target position of the part
GroupB Points	List<CogAlpsPointFeature>	Point features representing the current position of the part *Only shows when **Accept Multiple GroupB Point Sets** property is set as false.
GroupB Points List	List<List<CogAlpsPointFeature>>	A list of GroupB Points representing multiple run time parts' current positions. *Only shows when **Accept Multiple GroupB Point Sets** property is set as true.

Outputs

Name	Type	Description
Distances between GroupA and Transformed GroupB Points	CogDictionary	Average distances between Transformed GroupB points and GroupA points.
GroupA from GroupB Transform	CogTransform2DLinear	The transform that maps the points of GroupB to points of GroupA. *Only shows when **Accept Multiple GroupB Point Sets** property is set as false.
GroupA from GroupB Transforms	List<CogTransform2DLinear>	A list of transforms that maps GroupB point sets to GroupA point set. *Only shows when **Accept Multiple GroupB Point Sets** property is set as true.

Name	Type	Description
Transformed GroupB Points	List<CogAlpsPointFeature>	GroupB's Point features after applying transform, theoretically it should be very close to Group A line features

Properties



Name	Type	Description
Accept Multiple GroupB Point Sets	Boolean	True: this block will compute multiple transforms that map each GroupB point set to Group A point set. False: this block only computes one transform that map one GroupB point set to Group A point set.
Degrees of Freedom to Compute	Constant	Degrees of Freedom taking into consideration while calculating GroupB to GroupA linear transform

Stage Pose Computer

The Stage Pose Computer block is used to compute the required stage motion to properly align one or more parts. It can be used for two types of applications: alignment and assembly. When the type of application is selected using the ApplicationType property, the block reconfigures its pins appropriately. This block works for stationary and moving camera configurations.

Application types are as follows:

For Alignment applications, the block accepts a TrainFromRun transform. The transform can be computed externally using a PointsToPointsCenteringBlock, LinesToLinesCenteringBlock or any custom block that computes such a transform. The TrainFromRun transform will be used along with the current uncorrected stage position and the hand-eye calibration results to compute the relative and absolute stage pose parameters.

All input and output stage pose parameters will be in UnCorrectedHome2DFromStage2D. The motivation behind accepting and outputting uncorrected pose parameters is that the position is always reported by a motion control system. Similarly, the pose correction that is desired is conveyed to the motion control system.

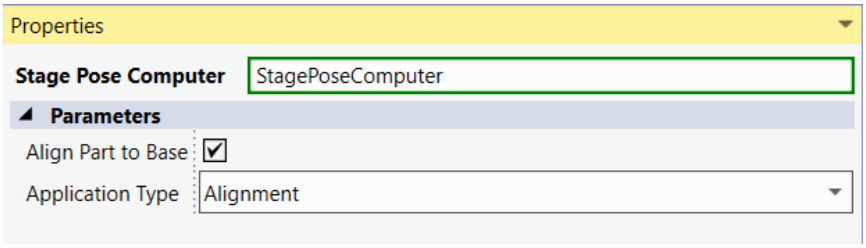
General Information

Class name: StagePoseComputerBlock

Namespace: Cognex.Designer.AlignPlus.Alignment

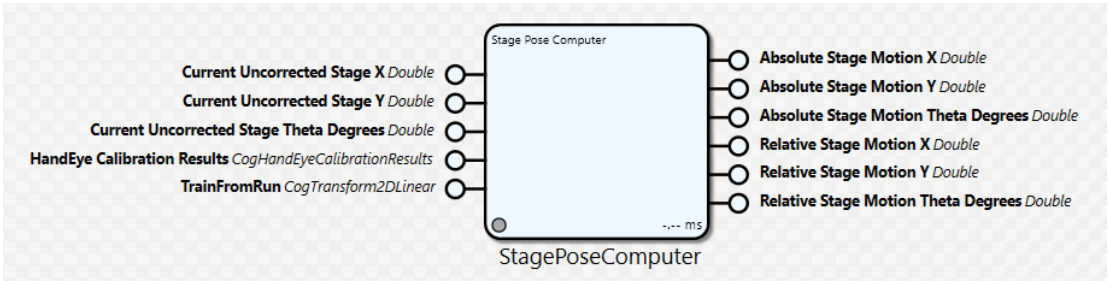
Assembly: Cognex.Designer.AlignPlus.Alignment.dll

Properties



Name	Type	Description
Align Part to Base	Boolean	Checked on: Align Part to Base Checked off: Align Part to Gripper
Application Type	Cognex.Designer.AlignPlus.Alignment.StagePoseComputerBlock.Application	Select the type of application as follows: • When the value of this property is **Alignment**, the block will configure itself for use in an alignment application. The TrainFromRun pin, which takes the trainFromRun transform as an input, will be displayed. • When the value of the property is **Assembly**, the block will configure itself for use in an assembly application. The TrainFromRun At Station With Stage and TrainFromRun At Station Without Stage pins will be displayed.

Alignment



Inputs

Name	Type	Description
Current Stage X	double	X component of stage's current position
Current Stage Y	double	Y component of stage's current position
Current Stage Theta Degrees	double	Theta component of stage's current position
HandEye Calibration Results	Cognex.VisionPro.CogHandEyeCalibrationResults	HandEye Calibration Results outputted from Handeye Calibrator on page 255 The results are used to convert the position reported by the stage to the actual position.
TrinFromRun	Cognex.VisionPro.CogTransform2DLinear	The TrainFromRun transform of a part. This pin is displayed if ApplicationType==Alignment

Outputs

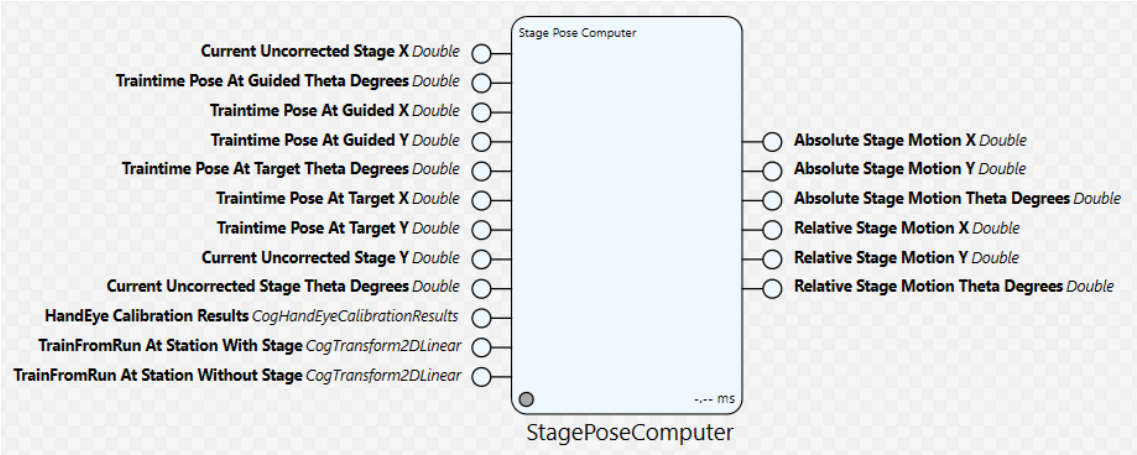
Name	Type	Description
Absolute Stage Motion X	Double	The absolute X position that the stage should be moved to in order to achieve the desired alignment/assembly
Absolute Stage Motion X	Double	The absolute Y position that the stage should be moved to in order to achieve the desired alignment/assembly
Absolute Stage Motion Theta Degrees	Double	The absolute rotation in degrees that the stage should be subject to in order to achieve the desired alignment/assembly
Relative Stage Motion X	Double	The X position relative to the current position that the stage should be moved to in order to achieve the desired alignment/assembly
Relative Stage Motion Y	Double	The Y position relative to the current position that the stage should be moved to in order to achieve the desired alignment/assembly
Relative Stage Motion Theta Degrees	Double	The rotation relative to the current rotational position in degrees that the stage should be subject to in order to achieve the desired alignment/assembly

Assembly

For Assembly applications, the block accepts two TrainFromRun transforms. One for a stationary part and the other is for a moving part. The two poses, along with the current uncorrected stage position and the hand-eye calibration results to compute the relative and absolute stage pose parameters.

For assembly applications the block assumes that during train time the parts are placed such that when assembled by a repeatable assembly mechanism the parts would have the desired assembly. The block will output stage positions that would assemble the run-time part and train-time part similarly.

When Assembly is chosen in "Application Type" property, the input pins of Stage Pose Computer will change as below:



Inputs

Name	Type	Description
Traintime Pose At Guided Theta Degrees	Double	Theta component of moving station's trained position
Traintime Pose At Guided X	Double	X component of moving station's trained position
Traintime Pose At Guided Y	Double	Y component of moving station's trained position
Traintime Pose At Target Theta Degrees	Double	Theta component of stationary station's trained position
Traintime Pose At Target X	Double	X component of stationary station's trained position

Name	Type	Description
Traintime Pose At Target Y	Double	Y component of stationary station's trained position
Current Uncorrected Stage Theta Degrees	Double	Theta component of moving station's current position
Current Uncorrected Stage X	Double	X component of moving station's current position
Current Uncorrected Stage Y	Double	Y component of moving station's current position
HandEye Calibration Results	CogHandEyeCalibrationResults	HandEye Calibration Results outputted from Handeye Calibrator on page 255 The results are used to convert the position reported by the stage to the actual position.
TrainFromRun At Station With Stage	CogTransform2DLinear	The TrainFromRun transform of the moving part. This pin is displayed if ApplicationType==Assembly
TrainFromRun At Station Without Stage	CogTransform2DLinear	The TrainFromRun transform of the stationary part. This pin is displayed if ApplicationType==Assembly

Outputs

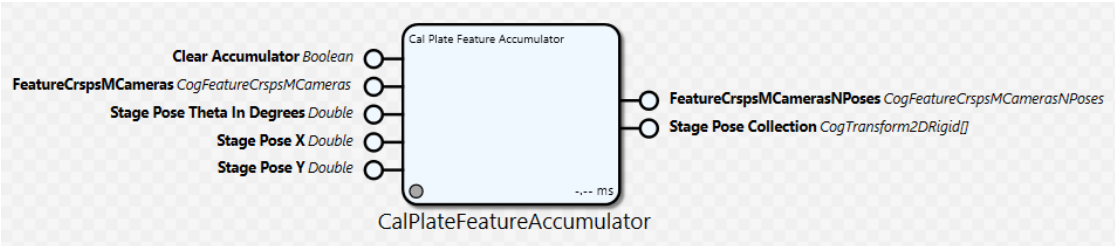
Name	Type	Description
Absolute Stage Motion X	Double	The absolute X position that the stage should be moved to in order to achieve the desired assembly
Absolute Stage Motion X	Double	The absolute Y position that the stage should be moved to in order to achieve the desired assembly
Absolute Stage Motion Theta Degrees	Double	The absolute rotation in degrees that the stage should be subject to in order to achieve the desired assembly
Relative Stage Motion X	Double	The X position relative to the current position that the stage should be moved to in order to achieve the desired assembly
Relative Stage Motion Y	Double	The Y position relative to the current position that the stage should be moved to in order to achieve the desired assembly
Relative Stage Motion Theta Degrees	Double	The rotation relative to the current rotational position in degrees that the stage should be subject to in order to achieve the desired assembly

Calibration

Cal Plate Feature Accumulator

The purpose of this block is to accumulate stage poses and the correspondence data extracted from a calibration target at those poses. Each time the block is executed, it adds the data from its input pins to a set of accumulated data, and outputs the current contents of the accumulated data on the output pins.

If the block is executed with a value of true on the Clear Accumulator pin, the accumulated data is cleared, then the current inputs are accumulated.



General Information

Class name: FeatureAccumulatorBlock

Namespace: Cognex.Designer.AlignPlus.Calibration

Assembly: Cognex.Designer.AlignPlus.Calibration.dll

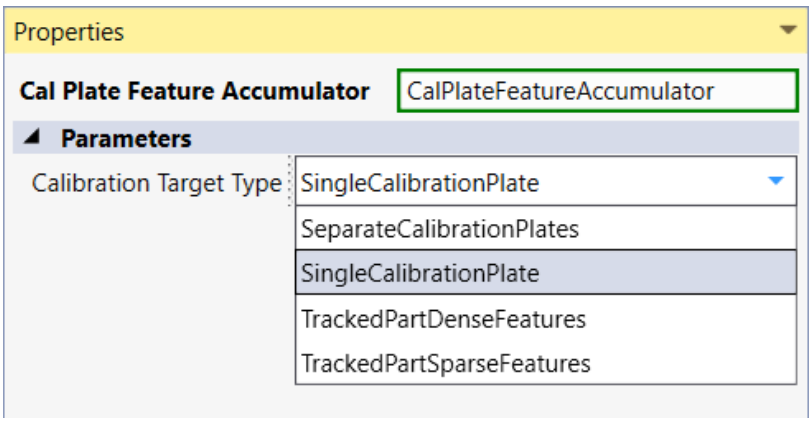
Inputs

Name	Type	Description
Clear Accumulator	Boolean	If true, the accumulator will first be cleared during execution of the block before accumulation of the other inputs.
FeatureCrspsMCameras	CogFeatureCrspsMCameras	The correspondence pair data from a set of cameras. It is expected that this input will be connected to the FeatureCrspsMCameras output pin of the CheckerGridFeatureExtractor block.
Stage Pose Theta In Degrees	Double	The rotation component (in degrees) of the stage position used to collect the correspondence data.
Stage Pose X	Double	The X component of the stage position used to collect the correspondence data.
Stage Pose Y	Double	The Y component of the stage position used to collect the correspondence data.

Outputs

Name	Type	Description
FeatureCrspsMCamerasNPoses	CogFeatureCrspsMCamerasNPoses	The accumulated of FeatureCrsps for all cameras at multi stage positions
Stage Pose Collection	CogTransform2DRigid[]	The accumulated stage pose data.

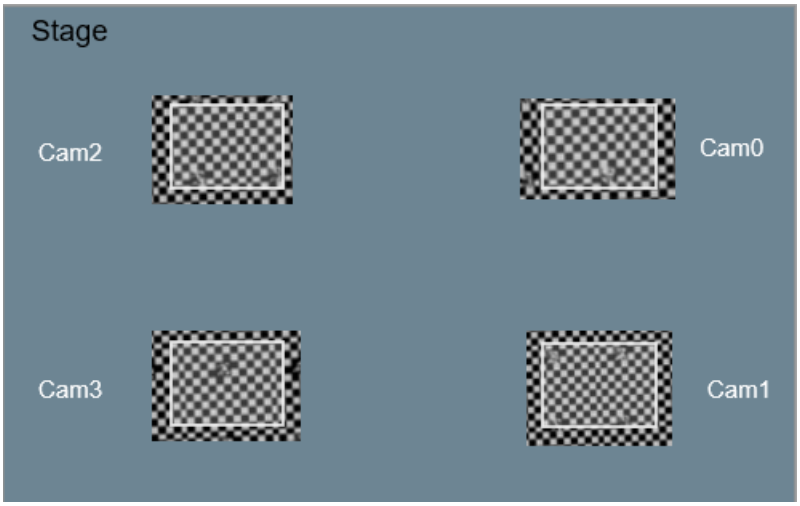
Properties



Calibration Target Type:

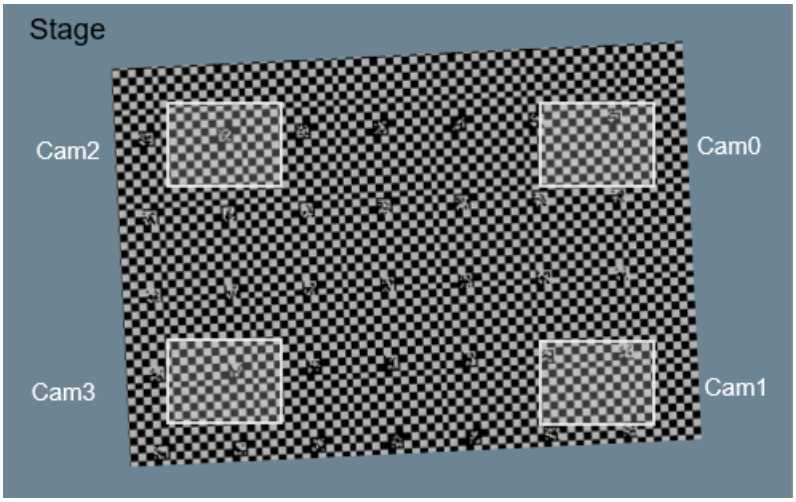
- **Separate Calibration Plates**

  Indicates that a separate calibration target is being used for each camera. In this mode, the checkerboard is used to compute lens and perspective distortions, but the motion device is used to compute the relative and absolute positions of each camera in Home2D. The accuracy of this computation depends upon the accuracy and extent of the rotation during the calibration process. The user should use this option only if a single calibration target cannot be used for the application.



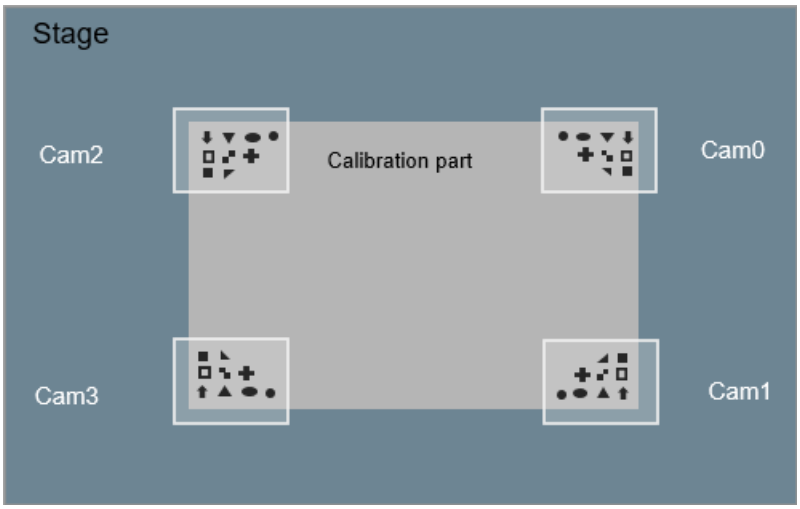
- **Single Calibration Plate**

  Indicates that a single calibration target is being used to hand-eye calibrate the multiple cameras. Following calibration, all cameras would have the ability to map their features to a common Plate2D. The relative position of the cameras are estimated accurately. The absolute cameras positions in Home2D is influenced by the accuracy and extent of rotation during the hand-eye calibration process.



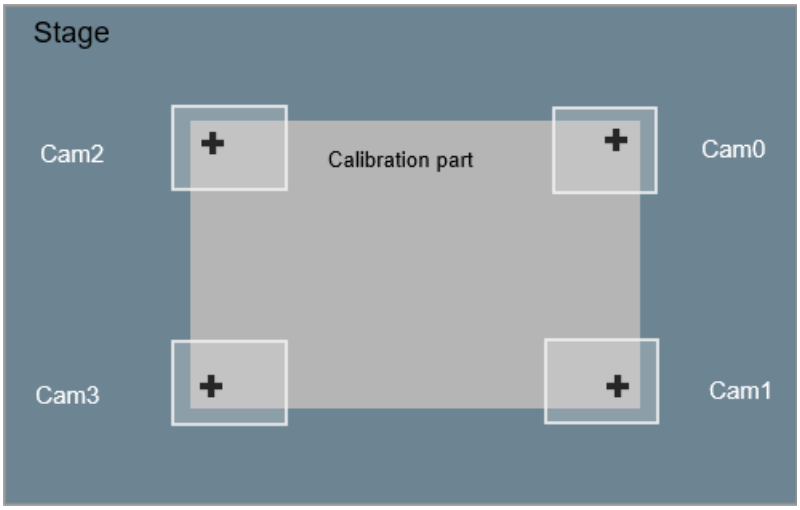
- **Tracked Part Dense Features**

  Used when the number of fiducial tracked during the hand-eye calibration process is more than three.



- **Tracked Part Sparse Features**

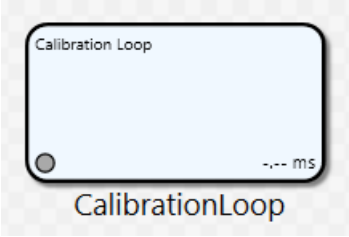
  Used when the number of fiducial marks is at-least one but less than three.



Calibration Loop

This block supports calibration of a motion stage. When executed in a sequence, it computes a set of predetermined poses for the motion stage and for each computed pose calls a script point. This script point is expected to execute a task that extracts the features from a calibration plate.

When performing hand-eye calibration it is recommended that the stage undergoes two sets of motion. The first set should translate the stage to various points on a rectangular grid. The second set should rotate the stage to various points on a circular grid. This block has a set of properties that control the position and the number of points on the rectangular grid and circular grid.



The purpose of this block is to generate poses for these points, in the following manner:

- Before translating the stage to points on the rectangular grid, the stage is first rotated to the mean rotational position for the circular grid.
- Before rotating the stage to points on the circular grid, the stage is first translated to the center of the rectangular grid.
- When moving along the rectangular grid, the stage will go to the various points on the grid in a serpentine fashion.

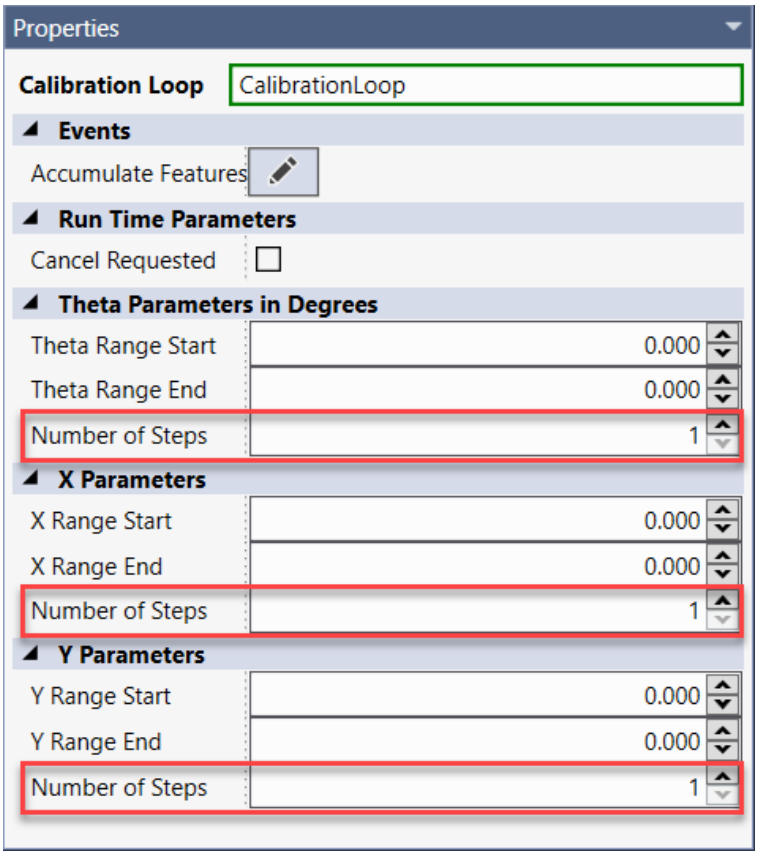
General Information

Class name: CalibrationLoopBlock

Namespace: Cognex.Designer.AlignPlus.Calibration

Assembly: Cognex.Designer.AlignPlus.Calibration.dll

Properties



Name	Type	Description
Theta Range Start	Double	Get/Set the starting position for the Theta axis (in degrees), used when generating stage poses on the circular grid.

Name	Type	Description
Theta Range End	Double	Get/Set the ending position for the Theta axis (in degrees), used when generating stage poses on the circular grid.
Number of Steps on Theta axis	Double	Get/Set the number of samples when rotating the stage along the Theta axis of the circular grid. This property has a minimum value of 1. Throws System.ArgumentOutOfRangeException: If the setter value is less than one.
X Range Start	Double	Get/Set the starting position for the X axis (in mm), used when generating stage poses on the circular grid.
X Range End	Double	Get/Set the ending position for the X axis (in mm), used when generating stage poses on the circular grid.
Number of Steps on X axis	Double	Get/Set the number of samples when rotating the stage along the X axis of the circular grid. This property has a minimum value of 1. Throws System.ArgumentOutOfRangeException: If the setter value is less than one.
X Range Start	Double	Get/Set the starting position for the Y axis (in mm), used when generating stage poses on the circular grid.
X Range End	Double	Get/Set the ending position for the Y axis (in mm), used when generating stage poses on the circular grid.
Number of Steps on X axis	Double	Get/Set the number of samples when rotating the stage along the Y axis of the circular grid. This property has a minimum value of 1. Throws System.ArgumentOutOfRangeException: If the setter value is less than one.
Cancel Requested	Boolean	Gets/sets if a cancel was requested. If this property is set to true before or during execution, the looping is cancelled.

These ranges and steps can also be set on Calibration Loop UI control.

Accumulate Features Script

This block supplies the script point "AccumulateFeatures". During execution this script point will be called for each pose in the generated set of stage poses. The script point will be called with suitable arguments to implement a single-pose feature extractor system.

bool AccumulateFeatures (Int32 Index, Double X, Double Y, Double Theta)

Name	Type	Description
Index	int	The zero-based iteration count in the current calibration operation. The first call to this script point receives a zero, the second call a one, etc.
X	Double	The X value of stage position that was used to acquire the images in this iteration
Y	Double	The Y value of stage position that was used to acquire the images in this iteration
Theta	Double	The Theta value of stage position that was used to acquire the images in this iteration

Here is a sample code for reference:

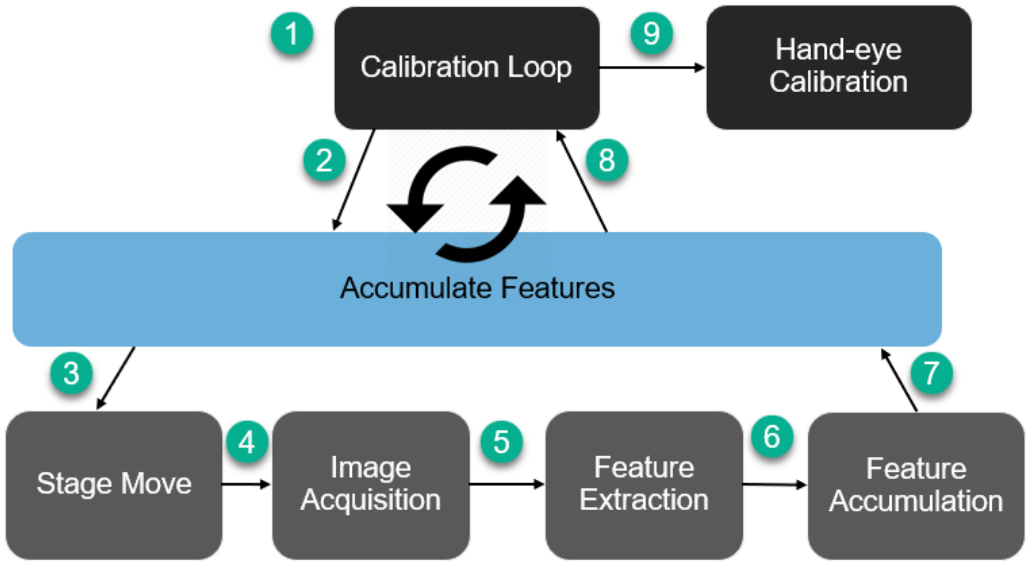
```
public bool AccumulateFeatures(int Index, double X, double Y, double Theta)
{
    // HE Calibration Compute Run
    message = String.Format("Acqurie Image & Feature extract Count = {0}", Index
       + 1);
    $LogData("Vision", message);

    $Stage0.CurPos.X = X;
    $Stage0.CurPos.Y = Y;
    $Stage0.CurPos.T = Theta;

    // Mode Select
    $Stage0.Calib.Compute = false;

    // Clear the accumulator for the first position
    if (Index == 0) $Stage0.Calib.Initialize = true;
    else $Stage0.Calib.Initialize = false;

    $Tasks.ImageAcquisition_Stage0.Run();
    $Tasks.Stage0_CalibPart.Run();

    return true;
}
```

In case user wants to cancel the Loop in the middle, “CancelRequested” property need to be set as true via scripting, such as:

```
$Tasks.Task.CalibrationLoop.CancelRequested = true;
```

Task Workflow

Here is a typical design for Calibration Loop：



Calibration Loop is the starting block, after getting all steps information, it generates a for-loop which iterate all points in shortest path of stage move. At each point, it calls Accumulate Features scripting, in which stage will be moved to desired position, then camera conduct image acquisition, feature is extracted by vision and all those features are collected before moving to the next point. Step 2-8 runs again for the next point and so on. Step 2-8 are normally done in a separate task.

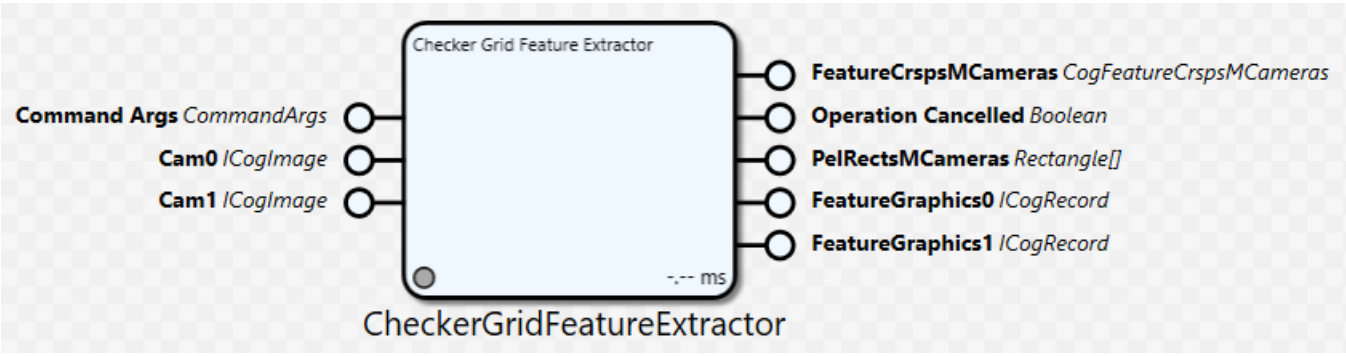
After all points’ features are collected, then it’s time to run the Hand-eye Calibration.

Checker Grid Feature Extractor

This block extracts the calibration features from a checkerboard calibration plate. It can be configured to extract calibration features from the images of multiple cameras, where the number of cameras is configured via a property. This block is a Cognex Designer wrapper for the VisionPro CogCalibFeatureExtractorCheckerboard operator and implements most of its functionality.

This feature extractor operates with a checkerboard style calibration plate consisting of a grid of alternating light and dark checkers. For each image of a checkerboard, the extractor locates the vertices of checkers in the image and generates a correspondence of vertex locations in image with vertex locations on the physical calibration plate coordinate system (Plate2D, see the remarks section of Cognex.VisionPro.CalibFix.CogCalibFeatureExtractorCheckerboardLabelModeConstants in the VisionPro documentation). The image point in the correspondence pair data will be in the selected space of the image.

The block supplies the script point "CurrentProgress". This script point will be called with suitable arguments to allow the monitoring of the feature extraction process during execution.



General Information

Class name: CheckerGridFeatureExtractorBlock

Namespace: Cognex.Designer.AlignPlus.Calibration

Assembly: Cognex.Designer.AlignPlus.Calibration.dll

Inputs

Name	Type	Description
Command Args	CommandArgs	CommandArgs object which contains description of command for setup-wizard program. For non-setup-wizard program, this input pin can be left as unlinked.
Cam0	ICogImage	The image from camera 0. Additional pins of this type are added based on the Number Of Cameras property.

Outputs

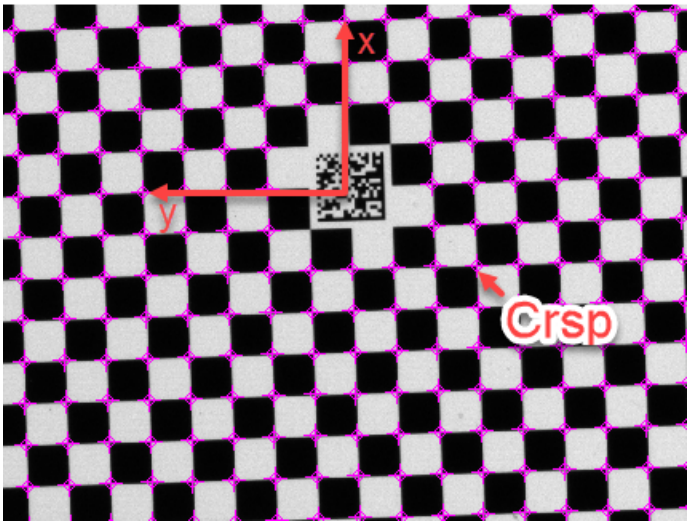
Name	Type	Description
FeatureCrspsMCameras	CogFeatureCrspsMCameras	The correspondence pair data extracted from all the input images.
Operation Cancelled	Boolean	True if the feature extraction was cancelled during execution.
PelRectsMCameras	Rectangle[]	The bounds of the input image, by default each rectangle is of the same size of image it refers to.
FeatureGraphics0	ICogRecord	CogRecord Creator on page 277 which shows all vertexes of tiles extracted by CheckerGridFeatureExtractor, Additional pins of this type are added based on the Number Of Cameras property.

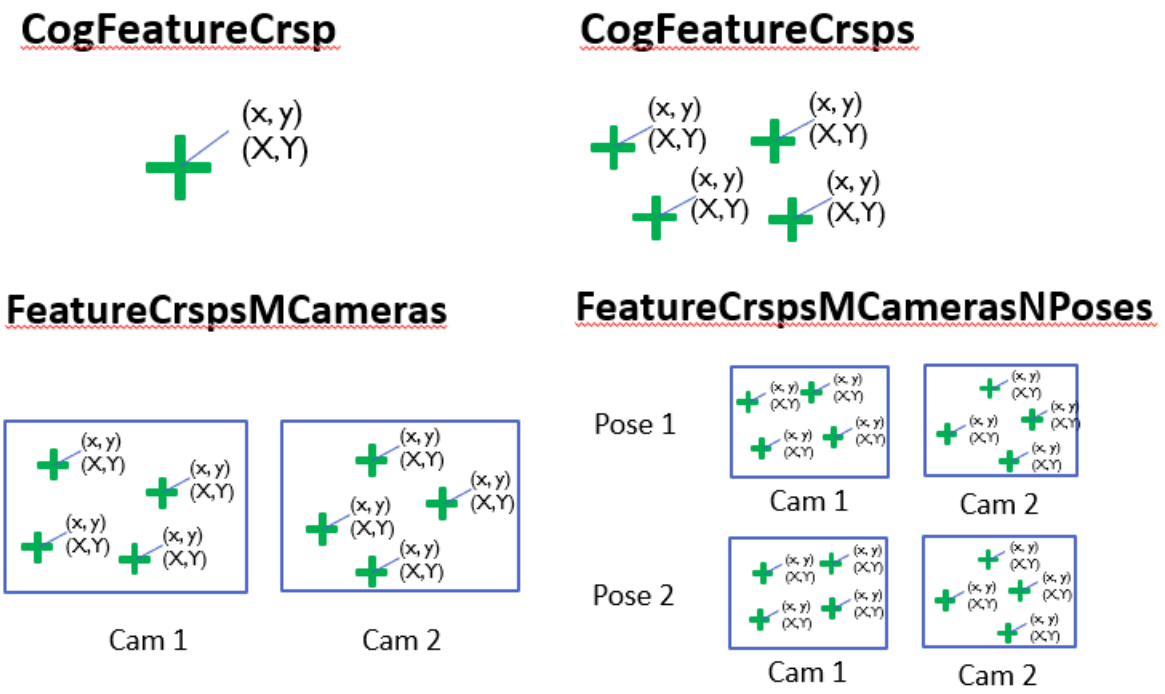
Definition of FeatureCrsp

Each point has two sets of coordinates: Raw2D coordinates(x,y) and Plate2D coordinates(X,Y), these two sets of coordinates then are stored in a structure called FeatureCrsp.

One point is represented by one FeatureCrsp, a group of points in one image then are called FeatureCrsps.

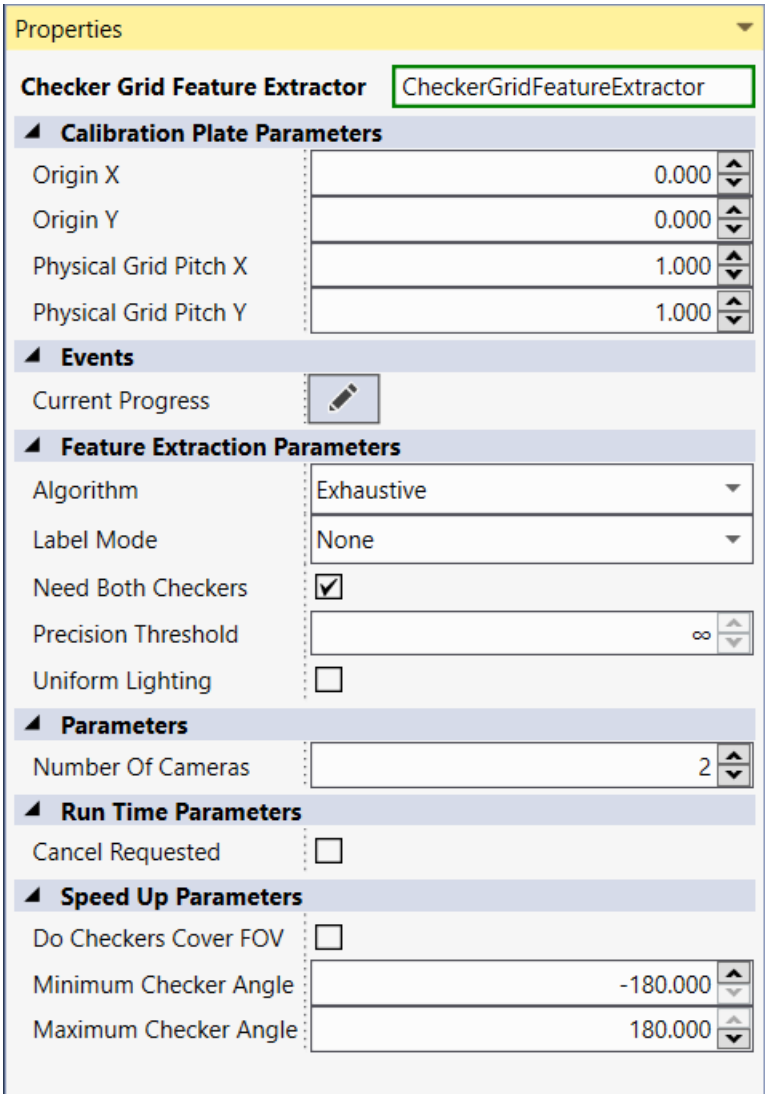
Multi cameras' FeatureCrsps there after is named as FeatureCrspsMCameras, and if that Multi cameras has features extracted at different stage locations, then all the features are grouped as FeatureCrspsMCamerasNPoses





For Checker Grid Feature Extractor, it extracts features from multi cameras at a certain position, so the major output is a FeatureCrspsMCameras object, which later is accumulated by Cal Plate Feature Accumulator.

Properties



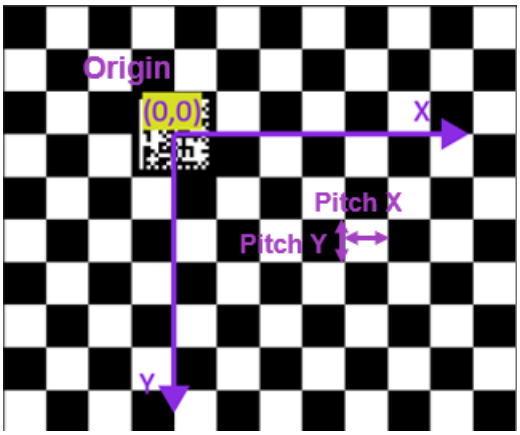
The properties are described below in different groups:

- **Calibration Plate Parameters**

Name	Type	Description
OriginX	Double	Gets/sets the the x value of the designated origin to be used for labeling of returned feature points. The vertex closest to point (OriginX, OriginY) will be used as the origin for point correspondence when Label Mode is CogCalibFeatureExtractorCheckerboardLabelModeConstants.UseOrigin. When not operating in CogCalibFeatureExtractorCheckerboardLabelModeConstants.UseOrigin mode, this property is ignored.
OriginY	Double	Gets/sets the the x value of the designated origin to be used for labeling of returned feature points. The vertex closest to point (OriginX, OriginY) will be used as the origin for point correspondence when Label Mode is CogCalibFeatureExtractorCheckerboardLabelModeConstants.UseOrigin. When not operating in CogCalibFeatureExtractorCheckerboardLabelModeConstants.UseOrigin mode, this property is ignored.
Physical Grid Pitch X		Gets/sets the physical units of grid pitch along the x-axis of the calibration plate coordinate system (Plate2D, see the remarks section of Cognex.VisionPro.CalibFix.CogCalibFeatureExtractorCheckerboardLabelModeConstants). It is the distance between any two adjacent checker vertices whenever the line joining them is parallel to the x axis of Plate2D. Throws System.ArgumentOutOfRangeException: The value is less than or equal to 0.

Name	Type	Description
Physical Grid Pitch Y		Gets/sets the physical units of grid pitch along the y-axis of the calibration plate coordinate system (Plate2D, see the remarks section of Cognex.VisionPro.CalibFix.CogCalibFeatureExtractorCheckerboardLabelModeConstants). It is the distance between any two adjacent checker vertices whenever the line joining them is parallel to the y axis of Plate2D. Throws System.ArgumentOutOfRangeException: The value is less than or equal to 0.

The figure below indicates the meaning of Origin, PitchX, and PitchY:



- **Feature Extraction Parameters**

Name	Type	Description
Algorithm	Cognex.VisionPro.CalibFix. CogCalibCheckerboardFeatureFinderConstants	Please refer to Feature Extractor
Label Mode	Cognex.VisionPro.CalibFix. CogCalibCheckerboardFiducialConstants	Please refer to Feature Extractor
Need both checkers	Boolean	Gets/sets the flag to indicate whether the tool should find only vertices shared by two interior light checkers. When set to true, the extractor will find only those vertices belonging simultaneously to two interior light checkers. An interior checker is one that does not touch the image boundary or the border of the calibration plate. When set to false, the extractor will attempt to find all vertices of all interior light checkers.
Precision Threshold	Double	Gets/sets the threshold for discarding vertices with excessive positional uncertainty, specified in pixels. Due to noise and distortion, there are errors in the computed vertex positions. The algorithm internally estimates the position uncertainty for all found vertices, and excludes those from the final result whose position uncertainty estimates exceed the threshold specified here. Throws System.ArgumentOutOfRangeException: If the input value is less than 0 in the setter.
Uniform Lighting	Boolean	Gets/sets the flag to indicate whether the checkerboard is expected to be uniformly illuminated in the run-time images. When set to true, the extractor expects the light checkers to be uniformly illuminated, and uses an efficient technique for finding the vertices which can improve the speed performance. However, if in fact the illumination is not uniform, this technique may not find certain vertices that are severely affected by the non-uniform lighting. When set to false, the tool performs better in presence of severe non-uniform lighting, and may find more vertices in these cases.

- **Parameters**

Name	Type	Description
Number of Cameras	Int32	Sets/Gets the number of cameras supplying images to the feature extractor. This property has a minimum value of MinNumberOfCameras (1) and a maximum value of MaxNumberOfCameras (16). The number of CamN and FeatureGraphicsN pins are each equal to the NumberOfCameras property. The available pins change immediately upon a change of this property. Throws System.ArgumentOutOfRangeException: If the setter value is less than MinNumberOfCameras or greater than MaxNumberOfCameras

- **Runtime Parameters**

Name	Type	Description
Cancel Requested	Boolean	Gets/sets if a cancel was requested. If this property is set to true before or during execution, the feature extraction is cancelled.

- **Speed Up Parameters**

Name	Type	Description
Do Checkers Cover FOV	Boolean	Gets/sets whether the checkers are expected to entirely cover the field of view. Note that the purpose of this property is to improve speed performance when the checker coverage is known. This property should only be set to true if it is known beforehand that the checkers will cover the entire image for each camera at each pose. Note that Do Checkers Cover FOV is only used when Algorithm is Cognex.VisionPro.CalibFix.CogCalibFeatureExtractorCheckerboardAlgorithmConstants.Exhaustive.
Minimum Checker Angle	Double	Gets/sets the minimum expected angle of checker orientations. Note that if the new value is larger than Maximum Checker Angle, then Maximum Checker Angle will be changed to the new Minimum Checker Angle. The purpose of Minimum and Maximum Checker Angle is to improve speed performance when the checker orientations are known. Minimum and Maximum Checker Angle are only used when Algorithm is Cognex.VisionPro.CalibFix.CogCalibFeatureExtractorCheckerboardAlgorithmConstants.Exhaustive.
Maximum Checker Angle	Double	Gets/sets the maximum expected angle of checker orientations. Note that if the new value is smaller than Minimum Checker Angle, then Minimum Checker Angle will be changed to the new Maximum Checker Angle. The purpose of Minimum and Maximum Checker Angle is to improve speed performance when the checker orientations are known. Minimum and Maximum Checker Angle are only used when Algorithm is Cognex.VisionPro.CalibFix.CogCalibFeatureExtractorCheckerboardAlgorithmConstants.Exhaustive

Current Progress Script

Current Progress script provides information on the current progress of feature extraction

CurrentProgress(Cognex.VisionPro.CalibFix.CogCalibFeaturesExtractedEventArgs CurrentProgress)

Script point called periodically during the execution of feature extraction.

Published Methods

Void ResetBlock()

This method resets this block to a default state.

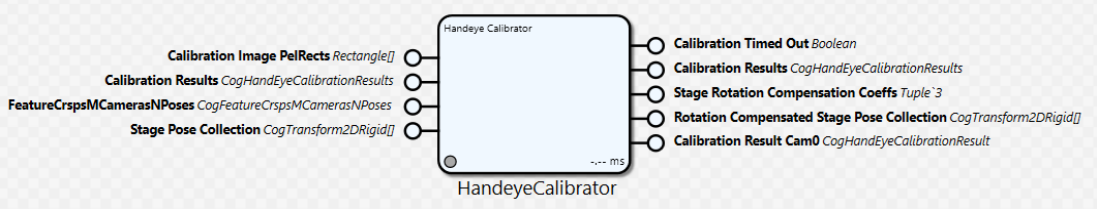
Handeye Calibrator

The HandEye Calibrator block does the work of computing hand eye calibration. This block is a Cognex Designer wrapper for the VisionPro CogHandEyeCalibrator operator and it implements most of its functionality. This block can be configured to perform hand-eye calibration on multiple cameras using the Number Of Cameras property.

The VisionPro CogHandEyeCalibrator has two overloads for the execute function:

- The first overload computes hand-eye calibration results using the corresponding point pairs and stage poses.
- The second overload recomputes the hand-eye calibration results using corresponding point pairs and stage poses, and reusing each camera's Raw2DFromCamera2D from a previous hand-eye calibration result.

This block implements functionality to support only the first overload.



General Information

Class name: MotionStageValidatorBlock

Namespace: Cognex.Designer.AlignPlus.Calibration

Assembly: Cognex.Designer.AlignPlus.Calibration.dll

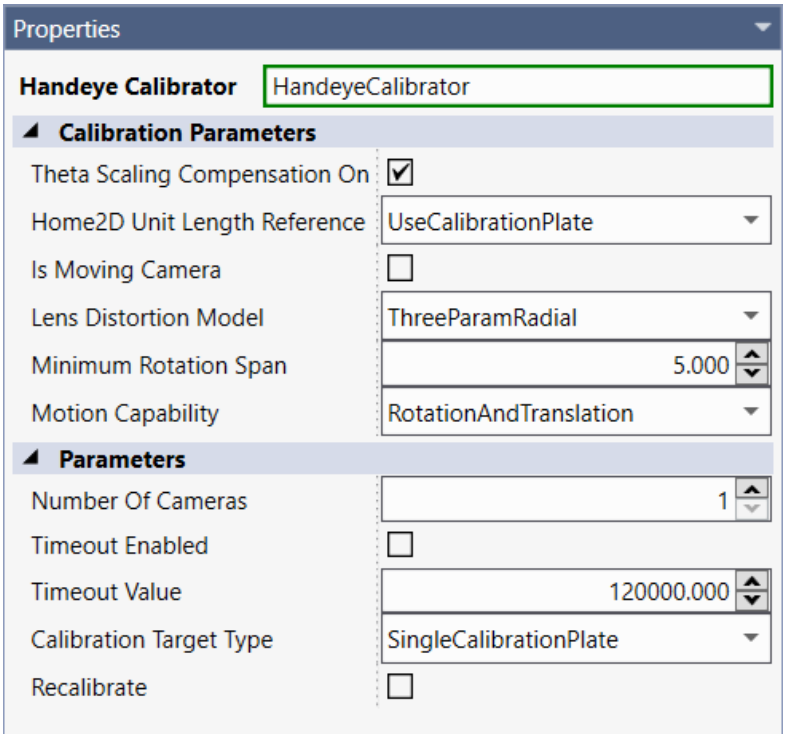
Inputs

Name	Type	Description
Calibration Image PelRects	Rectangle[]	The pelrects used to extract the correspondence point pairs. These pelrects are used to define the camera coordinate system (i.e. Camera2D) to be at the center of the pelrect. There should be one pelrect for each camera. There is no relationship between the pelrects of two separate cameras.
Calibration Results	CogHandEyeCalibrationResults	Contains a list of CogHandEyeCalibrationResult objects, one for each camera.
FeatureCrspsMCamerasNPoses	CogFeatureCrspsMCamerasNPoses	The correspondence point pairs to be used for calibration.
Stage Pose Collection	CogTransform2DRigid[]	The sequence of UncorrectedHome2DFromStage2D poses that were used to move the stage when the images that provided the correspondence point pairs were acquired.

Outputs

Name	Type	Description
Calibration Timed Out	Booleon	This output will be true if the calibration operator times out.
Calibration Results	CogHandEyeCalibrationResults	Handeye Calibration results which contains all spaces transforms and stage validation information
Stage Rotation Compensation Coeffs	CogTransform2DRigid[]	
Calibration Result Cam0	CogHandEyeCalibration	Contains the calibration result for camera 0. Additional pins of this type will be available based on the Number Of Cameras property.

Properties



These properties are described as below in two groups:

- **Calibration Parameters**

Name	Type	Description
Theta Scaling Compensation	Boolean	
Home2D Unit Length Reference	Cognex.VisionPro.CalibFix. CogHandEyeHome2DUnitLengthReferenceConstants	Specifies how unit length in Home2D is established. Refer to the VisionPro CogHandEyeCalibrator documentation for more details
Is Moving Camera	Boolean	Indicates if the cameras are moving or stationary. Refer to the VisionPro CogHandEyeCalibrator documentation for more details.
Lens Distortion Mode	Cognex.VisionPro. CogLensDistortionModelConstants	The optical distortion model to be used in computing Raw2DFromCamera2D. Refer to the VisionPro CogHandEyeCalibrator documentation for more details.
Minimum Rotation Span	Double	The minimum required rotation angle span in degrees among all input motion-x-y-theta poses for motion capabilities that allow rotation. When executed, this block will throw an exception if the rotation angle span among all input motion-x-y-theta poses is less than this minimum requirement. Refer to the VisionPro CogHandEyeCalibrator documentation for more details.
Motion Capability	Cognex.VisionPro.CalibFix. CogHandEyeMotionCapabilityConstants	The capability of the motion rendering device to render the motion of the calibration target. Refer to the VisionPro CogHandEyeCalibrator documentation for more details.

- **Parameters**

Name	Type	Description
Number of Cameras	Int32	Sets/Gets the number of cameras supplying images to the feature extractor. This property has a minimum value of MinNumberOfCameras and a maximum value of MaxNumberOfCameras. The total number of Calibration Result CamN pins is equal to the Number Of Cameras property. The number of pins changes immediately upon a change to this property. Throws System.ArgumentOutOfRangeException: If the setter value is less than MinNumberOfCameras or more than MaxNumberOfCameras
Timeout Enabled	Boolean	Enable or disable timeout for calibration. Refer to the VisionPro CogHandEyeCalibrator documentation for more details.
Timeout Value	Double	The calibration time out value in milliseconds. Refer to the VisionPro CogHandEyeCalibrator documentation for more details.
Calibration Target Type		
Recalibrate	Boolean	The target that is used to generate the features needed to perform hand-eye calibration.

Published Methods

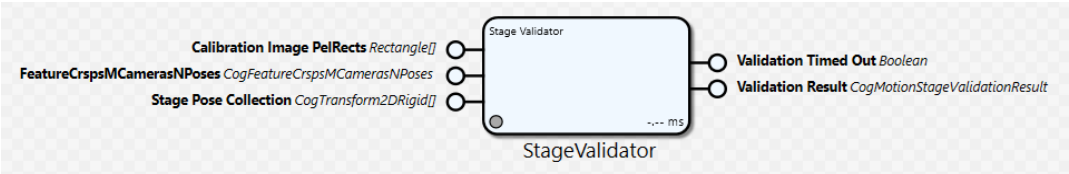
Void ResetBlock()

This method resets this block to a default state.

Stage Validator

The Motion Stage Validator Block performs motion stage validation. The purpose of this process is to verify that a stage moves to its commanded poses (X, Y, Theta), and to characterize certain types of systematic errors in the observed motion. This block is a Cognex Designer wrapper for the VisionPro CogMotionStageValidator operator and implements most of its functionality.

The VisionPro CogMotionStageValidator uses a fully configured CogHandEyeCalibrator object during execution. This block constructs a CogHandEyeCalibrator internally, but exposes its properties as Cognex Designer properties in order to help the user to configure it. All the properties of the CogHandEyeCalibrator object are exposed by the Motion Stage Validator Block except the MotionCapability property. The MotionCapability is inferred from the stage poses that are supplied as input.



General Information

Class name: HandEyeCalibrationBlock

Namespace: Cognex.Designer.AlignPlus.Calibration

Assembly: Cognex.Designer.AlignPlus.Calibration.dll

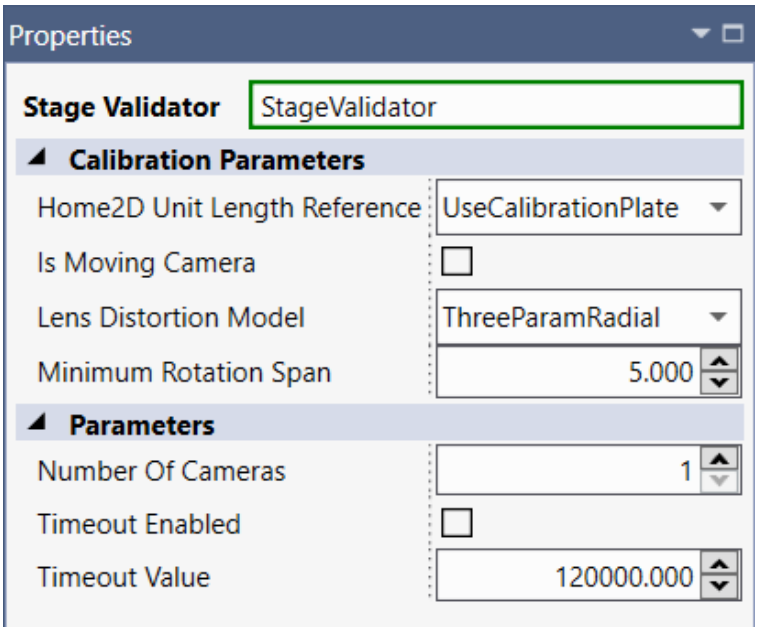
Inputs

Name	Type	Description
Calibration Image PelRects	System.Drawing.Rectangle[]	The pelrects used to extract the correspondence point pairs. These pelrects are used to define the camera coordinate system (i.e. Camera2D) to be at the center of the pelrect. There should be one pelrect for each camera. There is no relationship between the pelrects of two separate cameras.
FeatureCrspsMCamerasNPoses	Cognex.VisionPro.CogFeatureCrspsMCamerasNPoses	The correspondence point pairs to be processed.
Stage Pose Collection	Cognex.VisionPro.CogTransform2DRigid[]	The sequence of UncorrectedHome2DFromStage2D poses that were used to move the stage when the images that provided the correspondence point pairs were acquired.

Outputs

Name	Type	Description
Validation Timed Out	Boolean	This output will be true if the validation operation times out.
Validation Result	Cognex.VisionPro.CalibFix.CogMotionStageValidationResult	Contains the results of the motion stage validation process.

Properties



These properties are described as below in two groups:

- **Calibration Parameters**

Name	Type	Description
Home2D Unit Length Reference	Cognex.VisionPro.CalibFix. CogHandEyeHome2DUnitLengthReferenceConstants	Specifies how unit length in Home2D is established. Refer to the VisionPro CogHandEyeCalibrator documentation for more details
Is Moving Camera	Boolean	Indicates if the cameras are moving or stationary. Refer to the VisionPro CogHandEyeCalibrator documentation for more details.
Lens Distortion Model	Cognex.VisionPro. CogLensDistortionModelConstants	The optical distortion model to be used in computing Raw2DFromCamera2D. Refer to the VisionPro CogHandEyeCalibrator documentation for more details.
Minimum Rotation Span	Double	The minimum required rotation angle span in degrees among all input motion-x-y-theta poses for motion capabilities that allow rotation. When executed, this block will throw an exception if the rotation angle span among all input motion-x-y-theta poses is less than this minimum requirement. Refer to the VisionPro CogHandEyeCalibrator documentation for more details.

- **Parameters**

Name	Type	Description
Number of Cameras	Int32	Sets/Gets the number of cameras supplying images to the feature extractor. This property has a minimum value of MinNumberOfCameras and a maximum value of MaxNumberOfCameras. Throws System.ArgumentOutOfRangeException: If the setter value is less than MinNumberOfCameras or more than MaxNumberOfCameras
Timeout Enabled	Boolean	Enable or disable timeout for validation. Refer to the VisionPro CogMotionStageValidator documentation for more details.
Timeout Value	Double	The validation time out value in milliseconds. Refer to the VisionPro CogMotionStageValidator documentation for more details.

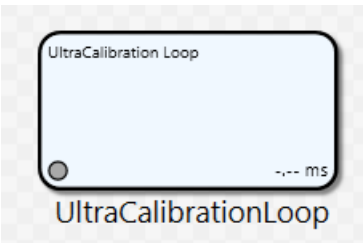
Published Methods

Void ResetBlock()

This method resets this block to a default state.

UltraCalibration Loop

The UltraCalibration process models non-linearities in the mapping from commanded motion device pose to actual motion device pose. The calibration process is very similar to the Hand-eye calibration process. The UltraCalibrationLoopBlock serves the same purpose as the HandEyeCalibrationLoopBlock. It has a set of properties that specify the motion range of the motion device, and properties that influence the number of poses for the calibration target. During the calibration process the block enables the vision application to move the calibration target to random positions within the specified range. The recommended X, Y, Theta Step for UltraCalibration is 4-7.



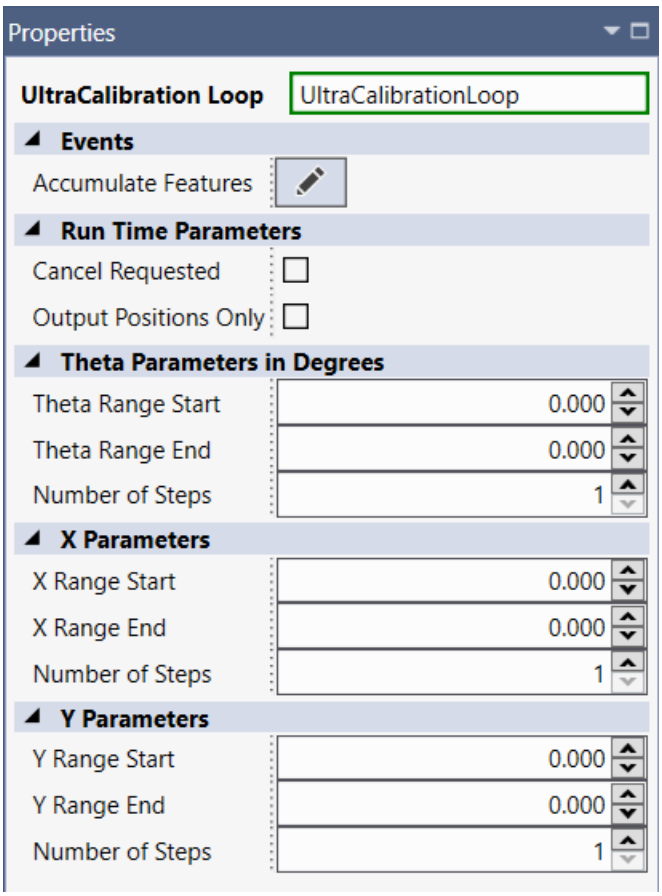
General Information

Class name: UltraCalibrationLoopBlock

Namespace: Cognex.Designer.AlignPlus.Calibration

Assembly: Cognex.Designer.AlignPlus.Calibration.dll

Properties



Name	Type	Description
Theta Range Start	Double	Get/Set the starting position for the Theta axis (in degrees), used when generating stage poses on the circular grid.
Theta Range End	Double	Get/Set the ending position for the Theta axis (in degrees), used when generating stage poses on the circular grid.
Number of Steps on Theta axis	Double	Get/Set the number of samples when rotating the stage along the Theta axis of the circular grid. This property has a minimum value of 1. Throws System.ArgumentOutOfRangeException: If the setter value is less than one.
X Range Start	Double	Get/Set the starting position for the X axis (in mm), used when generating stage poses on the circular grid.
X Range End	Double	Get/Set the ending position for the X axis (in mm), used when generating stage poses on the circular grid.
Number of Steps on X axis	Double	Get/Set the number of samples when rotating the stage along the X axis of the circular grid. This property has a minimum value of 1. Throws System.ArgumentOutOfRangeException: If the setter value is less than one.
X Range Start	Double	Get/Set the starting position for the Y axis (in mm), used when generating stage poses on the circular grid.

Name	Type	Description
X Range End	Double	Get/Set the ending position for the Y axis (in mm), used when generating stage poses on the circular grid.
Number of Steps on X axis	Double	Get/Set the number of samples when rotating the stage along the Y axis of the circular grid. This property has a minimum value of 1. Throws System.ArgumentOutOfRangeException: If the setter value is less than one.
Cancel Requested	Boolean	Gets/sets if a cancel was requested. If this property is set to true before or during execution, the looping is cancelled.

These ranges and steps can also be set via Calibration Loop UI control.

Accumulate Features Script

This block supplies the script point "AccumulateFeatures". During execution this script point will be called for each pose in the generated set of stage poses. The script point will be called with suitable arguments to implement a single-pose feature extractor system.

bool AccumulateFeatures (Int32 Index, Double X, Double Y, Double Theta)

Name	Type	Description
Index	int	The zero-based iteration count in the current calibration operation. The first call to this script point receives a zero, the second call a one, etc.
X	Double	The X value of stage position that was used to acquire the images in this iteration
Y	Double	The Y value of stage position that was used to acquire the images in this iteration
Theta	Double	The Theta value of stage position that was used to acquire the images in this iteration

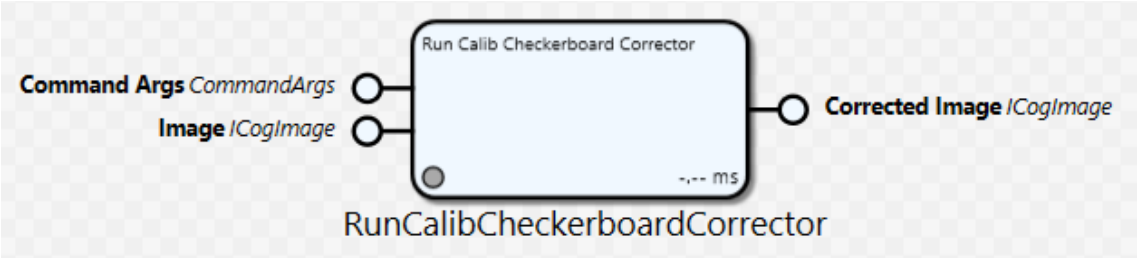
In case user wants to cancel the Loop in the middle, “CancelRequested” property need to be set as true via scripting, such as:

```
$Tasks.Task.CalibrationLoop.CancelRequested = true;
```

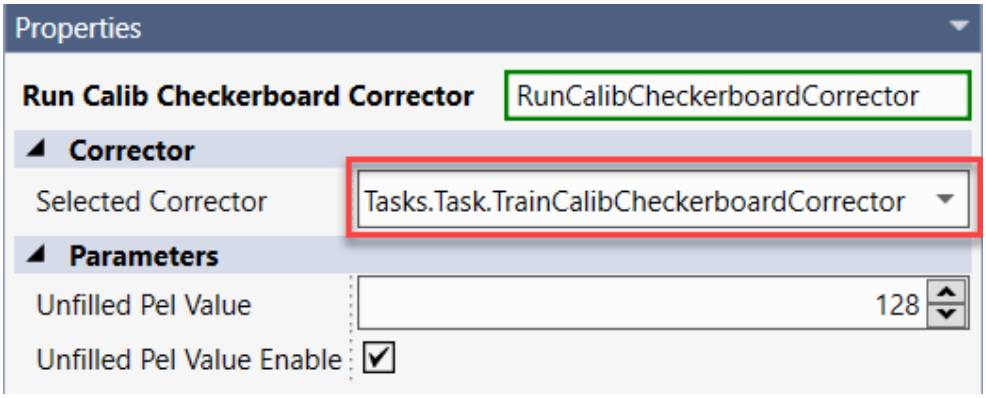
Image Corrector

Run Calib Checkerboard Corrector

The RunCalibCheckboardCorrector block holds a VisionPro CogCalibCheckerboardRunParams and refers to a TrainCalibCheckboardCorrector block. When executed, this block runs the referenced image corrector using the selected parameters.

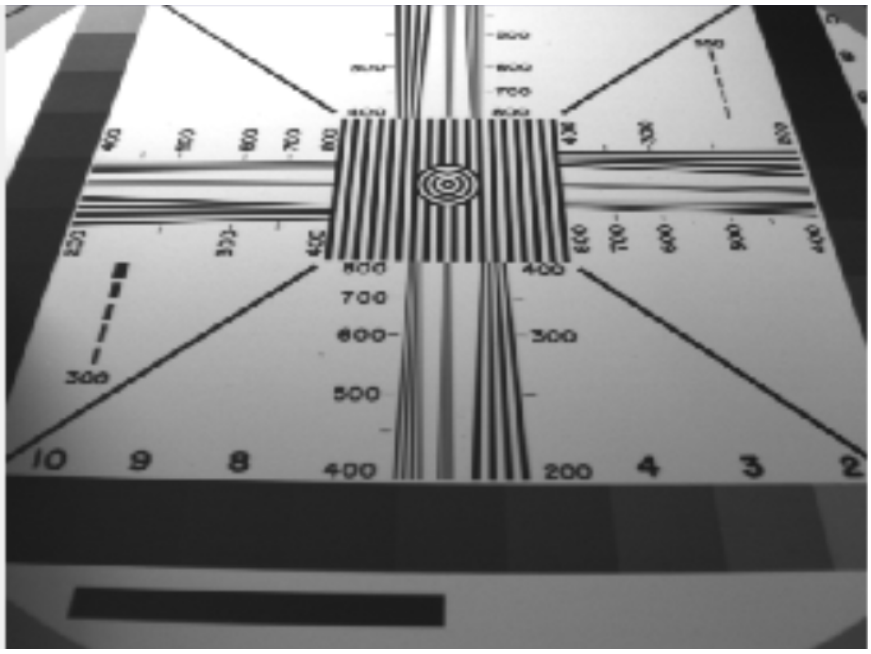


The referred TrainCalibCheckboardCorrector is selected in "Corrector" property.

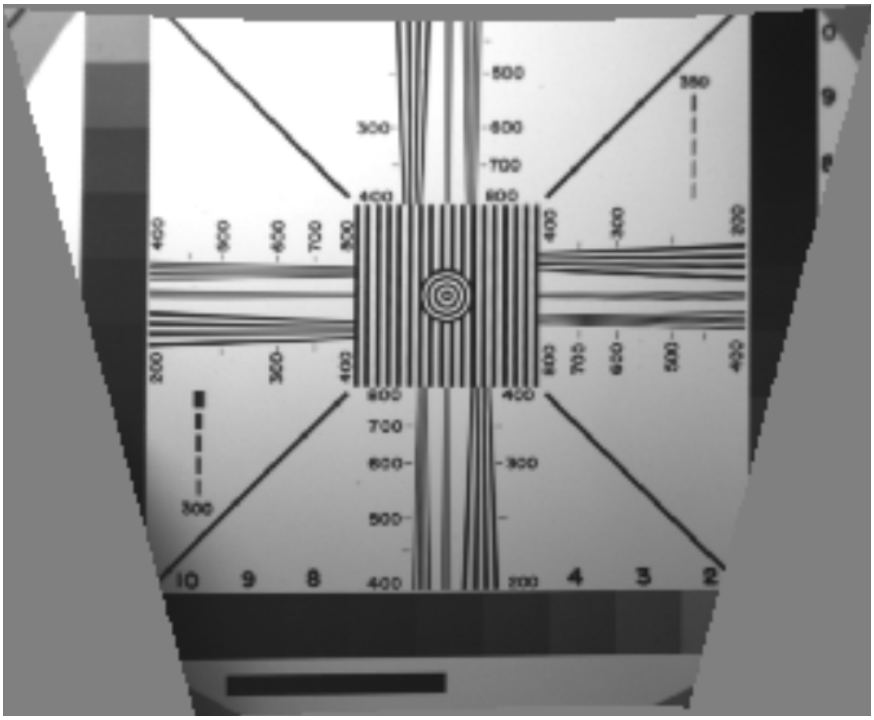


Here is an example of input raw image and its corrected image:

- Input image:



- Output image with graphics:



Note: In alignment application, it is not recommended to install camera at such a tilted angle which cause big perspective distortion in raw image as this will reduce the alignment accuracy.

General Information

Class name: RunCalibCheckerboardCorrector

Namespace: Cognex.Designer.AlignPlus.Calibration

Assembly: Cognex.Designer.AlignPlus.Calibration.dll

Inputs

Item	Type	Description
Image	ICogImage	Raw image acquired by that specific camera
Command Args	CommandArgs	Command information

Outputs

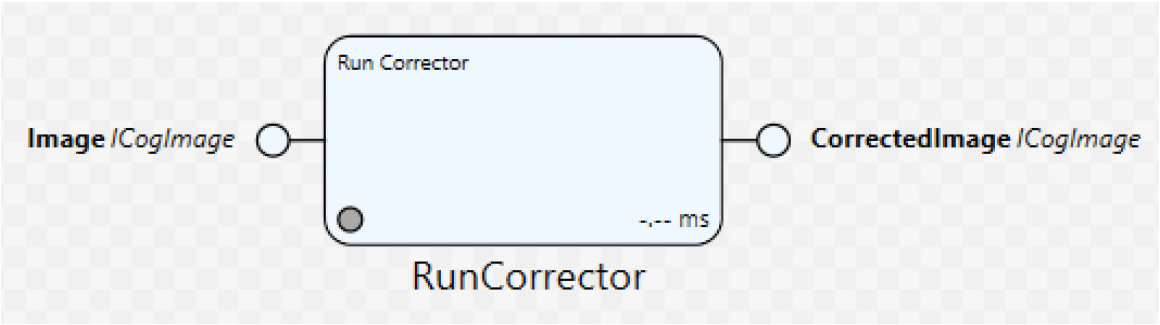
Item	Type	Description
Corrected Image	ICogImage	Corrected Image with Plate2D as default space

Properties

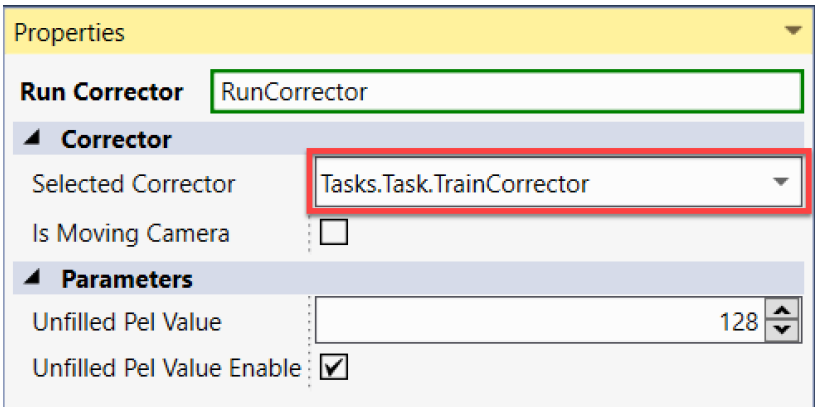
Name	Type	Description
Selected Corrector	String	Get/set the corrector (an instance of a TrainCalibCheckerboardCorrector block, generally in another sequence) to run when this block is executed.
Unfilled Pel Value Enable	Boolean	Get/set the unfilled pel value behavior. If true, unfilled pixels in the corrected image will be initialized using UnfilledPelValue. Otherwise unfilled pixels will be uninitialized. See the VisionPro CogCalibCheckboardRunParams.UnfilledPelValueEnabled property.
Unfilled Pel Value	Int32	Get/set the unfilled pixel value. If UnfilledPelValueEnable is true, unfilled pixels in the corrected image will be set to this value. See the VisionPro CogCalibCheckerboardRunParams.UnfilledPelValue property.

Run Corrector

The RunCorrector block holds a VisionPro CogCalibImageCorrectorRunParams and refers to a TrainCorrector block. When executed, this block runs the referenced image corrector using the selected parameters.



The referred TrainCorrector is selected in "Corrector" property.



General Information

Class name: RunCorrector

Namespace: Cognex.Designer.AlignPlus.ImageCorrector

Assembly: Cognex.Designer.AlignPlus.ImageCorrector.dll

Inputs

Item	Type	Description
Image	ICogImage	The input image to correct

Outputs

Item	Type	Description
CorrectedImage	ICogImage	The output corrected image

Properties

Name	Type	Description
Selected Corrector	String	Get/set the corrector (an instance of a TrainCorrector block, generally in another sequence) to run when this block is executed.

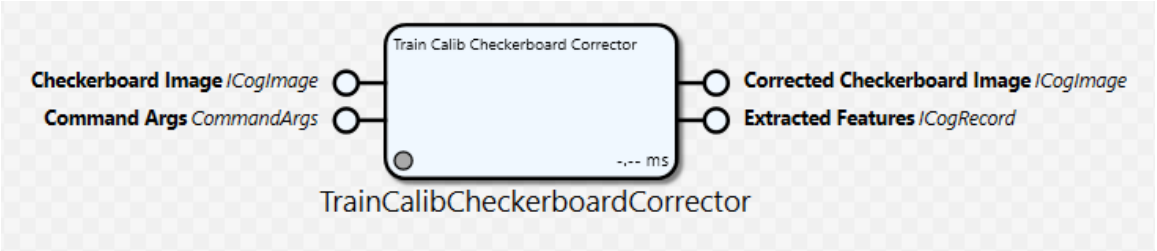
Name	Type	Description
Is Moving Camera	Boolean	Get/set if the corrector should be run for a moving camera configuration. If false, specifies a stationary camera configuration. This flag must match the IsMovingCamera property of the SelectedCorrector. If it does not, an exception will be thrown when this block is executed. If true, this block will show input pins for X, Y and Theta. These pins must be connected to inputs representing the current stage position.
Unfilled Pel Value Enable	Boolean	Get/set the unfilled pel value behavior. If true, unfilled pixels in the corrected image will be initialized using UnfilledPelValue. Otherwise unfilled pixels will be uninitialized. See the VisionPro CogCalibImageCorrectorRunParams.UnfilledPelValueEnabled property.
Unfilled Pel Value	Int32	Get/set the unfilled pixel value. If UnfilledPelValueEnable is true, unfilled pixels in the corrected image will be set to this value. See the VisionPro CogCalibImageCorrectorRunParams.UnfilledPelValue property.

Train Calib Checkerboard Corrector

The TrainCalibCheckerboardCorrector block holds a VisionPro CogCalibCheckerboardTool. When executed, this block runs checkerboard calibration on input image and outputs the corrected image as well as feature graphics.

The input image here must be image from one of the following calibration plates:

- StandardRectangles
- DataMatrix
- DataMatrixWithGridPitch
- DotGridAxes



General Information

Class name: TrainCalibCheckerboardCorrector

Namespace: Cognex.Designer.AlignPlus.Calibration

Assembly: Cognex.Designer.AlignPlus.Calibration.dll

Inputs

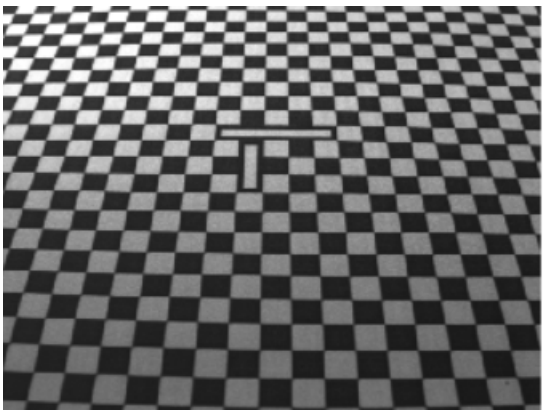
Item	Type	Description
Checkerboard Image	ICogImage	An image of the calibration target
Command Args	CommandArgs	Command for current task

Outputs

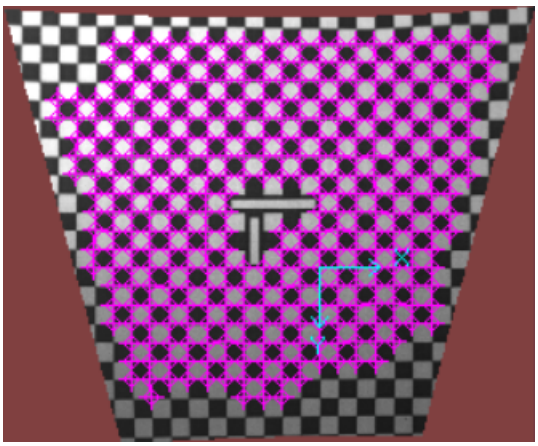
Item	Type	Description
Corrected Checkerboard Image	ICogImage	After training, the result of running image correction on the checkerboard image.
Extracted Features	ICogRecord	Features graphics extracted by CogCalibCheckerboardTool.

Here is an example of input raw image and its corrected image:

- Input image:

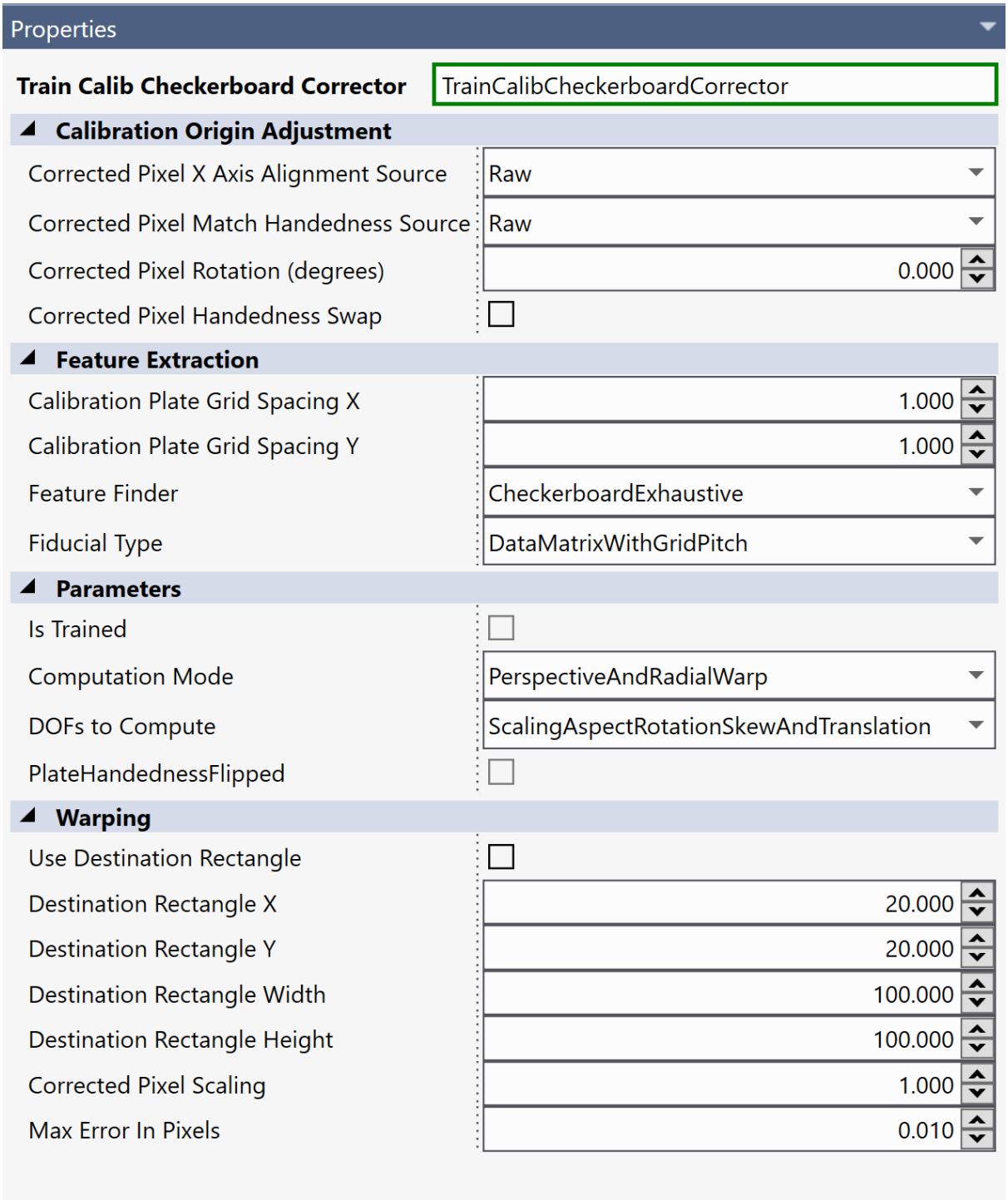


- Output image with graphics:



Note: In alignment application, it is not recommended to install camera at such a tilted angle which cause big perspective distortion in raw image as this will reduce the alignment accuracy.

Properties



These properties are described below in different groups:

- **Calibration Origin Adjustment**

Name	Type	Description
Corrected Pixel X Axis Alignment Source	Cognex.VisionPro.CalibFix.CogCalibCheckerboardAdjustmentSpaceConstants	Get/set the Corrected Pixel X Axis Alignment Source. • Plate(RawCalibred): during correction the image pixels will be rotated such that in the corrected image, the Home2D X-axis is oriented in the same direction as the corrected image X-axis (# space). • Raw(Uncalibred): there is no effect during image correction. Any additional rotation specified by the CorrectedPixelRotation property will be applied after this X-axis alignment is complete. If this property is changed, this corrector will become untrained.
Corrected Pixel Match Handedness Source	Cognex.VisionPro.CalibFix.CogCalibCheckerboardAdjustmentSpaceConstants	Specifies whether the output handedness is the same in Raw2D space(Uncalibrated space), or in Plate2D space (RawCalibrated space).
Corrected Pixel Rotation (degrees)	Double	Get/set the rotation, in degrees, of the corrected image. This property allows you to rotate the image during correction. See the VisionPro CogCalibCheckboardTool.Calibration.OwnedWarpParams.WarpRotation property. If this property is changed, this corrector will become untrained.
Corrected Pixel Handedness Swap	Boolean	Get/set the handedness of the corrected space. If this property is changed, this corrector will become untrained.

- **Feature Extraction**

Name	Type	Description
Calibration Plate Grid Spacing X	Double	Gets/sets the physical units of grid pitch along the x-axis of the raw calibrated space (see the Calibration and Fixturing section of VisionPro documentation). It is the distance between any two adjacent checker vertices whenever the line joining them is parallel to the x axis of calibration plate.
Calibration Plate Grid Spacing Y	Double	Gets/sets the physical units of grid pitch along the y-axis of the raw calibrated space (see the Calibration and Fixturing section of VisionPro documentation). It is the distance between any two adjacent checker vertices whenever the line joining them is parallel to the y axis of calibration plate.
Feature Finder	Cognex.VisionPro.CalibFix.CogCalibCheckerboardFeatureFinderConstants	Gets/Sets the algorithm used to find vertices on the calibration plate (refer to the CogCalibCheckerboardFeatureFinderConstants documentation in the VisionPro documentation).
Fiducial Type	Cognex.VisionPro.CalibFix.CogCalibCheckerboardFiducialConstants	Gets/Sets the style of fiducial mark present on the calibration plate (refer to the CogCalibCheckerboardFiducialConstants documentation in the VisionPro documentation).

- **Parameters**

Name	Type	Description
Is Trained	Boolean	Get if this corrector block is currently trained.
Computation Mode	Cognex.VisionPro.CalibFix.CogCalibFixComputationModeConstants	Gets/Sets the modes for computing calibration and fixturing transformations (refer to the CogCalibFixComputationModeConstants documentation in the VisionPro documentation).
DOFs to Compute	Cognex.VisionPro.CalibFix.CogCalibCheckerboardDOFConstants	This enumeration specifies which degrees of freedom will be allowed when computing the best-fit linear transformation between the uncalibrated points and the raw calibrated points. It is used only when the ComputationMode is linear.
PlateHandednessFlipped	Boolean	Specifies the handedness of the calibrated space. • False: the calibrated space will have the same handedness as the raw (unadjusted) calibrated space. • True: it will have the opposite handedness.

- **Warping**

Name	Type	Description
Use Destination Rectangle	Boolean	Get/set a boolean indicating if the "DestinationRectangle" properties will be used during training. The destination rectangle specifies which pixels will be present in the corrected image. The "DestinationRectangle" properties are in '.' space. See the VisionPro CogCalibCheckerboardTool.Calibration.OwnedWarpParams.WarpDestinationRectangle property. If UseDestinationRectangle is false (or not connected), all of the pixels in the corrected image will be output. Changes to this or any of the "DestinationRectangle" properties will cause this corrector to become untrained.
Destination Rectangle X	Double	Get/set the X origin of the destination rectangle. See the UseDestinationRectangle property.
Destination Rectangle Y	Double	Get/set the Y origin of the destination rectangle. See the UseDestinationRectangle property.
Destination Rectangle Width	Double	Get/set the width of the destination rectangle. See the UseDestinationRectangle property.
Destination Rectangle Height	Double	Get/set the height of the destination rectangle. See the UseDestinationRectangle property.
Corrected Pixel Scaling	Double	Get/set an additional scaling factor to be applied during correction. See the VisionPro CogCalibCheckerboardTool.Calibration.OwnedWarpParams.WarpScaling property. If this property is changed, this corrector will become untrained.
Max Error In Pixels	Double	Get/set the maximum error allowed during image correction, measured in pixels of the training image. See the VisionPro CogCalibCheckerboardTool.Calibration.OwnedWarpParams.WarpMaxErrorInPixels property.

Methods

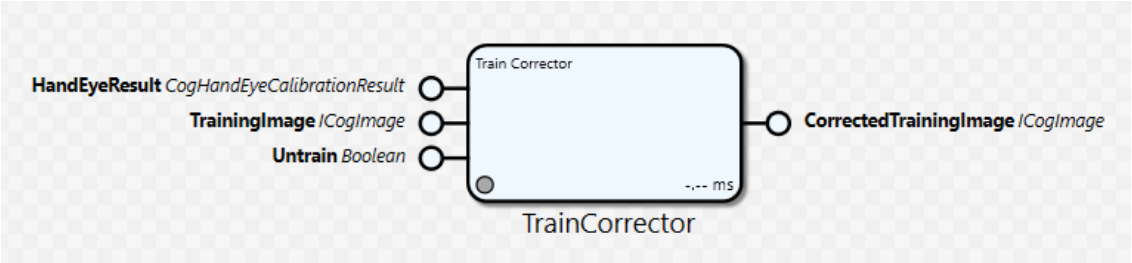
Void Uncalibrate()

Untrains this image corrector. This function has no effect if the corrector is already untrained.

Train Corrector

The TrainCorrector block holds a VisionPro CogCalibImageCorrector. When executed, this block uses its inputs to train the image corrector.

To run a trained image corrector, use the RunCorrector block.



General Information

Class name: TrainCorrector

Namespace: Cognex.Designer.AlignPlus.ImageCorrector

Assembly: Cognex.Designer.AlignPlus.ImageCorrector.dll

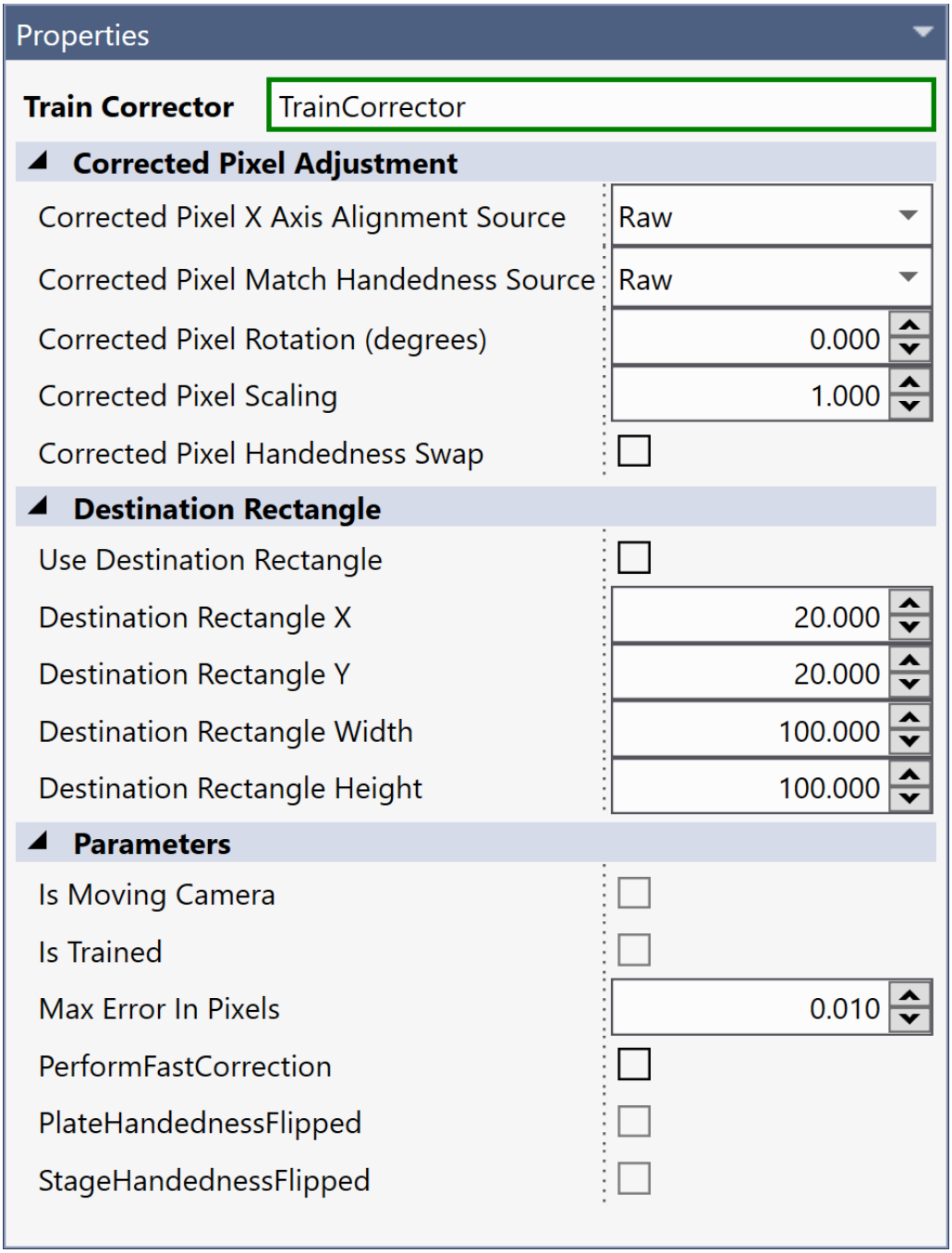
Inputs

Item	Type	Description
HandEyeResult	CogHandEyeCalibrationResult	Hand-eye Calibration result for specific camera
TrainingImage	ICogImage	A reference image to use during training.
Untrain	Boolean	If true, this corrector will be untrained when executed in a sequence. If false or unconnected, this corrector will be trained when executed.

Outputs

Item	Type	Description
CorrectedTrainingImage	ICogImage	After training, the result of running the image corrector on the TrainingImage.

Properties



These properties are described below in different groups:

- **Corrected Pixel Adjustment**

Name	Type	Description
Corrected Pixel X Axis Alignmen Source	Cognex.Designer.AlignPlus.ImageCorrector.CorrectorXAxisAlignmentSource	Get/set the source whose X axis will be aligned to the X axis of the corrected image • Plate: during correction the X axis of the corrected image is aligned to the X axis of the calibration plate • Raw: During correction the X axis of the corrected image is aligned to the X axis of the raw image • Stage: During correction the X axis of the corrected image is aligned to the X axis of the stage If this property is changed, this corrector will become untrained.
Corrected Pixel Match Handedness Source	Cognex.Designer.AlignPlus.ImageCorrector.CorrectorMatchHandednessSource	Allows the specification of the coordinate system whose handedness will be the same as the handedness of the corrected image. • Raw- The corrected image will have the same handedness as the raw image (Raw2D) • Plate - The corrected image will have the same handedness as the calibration plate (Plate2D) • Stage - The corrected image will have the same handedness as the stage (Stage2D)
Corrected Pixel Rotation (degrees)	Double	Get/set the rotation, in degrees, of the corrected image. This property allows you to rotate the image during correction. See the VisionPro CogCalibImageCorrector.CorrectedPixelRotation property. If this property is changed, this corrector will become untrained.
Corrected Pixel Scaling	Double	Get/set an additional scaling factor to be applied during correction. See the VisionPro CogCalibImageCorrector.CorrectedPixelScaling property. If this property is changed, this corrector will become untrained.
Corrected Pixel Handedness Swap	Boolean	Get/set the handedness of the corrected space. See the VisionPro CogCalibImageCorrector.SwapCorrectedHandedness property. If this property is changed, this corrector will become untrained.
Is Trained	Boolean	Get if this corrector block is currently trained.
Is Moving Camera	Boolean	Get if this corrector block was trained with a moving camera hand-eye result. Returns true if so, or false if this block was trained with a stationary camera hand-eye result. Returns false if this block is not trained.
Max Error In Pixels	Double	Get/set the maximum error allowed during image correction, measured in pixels of the training image. See the VisionPro CogCalibImageCorrector.MaxErrorInPixels property.

- **Destination Rectangle**

Name	Type	Description
Use Destination Rectangle	Boolean	Get/set a boolean indicating if the "DestinationRectangle" properties will be used during training. The destination rectangle specifies which pixels will be present in the corrected image. The "DestinationRectangle" properties are in '.' space. See the VisionPro CogCalibImageCorrector.DestinationRectangle property. If UseDestinationRectangle is false (or not connected), all of the pixels in the corrected image will be output. Changes to this or any of the "DestinationRectangle" properties will cause this corrector to become untrained
Destination Rectangle X	Double	Get/set the X origin of the destination rectangle. See the UseDestinationRectangle property.
Destination Rectangle Y	Double	Get/set the Y origin of the destination rectangle. See the UseDestinationRectangle property.
Destination Rectangle Width	Double	Get/set the width of the destination rectangle. See the UseDestinationRectangle property.
Destination Rectangle Height	Double	Get/set the height of the destination rectangle. See the UseDestinationRectangle property.

- **Parameters**

Name	Type	Description
Is Moving Camera	Boolean	Get if this corrector block was trained with a moving camera hand-eye result. Returns true if so, or false if this block was trained with a stationary camera hand-eye result. Returns false if this block is not trained.
Is Trained	Boolean	Get if this corrector block is currently trained.
Max Error In Pixels	Double	Get/set the maximum error allowed during image correction, measured in pixels of the training image. See the VisionPro CogCalibImageCorrector.MaxErrorInPixels property.
PerformFastCorrection	Boolean	When true, instead of using the corrector to correct the image, the input image is passed without any correction. The Raw2DFromHome2D transform in hand-eye calibration result to compute an approximate imageFromClient transform
PlateHandednessFlipped	Boolean	Specifies the handedness of the Plate space. • False: the calibrated space will have the same handedness as the raw (unadjusted) calibrated space. • True: it will have the opposite handedness.
StageHandednessFlipped	Boolean	Specifies the handedness of the Stage space. • False: the Stage space will have the same handedness as the raw (unadjusted) calibrated space. • True: it will have the opposite handedness.

Methods

Void Untrain()

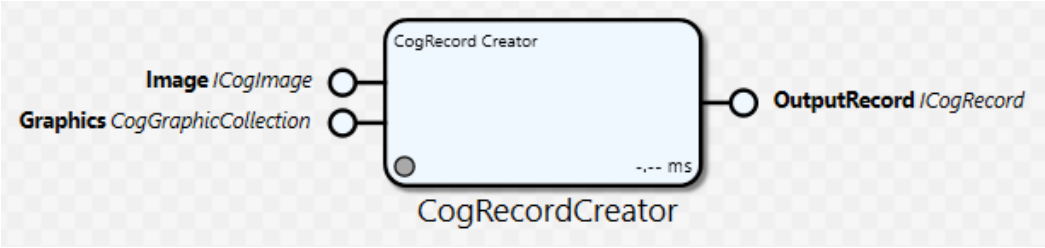
Untrains this image corrector. This function has no effect if the corrector is already untrained.

Utilities

CogRecord Creator

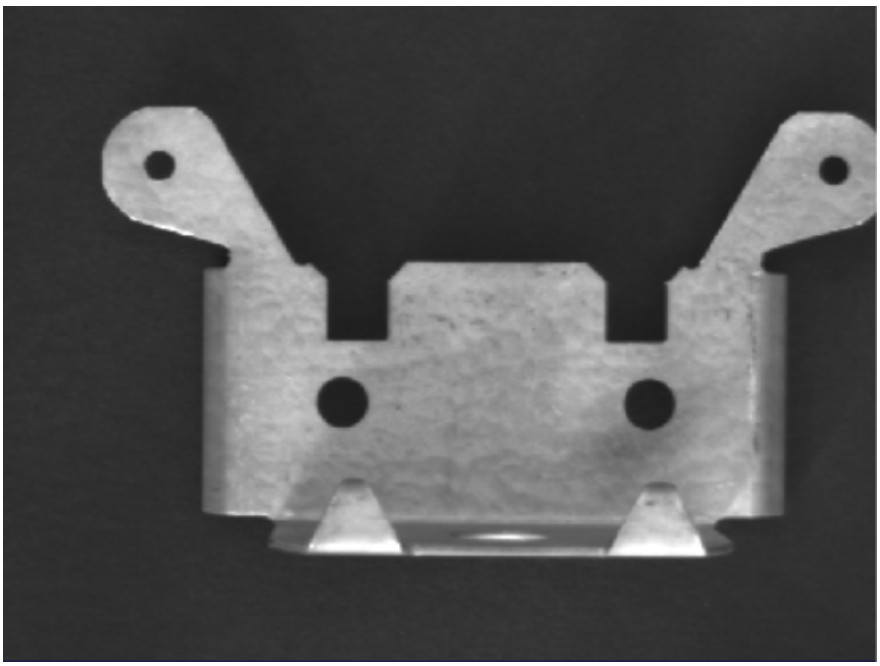
When executed, the CogRecordCreator block creates a VisionPro CogRecord object from an image and any number of VisionPro CogGraphicCollections.

The graphics from all input CogGraphicCollections are added to the output CogRecord, such that this block can be used to merge graphics from multiple sources into a single CogRecord for display. When used with no CogGraphicCollection inputs, a CogRecord containing just the input image is created. This is commonly used to clear graphics from a display.

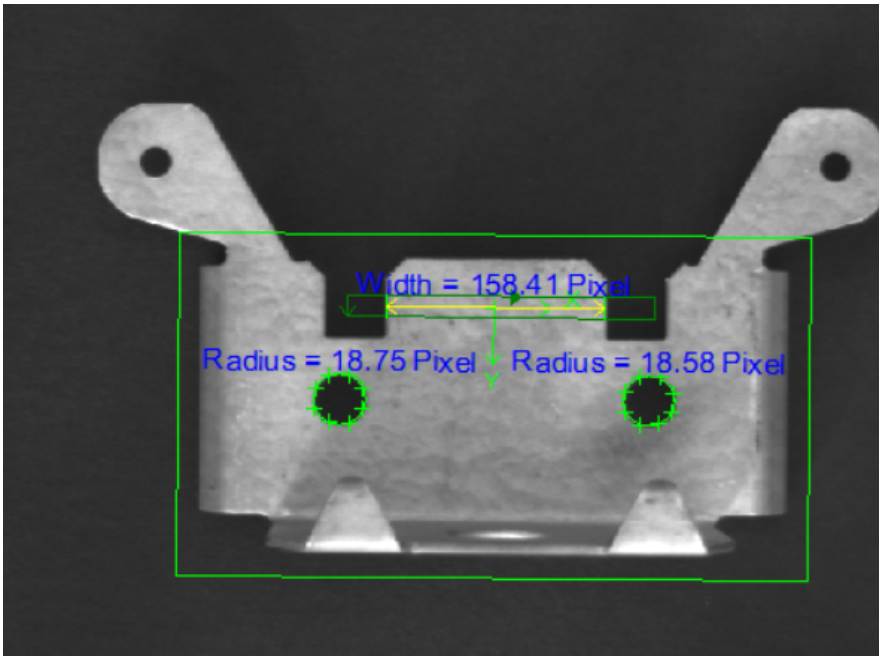


Here is an example to show the difference between ICogImage and ICogRecord:

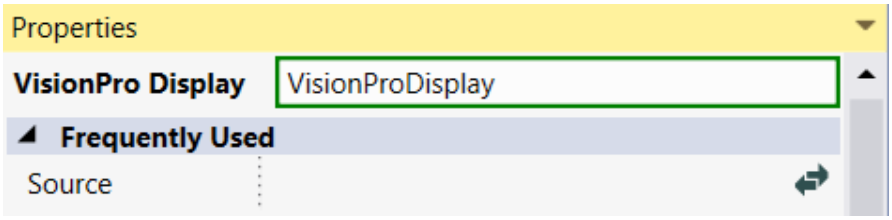
ICogImage:



ICogRecord



CogDisplay or CogRecordDisplay on HMI can either choose ICogImage or ICogRecord to display, this can be customized by changing Display's "Source" property.



General Information

Class name: CogRecordCreatorBlock

Namespace: Cognex.Designer.AlignPlus.Utils

Assembly: Cognex.Designer.AlignPlus.Utils.dll

Inputs

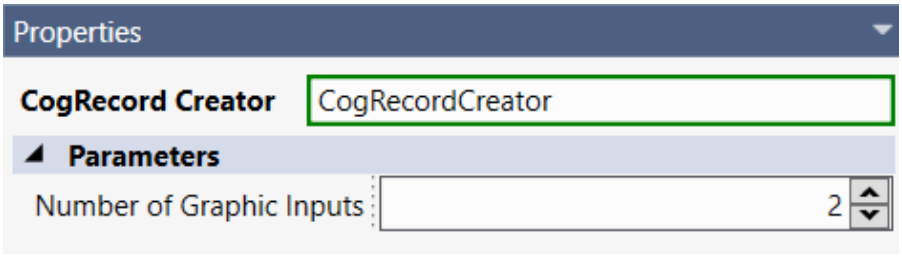
Name	Type	Description
Image	ICogImage	The input image
Graphics	CogGraphicCollection	A CogGraphicCollection to add to the output record.

Outputs

Name	Type	Description
OutputRecord	ICogRecord	The output CogRecord object

Properties

Additional pins of this type are created based on the Number of Graphic Inputs property.

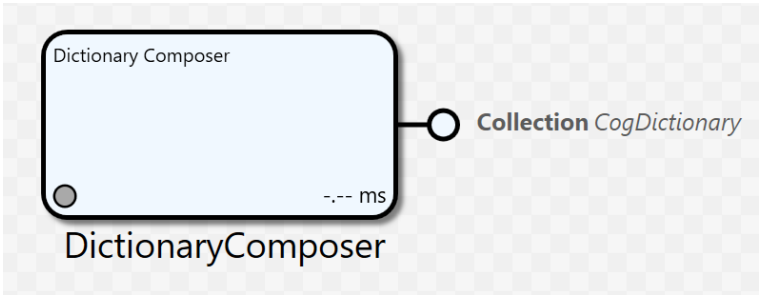


Name	Type	Description
Number of Graphic Inputs	Int32	Set/get the number of input graphic collections. This block will create an input pin for each graphic collection. Must be >= 0 and <= 16. The default is 1. If the value on a graphic input pin is non-null, then the given graphic collection is added to the output record. If the pin is unconnected or has a value of null, then it does not contribute to the output record. Throws System.ArgumentOutOfRangeException: If the setter value is less than one or greater than 16

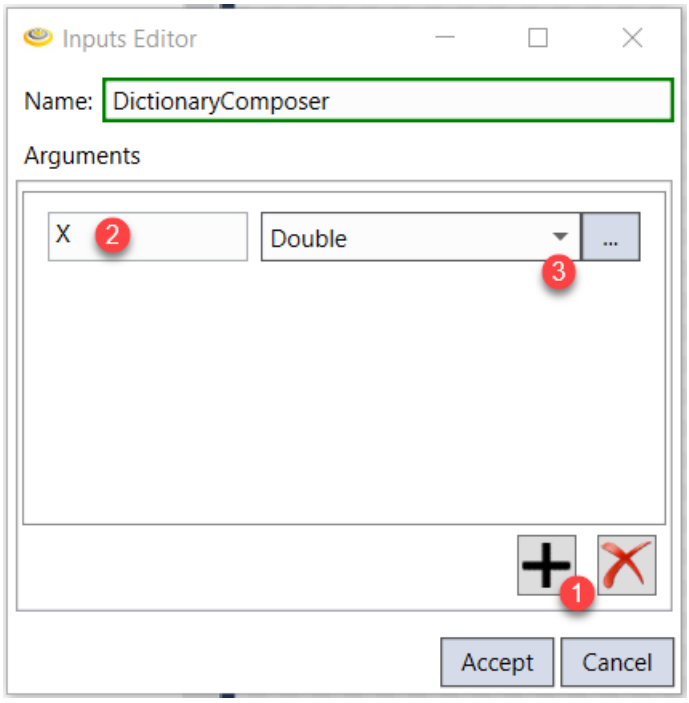
Dictionary Composer

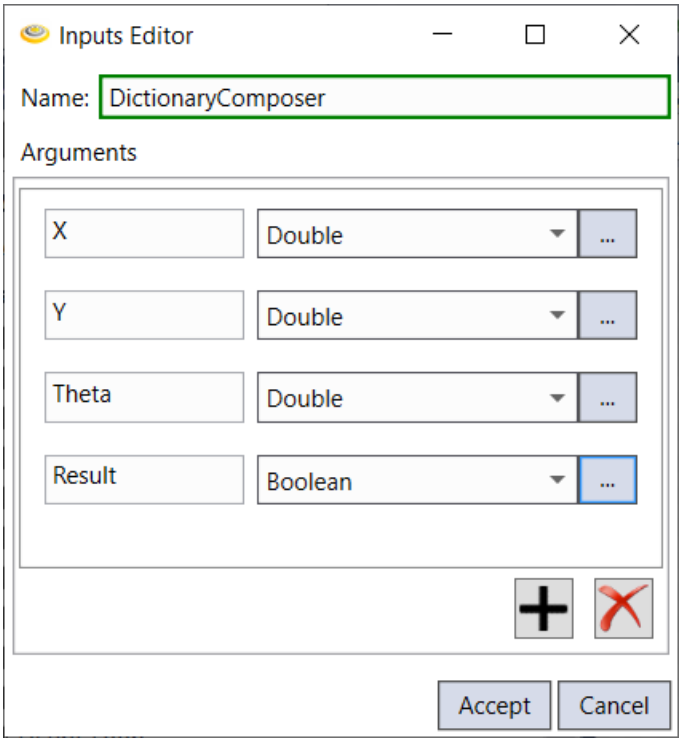
Dictionary Composer is a tool block that compiles a set of input data into a Dictionary object.

The input pin names, quantity, and types need to be configured by user, those input pin names will be Keys in the output dictionary, their values will be the data under corresponding Keys.

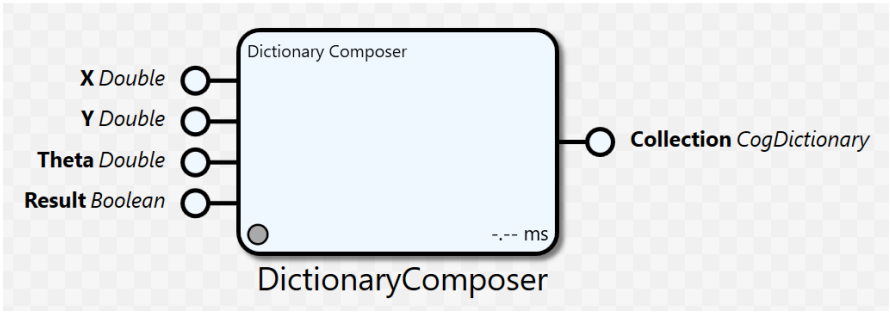


Double click this block to enter into edit mode and add input pins:



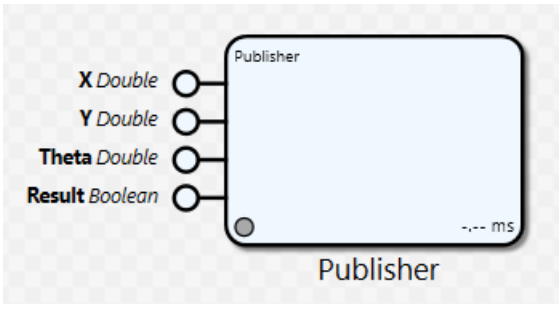


After editing, those elements will appear as input pins ready to be connected to data sources.



Publisher

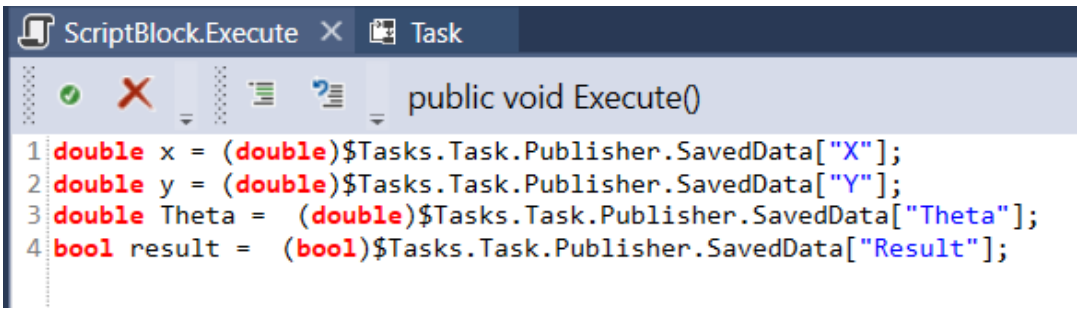
Publisher compiles multiple input data into a CogDictionary object and make it public so that it can be referred by a Subscriber in any task or visited in any script.



Here are three major features of Publisher:

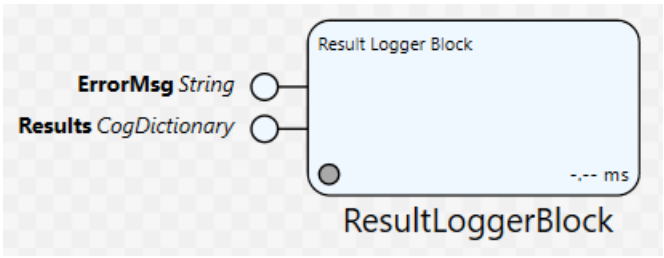
1. Publisher doesn't generate data by itself, but publicize other's outputs.
2. Publisher's data is saved as a dictionary named "SavedData".

3. Publisher's data can be accessed through scripting or Subcriber.



Result Logger Block

Result Logger Block feed data and file name to Multifile Log Writer to be logged as CSV file.

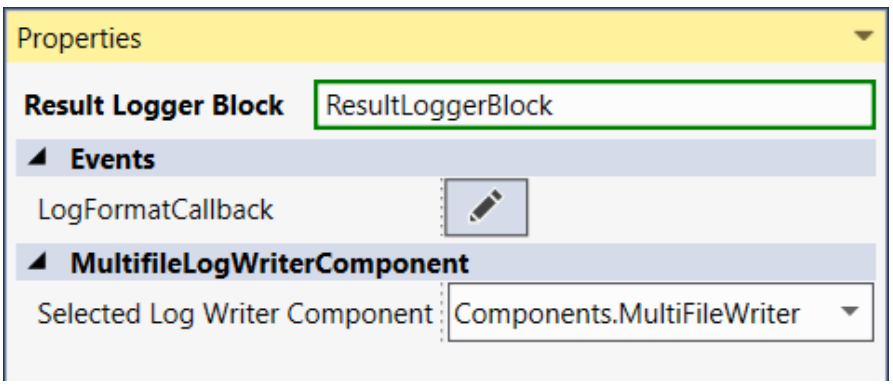


Inputs

Name	Type	Description
ErrorMsg	string	error message
Results	CogDictionary	Data needs to be written to CSV file, organized in CogDictionary structure

Properties

A Multifile Log Writer component need to be chosen as its server in the property.



In LogFormatCallback, Result Logger Block can define the file name and the data to be written into that CSV file. Here is a sample code:

```
public void Execute(CogDictionary resultsDict, List<string> orderedResultKeys, string errorMsg, List<string> csvHeader, List<string> csvData)
{
    // Add Timestamp and Error Message to the front of the CSV record
    csvHeader.Add("TimestampUTC");
    csvData.Add(DateTime.UtcNow.ToString("yyyyMMdd HH:mm:ss.fff"));

    csvHeader.Add("ErrorMsg");
    csvData.Add(errorMsg == null ? "" : errorMsg);

    // Use our default object-to-CSV conversion function to process the remaining results.
    // Because dictionaries do not preserve key order, we iterate over "orderedResultKeys"
    foreach (String key in orderedResultKeys)
    {
        $LogHelper.AppendToCSVRecord(key, resultsDict[key], csvHeader, csvData);
    }
}
```

Here is an example of the results in CSV file:

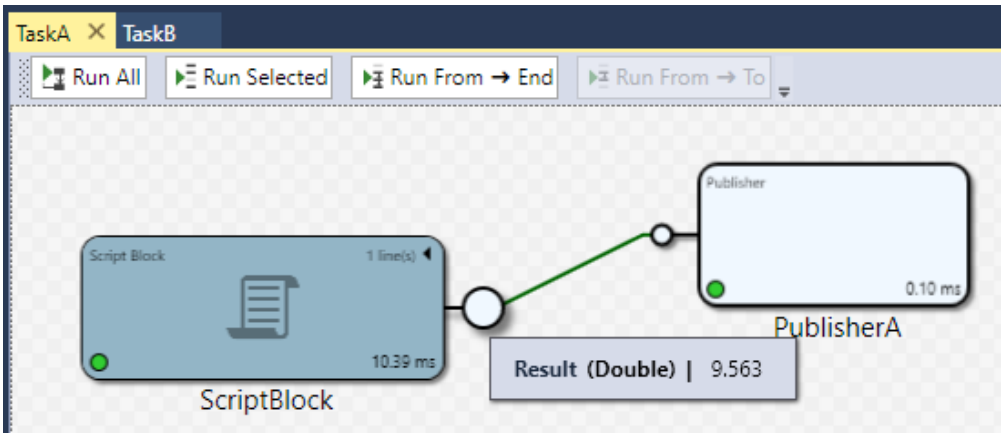
TimestampUTC	ErrorMsg	Absolute StageMotionX	Absolute StageMotionY	AbsoluteStageMotionThetaDegrees	Relative StageMotionX	Relative StageMotionY	RelativeStageMotion ThetaDegrees
20200215 11:49:53.913		2.163	-4.545	0.014	2.163	-4.545	0.014
20200215 12:21:02.782		2.163	-4.545	0.014	2.163	-4.545	0.014

Subscriber

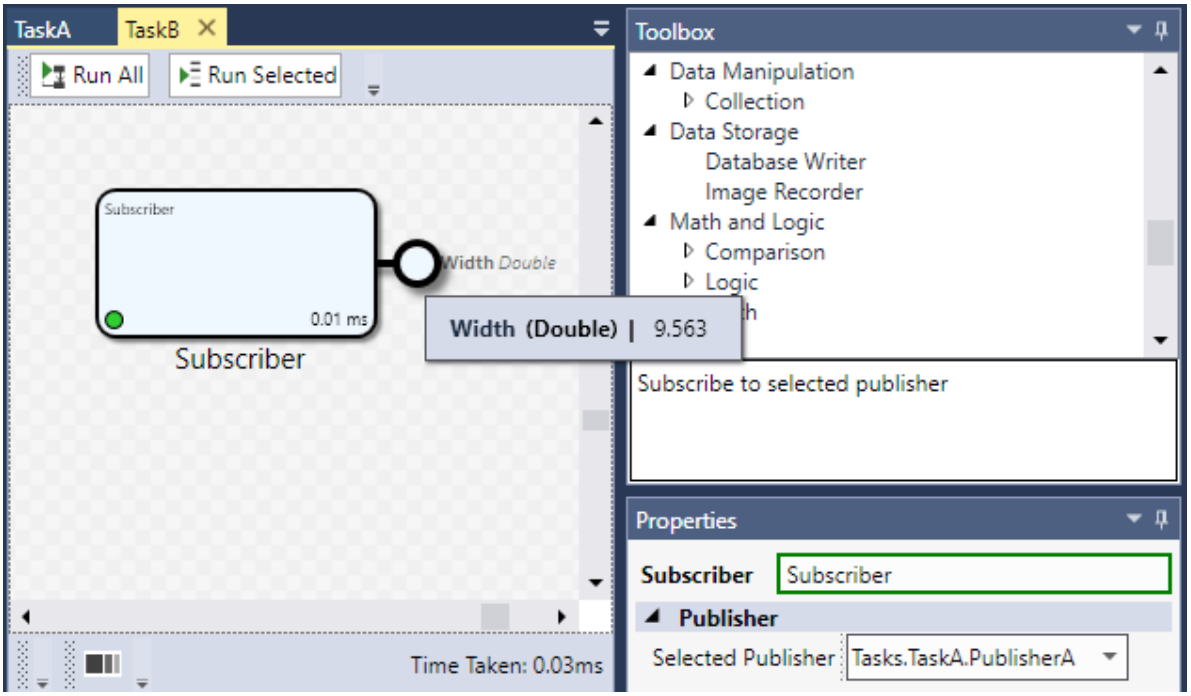
Subscriber is a tool block that refers to specific Publisher and generate the output pins which are the same with the referred Publisher's input pins. Subscriber and its referred Publisher does not need to be in the same task.

Here is an example of how Subscriber works:

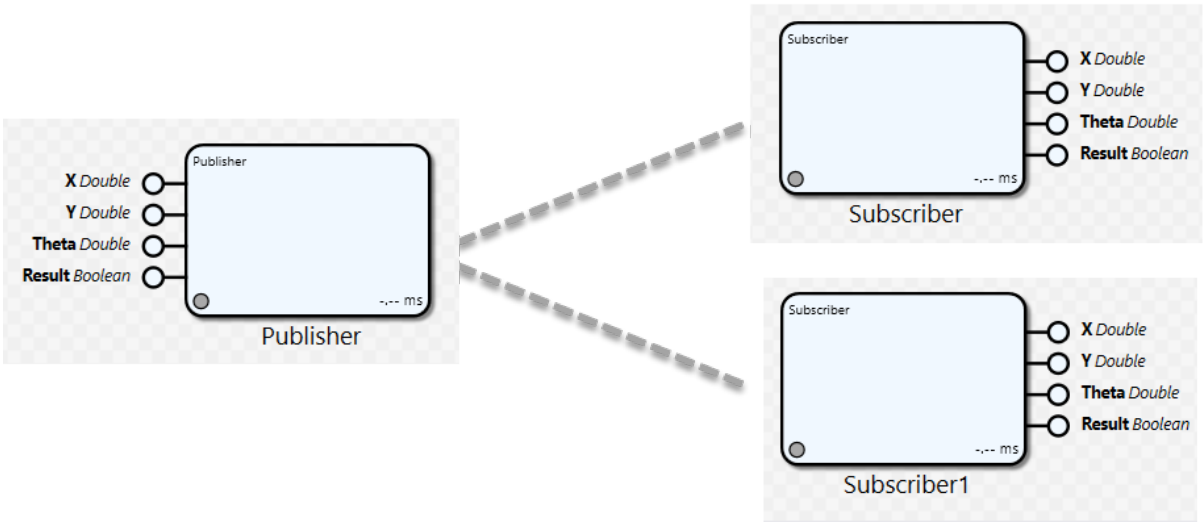
1. PublisherA publishes a result data in TaskA



2. Subscriber in TaskB refers to PulbisherA, such that it got the result data passed over as an output.



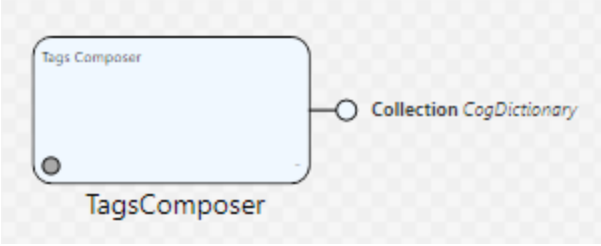
One Publisher can have multiple Subscribers or no Subscriber, Whereas Subscriber has to have one Publisher referred.



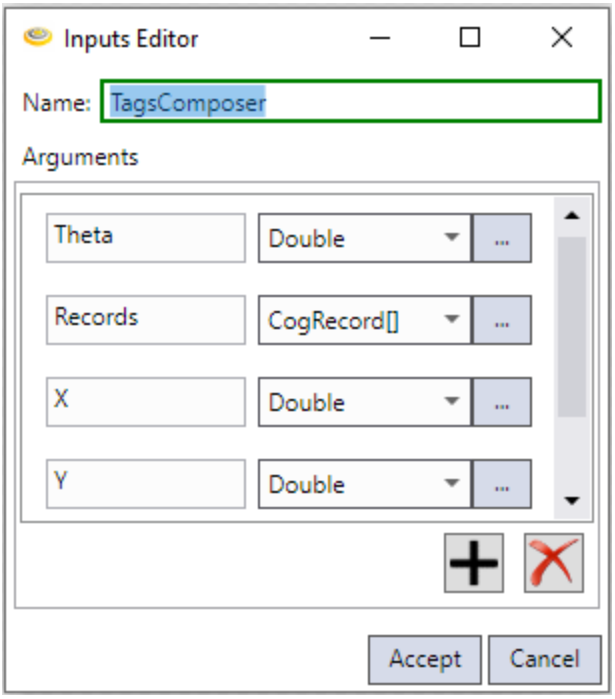
Publisher and Subscriber are convenient to share data among tasks as no tags need to be created to pass those data.

Tags Composer

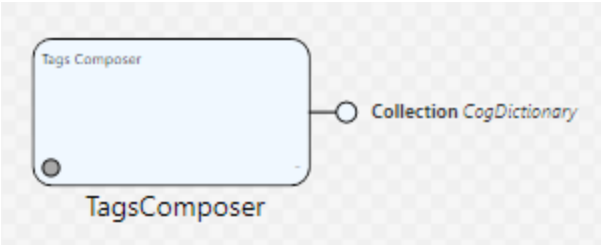
Tags Composer compiles elements added by user together and outputs a CogDictionary object, each element is considered as a tag within Tag Composer. These tags' values can be read and written in scripts, or by HMI controllers that have the compatible data type.



Double click this block to enter edit mode and add tag elements. The element can be any type.



After editing, the block has no change in appearance.

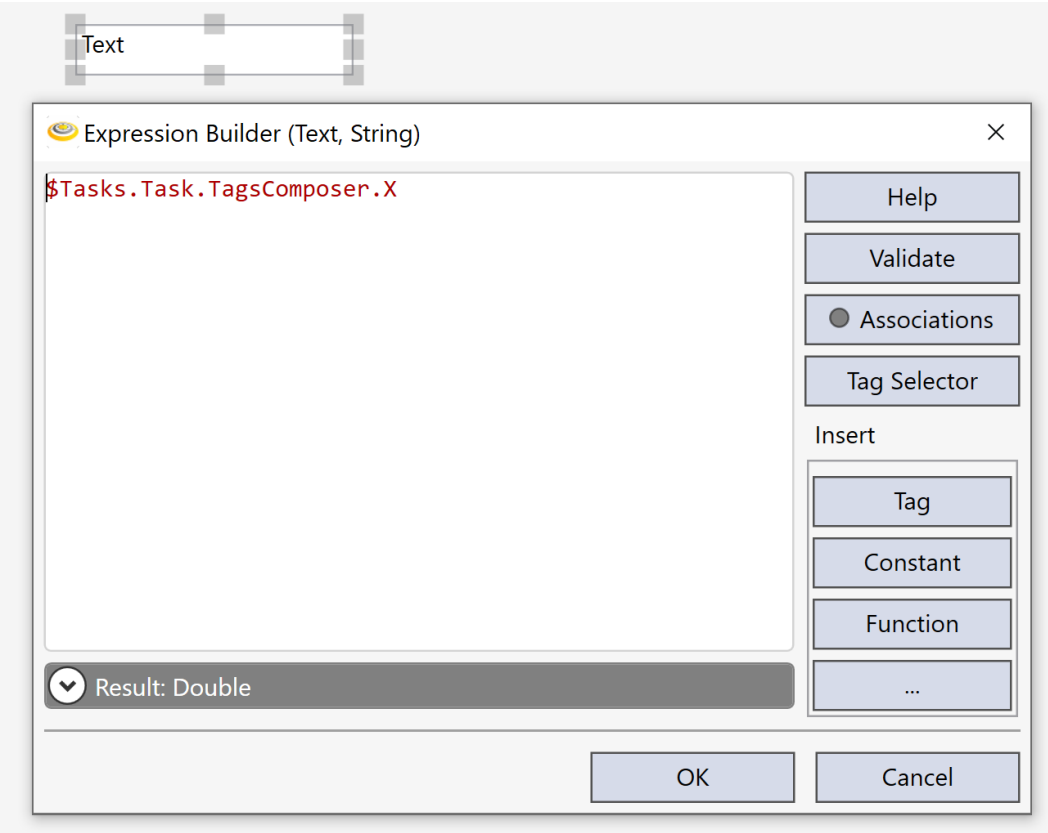


Here are two examples of how a Tag Composer is being used:

1. Change tag element in script:

   $Tasks.Task.TagsComposer.X = 9.923;

2. Refer the element in Text property of a text box:



Note: Once a new Tag Composer is used, it's TagValues should be added and saved in a recipe. Otherwise, its data will get lost after the program is closed.

How To

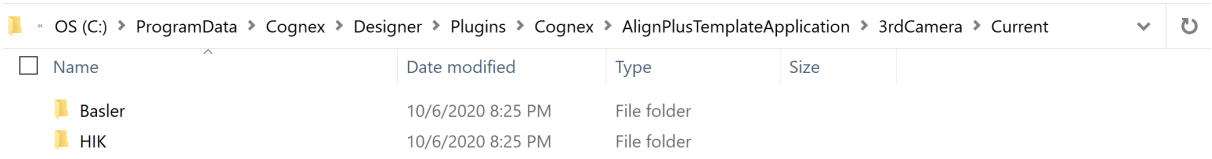
How To ... Acquire

How to add 3rd party acquisition plugin

Besides Cognex CIC cameras and other VisionPro supported GigE cameras, AlignPlus also supports 3rd party camera image acquisition. However, the prerequisite is that 3rd party SDK to be installed, and specific acquisition plugins to be manually placed in a given location before launching AlignPlus program.

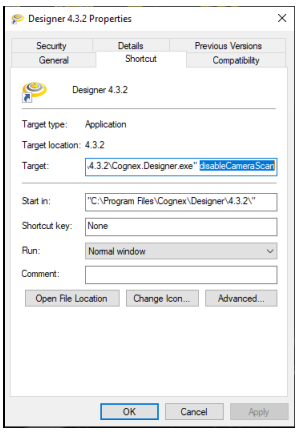
Follow the steps below to add the 3rd party acquisition plugin:

1. Install the camera's SDK (Please contact Cognex engineer for specific version of the SDK that goes with the acquisition plugin)
2. Copy the camera's acquisition plugin files to "C:\ProgramData\Cognex\Designer\Plugins\Cognex\AlignPlusTemplateApplication\3rdCamera\Current". Cognex Designer will only load the 3rd party acquisition plugins from this folder on start-up (Please contact Cognex engineer to get these acquisition plugin files).



   If certain camera brand is not needed, you can move its plugin files to "C:\ProgramData\Cognex\Designer\Plugins\Cognex\AlignPlusTemplateApplication\3rdCamera\Database", wherein plugins will be ignored by Designer.

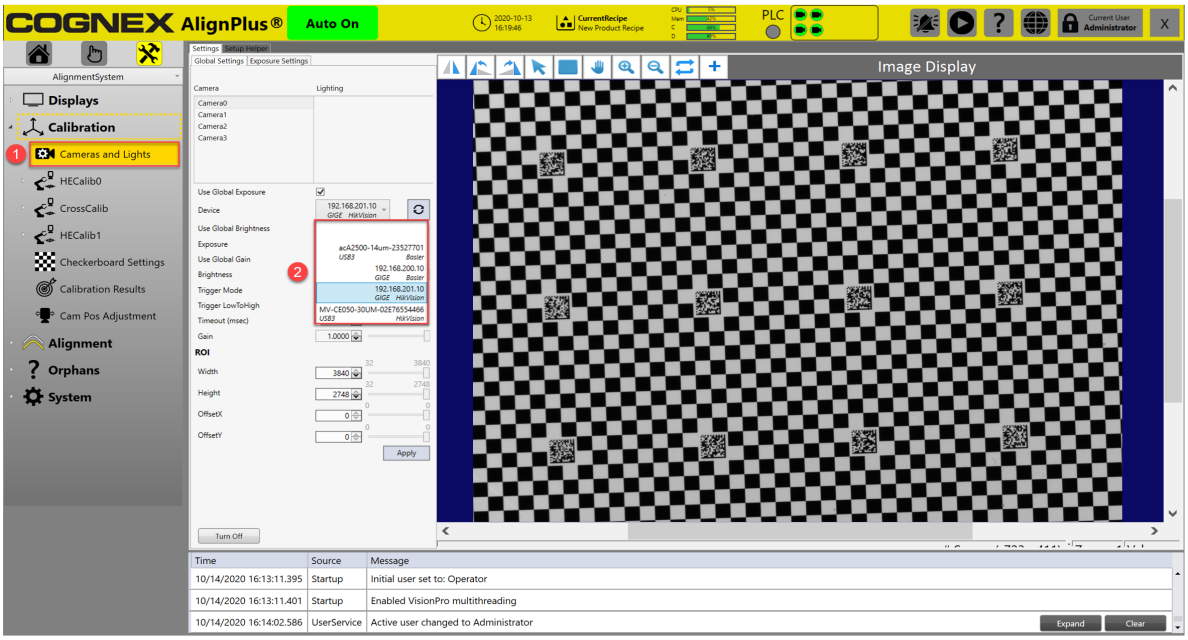
3. Add "**disableCameraScan**" launch option to Designer's shortcut to make Designer exclude the default VisionPro acquisition function and enable the 3rd party acquisition functions.



> **Note**: For USB3 and high-speed cameras, Cognex recommends the use of the 3rd party acquisition plugin for optimal performance.

4. Run Designer, open an AlignPlus Program and run it.

   From Device drop down list on "Cameras and Lights" page under Calibration category in setup mode of the program, you can find all the available 3rd party cameras with IP address/serial number and brand name information attached.



Note: For Basler GigE Cameras, if the brand name shown in the list is "Cognex", it means the default VisionPro acquisition function overrides Basler's acquisition plugin. If it shows "Balser" instead, it indicates that Basler's acquisition plugin is taking effect.

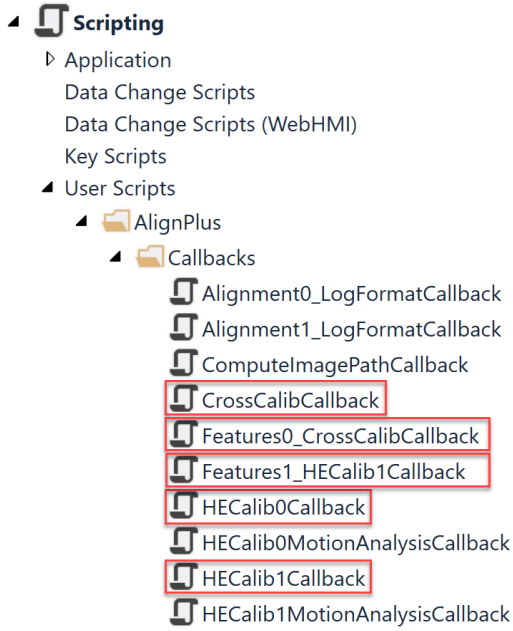
How To ... Control Devices

How to command stage to move

Before a task starts image acquisition, the stage should move to its target pose first. The control of this movement could be done by an external motion controller before it calls the task(motion guided) or done by the vision system which commands the stage to move before image acquisition starts(vision guided). This topic will walk through how the vision system commands the stage to move and waits until the movement is done before starting image acquisition.

Where to command stage to move

Every image acquisition block inside a calibration or feature finding task have one corresponding call back function which allows user to customize before or after image acquisition. These acquisition callback functions can be found in Scripting/User Scripts/AlignPlus/Callbacks category, named as <Task Name>Callback.



For example, the image acquisition callback function for a hand-eye calibration task "HECalib0" is "HECalib0Callback". The content of HECalib0Callback script is implemented by configuration wizard during application generation. It performs in three different acquisition states: InitializingAcquistion, ReadyToAcquire, and FinishedAquisition.

```
public void Execute(Cognex.Designer.AlignPlus.Utils.CommandArgs Command, string ExecutionState)
switch(ExecutionState)
{
    case "InitializingAcquisition":
        //
        if (Command.OtherArgs.ContainsKey("X") && Command.OtherArgs.ContainsKey("Y") && Command.OtherArgs.ContainsKey("ThetaInDegrees"))
        {
            var nextPos = new Dictionary<string, double>();
            nextPos["X"] = (double)Command.OtherArgs["X"];
            nextPos["Y"] = (double)Command.OtherArgs["Y"];
            nextPos["ThetaInDegrees"] = (double)Command.OtherArgs["ThetaInDegrees"];

            $Stage0Move(nextPos);
        }
        //
        break;
    case "ReadyToAcquire":
        var curPos = new Dictionary<string, double>();
        $Stage0GetPosition(curPos);

        if (Command.OtherArgs.ContainsKey("SubCommand1") && "AccumulateFeatures".Equals(Command.OtherArgs["SubCommand1"]))
        {
            Command.SetMotionParameters(curPos["X"], curPos["Y"], curPos["ThetaInDegrees"]);
        }
        else
        {
            Command.SetRawMotionParameters(curPos["X"], curPos["Y"], curPos["ThetaInDegrees"]);
            var stepID = -1;
            stepID = $MachineDescFunctions.GetStepIDFromComponentName("HECalib0");
            var compensatedRotation = $MachineDescFunctions.GetCompensatedRotationFromRawInDeg(stepID, curPos["ThetaInDegrees"]);
            Command.SetMotionParameters(curPos["X"], curPos["Y"], compensatedRotation);
        }
        break;
    case "FinishedAcquisition":
        break;
    default:
        // Raise alarm because of the unrecognized execution state.
        break;
}
```

Get target pose

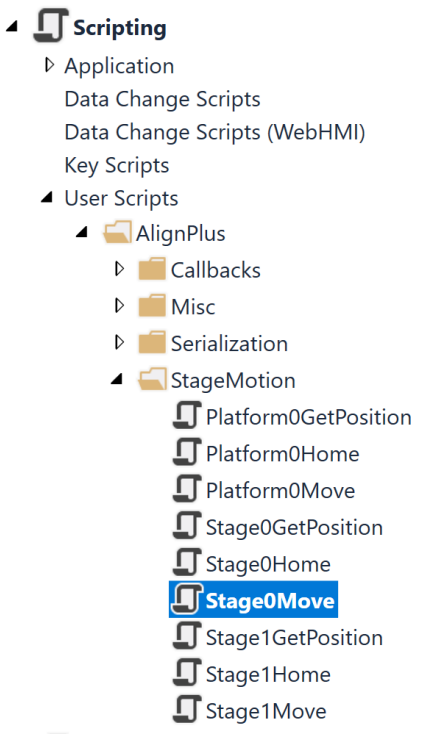
Command Stage to Move

After stage move is done, read stage's current pose

InitializingAcquistion

HECalib0Callback is first called with **InitializingAcquisition** state. In this state, the target pose of stage is extracted from Command, and sent to "Stage0Move" script. The specific code inside "Stage0Move" to command the stage to move needs to be implemented by an engineer according to specific stage type and communication protocol.

The stage move script lies under Scripting/User Scripts/AlignPlus/StageMotion.



ReadyToAcquire

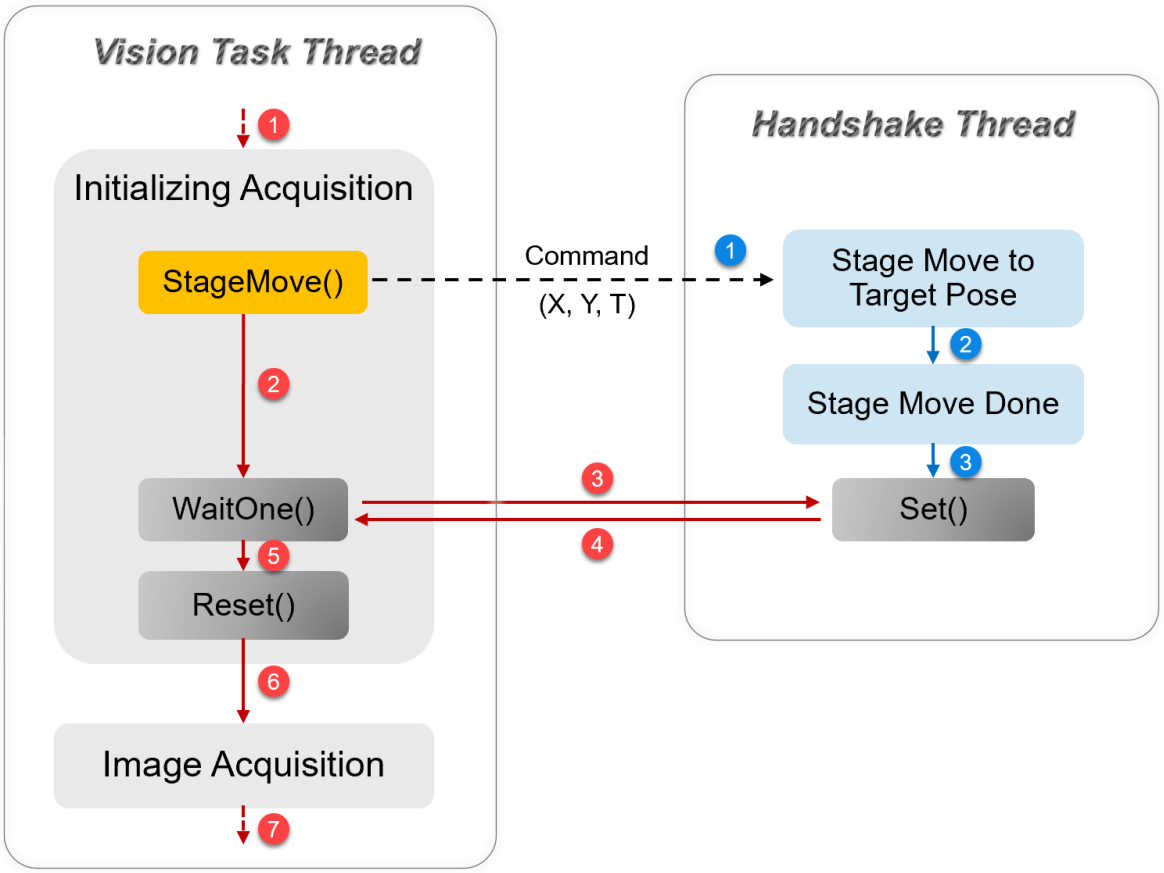
After the movement is done and before the camera starts to acquire image, HECalib0Callback is called again with **ReadyToAcquire** state. In this state, the vision system can acquire the stage's current pose through "Stage0GetPosition" script. "Stage0GetPosition" script here is also empty and requires an engineer to implement according to specific stage type and communication protocol.

FinishedAcquisition

After image acquisition is done, HECalib0Callback will be called with FinishedAcquisition state, where user can add some other custom actions if needed.

How to wait stage move done

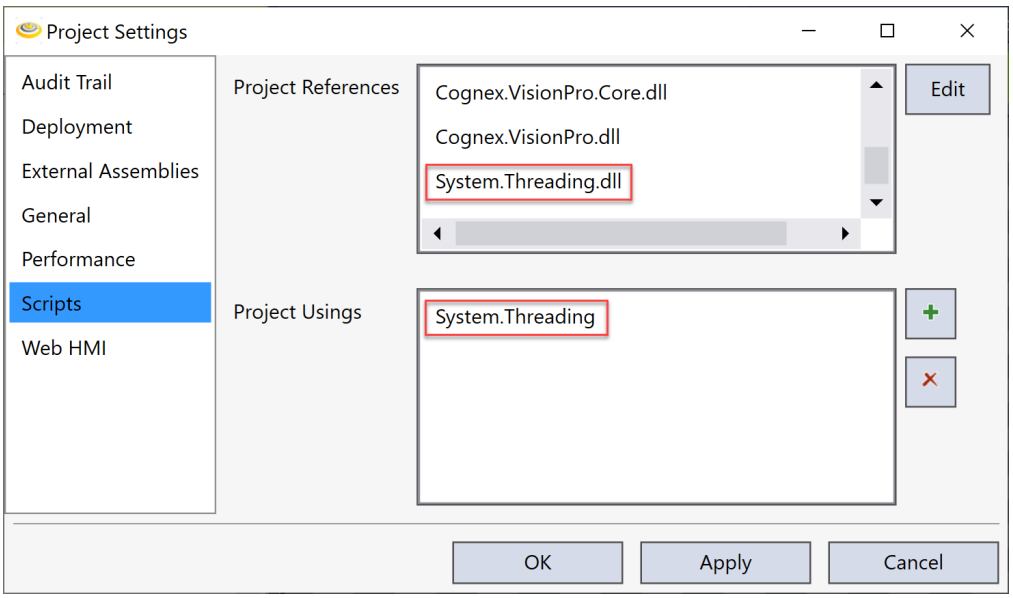
After the vision system send out stage move command, it needs to wait until the move is done before continuing image acquisition. A simple way to hold current vision task thread while waiting for the stage move done signal in another thread is to use a ManualResetEvent object. Here is the diagram that shows how a ManualResetEvent object works in controlling work flow between two threads.



In InitlizingAcqusition state of a vision task, after stage move command is sent, the ManualResetEvent object will block current vision task thread (without freezing the program) by calling **WaitOne()** event. After the requested stage finishes moving in the handshake thread, the same ManualResetEvent object will call a **Set()** event, through which the suspended vision task thread will be released, and continue to run the next line which makes ManualResetEvent object be **Reset()**. What follows is for the vision thread to continue image acquisition.

Below are the steps of adding and using a ManualResetEvent object.

1. Add "System.Threading.dll" to project references and "System.Threading" to space names in System/Settings/Scripts.



2. Create a tag named **HECalib.MoveDoneEvent** as ManualResetEvent object.
3. Initialize the tag in OnStartup script.

   $HECalib.MoveDoneEvent = new ManualResetEvent(false);
4. In InitializingAcquisition state of an image acquisition callback, set the tag to start a waiting event (it will return false if it is timed out).

   $HECalib.MoveDoneEvent.WaitOne($SystemData.HECalib.Timeout);
5. In the handshake thread, once the vision system receives the stage's move done signal, set the event.

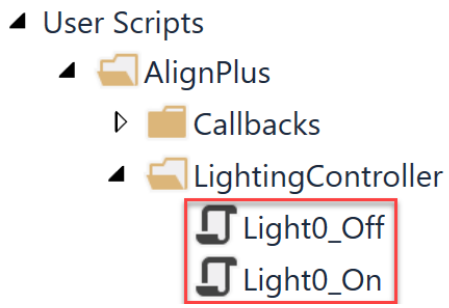
   $HECalib.MoveDoneEvent.Set();
6. Back to InitializingAcquisition state of image acquisition callback, reset the event.

   $HECalib.MoveDoneEvent.Reset();

Where to control light

If a lighting controller device is added in the wizard configuration, there will be a pair of light on and light off callbacks for user to control the actual lighting hardware on and off after the application is generated.

These callbacks are can be found in Scripting/User Scripts/LightingController category.



Light On

Light On function is called to turn on the light before an image acquisition that uses the light to acquire. For more information about how to enable a light for an image acquisition and how to set its intensity for each channel, please refer to Cameras

and Lights for Calibration on page 75 and Camera and Lights for Alignment on page 108.

The signature of Light On is as below:

public void *Light_On(Int32[] Channels, Int32[] Intensities)*

Input Name	Type	Description
Channels	Int32[ ]	The indexes of channels which should be turned on for current image acquisition
Intensities	Int32[ ]	The intensities of each channels which should be turned on

Light Off

Light Off function is called after associated image acquisition is finished. The signature of it is as below:

public void *Light_Off(Int32[] Channels*)

Input Name	Type	Description
Channels	Int32[ ]	The indexes of channels which should be turned off for current image acquisition

Note:
If there are multiple image acquisition positions for one vision task and a light device is used for each position, then during run time, the light will be turned on and off at each acquisition position.

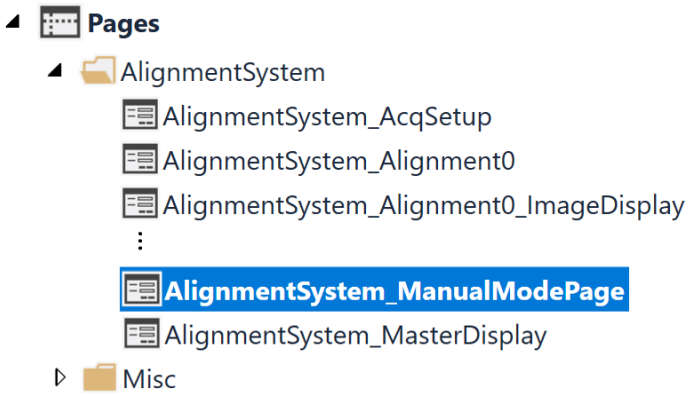
To control an actual light device, user needs to implement Light On and Light Off scripts based on specific light controller and its communication protocol.

How To ... Change UI

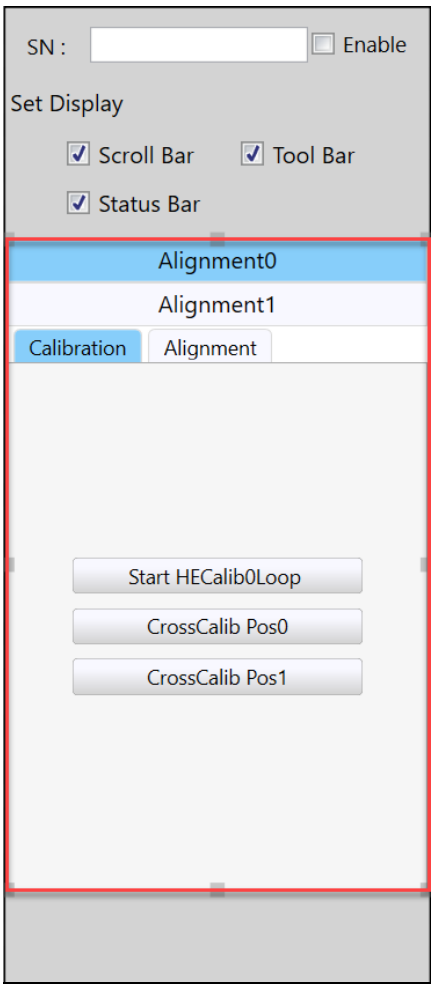
How to change manual button properties

If the size of the manual button should be changed to fit its text display, or its text font size should be adjusted, you can modify them through the properties of manual trigger HMI. Follow the steps below to change manual button properties:

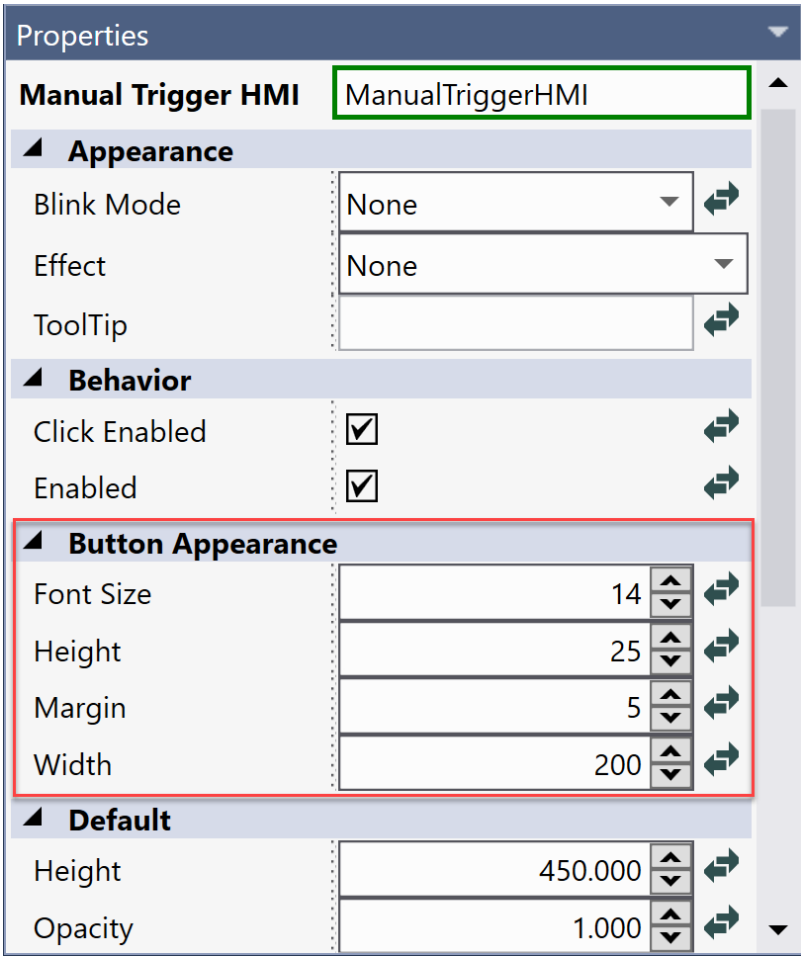
1. Under edit mode of the program, open "Alignment_ManualModePage" within "AlingmentSystem" page category



2. Select the Manual Trigger HMI controller



3. From its properties, find "Button Appearance" group where you can see font size, height, margin and width properties of buttons.

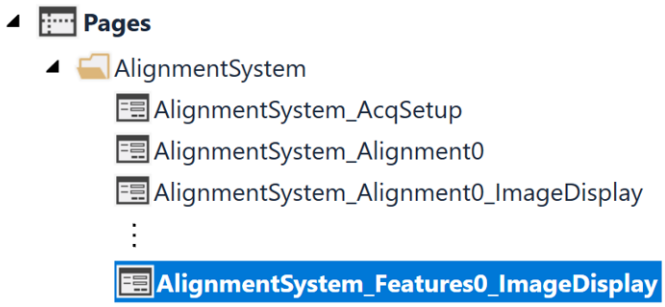


4. Change these property values to make them suit requirements. The changes of these values will have effects on all the manual buttons to keep them unified.

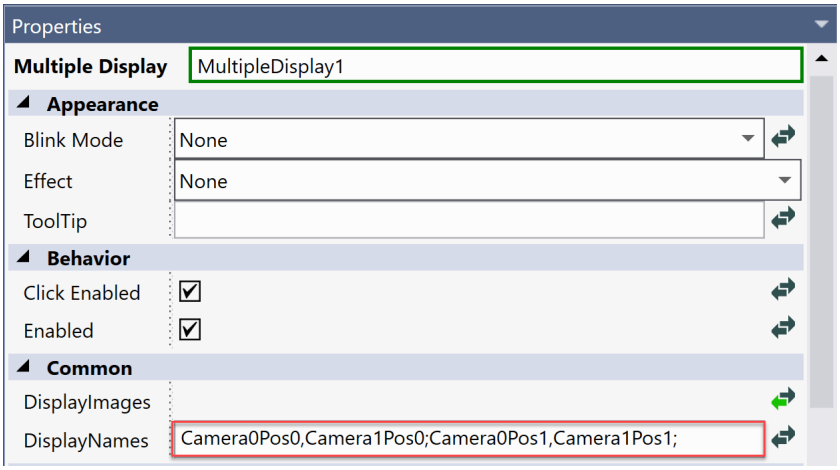
How to add custom graphics on image display

Some applications may require extra graphics to be added onto default image display, this topic will guide you the process though an example of adding labels and markers of features' coordinates on feature finding displays.

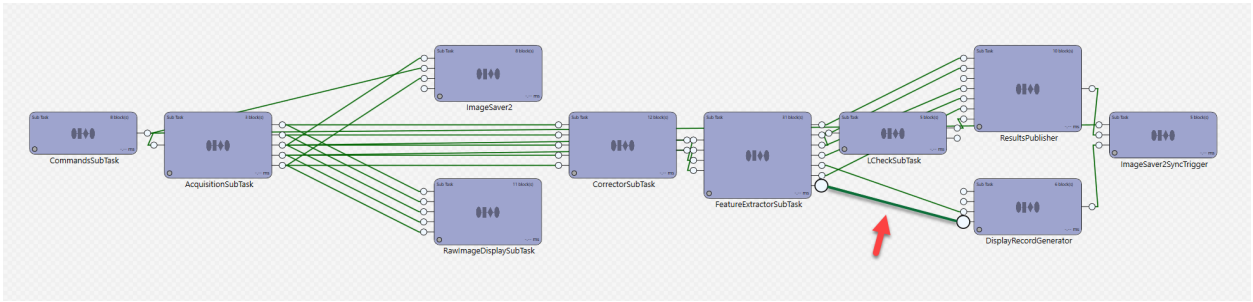
Each image display on AlignPlus is a Multiple Display control which displays multiple VisionPro records at the same time. Each record has its own unique name in its multiple display. In this example, the feature finding multiple display has four records in it: Camera0Pos0, Camera1Pos0, Camera0Pos1, Camera1Pos1. You can find these names by opening the image display page and check its "DisplayNames" property.



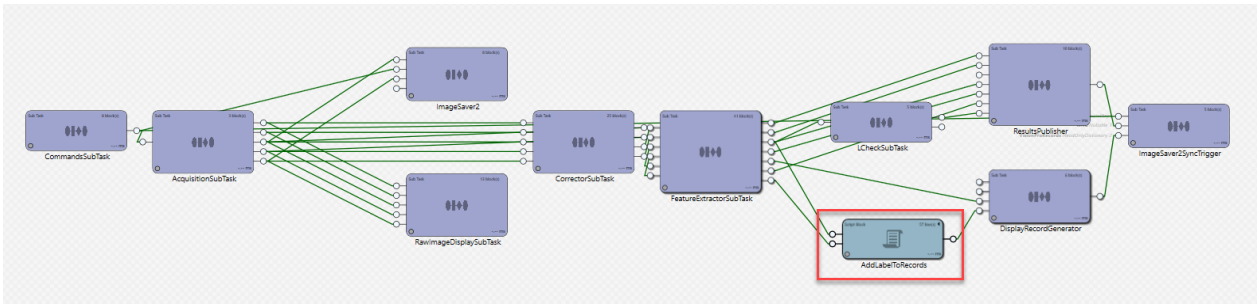
Different record names are separated with ";" or ",".



The feature finding image display uses VisionPro records from DisplayRecordGenerator in feature finding task. By default, DisplayRecordGenerator directly uses the records output from FeatureExtractorSubTask.

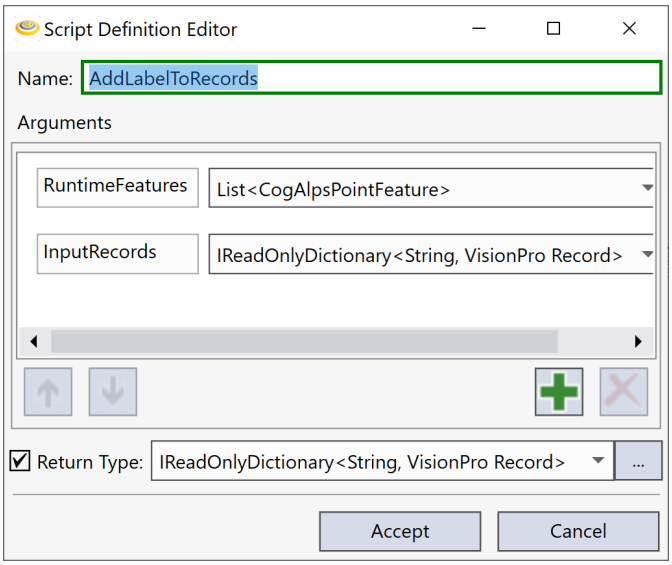


To add labels on display, we can insert a script block between FeatureExtractorSubTask and DisplayRecordGenerator to add labels to default records.



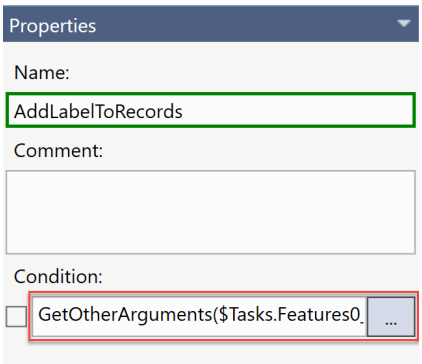
Here are the steps:

1. Add a script block to feature finding task, rename it as "AddLabelToRecords".
2. Set its input and output pins as below:



RuntimeFeatures will provide features coordinate information for labels.

3. Link the input pins to FeatureExtractorSubTask's thrid and last pin, the ouput pin to DisplayRecordGenerator last input pin as shown in modified task above.
4. Set the script block's condition the same as FeatureExtractorSubTask's, so that it also only gets called when all images are acquired at all positions.



5. Add "System.Drawing.dll" and "Cognex.Designer.AlignPlus.Alignment.dll" to script block's references
6. Add "System.Drawing", "Cognex.Designer.AlignPlus.Alignment", "Cognex.VisionPro.Implementation, and "Cognex.VisionPro" to script block's name spaces.

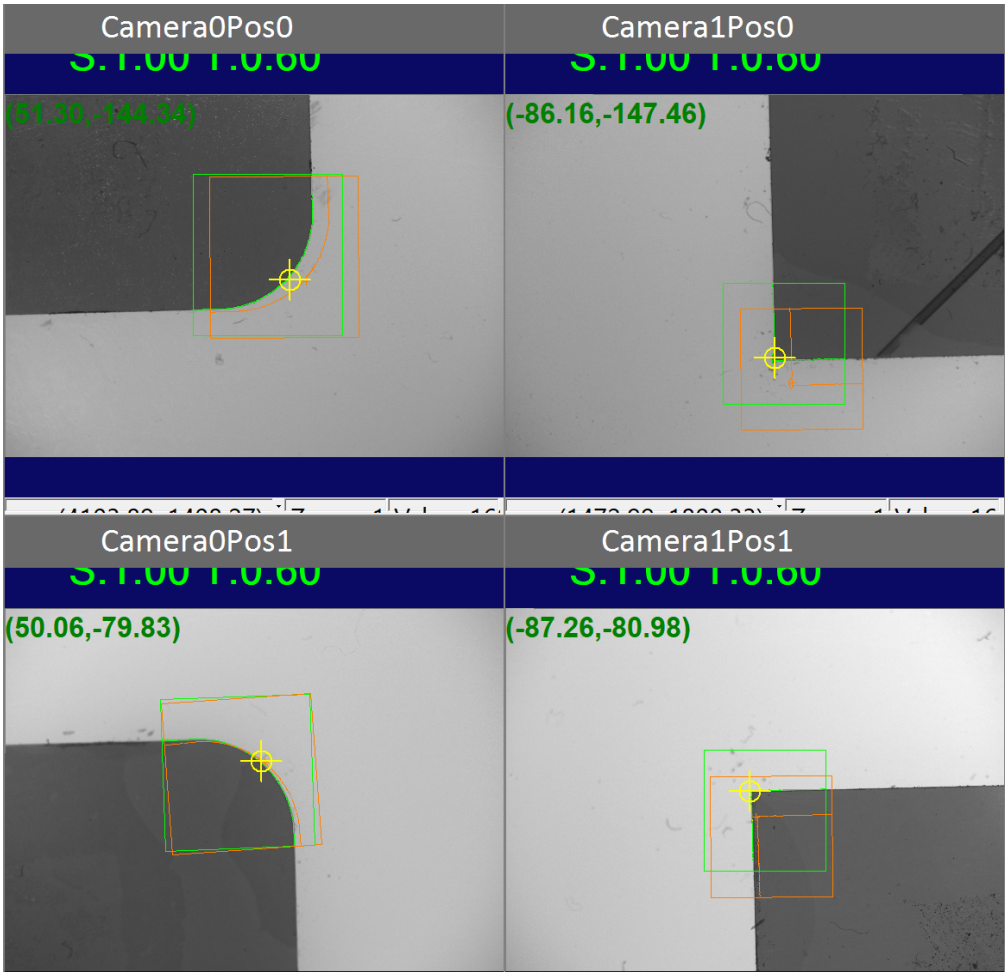
7. Within the script block, implement the following code.

AddLabelToRecords.Execute

public System.Collections.Generic.IReadOnlyDictionary<string, Cognex.VisionPro.ICogRecord>

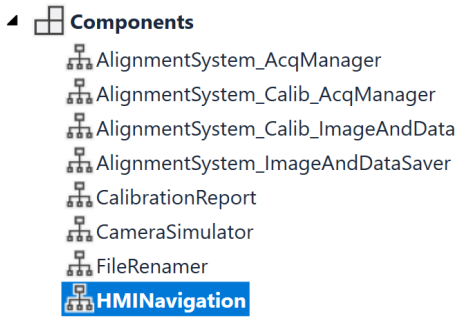
```
//Clear previous Feature Label records
foreach(var records in InputRecords)
{
    if(records.Value.SubRecords.ContainsKey("FeatureLabel"))
        records.Value.SubRecords.Remove("FeatureLabel");
}

foreach(var feature in RuntimeFeatures)
{
    //Match feature with records
    string recordKey = "";
    if(feature.CameraIndex == 0) recordKey = "Camera0Pos0";
    else if(feature.CameraIndex == 1) recordKey = "Camera1Pos0";
    else if(feature.CameraIndex == 2) recordKey = "Camera0Pos1";
    else if(feature.CameraIndex == 3) recordKey = "Camera1Pos1";

    if(recordKey != "")
    {
        //Get current image
        CogImage8Grey image = InputRecords[recordKey].Content as CogImage8Grey;

        //Create label
        CogGraphicLabel label = new CogGraphicLabel();
        label.Text = string.Format("({0:0.00},{1:0.00})", feature.FeatureLocationX, feature.FeatureLocationY);
        label.X = 0;
        label.Y = 50;
        label.Color = (feature.IsValid) ? CogColorConstants.DarkGreen : CogColorConstants.Red;
        label.Alignment = CogGraphicLabelAlignmentConstants.TopLeft;
        label.Font = new Font("Arial", 10, FontStyle.Bold);
        label.LineWidthInScreenPixels = 3;
        label.SelectedSpaceName = "*";

        //Create marker
        CogPointMarker Marker = new CogPointMarker();
        Marker.GraphicType = CogPointMarkerGraphicTypeConstants.Crosshair;
        Marker.X = feature.FeatureLocationX;
        Marker.Y = feature.FeatureLocationY;
        Marker.Color = CogColorConstants.Yellow;
        Marker.LineWidthInScreenPixels =2;
        Marker.SizeInScreenPixels = 50;
        Marker.SelectedSpaceName = image.SelectedSpaceName;

        //Create Graphics
        CogGraphicCollection graphics = new CogGraphicCollection();
        graphics.Add(label);
        graphics.Add(Marker);

        //Create CogRecord
        CogRecord labelRecord = new CogRecord("FeatureLabel", typeof(CogGraphicCollection),
        CogRecordUsageConstants.Result, false, graphics, null);

        //Add new record to default VisionPro Records
        InputRecords[recordKey].SubRecords.Add(labelRecord);
    }
}

return InputRecords;
```

8. Run the feature finding task and see the result.



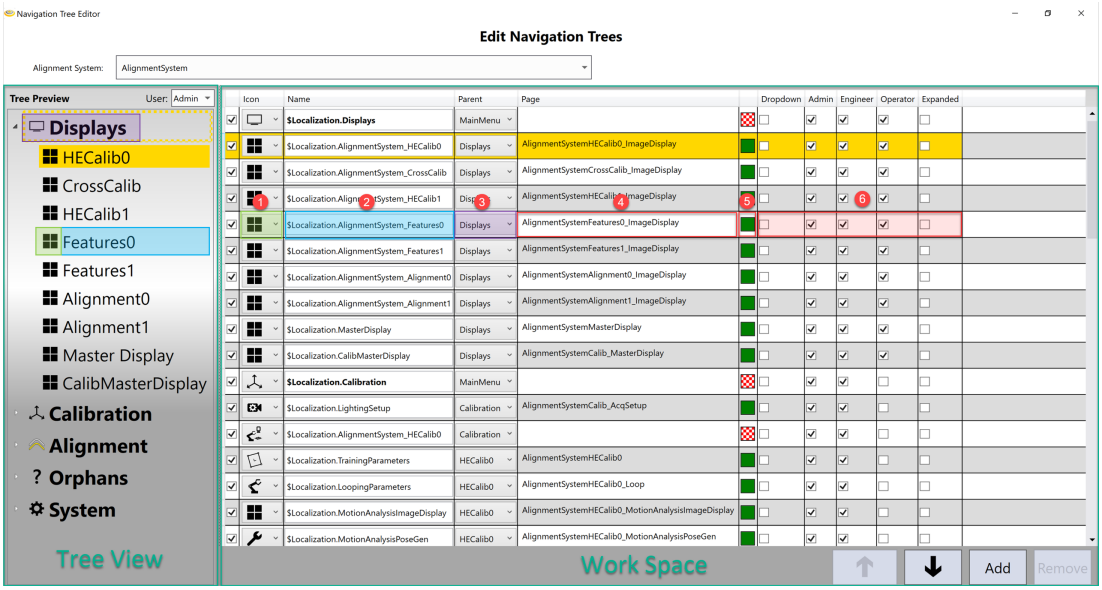
How to customize UI using Navigation Tree

Navigation tree manages page display in setup mode, it is automatically generated by configuration wizard during program generation with Displays, Calibrations, Alignment and System top-down tree structure. However, it also allows user to move, add, rename, hide, or delete a page in edit mode to satisfy certain HMI customization requirement. To enter its edit mode, double click "HMINavigation" under Components in the AlignPlus program. There will be a pop-up page for you to edit.



HMINavigation edit page consists of two areas: Tree View and Work Space.

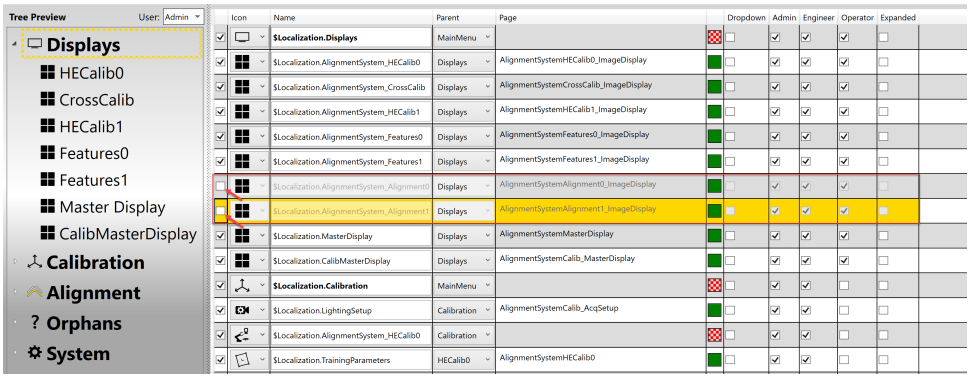
Tree View area shows a preview of the result navigation tree. Work Space area is for editing each page's categories, icons and user accesses.



1	Icon for current page
2	Display name for current page on HMI, here localization is used to translate the name into local language
3	Parent group the current page belongs to
4	The name of the page to be displayed
5	An auto-generated mark indicating whether the selected page is a valid page in edit mode
6	• Dropdown: When it is checked on, navigation tree will expand all sub pages of current page in run time, but user can choose to collapse them on HMI. • Expands: When it is checked on, navigation view will forcefully expand all sub pages and refuse to be collapsed. • Admin/Engineer/Operator: when it is checked on, the page will be visible for Admin/Engineer/Operator, otherwise, it is invisible to its selected user.

Hide unnecessary pages

AlignPlus displays all available pages in navigation tree by default though not all of them are always necessary. It is up to user to decide whether to hide some. To hide a page, just check off the front check box of a selected page.

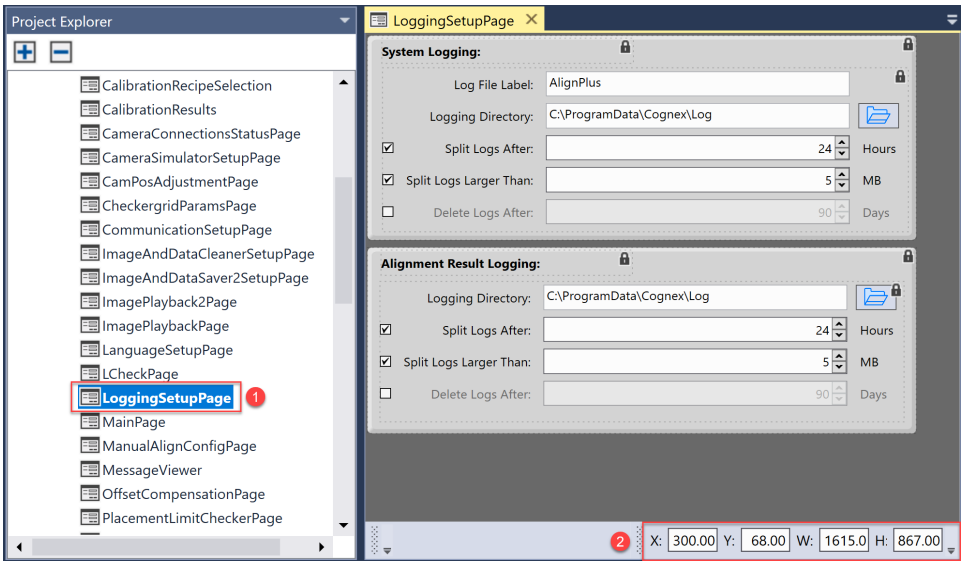


Add a new page

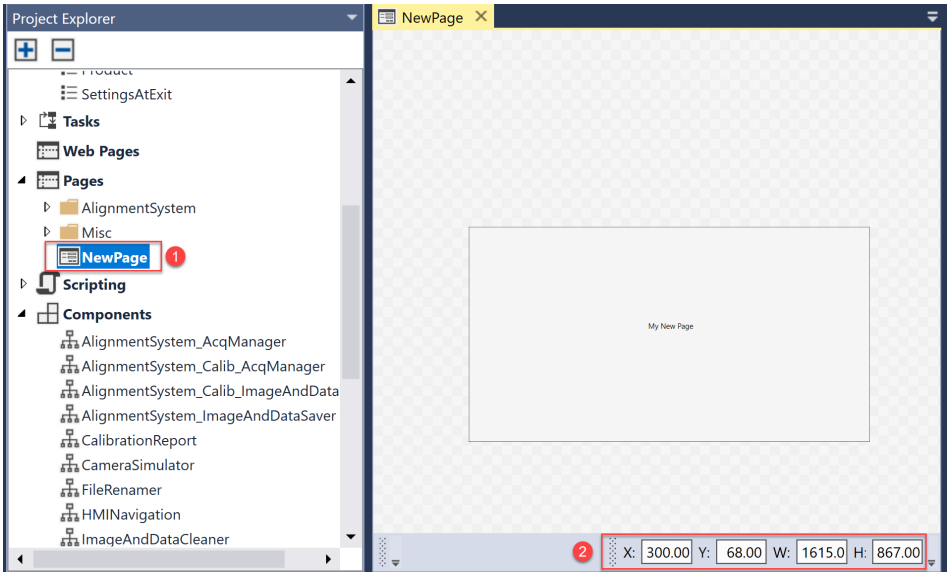
Follow the steps below to add a new page.

1. Create a new page and set it's size the same with other reference pages in the same group.

   For example, here is the property information of "LoggingSetupPage":

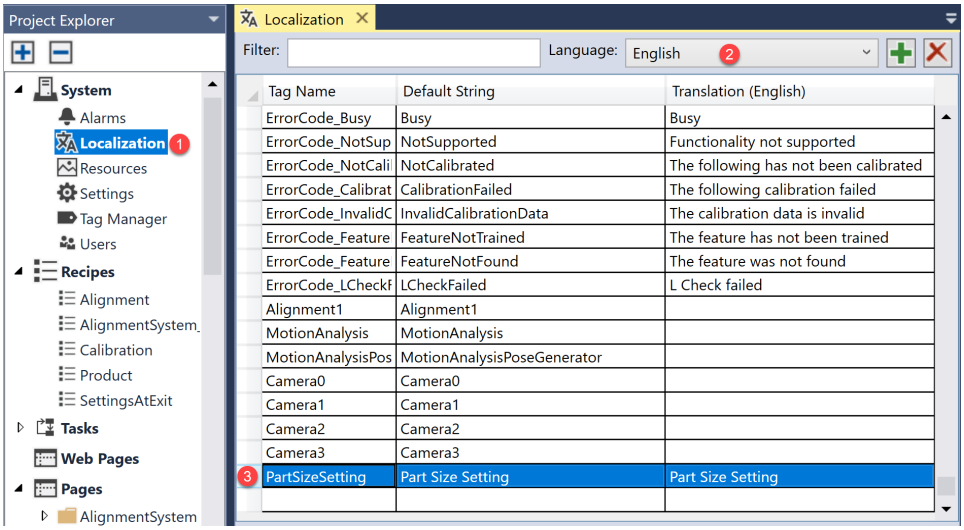


   Apply the same X, Y, W and H of the reference page to the new page.



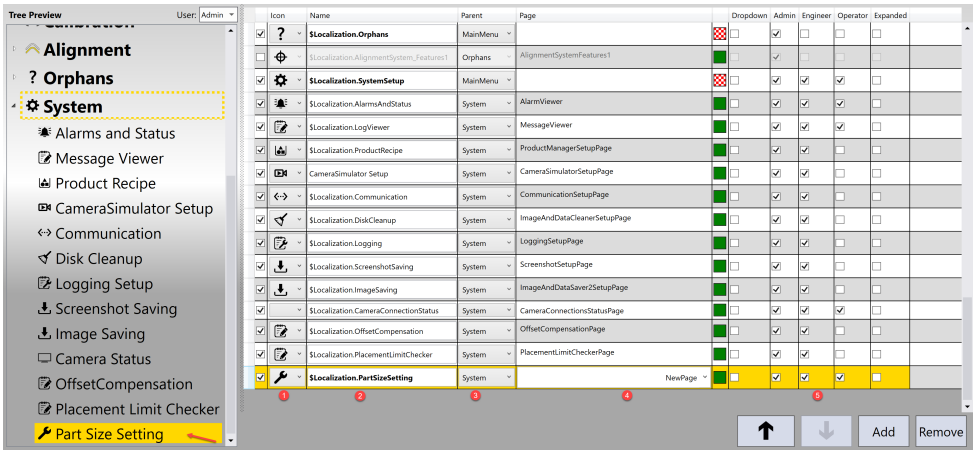
2. Edit the new page according to customer's requirement

3. Add a localization tag for the page name

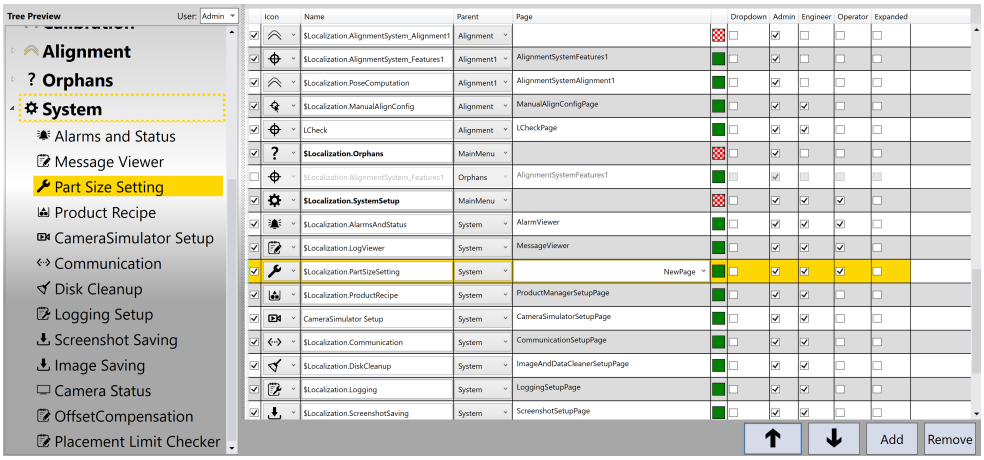


4. Add the page to HMINavigation component

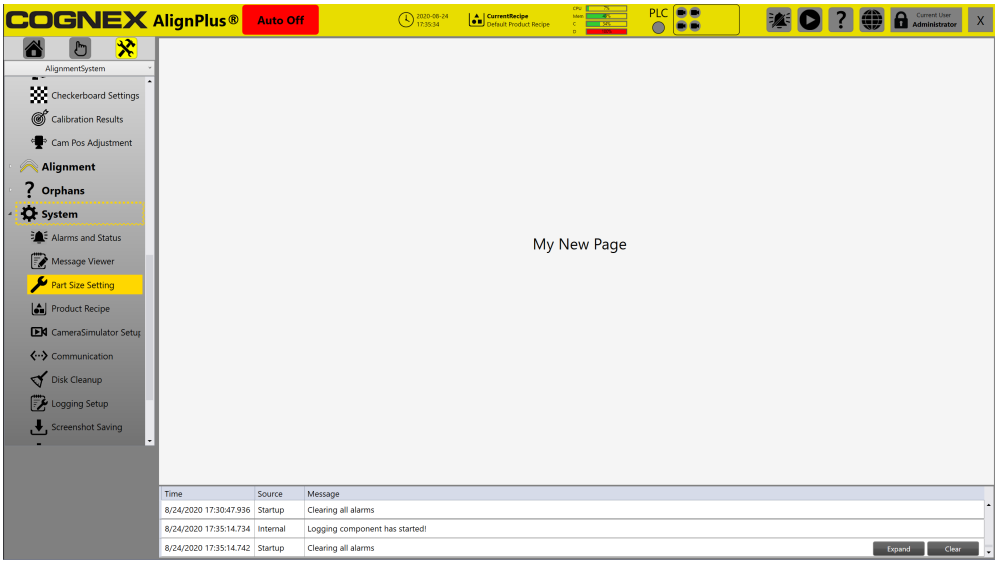
Click "Add" button, choose an icon for it, use the newly created localization tag for "Name", choose a parent node (here is "System"), and then select the page name you created and check the user accesses.



5. After all is set, the new page will automatically be added under the button of the parent group. The displayed name in tree view is what you have defined in localization for the current language. If you would like to move the page to a different position within the group, just click "↑" or "↓" buttons to move it up or down.



6. Run the program and see if new page is added properly.



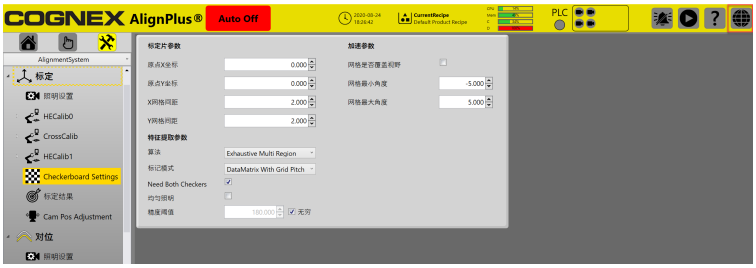
Remove a page

Select the unwanted page, and then click "Remove" button to remove it. This only applies to customized pages, if it is an automatically generated page, user can not delete it but can hide it instead.

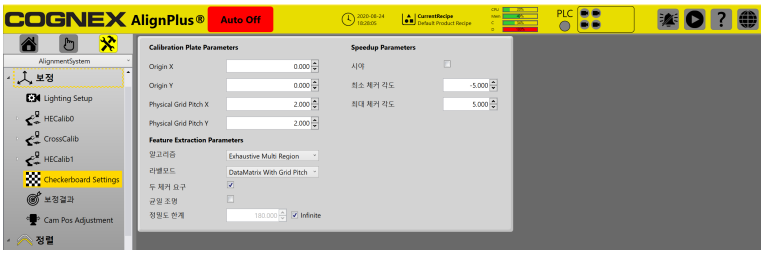
How to localize

AlignPlus provides Chinese and Korean language localization for HMI, user just needs to select the language on the top bar of the main GUI. Here is an example of default Chinese and Korean localization results.

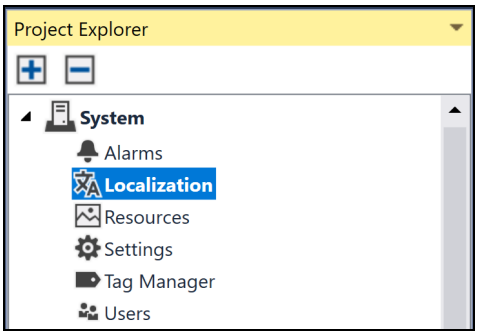
- Chinese



- Korean



Notice that, not all items are translated. However, in case where a particular item needs to be changed to a different name to make it more meaningful in a specific application, user can utilize Localization function to rename it. Localization function can be found in "System" category in edit mode. Double click it to open the edit window.

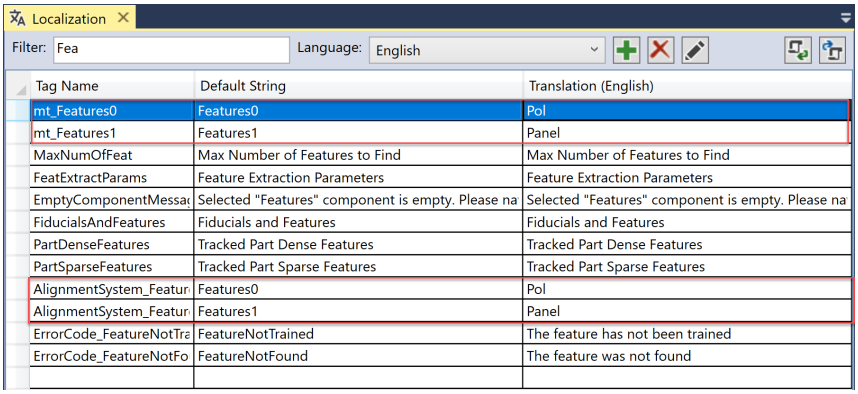


Localization maintains one table of tag names, default names, and translated names for each language type. **Filter** here can be used to search specific tags using keywords; **Language** is the target language that one would like to translate into. Column **Tag Name** lists names of tags used for localization. Column **Default String** lists default names of localization tags shown on HMI when English is chosen. Column **Translation** lists the translated names of localization tags for the current selected language.

Tag Name	Default String	Translation (Chinese)
mt_Alignment0	Alignment0	
mt_Features0	Features0	
mt_CrossCalib	CrossCalib	
mt_HECalib0	HECalib0	
mt_Alignment1	Alignment1	
mt_Features1	Features1	
mt_HECalib1	HECalib1	
Alignment0	Alignment0	
Setup	Setup	设置
NewRecipe	New Recipe	新配方
LoadRecipe	Load Recipe	读取配方
SaveRecipe	Save Recipe	保存配方
DeleteRecipe	Delete Recipe	删除配方
SaveRecipeAs	Save Recipe As	保存配方为
Save	Save	保存

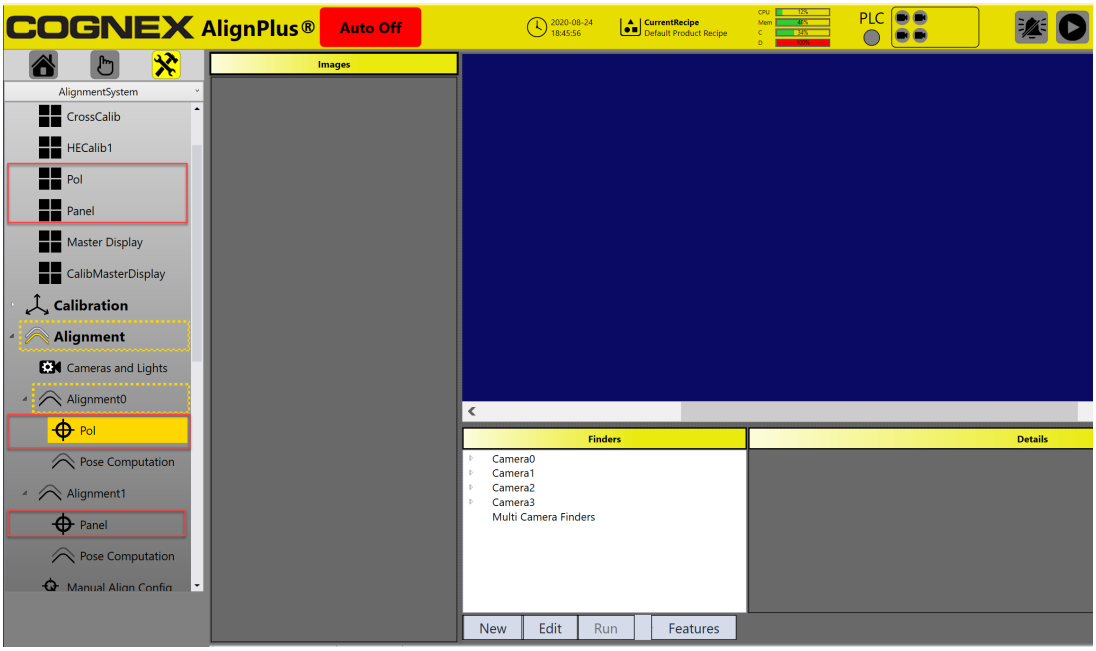
Here is an example to rename feature finders:

In this project, "Features0" and "Features1" are not meaningful to users, instead, specific part name such as "Pol" or "Panel" makes more sense. Therefore, you can translate "Features0" and "Features1" to "Pol" and "Panel" .

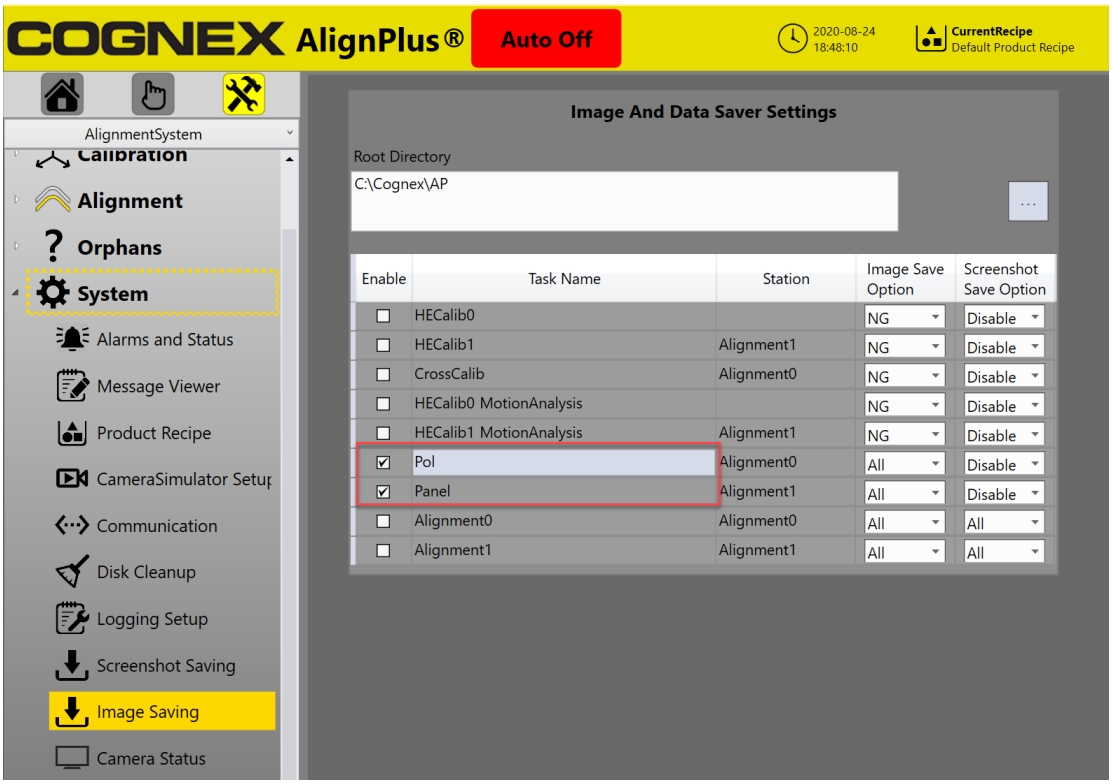


After the translation, on run time HMI you will find that the former "Features0" and "Features1" have been replaced with "Pol" or "Panel".

- Name changes in Navigation tree:



- Name changes in specific pages:



How To ... Change Log

How to get alignment result in AlignPlus program

The final alignment/assembly result is available in **SendResultToPLCCallBack** function.

Every time when a vision task finishes running, it will call TaskScheduler's call back function **AcknowledgeTaskResults**, in which **UpdateResult** will be called. UpdateResult then will call SendResultToPLCCallBack where you can get the alignment result if the vision task is triggered by a GP/GPA/LFGP command.

The signature of SendResultToPLCCallBack is:

public void SendResultToPLCCallBack(string commandResult, Int stepID, Int GPStepID)

Input Name	Type	Description
commandResult	string	The result string at given command
stepID	int	The StepID of the current task being called by the command
GPStepID	int	The StepID of the pose computer task when LFGP command is used

Below is an example of getting x, y, theta values from GP, GPA, or LFGP command result string:

SendResultToPLCCallBack.Execute

public void Execute(string commandResult, int stepID, int GPStepID)

```
#region Cognex Generated
// This is an automatically generated script. Do not edit inside this region.
// Edits outside this region are ok and will be preserved.
/* DO NOT EDIT */ var resultArray = commandResult.Split(',').Select(data => data.Trim()).ToArray<string>();
/* DO NOT EDIT */ string strCommandName = "";
/* DO NOT EDIT */ int iStatus = 0;
/* DO NOT EDIT */
/* DO NOT EDIT */ strCommandName = resultArray[0];
/* DO NOT EDIT */ iStatus = Convert.ToInt32(resultArray[1]);
#endregion Cognex Generated

if(resultArray[0]=="GP"||resultArray[0]=="GPA"||resultArray[0]=="LFGP")
{
    int status = Convert.ToInt32(resultArray[1]);
    if(status>0)
    {
        double x = Convert.ToDouble(resultArray[4]);
        double y = Convert.ToDouble(resultArray[5]);
        double theta = Convert.ToDouble(resultArray[7]);
    }
}
```

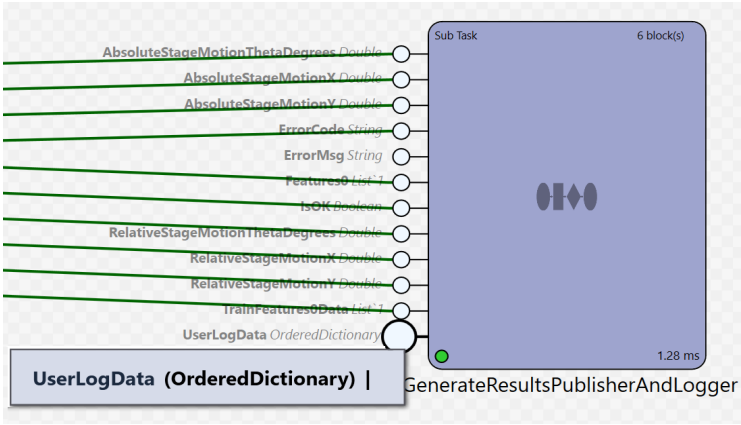
How to add data in alignment log

When more items need to be added into the default alignment log file (see more information in Logging Setup on page 172) , two ways of adding them are at your disposal.

Add by script block in alignment task

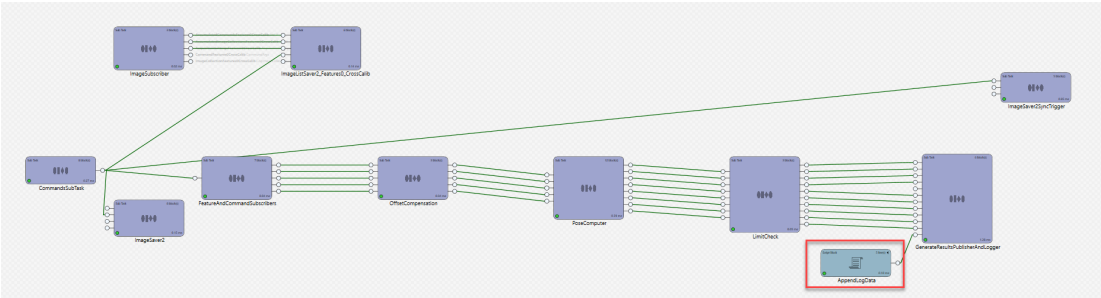
This way applies when extra data is to be added to the right side of the last column in alignment log .

In alignment task, GenerateResultsPublisherAndLogger sub task has an open input pin named "UserLogData", this pin provides a window for user to add extra data to current station's alignment log file.

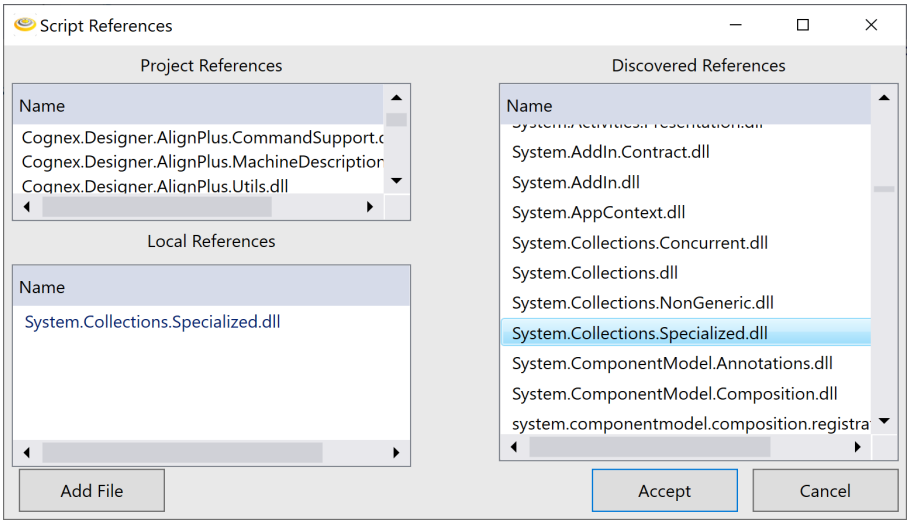


Suppose you want to log tact time of feature finder task, here are the steps:

1. Add a script block in alignment task, and configure its output pin as an OrderedDictionary object
2. Link the output pin to "UserLogData" input pin of GenerateResultsPublisherAndLogger



3. Within the script block, add "System.Collections.Specialized.dll" to its references.



4. Add the following code in the script block (The task name here should be the alignment task name in your own program).

```
public System.Collections.Specialized.OrderedDictionary Execute()
System.Collections.Specialized.OrderedDictionary Results = new System.Collections.Specialized.OrderedDictionary();
double time = $Tasks.Features0_CrossCalib.ExecutionTime;
Results.Add("TactTime_Features0",time);

return Results;
```

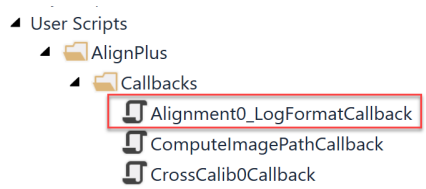
5. Run program and run feature finding and alignment, then check the log file. There should be a extra column named "TactTime_Features0" at the end of the log file with tact time data.

AM	AN	AO	AP	AQ
Features0_	Features0_	Features0_	Features0_	TactTime_Features0
-0.093	-0.093	1	1	248.227

Add in log format callback function

If you want to insert columns in the default alignment log extra data, you achieve so in the log format call back function.

At the end of an alignment task, AlignPlus will call a script named as "<Alignment Name>_LogFormatCall" where alignment data is logged. The **Alignment Name** here is the name of the task. If a program has multiple alignment tasks, there would be multiple log format callbacks so that each task will have its own logging function.



The signature of log format call back is:

public void AlignmentTaskName_LogFormatCallback(CogDictionary resultsDict, List<string> orderedResultKeys, string errorMsg, List<string> csvHeader, List<string> csvData)

Here are its arguments.

Argument	Type	Description
ResultsDict	CogDictionary	A dictionary of all saved alignment results

Argument	Type	Description
orderedResultKeys	List<string>	A list of key names whose data needs to be logged
errorMsg	string	Error message to log when alignment task fails
csvHeader	List<string>	The list of column names, by default it is the same with orderedResultKeys
csvData	List<string>	The list of data in string format extracted from ResutsDict using key names from orderedResultKeys

The following code shows an example to insert tact time after "ErrorCode" column.

Alignment0_LogFormatCallback.Execute | GenerateResultsPublisherAndLogger | OnStartup.Exec

public void Execute(Cognex.VisionPro.CogDictionary resultsDict, Syster

```
// Add Timestamp and Error Message to the front of the CSV record
csvHeader.Add("Timestamp");
csvData.Add(DateTime.Now.ToString("yyyyMMdd HH:mm:ss.fff"));

csvHeader.Add("ErrorMsg");
csvData.Add(errorMsg == null ? "" : errorMsg);

$System.Debug.Break();
// Use our default object-to-CSV conversion function to process the remaining results.
// Because dictionaries do not preserve key order, we iterate over "orderedResultKeys"
foreach (String key in orderedResultKeys)
{
    $LogHelper.AppendToCSVRecord(key, resultsDict[key], csvHeader, csvData);
    if(key=="ErrorCode")
    {
        double time = $Tasks.Features0_CrossCalib.ExecutionTime;
        $LogHelper.AppendToCSVRecord("TactTime", time, csvHeader, csvData);
    }
}
```

Here is the alignment log result:

	A	B	C	D	E	F	G	H	I	J	K	L	M
	Timestamp	ErrorMsg	AbsoluteSt	AbsoluteSt	AbsoluteSt	RelativeSta	RelativeSta	RelativeSta	IsOK	ErrorCode	TactTime	TrainFeatu	TrainFeatu
	20200814 19:58:38.897		0	0.001	0.098	0	0.001	0.098	TRUE		178.44	TRUE	TRUE

How To ... Change Recipe

How to modify recipe

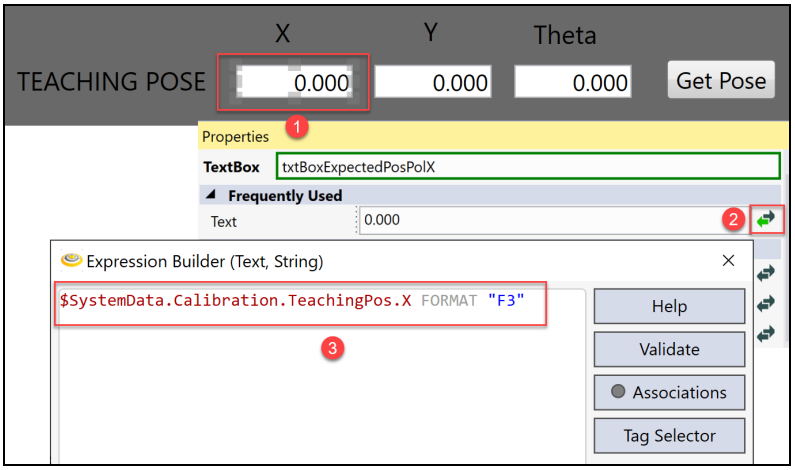
Modify Product Recipe

There are times when extra tags need to be added to Acquisition, Calibration, or Alignment recipes. For example, hand-eye calibration teaching pose (the center pose of calibration loop) should to be save into Calibration recipe so that it goes with specific calibration recipes. Here are the steps to achieve it:

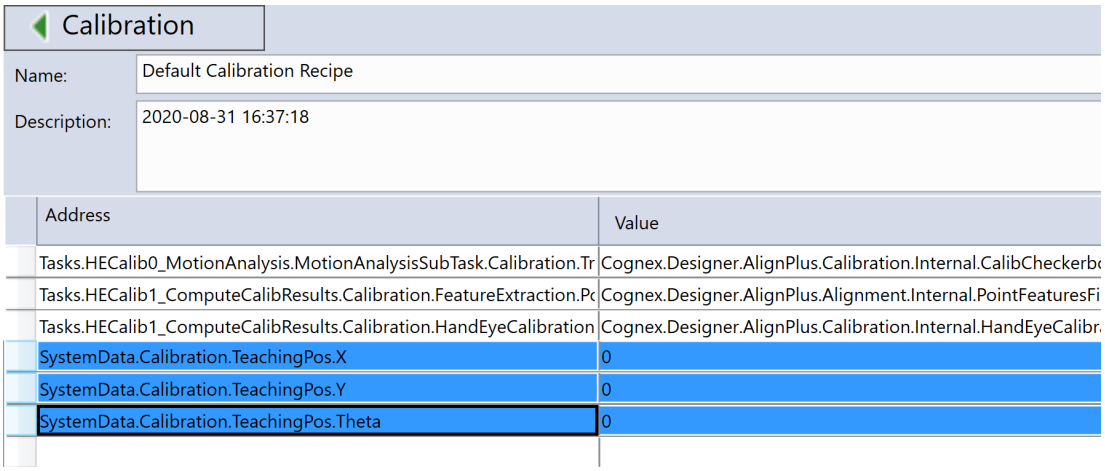
1. Create tags for teaching pose X, Y, Theta values

Name	Comment	Data Type
SystemData.Calibration.TeachingPos.Theta		Double
SystemData.Calibration.TeachingPos.X		Double
SystemData.Calibration.TeachingPos.Y		Double

2. On the HMI, create text boxes to show current teaching pose, and bind X, Y, Theta values with these created tags, respectively.



3. Open one of the Calibration recipes and, add these three tags into it.



4. Create a button besides the text boxes on HMI; within its OnButtonClicked event, add code to obtain teaching pose.

   Here is an example of using stage's current pose as teaching pose:

```
btnGetTargetPosPol.OnClick
public void OnClick(Object sender, EventArgs e)
1 //Read Stage's current pose
2 Dictionary<string, double> XYThetaInDegrees = new Dictionary<string, double>();
3 $Stage0GetPosition(XYThetaInDegrees);
4
5 //Set current pose as teaching pose
6 $SystemData.Calibration.TeachingPos.X = Convert.ToDouble(XYThetaInDegrees["X"]);
7 $SystemData.Calibration.TeachingPos.Y = Convert.ToDouble(XYThetaInDegrees["Y"]);
8 $SystemData.Calibration.TeachingPos.Theta = Convert.ToDouble(XYThetaInDegrees["ThetaInDegrees"]);
9
```

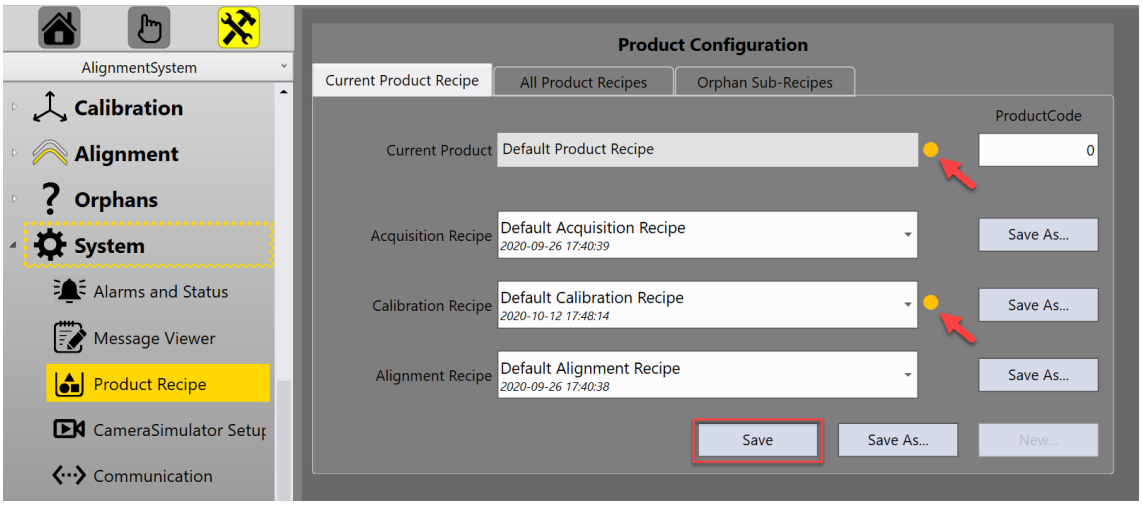
Enable Recipe Saving

When user changes the value of a customized tag which has been manually added to one of the acquisition/calibration/alignment recipes, you may want to remind him/her to save recipe before the program is closed. However, the product recipe page in this case shows nothing is changed and thus no need to save recipe. This is because the product manager only monitors the value changes of auto-generated tags when the application was generated by

configuration wizard, and decides whether to enable recipe saving or not. To change this, you can add a script as below to force product recipe to recognize data change and enable its recipe saving function:

$SystemState.CalibrationRecipeDirty = true;

After the script above is called, the product manager will render the product recipe page as below for you to manually save recipe or remind you of recipe saving if you directly exit program without saving.



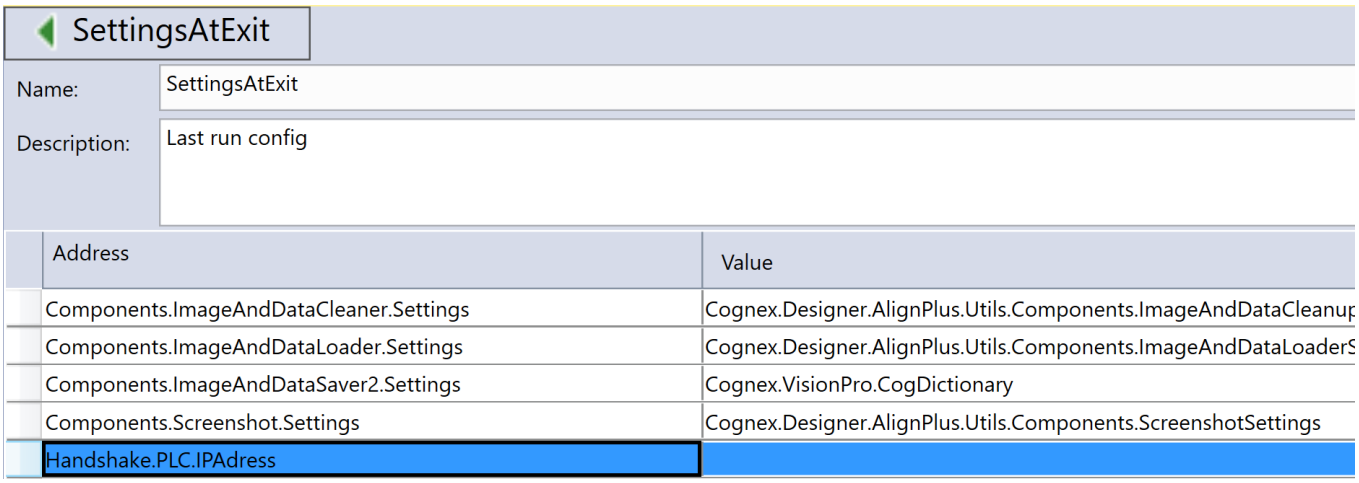
If you want to save recipe right after the tag's value is changed, you can also call product manager to save current recipe after notifying the changes.

$SystemState.CalibrationRecipeDirty = true;

$Components.ProductManager.SaveCurrentRecipe();

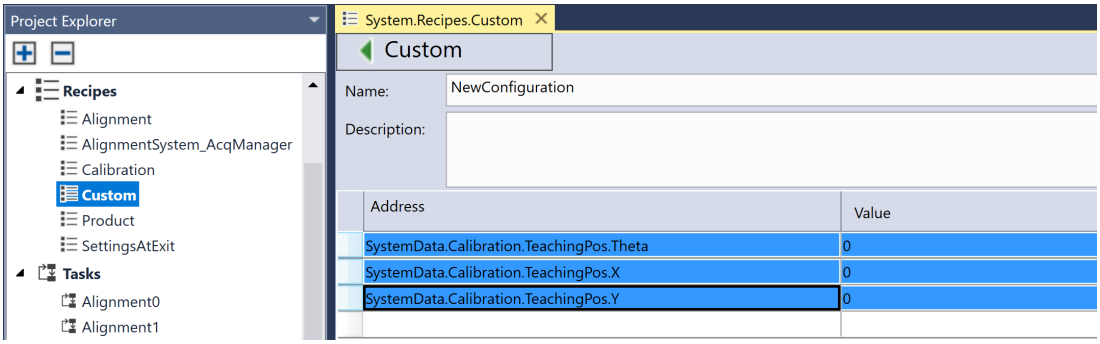
Modify Exit Recipe

If values of some tags should be preserved after the program is rebooted(for example: PLC's IP address), you can add these tags to SettingsAtExit recipe. This recipe will always execute recipe saving automatically before program is closed, you do not need to add any extra script to save them.



How to manage other recipes

In case you do not want to make changes to default acqusition/calibration/alignment recipes, you can create your own custom recipe to manage extra tags. For detailed steps, please refer to Cognex Designer User Guide/Developing a Cognex Designer Project/Recipes on how to create a new recipe, manage it on UI, etc.



You can create separate recipe saving function for custom recipes, or utilize the default product manager for custom recipe saving.

Here are steps for the latter option:

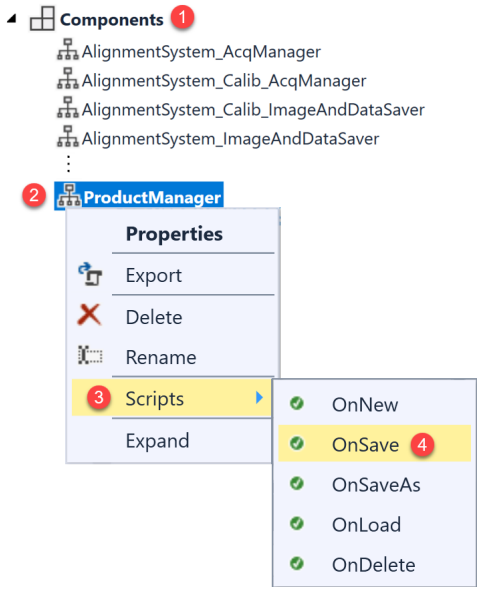
1. Mark product recipe as dirty when there is any data change in your custom recipe

   $SystemState.CalibrationRecipeDirty = true;

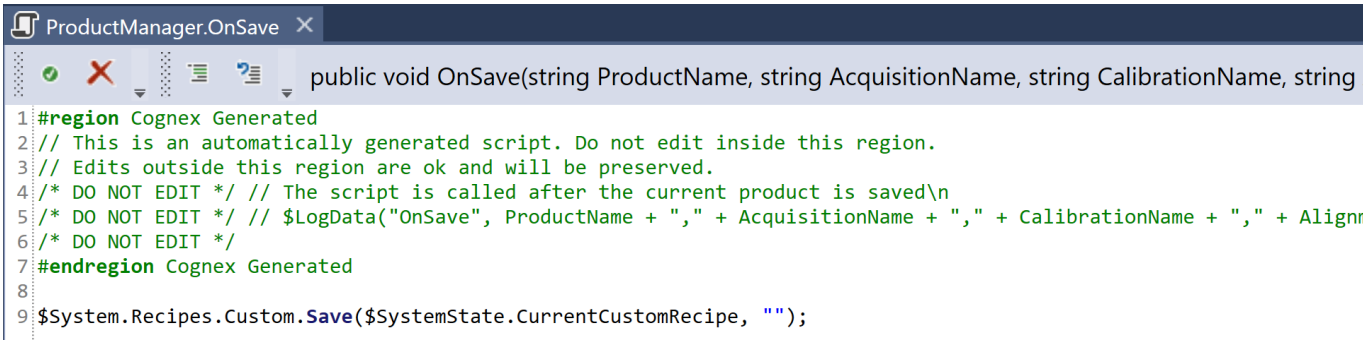
   Note: You can also mark acquisition or alignment recipe as dirty, though none of them including calibration recipe has any change, this function only informs product manager to be ready to save.

2. Add custom recipe saving function in current product manager.

   "OnSave" script of ProductManager component is a call back function whenever product manager saves recipe.



   Add your custom recipe saving function here will make product manager add your custom recipe saving to its saving list.



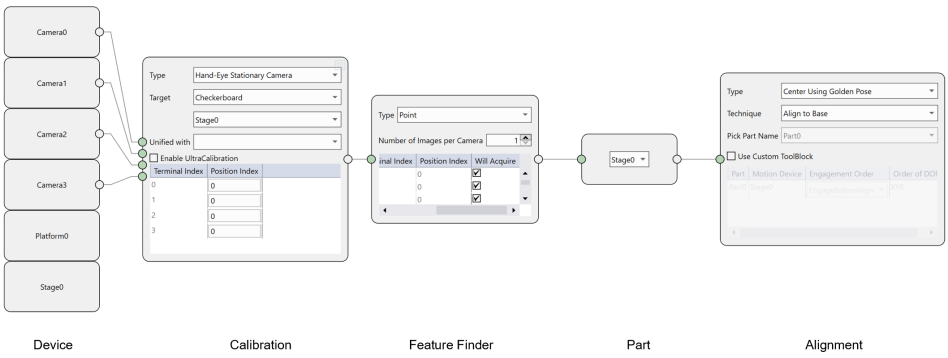
How To ... Change workflow

How to change 4-Point Align to 1-Point Align

AlignPlus program allows user to change the quantity of alignment points without rerunning configuration wizard if it does not involve recalibration. This topic will introduce two examples of this type of changes: 4-camera based, and camera shuttling based.

4-Camera Based

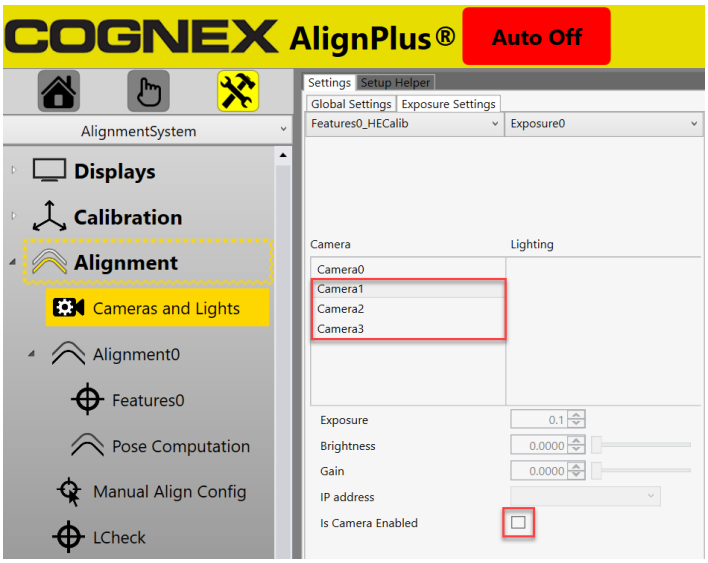
In this application, one-part alignment used to use four cameras needs to be modified to only use one camera due to model change.



The places where user needs to handle for model change are camera acquisition, vision tool, and recipe. Follow these steps to change model:

1. Use "Save As" function to create a copy of current product recipe
2. Disable unused cameras

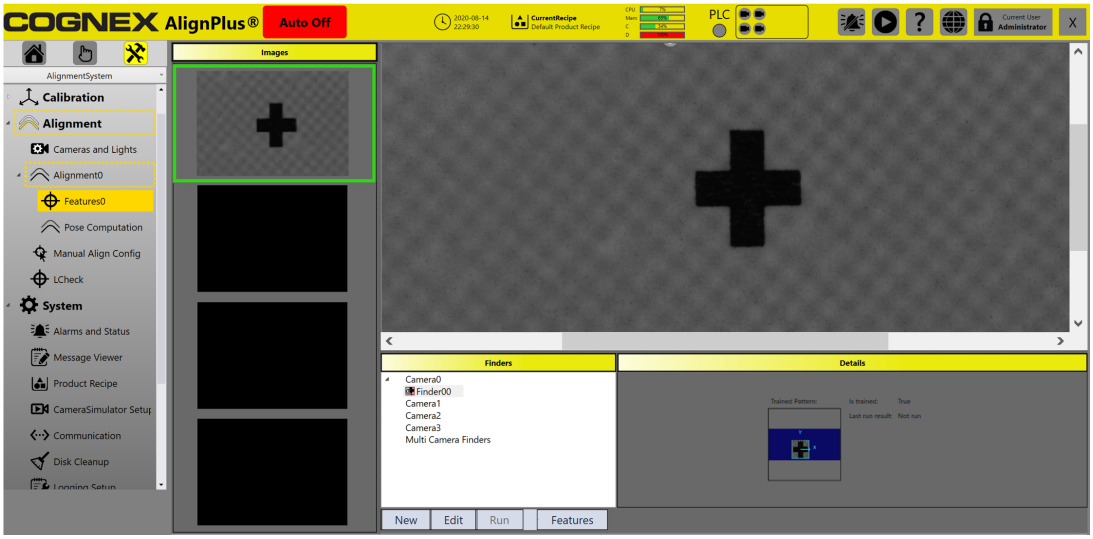
   Disable unused cameras in feature finder's exposure setting under **Alignment/Camera and Lights/Exposure Settings**.



3. Save current product recipe to save camera disable/enable information to Alignment recipe.

4. Modify vision task to find features of new model

   Open feature finder setup page after one image acquisition. Delete feature finders under unused cameras. Use one PatMax finder to locate one feature or two point finders (PatMax finder/Corner finder/Custom Point Finder) to locate two features on new model under the remaining camera.

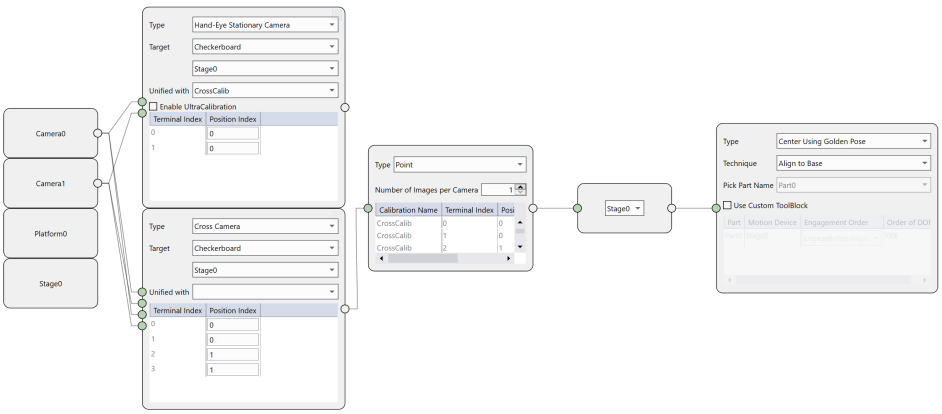


5. Train golden pose for new model
6. Save current product recipe again to save golden pose and vision tools into Alignment recipe.

After this, the program is ready to run.

Camera Shuttling Based

In model change of this application, one-part alignment used to use 2 shuttling cameras acquiring four images now needs to be changed to only use one camera acquiring one image at one position.



The steps to achieve this is the same as in the 4-camera based case except that:

- In 4-camera based application, the command to call feature finding task does not change before and after model change.

- In camera shuttling based application, the commands are different in quantity and parameter before and after model change.

  Before model change, two commands need to be sent to vision system to finish feature finding. One is for acquire only, the other is for acquire and process.

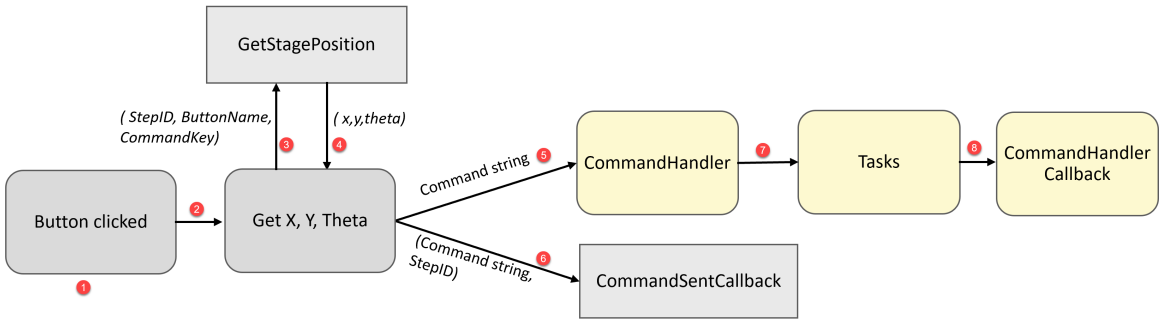
Position	Action
Position 0	Acquire Only
Position 1	Acquire and Process

  After model change, since position 1 does not need to acquire image anymore, only one command is needed for position 0: acquire and process, to run feature finder.

Position	Action
Position 0	Acquire and Process

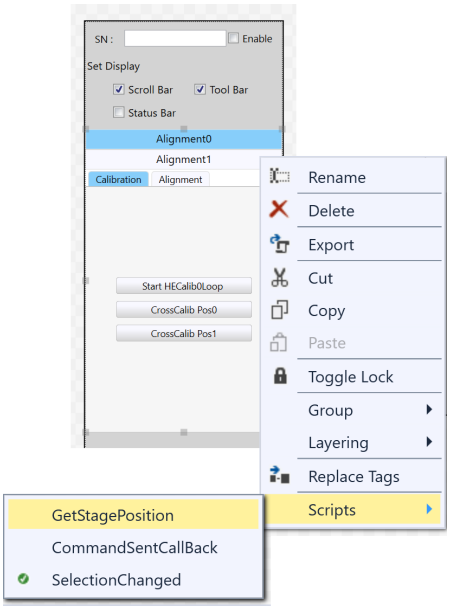
How to get stage's current pose for manual buttons

By default, manual trigger HMI controller assumes current stage is always at its origin by setting X, Y, Theta parameters in commands as 0. This works well if the purpose of manual trigger is to configure vision tasks or test program workflow, but not when the purpose is to further test align accuracy since stage's current pose is not involved in pose calculation. However, manual trigger HMI controller also provides a callback function which allows user to acquire stage pose before sending the manual triggered command out to program. The callback function is named "GetStagePosition", and here is the workflow of manual trigger process wherein GetStagePosition is positioned.



Once any button on manual trigger HMI is clicked in run time mode, the manual trigger HMI controller will first call GetStagePosition where stage pose can be obtained and sent back to manual trigger controller. Following that, the controller will send the command with updated X, Y, Theta parameters of stage pose to CommandHanlder to let the program finish the rest of the vision task. Meanwhile, manual trigger controller will also call another callback function named "CommandSentCallback" where user can get the sent-out command string. CommandSentCallback can be used for logging purpose or user can just leave it as it is since it has no effects on the main process.

The two callback functions can be found under "Scripts" option of context menu by right-clicking the manual trigger HMI controller in test mode.



GetStagePosition

The signature of GetStagePosition is as below:

public System.Tuple<double, double, double> GetStagePosition(int StepID, string ButtonName, string CommandKey)

Inputs

Item	Type	Description
StepID	Integer	The StepID of requested task
ButtonName	String	The name of button that has been triggered
CommandKey	String	The Command Key that of the command that sent by the triggered button

Output

The output is x, y, theta values in Tuple format.

Note: All buttons on the same manual trigger HMI share the same GetStagePosition callback function, user can decide which stage's pose is to return depending on input StepID.

CommandSentCallBack

The signature of CommandSentCallBack is as below:

public void CommandSentCallBack(int StepID, string cmdSent)

Inputs

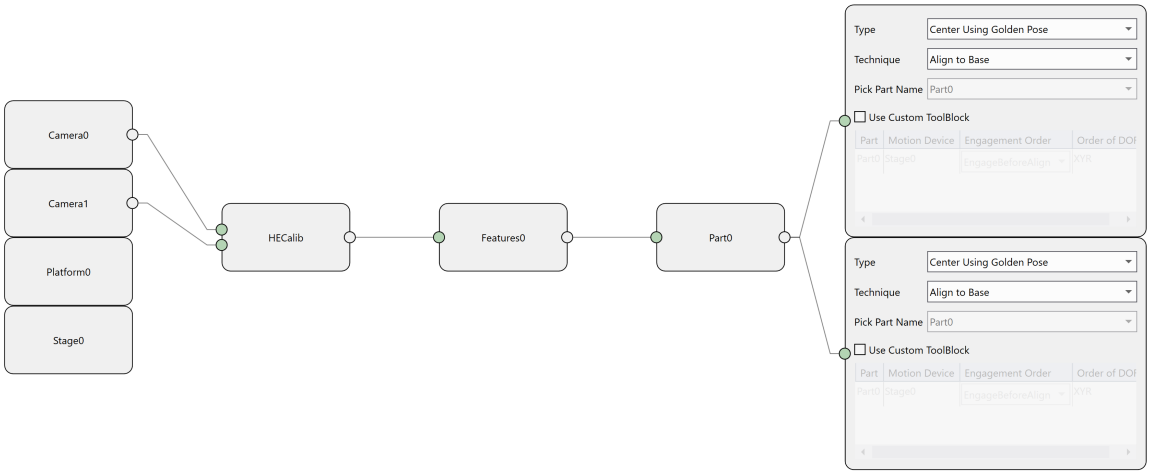
Item	Type	Description
StepID	Integer	The StepID of requested task behind the triggered button
cmdSent	String	The final command string sent to CommandHandler which later calls corresponding task

How to run two independent pose computers for two parts in one image

In some alignment applications there are two independent parts within the same field of views of cameras which require to be located separately. In the example below, there is a white protect film on the black screen, the relative pose between film and screen is not fixed. The vision system's task here is to find where the film is(marked in red) and then locate where the screen is(marked in blue), so that the machine can first remove the film, and then align the screen.

Wizard Configuration

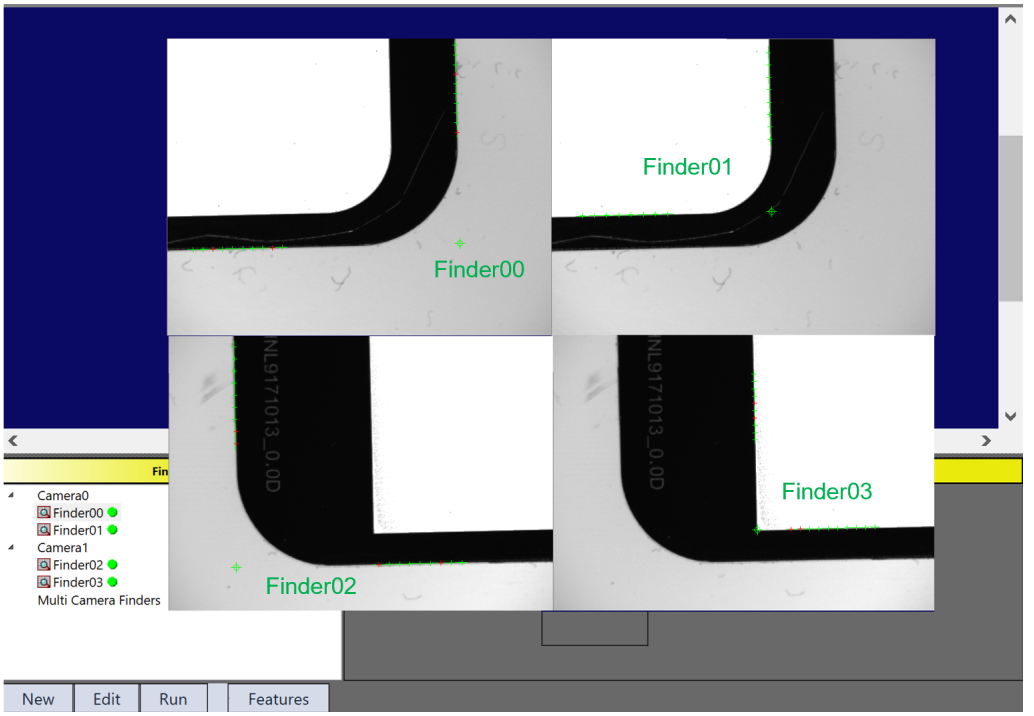
In these kind of applications, user usually does not want to acquire images twice for each part and run two independent features finder as that will increase processing time. Rather, user would like to acquire once, run one features finder to find all features together, and compute two poses using features from each part. To achieve that, here is how wizard should be configured.



In this configuration, there are two alignment components, both of which use all features found in Features0 for pose computing as AlignPlus cannot tell which feature belongs to which part. Therefore, the auto-generated alignment tasks will run through exactly the same process with the same inputs and outputs. However, this configuration provides a good structure for you to customize upon.

Feature Finding

To start with, we need to find all features first as shown below:



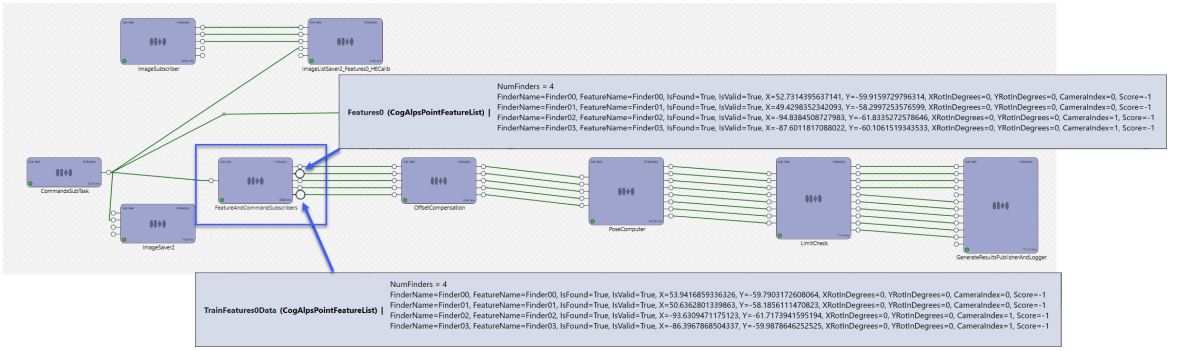
There are four corner finders inside Features0, Finder00 and Finder 02 find the two corners of the screen; Finder01 and Finder03 locates the corners the film.

Customization

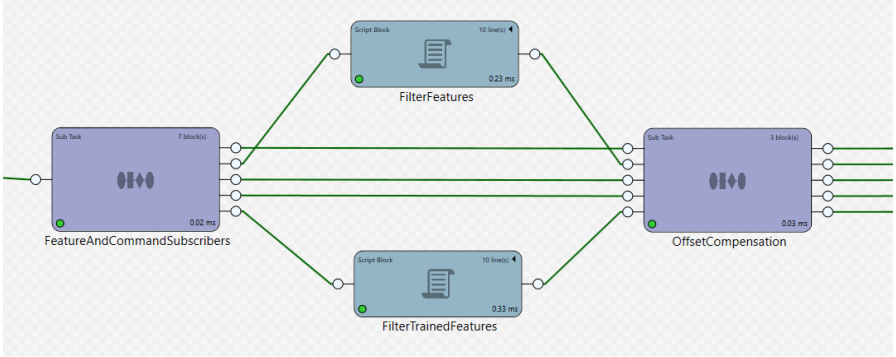
The goal is to let alignment0 run pose computing for film, and alignemnt1 for screen. Instead of all alignment tasks using all features, we can filter the features inputs so that each task only receives features it absolutely needs.

Here is where filtering takes place:

The FeatureAndCommandSubscribers subtask in the alignment task is responsible for subscribing train time and run time features and their commands from corresponding feature finding task. Its second output pin is a list of run time features, last output pin is a list of train time features. As displayed below, it uses all features from the feature finder by default .

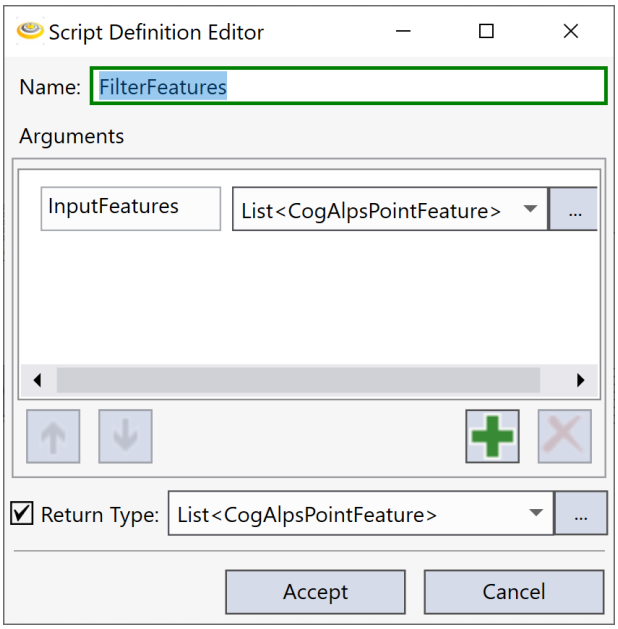


To filter out those unwanted train time and run time features, one can insert two script blocks between FeatureAndCommandSubscribers and OffsetCompensation.

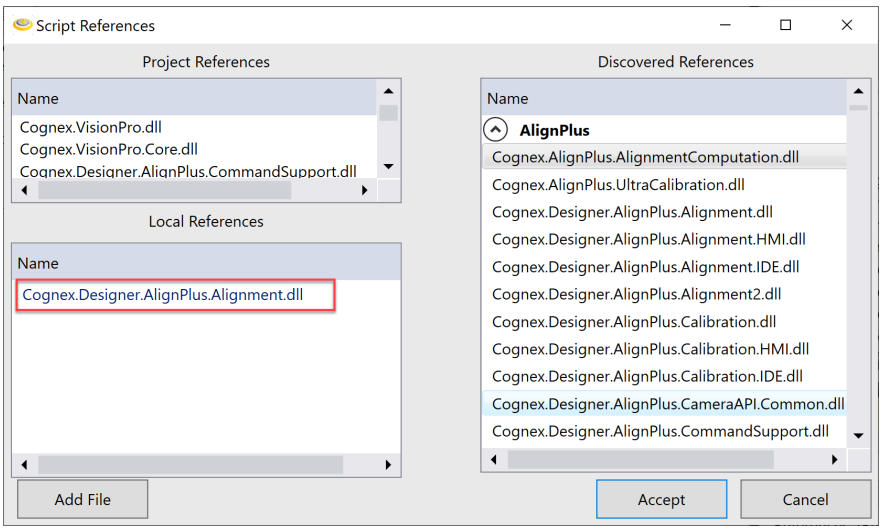


For Alignment0, only features from film are needed (Finder01 and Finder03). Here are the steps of customization:

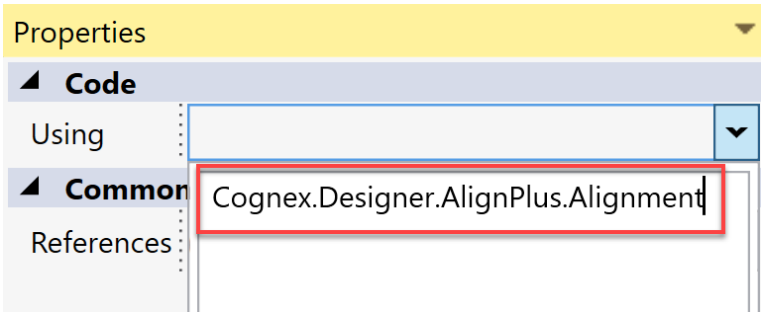
1. Add a script block named as "FilterFeatures", configure its input and output as below:



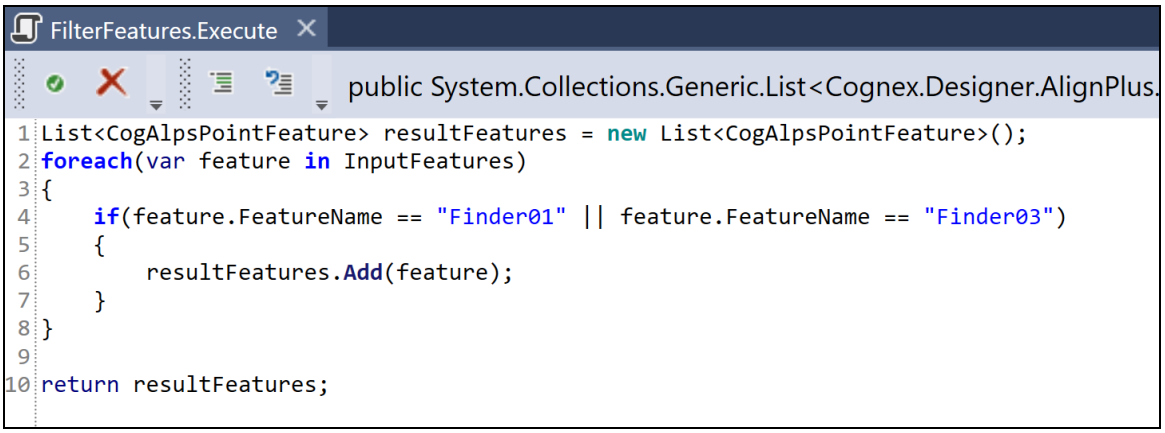
2. Add Cognex.Designer.AlignPlus.Alignment.dll to its references.



3. And "Cognex.Designer.AlignPlus.Alignment" to its name space.

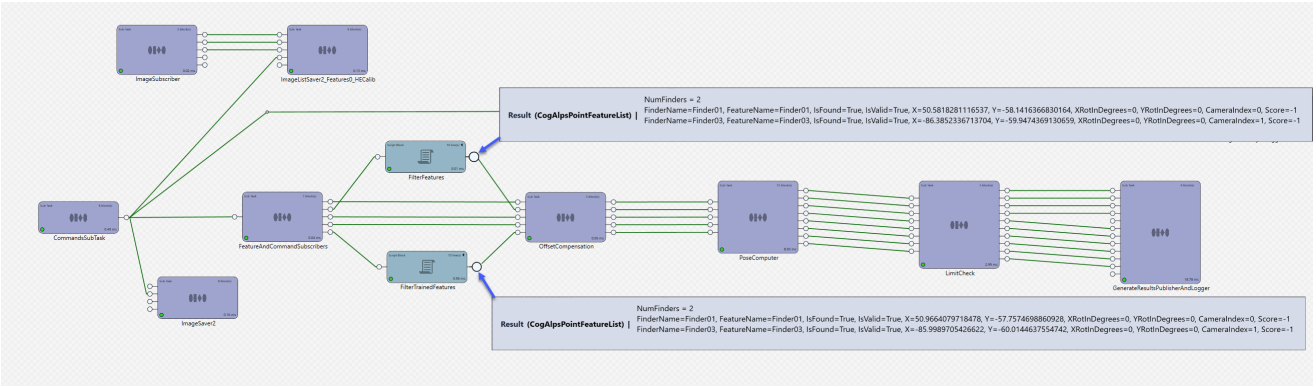


4. Inside the script, input the following code:



5. Copy the block and rename it as "FilterTrainFeatures", link its input pin to trained features of FeatureAndCommandSubscribers subtask, and output pin to OffsetCompensation subtask.

After the customization, the alignment0 will only handle features from film as shown below:



For Alignment1, the customization is exactly the same except the feature names in its script blocks should be "Feature00" and "Feature01" representing the screen part.

Reference

Space Tree

In AlignPlus, many spaces are involved in calculation, each space serves a different purpose, in order to understand which space are available in AlignPlus and how to use them, here we need first understand the concept of space tree.

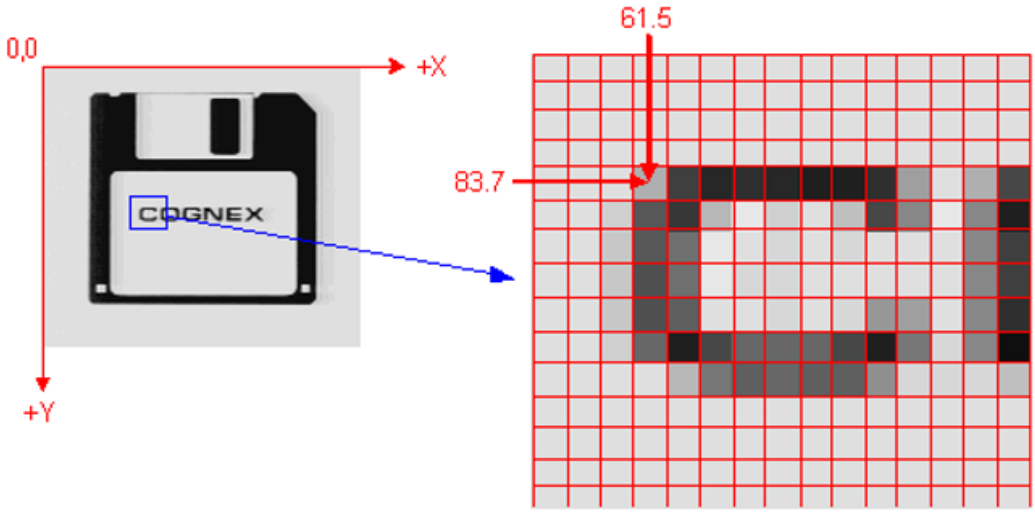
Space tree is a coordinate class in VisionPro to track spatial transformations applied during image processing as well as map coordinates from root space to any defined user space and back again. The coordinate space tree for any image can be accessed through its CoordinateSpaceTree property.

Root Space and Pixel Space

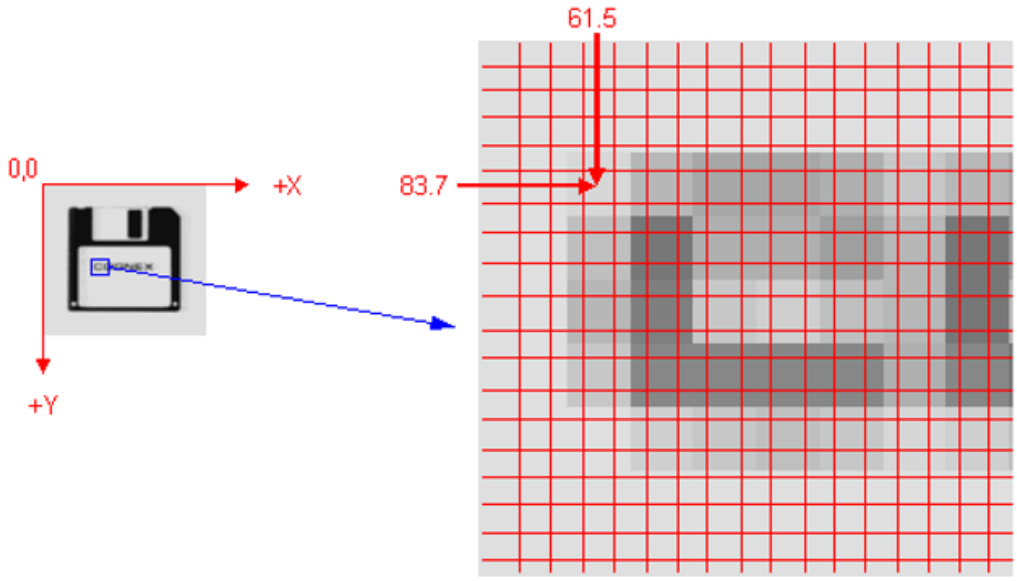
For every image that is acquired from camera or loaded from image file, VisionPro will give it a root space(named as "@") which originated at upper-left corner of the image. Root space is exactly the same with image space(named as "#") before image undergoing any image processing.

However, if original image is being clipped, resized or transformed, though image space alters based on new image after every action, root space remain the original space in which each pixel's coordinates is tracked and mapped to original image. Whenever a VisionPro tool modifies the pixels of an acquired image, it automatically updates the root space to reflect any changes in the location of image features.

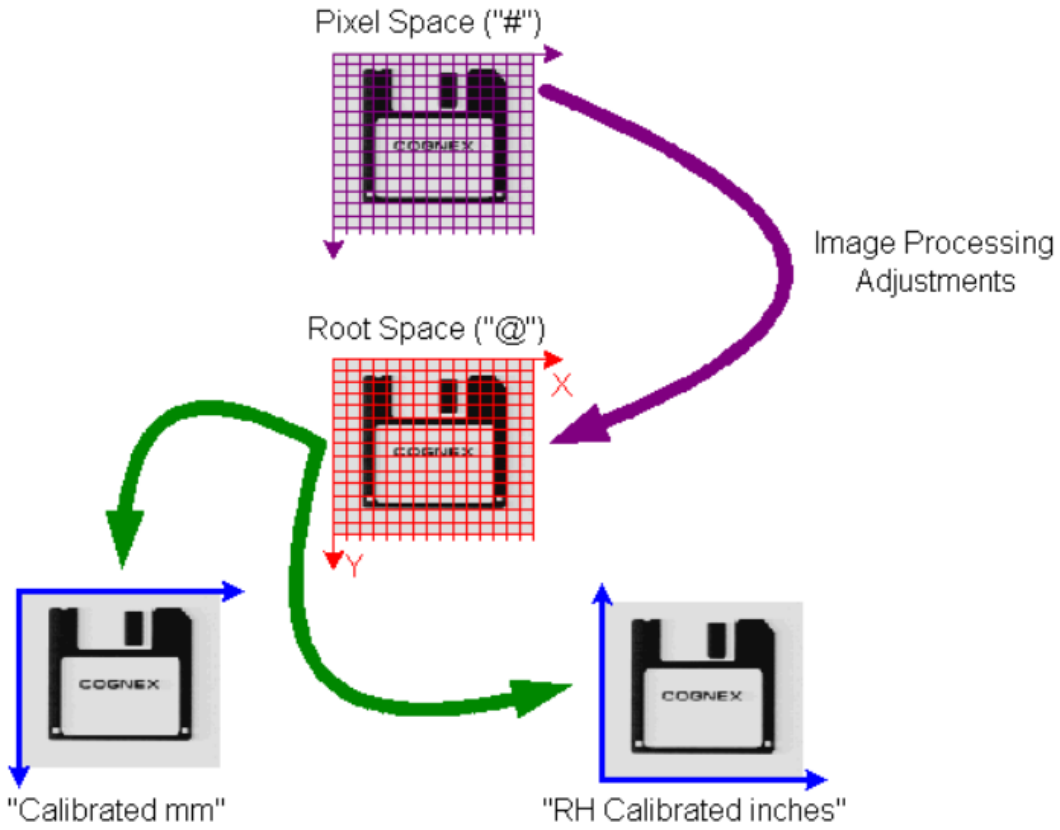
Here is an example, the pixel is at (83.7, 61.5) in the original image.



After the image is scaled down by 2, the corresponding point in new image's pixel space should be at (41.85, 30.75). However, if user choose to output coordinates in root space, it is still (83.7, 61.5).



The following figure shows how pixel space is related to the root of the coordinate space tree:



In this figure, the purple arrow represents the transformation between the pixels of the image and its root space; this transformation incorporates all of the image processing adjustments. You can obtain the transformation between the pixels of the image and its root space through the image's PixelFromRootTransform property. Each of the green arrows represents a transformation between root space and user defined space (which will be introduced in the following paragraph) in the image's coordinate space tree.

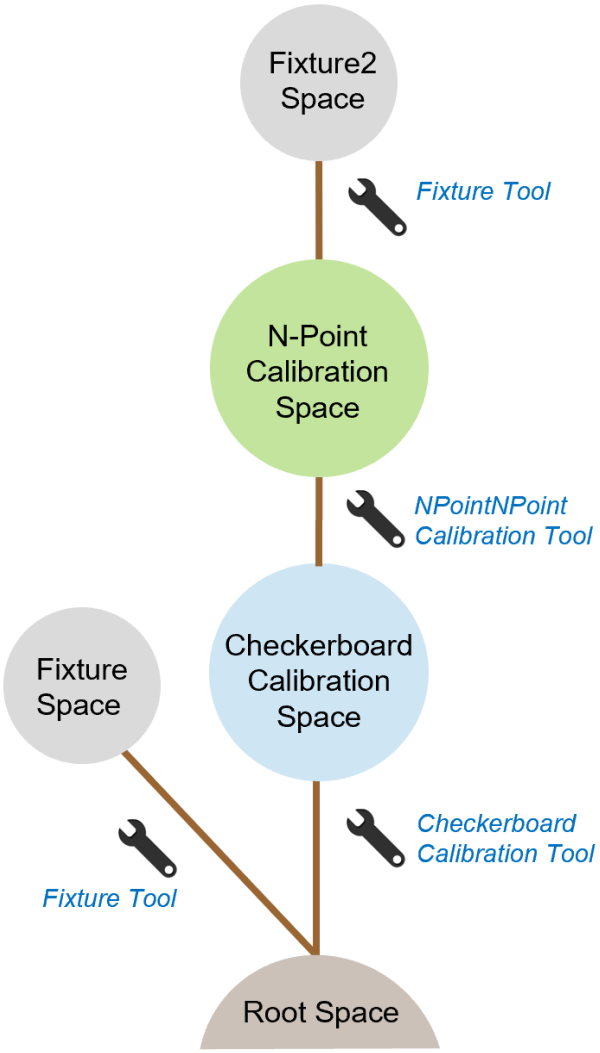
User Spaces

User can build their own spaces upon root space using certain VisionPro Tools like generating branches on the trunk of a tree. Each user space in the space tree has a parent coordinate space related by one or more transformations. All user spaces can trace their ancestry back to the root space.

New coordinate spaces can be generated through the following VisionPro Tools:

- CogFixtureTool
- CogFixtureNPointToNPoint Tool
- CogCalibNPointToNPoint Tool
- CogCheckerboardCalibration Tool
- Manually configuring and passing a 2D Transform (available)

Here is an example of space tree of one image. The image was first processed by CogFixtureTool which added a "Fixture Space" to the root space, and later checkerboard calibrated which added a "Checkerboard Calibration Space". The corrected image of checkerboard calibration then runs NPointNPoint calibration which added a "N-Point Calibration Space". At last, another Fixture Tool which creates a "Fixture2 Space".



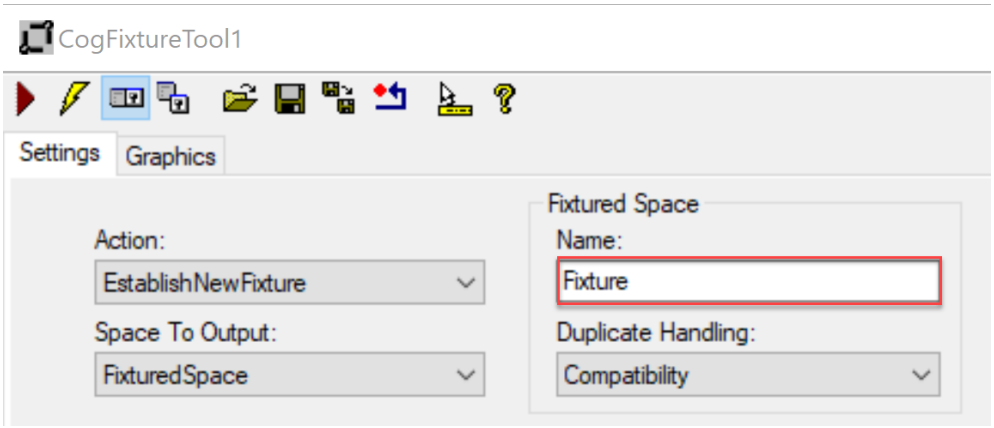
After all those spaces are built, user can pay visit to every space through the image's CoordinateSpaceTree property in which spaces are stored with parent coordinate spaces tracing back to root space. These spaces are outputs of every tool, but they don't belong to tool itself, they belong to the image being processed.

Item	Space Name	Get From	Unit	Space Transforms
1	@	Born	pixel	@ <=> Image Space
2	@\Fixture	CogFixtureTool	pixel	@ <=> Fixture Space
3	@\Checkerboard Calibration	CogCalibCheckerboard Tool	mm/inch	Checkerboard Calibration <=> Root Space
4	@\Checkerboard Calibration\NPoint Calibration	CogCalibNPointNPoint Tool	mm/inch	NPoint <=> Checkerboard Calibration
5	@\Checkerboard Calibration\NPoint Calibration\Fixture2	CogFixtureTool	mm/inch	Fixture2 <=> NPoint

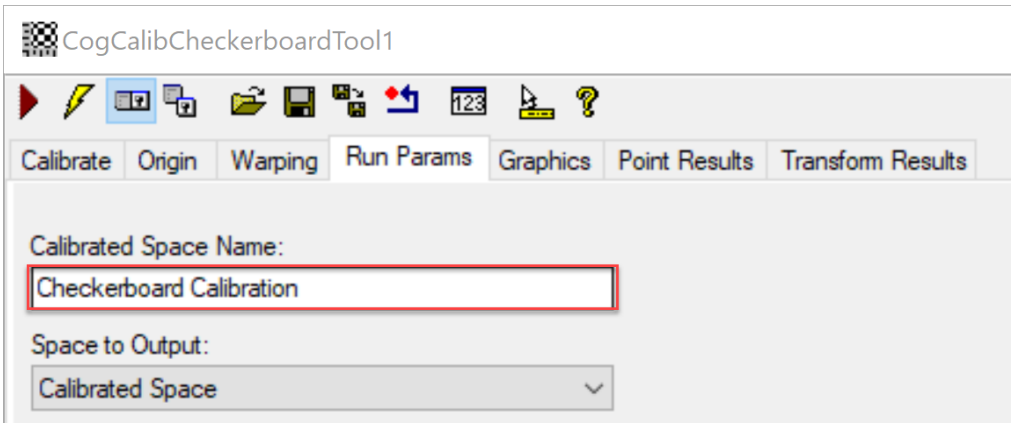
Space Names

Each calibration tool has an default calibrated space name.

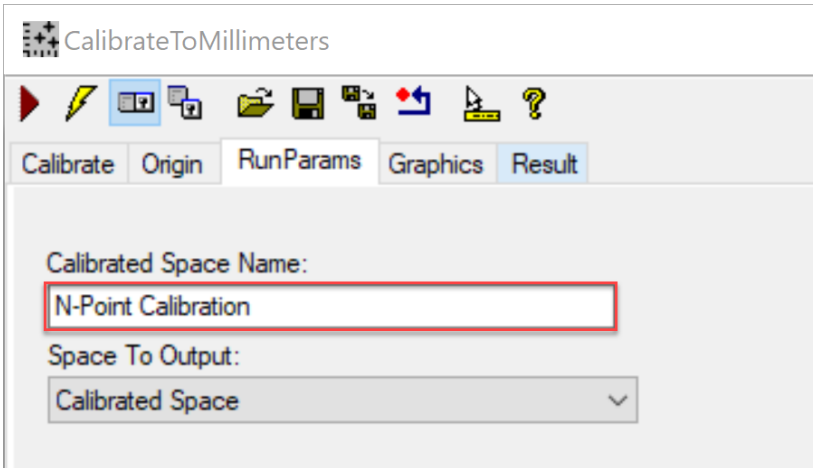
Default calibrated space name of CogFixtureTool is "Fixture".



Default calibrated space name of CogCalibCheckerboardTool is "Checkerboard Calibration".



Default calibrated space name of CogCalibNPointToNPointTool is "N-Point Calibration".

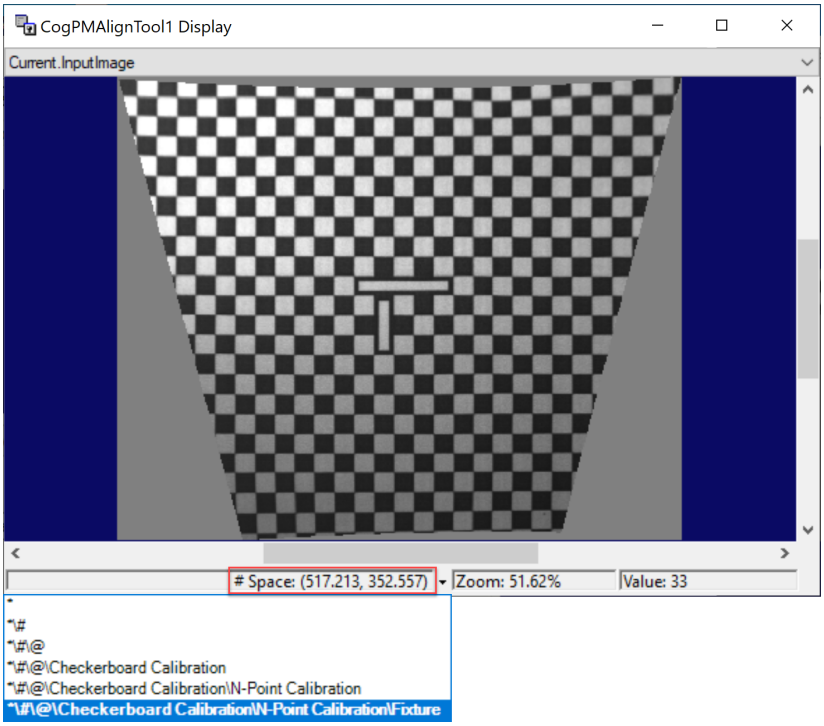


However, user can rename it to make it unique in one space tree.

Note: All space names in one image should be unique, otherwise VisionPro will throw a name space conflict exception. If the same tool is used more than one times in one image, then user needs to manually change their calibrated space names to make them unique.

View Space Tree

All spaces within a space tree of an image can be viewed in VisionPro Floating Display by clicking  button in the toolbar of VisonPro job/tool edit window. By mouse-hovering over the image, the Display will show pointer's coordinates in current selected space. You can select a different space in the drop-down list below and see how coordinates change in different spaces.



However, the change of selected space in Floating Display is only for display purpose and will not affect image's current selected space.

Space Selection

Selected Space

At any given time, one space within the coordinate space tree is designated as the selected space for a given image. The selected space is the coordinate system in which all VisionPro tools that operate on the image return results (such as locations and distances) and in which the tools interpret input data (such as regions of interest). User can set the selected space for an image using the image's SelectedSpaceName property.

When you create a new image using a calibration or fixturing tool, the tool adds a new coordinate space to the input image's coordinate space tree and automatically selects that space as the new image's selected space name, so does the other calibration tools.

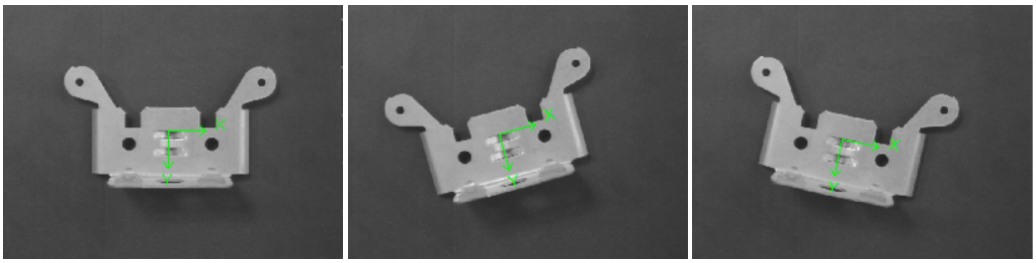
Different space serves different purpose

- **Fixture space**

  Fixture space created by fixturing tool is a user space defined based on parts' run time position. By using this space, other tools' ROI (region of interest) can follow the part.

  Checkerboard calibration tool does changes as follow:

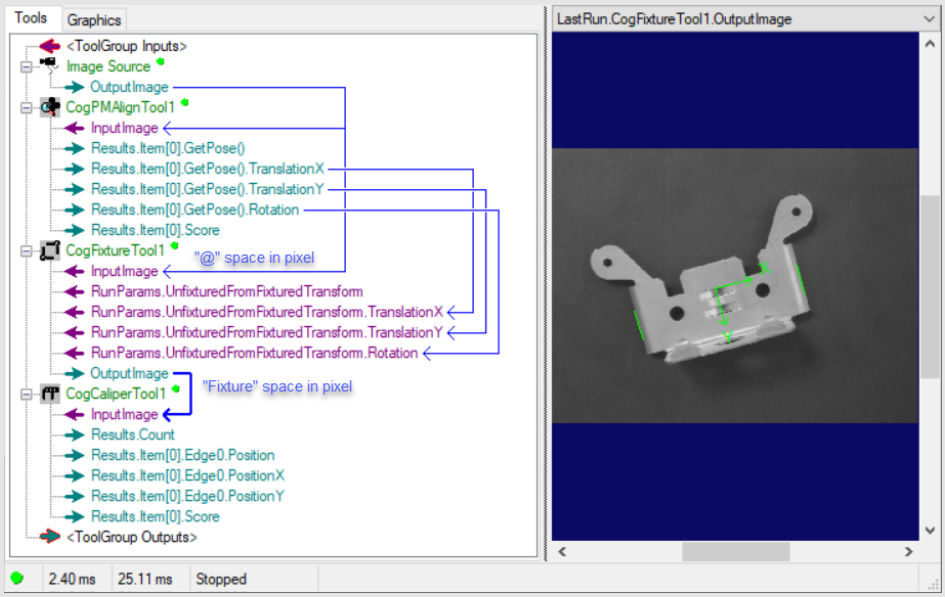
  1. Establish transform between input image's space and Fixture space.
  2. Add "Fixture" space to image space tree.
  3. Set "Fixture" space as selected space of its output image.



Note:
CogFixtureTool doesn't change unit of input image. Therefore, if input image is in pixel, then output image is also pixel. If input image is in mm, then output image will also be in mm.

Here is an example:

In the vision task below, the CogFixtureTool runs on raw image, and output an new image with a new space "Fixture" as its selected space. Then the follow-up CogCaliperTool result which measures the part width will be in pixel, not mini meter.



- **Checkerboard Calibration Space**

  Calibration space is generated by checkerboard calibration tool. Checkerboard calibration tool does the following changes:

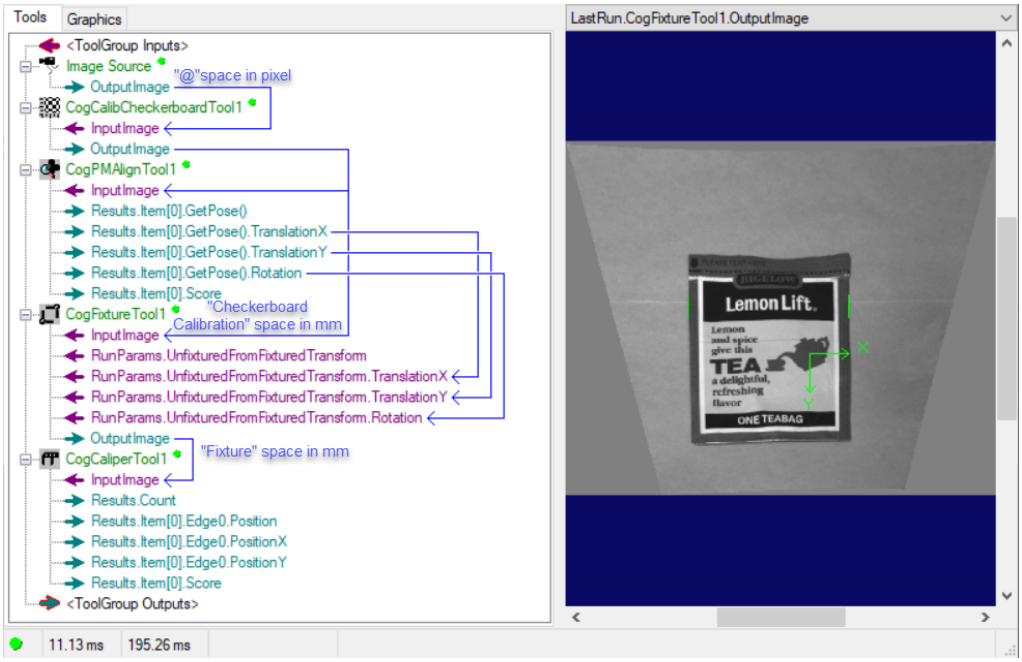
  1. Establish transform between input image's space and checkerboard calibration space.
  2. Add "Checkerboard Calibration" space to image space tree.
  3. Set "Checkerboard Calibration" space as selected space of its output image.
  4. No matter what unit input image space is using, "Checkerboard Calibration" space uses real world unit (mm or inch depending on user's input) as its unit.

  Every VisionPro tool which directly use this output image will get the result(such as location, measurement) based on checkerboard calibration space.

  The most typical vision application using checkerboard calibration space is measurement because measurement requires no absolute positions of features but their relative distances.

  In order to let measurement tool follow run time part, CogFixtrueTool is often used after checkerboard calibration and followed by measurement tool. Here is the one example:

  CogFixture tool works on the output image of CogCalibCheckerboardTool whose selected space is "Checkerboard Calibration". And then CogFixtureTool outputs image with a new space named "Fixture" as its selected space. The measurement tool CogCaliperTool works on Fixture space. This time the measurement result will be in mm instead of pixel.

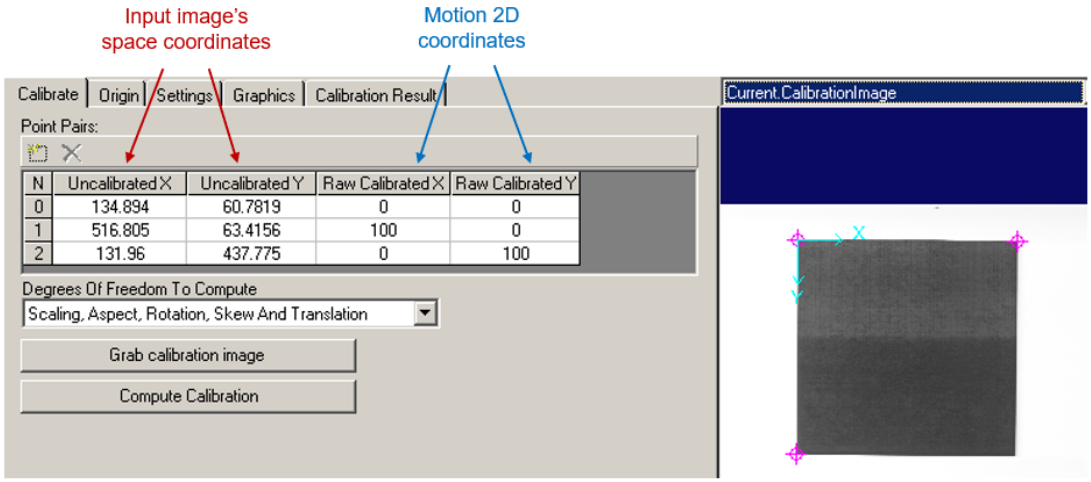


- **NPoint Space**

  NPoint space is the default space created by NPointNPoint Tool.

  NPointNPointTool maps the input image's space to user defined space(such as Stage 2D space) using shared paired features whose coordinates in both spaces are known. NPointNPointTool does the following changes:

  1. Establish transform between input image selected space and motion 2D space.
  2. Add "NPoint" space(stands for Motion 2D space) to image space tree.
  3. Set "NPoint' space as selected space of its output image.
  4. No matter what unit input image space is using, "NPoint" space use mm/inch based on user's input for Motion 2D coordinate.



  Every VisionPro tool which directly uses NPointNPoint tool's output image will get the result(such as location, measurement) based on NPoint space. NPointNPoint tool is wildly used in object locating applications such as picking or placing, it has similar function as hand-eye calibration in AlignPlus. The different is NPointNPoint tool only uses few points to map Raw2D to Motion 2D, and requires very skillful teaching process and manual inputs, so it's much less convenient and also has low accuracy. However in applications which doesn't require high-accuracy, NPointNPoint tool is an good option.

- **Home2D Space**

  The default selected space of output image of hand-eye calibration tool is named as "Home2D". Every VisionPro tool which directly use this output image will get the result(such as location, measurement) based on Home2D space.

  Similar as NPoint Space, Home 2D Space is also related to Motion 2D, but not equal to Stage 2D, it is ideal orthodox coordinate which shares the same origin as Stage2D when motion is at initial position. An transform between Stage2D and Home2D is maintained in Hand-eye calibration result.

  In picking, placing or alignment applications, features on image are extracted and located in Home2D, then vision calculated how much motion should move first in Home2D, later mapped to Stage2D and forward it to motion. So Home2D played an very important role in such applications.

Change Selected Space

There are mainly three ways to change selected space for different purposes:

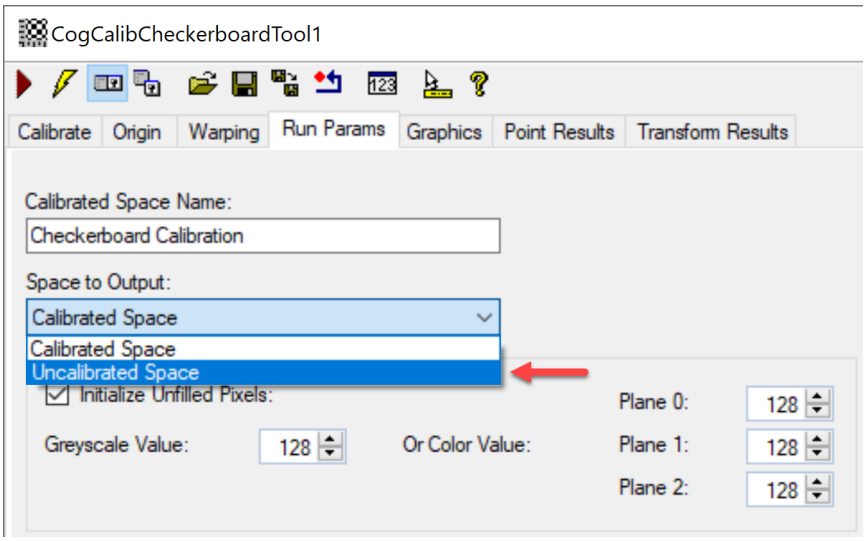
1. Use different input image to change selected space

   As shown above, different fixturing/calibration tool has different selected space for output image. By using those output images as input image, vision tool will work on the corresponding selected space.

2. Manually change default space of an output image

   User can also manually change selected space for output image in fixturing/calibration tool.

   By default, those tools will set the newly generated space as selected space. However, if user choose "Uncalibrated Space", then the output image's selected space will not change, and be the same as input image's.



3. Manually change via scripting

   At any time, user can specify selected space of a particular image via scripting.

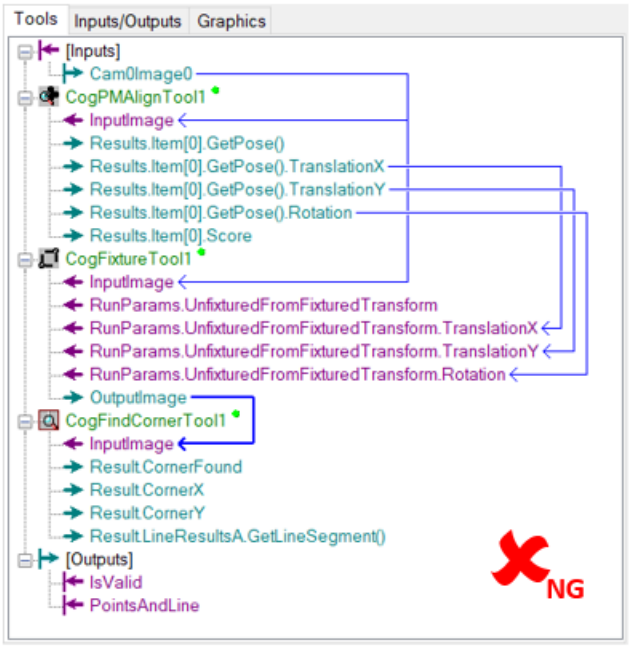
   For example:

```
InputImage.SelectedSpaceName = "Checkerboard Calib";
```

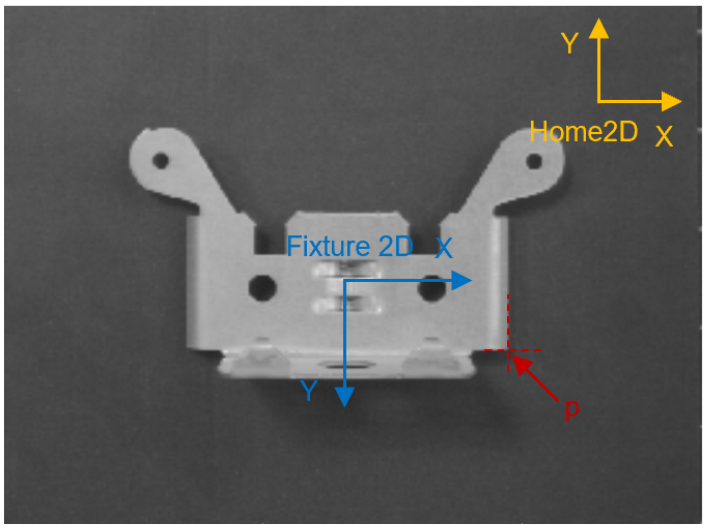
Use Different Space for ROI and Returned Result

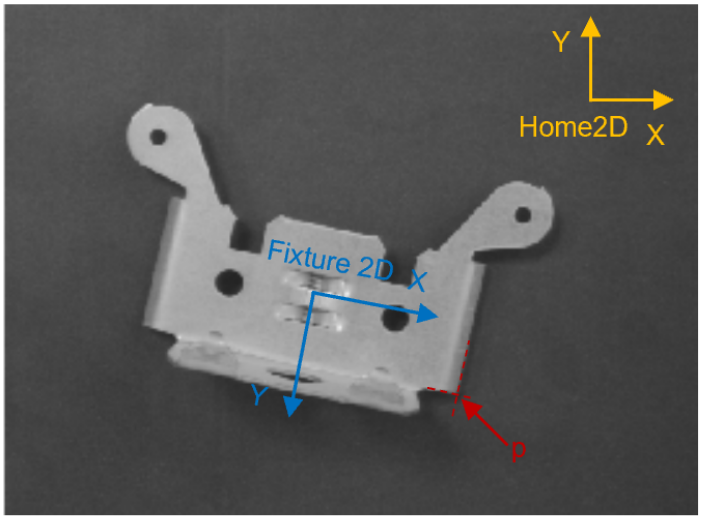
In some applications, the vision tools needs to use different spaces for ROI and returned result respectively.

In the following example, Cam0Image0 is an image in Home2D space. CogFindCornerTool wants to locate the corner point of the image. If CogFindCornerTool use output image from CogFixtureTool, both CogFindCornerTool's ROI and returned result will based on current run time part's Fixture space. However, vision actually wants to locate corner in Home2D so that it can guide how much motion should move. So CogFindCornerTool should report corner point result in Home2D instead of in Fixture space.



In the picture below, corner point's coordinates in Fixture 2D doesn't change in two images, but its Home2D coordinates actually changed a lot.

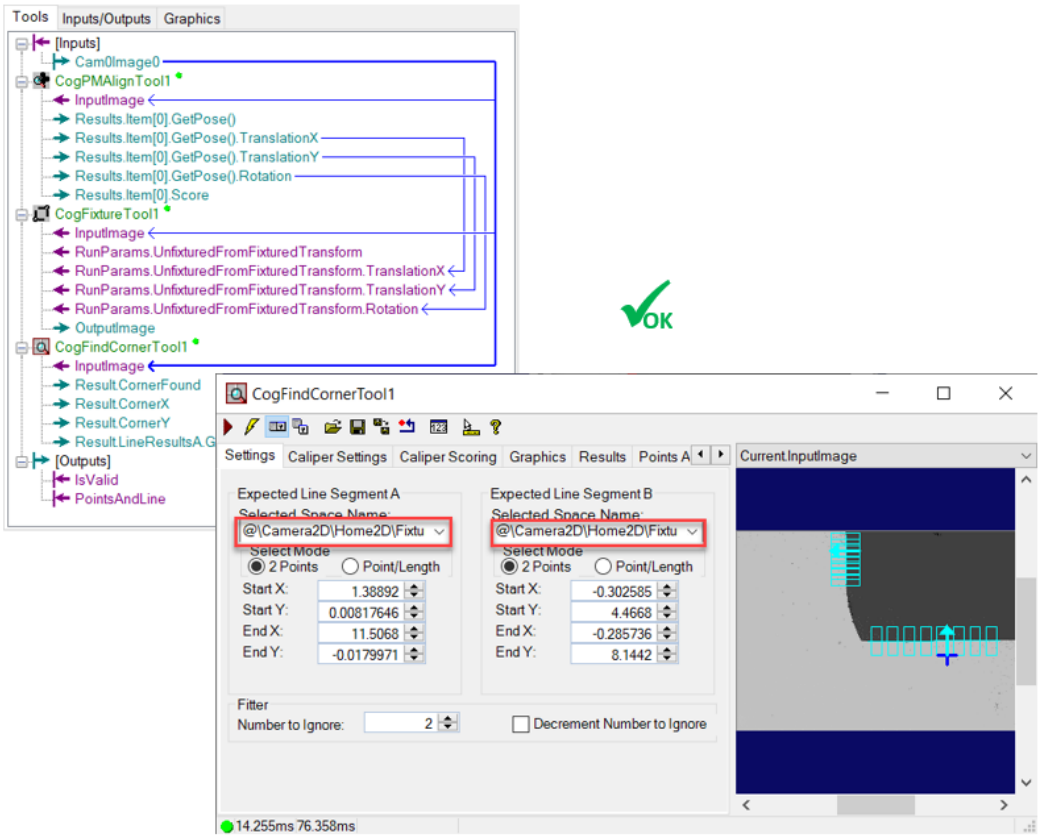




In order to follow the fixture space to find corner and report result in Home2D, CogFindCornerTool needs to use different space for ROI and returned result.

Here is the way to configure it:

- Use input image with Home2D as selected space
- Manually select fixture space for ROI selected space



Get Transform in Space Tree

The coordinate space tree includes methods that let user obtain a composed transformation that can map points between any two spaces anywhere in the coordinate space tree.

If user request a transformation using the GetTransform method of an image, what's needed only is the names of the coordinate spaces between which user want to transform points. In VisionPro, "." stands for the currently SelectedSpaceName, "@" stands for the root space, "#" stands for pixel space. For user spaces, just use their names.

Here is an example to get the transform between Home2D and root space:

```
CogTransform2DLinear Home2DFromRoot2D = InputImage.CoordinateSpaceTree.GetTransform("Home2D","@") as CogTransform2DLinear;
```

Space Tree in AlignPlus

In AlignPlus, there are mainly two spaces to consider in space tree:

1. Checkerboard Calibration Space (corresponding to Plate2D)
2. Home2D Space

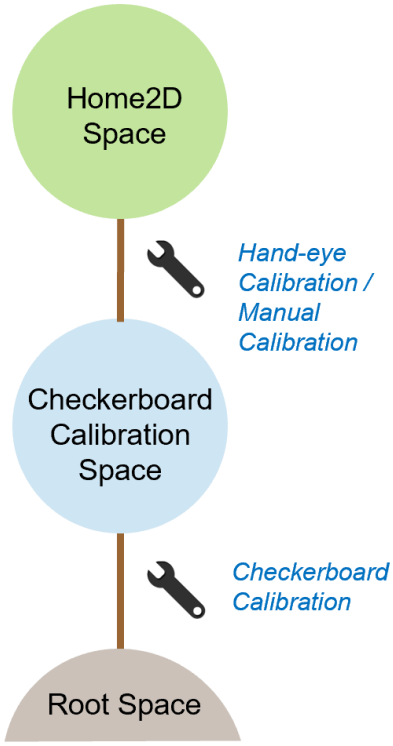
Checkerboard calibration space is a by-product of checkerboard calibration, it's built on corrected image and originated at Plate2D.

Home2D space is generated through hand-eye calibration or manual calibration which maps Root2D (equal to Raw2D) to Home2D so that every feature located in the image can output Home2D coordinates.

Space Tree generated by Hand-eye Calibration or Manual Calibration

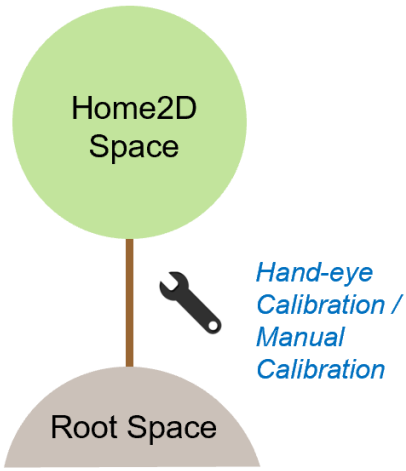
- Checkerboard-based hand-eye calibration or hybrid manual calibration

  Both checkerboard-based hand-eye calibration and hybrid manual calibration use checkerboard to correct lens distortion and camera perspective distortion. So the space tree for each camera will have both checkerboard calibration space and Home2D:



> **Note**: Once root space to checkerboard calibration space, and checkerboard calibration space to Home2D space transforms are established, Home2D to root space transform is also established. User can directly use image's GetTransform method to get transform between any two spaces in space tree. (To get Home2D from root space transform, call GetTransform("Home2D", "@")).

- Part-based Hand-eye Calibration or Non-hybrid Manual Calibration directly build Home2D upon root space.

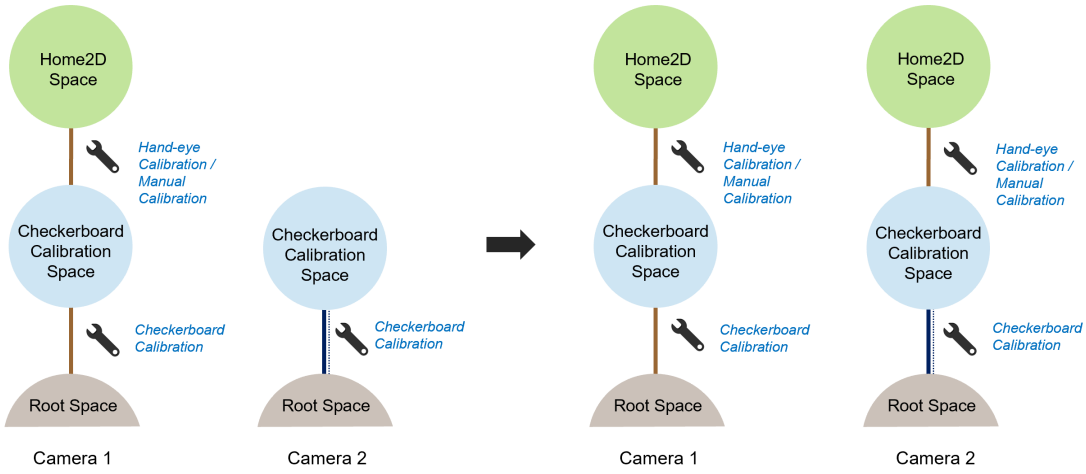


Spaces generated by Cross Calibration

Cross calibration itself doesn't create new Home2D, but maps Home2D from one set of cameras to another set of cameras as they all share the same Home2D.

- Checkerboard-based Cross Calibration

  For checkerboard-based cross calibration, the same calibration plate was used for Camera1 and Camera2, since Camera1's Home2D and checkerboard calibration space transform is already known, Camera2 then can copy this transform and attached it to Camera 2. After calibration, both cameras have their own root space to checkerboard calibration space transforms but share the same checkerboard calibration space to Home2D transform.



- Part-based Cross Calibration

  In part based cross calibration, one run time part or dummy part was transferred from Camera1 to Camera2. The features on part under Camera1 already have Home2D coordinates. When they're moved to under Camera2, they'll have new Root2D coordinates. However, their Home2D coordinates doesn't change, therefore, Home2D from Camera2's Root2D transform can be calculated by mapping the two coordinate pairs. After calibration, both cameras have their own Root2D to Home2D transforms. Whereas, the Home2D coordinate here are the same.