

Ether-Inspect 11 Template Documentation

1 Contents

2	Template Overview	4
2.1	What is a template	4
2.2	Template Types	4
2.3	Compatibility	5
3	EtherInspect Template	6
4	EtherInspectViDi	7
5	BeadInspect	8
5.1	Setting Pattern Tools for Fixturing	9
5.2	Two Point Fixturing Tool	9
5.3	Image Masking Tool	10
5.4	Beads Tool	11
5.4.1	Common Settings	11
5.4.2	Individual Setting	12
5.4.3	Generating Segments	12
5.4.4	Master Segments	13
6	TrayInspect Template	14
6.1	Image Processing	15
6.2	Tray Definitions	15
6.2.1	Common Settings	15
6.2.2	Tray Tool	16
6.2.3	Display Options	17
6.3	Inspection of Pockets	18
6.4	Check for Part Present	19
6.5	TrevistaViDi	19
7	Edge Learning	20
7.1	Classify Tool	20
7.2	Segment Tool	21
8	Trevista	24
8.1	Trevista Tool	25
8.1.1	Tool Result Images	27
8.1.2	Floating Point to Greyscale Mapping	28
9	Alignment	30
9.1	Checkerboard Calibration	31
9.2	Real Part Calibration	32
9.3	Calibration Report	33

9.4	Alignment Finder.....	35
9.4.1	Pattern Feature Finder	36
9.4.2	Corner Feature Finder	37
9.5	Alignment Compute	38

2 Template Overview

2.1 What is a template

A template in EtherInspect is a preconfigured set of tools used for a particular application. A template serves as a base to build your job off.

2.2 Template Types

There are several templates in EtherInspect

- EtherInspect
 - Standard EI11 project with general VisionPro tools
- EtherInspectViDi
 - EtherInspect template with ViDi tool to load in VisionPro Deep Learning and ViDi workspaces for inspection
- BeadInspect
 - A combination of VisionPro and custom EtherInspect tools to inspect the bead in a bead sealant application
- TrayInspect
 - Runs job inspection on multiple pockets of a tray formation and/or grid
- TrayInspectViDi
 - TrayInspect template with ViDi tool
- EdgeLearning
 - Template for standard job inspection similar to EtherInspect but with added Cognex Edge Learning tools including Segment and Classify
- EdgeLearningViDi
 - Edge Learning template with ViDi tool
- Trevista
 - Used with Trevista Light dome alongside Edge Learning tools to run surface inspection
- TrevistaViDi
 - Trevista template with ViDi tool

2.3 Compatibility

Depending on the VisionPro and VisionPro Deep Learning firmware versions, certain templates will be excluded from the list. For more information on compatibility with templates and their target VisionPro and VPDL version, refer to the table below:

Template Name	VisionPro Version	VPDL/ViDi Version
<i>EtherInspect</i>	9.5 SR2, 9.6 SR2, 9.8, 9.10, 9.20	ViDi 4.1 VPDL 1.1, 3.0, 3.1, 3.2
<i>EtherInspectViDi</i>	9.5 SR2, 9.6 SR2, 9.8, 9.10, 9.20	ViDi 4.1 VPDL 1.1, 3.0, 3.1, 3.2
<i>BeadInspect</i>	9.5 SR2, 9.6 SR2, 9.8 9.10, 9.20	ViDi 4.1 VPDL 1.1, 3.0, 3.1, 3.2
<i>TrayInspect</i>	9.5 SR2, 9.6 SR2, 9.8 9.10, 9.20	ViDi 4.1 VPDL 1.1, 3.0, 3.1, 3.2
<i>TrayInspectViDi</i>	9.5 SR2, 9.6 SR2, 9.8 9.10, 9.20	ViDi 4.1 VPDL 1.1, 3.0, 3.1, 3.2
<i>EdgeLearning</i>	9.10, 9.20	VPDL 3.1, 3.2
<i>EdgeLearningViDi</i>	9.10, 9.20	VPDL 3.1, 3.2
<i>Trevista</i>	9.20	VPDL 3.2
<i>TrevistaViDi</i>	9.20	VPDL 3.2

3 EtherInspect Template

The **EtherInspect** template is a standard template that contains commonly used VisionPro tools to run job inspection. These tools include the following:

- Calibrate
 - Used to calibrate images and convert from image coordinates to real-life measurements
- Image Filtering (CogIPOneTool)
 - Used to set filters on images; Creates new image filter in list to display separately from raw image
- Image Conversion
 - Converts images to greyscale
- Fixturing
 - Sets fixture point based on point of interest
- Pattern Tool
 - Used to find match features in a pattern within the image
- Histogram
 - Analyzes pixel greyscale values to generate a histogram of pixel values and weights
- Blob
 - Used to find regions of similarly weighted pixels (blobs)
- Measure
 - Utilizes calipers to measure distance between two points in an image
- LineMax
 - Finds a line segment that best fits a given region
- Find Line
 - Uses a series of caliper to find a line within a given region
- Find Circle
 - Uses calipers to find a circle within a given region
- Codes
 - Finds a barcode/data matrix code within a given region
- Color Segmenter
 - Separates pixels of a target color from the remaining image to create an image filter
- 2D Geometry
 - Custom EtherInspect tool for detecting certain geometrical figures given a series of points/lines; Can gather measurement data from figures for further analysis
- Cross Section
 - Takes a thin slice of a 3D range image and allows users to make measurements and find tolerances
- Find Ellipses
 - Uses a series of calipers to find an ellipse within a given region
- Image Crop
 - Custom EtherInspect tool used to crop a given region within an image and creates a separate filter for the cropped image
- Image Masking
 - Masks an image within target regions specified by the user with a given pixel value; Creates a separate image filter to display masked image
- Parallel
 - Allows user to allocate multiple threads to run certain toolblocks
- Custom Toolblock
 - Allows user to create a custom VisionPro toolblock
- Scripting

- Allows user to create a custom script that runs after inspection
- Pre job scripting
 - Allows user to create a custom script that runs prior to running job inspection

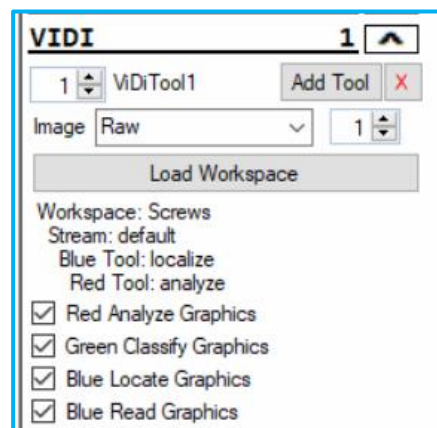
Information on all of the above tools can be found within the main EtherInspect 11 documentation.

4 EtherInspectViDi

The **EtherInspectViDi** template contains the same tools as the EtherInspect template but adds the ViDi tool to the job.

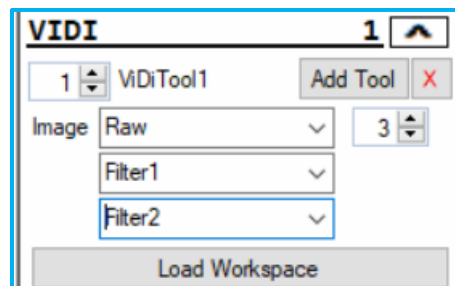
The **ViDi** template is used to load and run VisionPro Deep Learning and/or ViDi workspaces. This utilizes machine learning to detect features within an image. The ViDi tool accepts **.VRWS** files.

Note: ViDi workspaces can take time to load into an EtherInspect job due to having to load in the machine learning model associated with it.



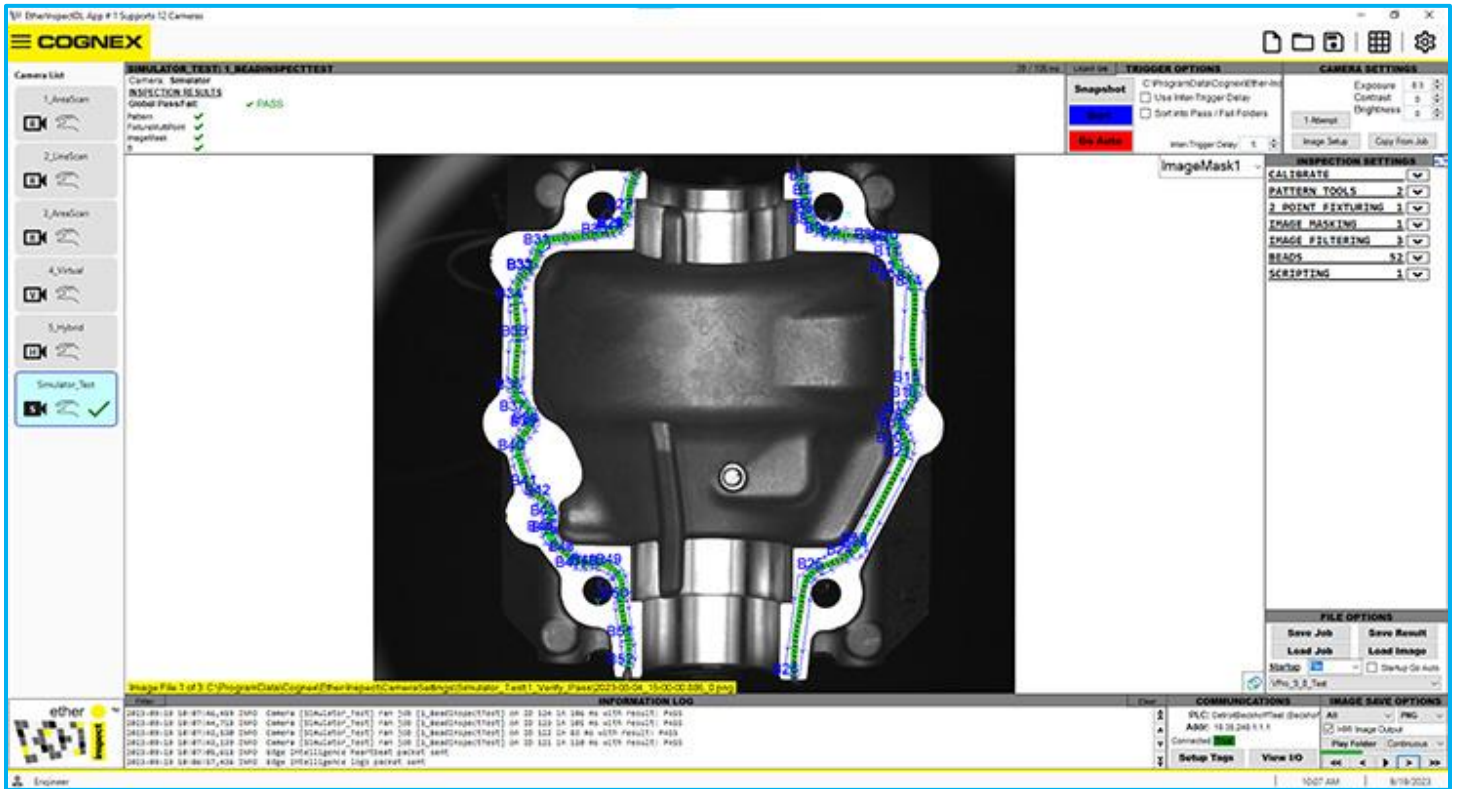
The graphics for the ViDi tool can be shown or hidden from the image display. The red tool displays as red contours. The green tool places a green rectangle around the region that matches the classification. The blue tool shows the centerpoint of the object that it has located.

EtherInspect supports **Multi-Channel Deep Learning** allowing the user to add multiple images to run on the same ViDi workspace.



5 BeadInspect

The **BeadInspect** template uses a custom Beads tool to combine multiple calipers to inspect bead sealer on tires.

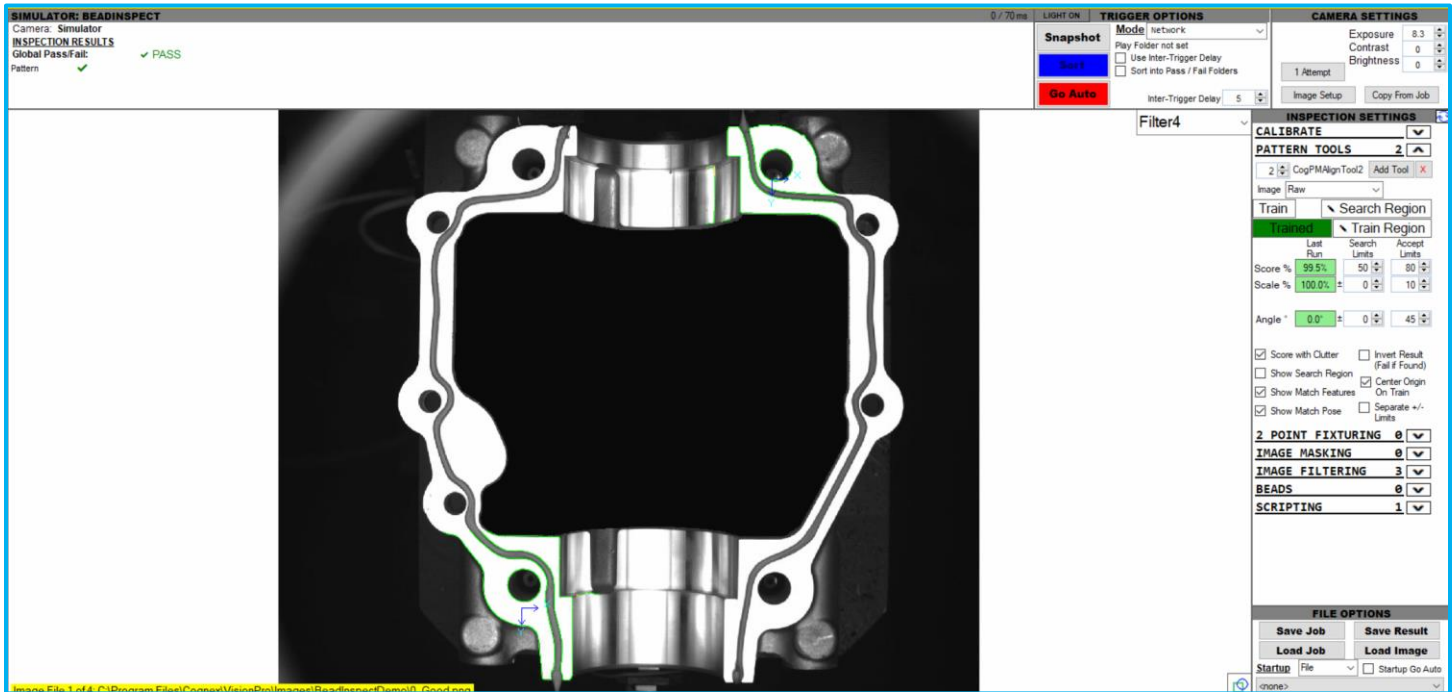


There are four main tools used specifically with the BeadInspect template:

- Pattern Tool (PatMax)
- 2 Point Fixturing (CogNPointToPoint)
- Image Masking (Custom EtherInspect Tool)
- Beads Tool (Custom EtherInspect Tool)

5.1 Setting Pattern Tools for Fixturing

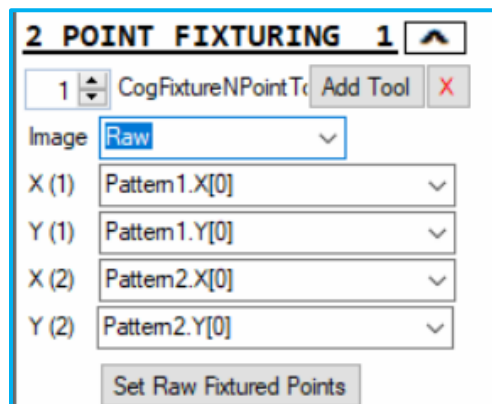
Prior to using the 2 Point Fixturing tool (CogNPointToPoint), two pattern tools need to be used as fixture points for 2 point fixturing. Make sure to use unique regions in the image for the pattern tools for the fixturing to properly work. In the example below, two regions are used: the bottom left and bottom right of this cylindrical encasement.



5.2 Two Point Fixturing Tool

Once the pattern tools are set, 2 point fixturing can be used to set an overall fixture between those two points. This helps with aligning the calipers that will be generated from the Beads tool for proper bead inspection.

To set the tool, an input image needs to be provided. In this example, the “Raw” image will be used. The X and Y coordinates of two separate points will need to be set. To do this, use the X and Y of each pattern found by the pattern tools to set the two-point fixturing. After setting the points, click **Set Raw Fixtured Points** to set the fixture.



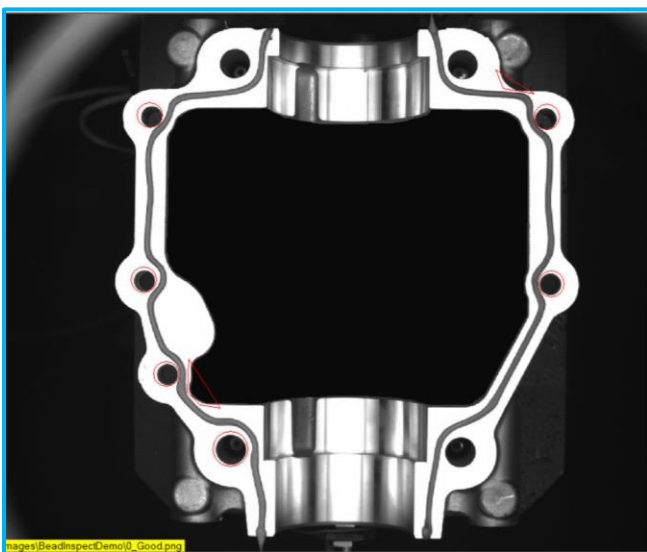
5.3 Image Masking Tool

After setting the two-point fixture, an image mask may be needed. The **Image Masking** tool allows users to set multiple masks to cover regions of an image. This is useful when areas of the bead can easily blend in with other elements of the image.

To set the image masking tool, select the **Edit Region** button to go into special edit mode. From here, different shapes can be added to specify the regions to mask.



After selecting the image shape to use, selecting the  button will add the shape to the image. This can then be modified to cover the region of interest. To remove the shape, select the  button.



5.4 Beads Tool

The **Beads** tool utilizes a series of calipers to inspect the bead on an encasement. The beads tool is split into two sections:

- Common Settings
 - General settings for the entire series of calipers
- Individual Settings
 - Settings for individual calipers

5.4.1 Common Settings

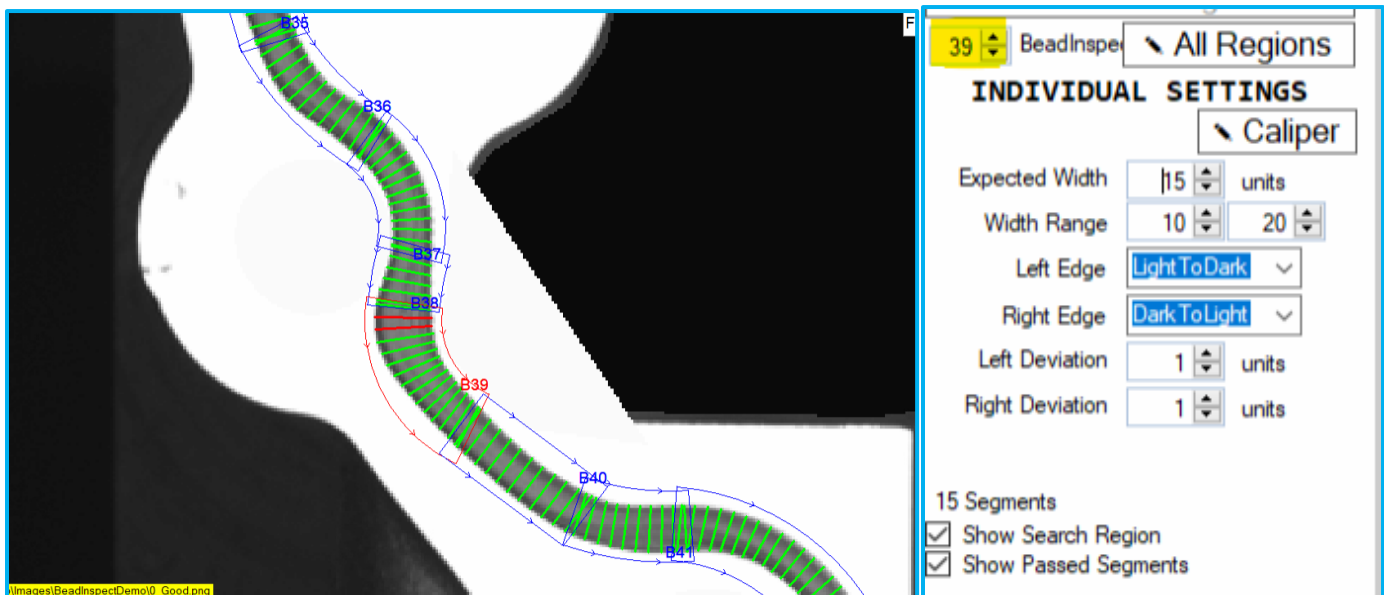
Common settings apply to all calipers that are not defined using the individual settings. This serves as a default when creating and setting new calipers.

There are several settings included in the common settings section:

- Expected Width
 - Expected width of caliper needed to pass inspection
- Width Range
 - Range of allowable caliper widths that will pass inspection
- Scoring mode
 - Algorithm to determine scoring of each caliper
- Right Edge
 - Contrast setting to find right edge of bead
- Left Edge
 - Contrast setting to find left edge of bead
- Min Contrast
 - Minimum contrast required to detect edge of bead
- Edge Width
 - Expected width of edge of bead
- Caliper Overlap
 - Amount of overlap allowable for calipers
- Exceptions
 - Number of failed calipers to ignore before failing inspection
- Left Deviation
 - Amount of deviation for left edge
- Right Deviation
 - Amount of deviation for right edge

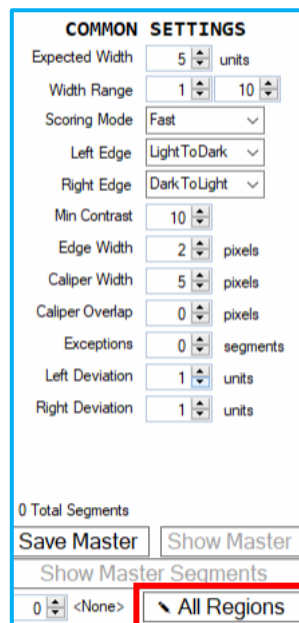
5.4.2 Individual Setting

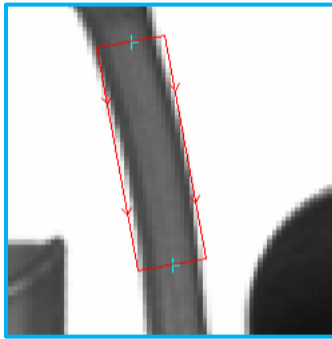
Individual settings apply to a specific caliper. These settings are usually edited during edge cases where certain calipers fail inspection on an expected PASS image. The expected width, width range, left and right edge and deviations can all be edited individually from the overall default. To select a specific caliper, change the number to the left of the individual caliper section to match with the number of the bead that is to be edited.



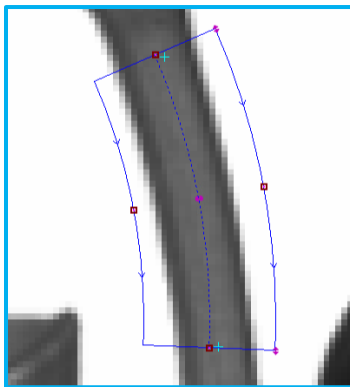
5.4.3 Generating Segments

To generate a segment to create a series of calipers in that region, click the **All Regions** edit button. This will open special edit mode. From here, double click the first point that the segment should start from. Then, double click the second point that the segment should end at. This will generate a region that can be edited to cover the bead.

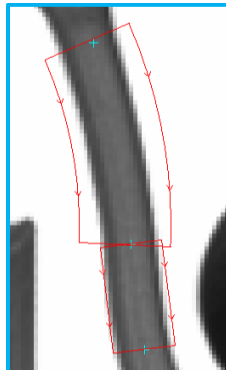




The middle point in the generated segment can be clicked and dragged to turn the segment. This can help with matching areas of the bead that curve around a point. The outer points can be used to increase the size of the segment. The arrow to the top right and bottom left of the segment can be clicked and dragged to pull and expand the segment.

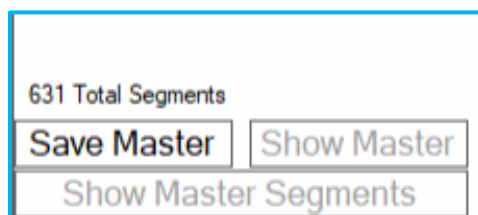


Any subsequent segments can be generated by double clicking on the next look-ahead point. This will generate a segment from the end point of the previous segment to the end point of the next segment.



5.4.4 Master Segments

The Bead tool uses master segments as a point of reference to calculate deviations from those segments. Once master segments are set, the overlay of the master segments can be viewed as reference to see whether certain calipers fit within that region. The image used to generate the master segments can also be viewed at any time.



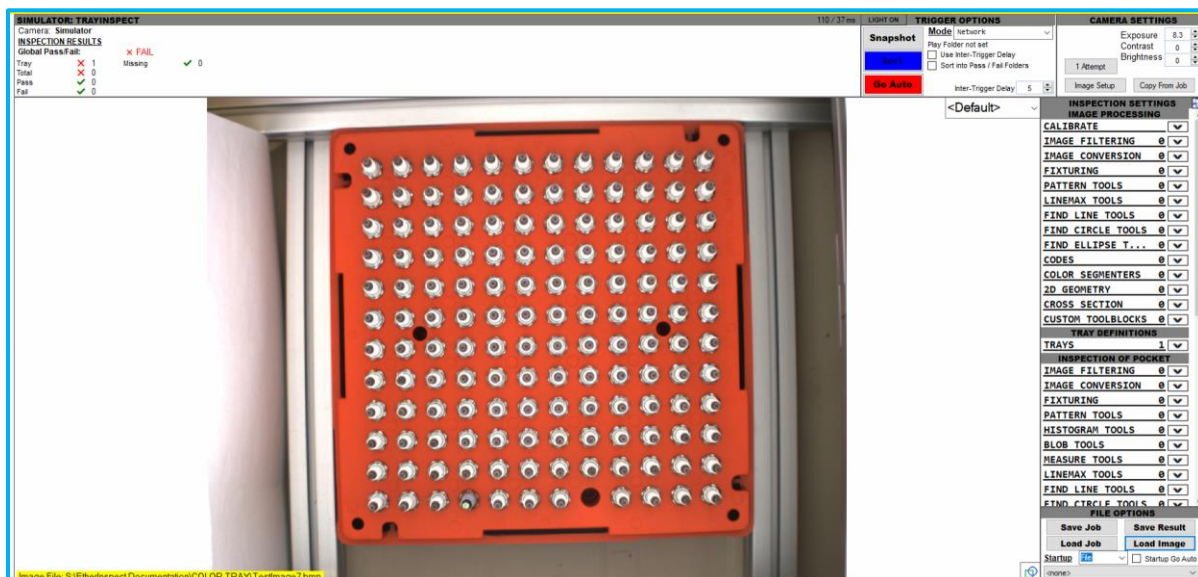
6 TrayInspect Template

The **TrayInspect** template is used to inspect an array/series of regions within an image using the same tools for each. Each region is referred to as a **pocket** and the overall array of pockets is a **tray**. The inspection settings for the TrayInspect template are split into 5 sections:

- Image Processing
 - Prepares the image for inspection whether that be converting the image to greyscale or using a color segmenter to filter out certain colors in an image
- Tray Definitions
 - Can define common settings for tray region and properties of the tray including number of rows and columns and inspection region
- Inspection of Pockets
 - Run VisionPro and EtherInspect tools on individual pockets
- Check for Part Present
 - Run VisionPro and EtherInspect tools to check whether target region contains part or is missing
- Script for Whole Tray
 - Runs script on entire tray

The general workflow for setting up a trayInspect job is as follows:

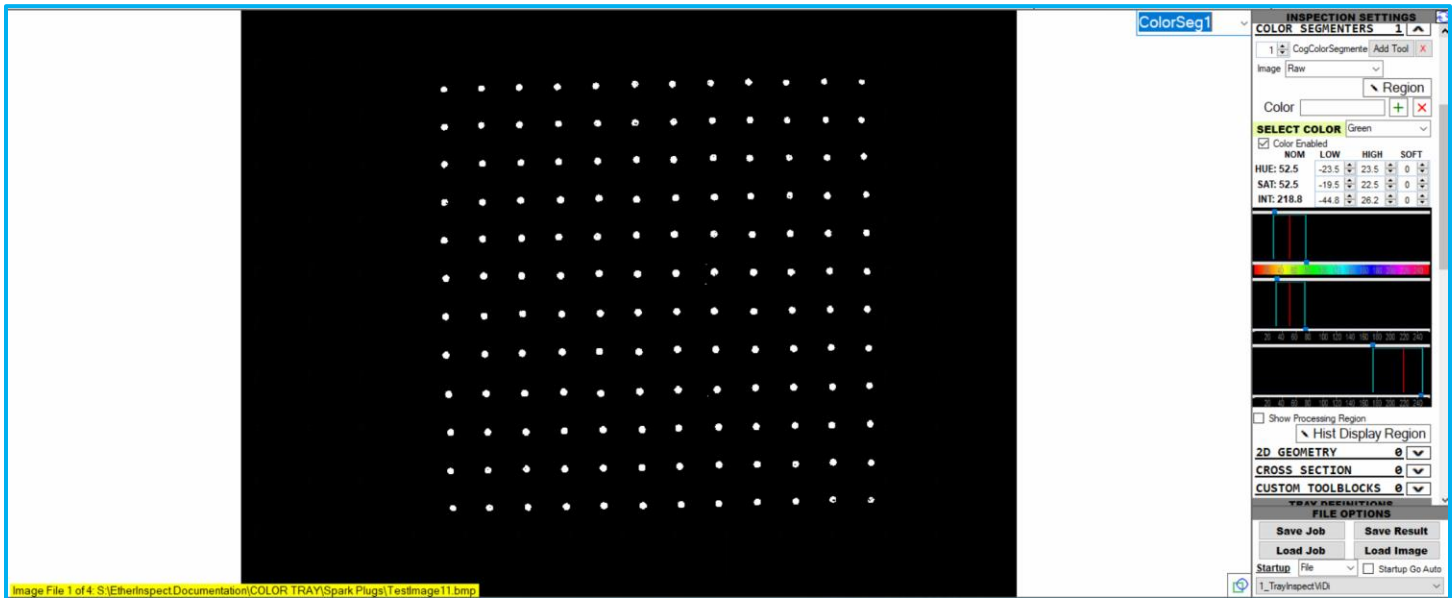
1. Prepare image for inspection using Image Processing section (optional)
2. Define the tray to run a series of inspections
 - a. Number of rows
 - b. Number of columns
 - c. Tray to use
 - d. Inspection image to use
 - e. Region shape, size and locations
3. Set tools to run inspection on each pocket in the tray
 - a. Examples include Blob tool, histogram, pattern, etc.
4. Set tools to check whether part is missing or present
 - a. What is the difference between a missing part and an incorrect part?
5. Custom Script for tray (optional)



6.1 Image Processing

Image processing is used to prepare an image for inspection. This may include filtering out colors that are not the target color to reduce noise, finding certain object within an image to use as fixture points, etc.

In the example below, a tray filled with spark plugs is being used for inspection. Any missing or different colored spark plug tips will be rejected. A color segmenter tool is used below to separate the tips of the spark plug image from the rest of the tray.



6.2 Tray Definitions

A tray will need to be defined indicating an array/series of inspection regions to run the same tools on. The tray is split into pocket regions equidistant from one another. There are two sections when defining a tray:

- Common Settings
- Tray Definition

6.2.1 Common Settings

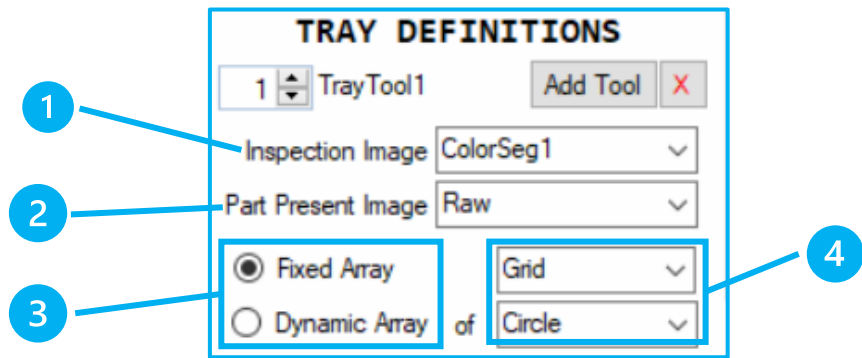
Defines criteria for passing overall inspection and settings for a single pocket. Each setting is defined below:

- Max Acceptable Missing
 - Maximum amount of pockets missing the target part before inspection fails
- Part Present Check
 - Can run check for whether part is present when overall pocket inspection fails, force the check each time, or skip checking for missing parts
- Draw All Tool Graphics
 - Display graphics for all tools in each pocket
- Tray
 - Specify which tray tool to use for inspection
- Region
 - Define which region number the settings should be adjusted for
- Save Region Image
 - Saves region image to disk

In the example above, the settings were kept at their default values.

6.2.2 Tray Tool

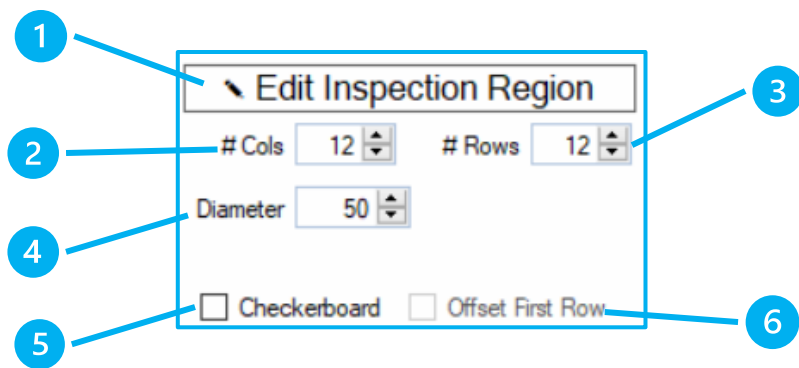
A tray tool is used to define the array of regions and region size/shape. Each tray tool defined can be set up as either a **fixed** array or **dynamic** array. This means that the point where the tray tool starts is in a fixed location. Dynamic arrays changes the number of rows/columns based on a source array. Settings are defined below:



Number	Name	Description
1	Inspection Image	Image to use for inspection
2	Part Present Image	Image to use for part present check
3	Array Type	Type of array to use for tray
4	Region Definition	Defines region shape and layout for tray

Fixed Array

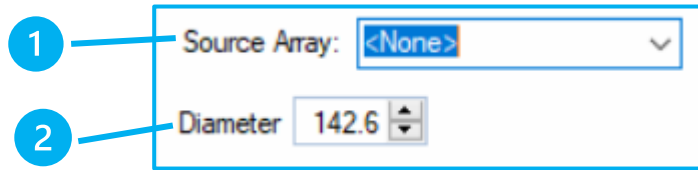
If **fixed** array is selected for the array type, the following controls will be used to define the tray:



Number	Name	Description
1	Edit Inspection Region	Edit the main inspection region to overlay the tray onto
2	Number of Columns	Specify number of columns in the tray
3	Number of Rows	Specify number of rows in the tray
4	Diameter (Circle)	Specify diameter of circular region for pockets (rectangle uses width and height)
5	Checkerboard	Use every other pocket for inspection
6	Offset First Row	Offset checkerboard region by 1 pocket

Dynamic Array

If **dynamic** array is selected for the array type, the following controls will be used to define the tray:



Number	Name	Description
1	Source Array	A collection of VPro tools used as a base reference for the array
2	Diameter (Circle)	Diameter of circular region (Rectangle uses width and height)

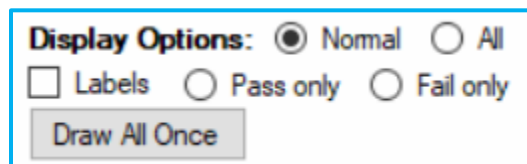
Valid options for a source array include:

- Directly link with VisionPro:
 - CogBlobCollection (Output of Blob tool)
 - CogPMAAlignResults (Output of PatMax tool)
 - Cog CaliperResults (Output of Caliper tool in single edge mode)
- Requires Custom ToolBlock
 - List<CogTransform2DLinear>
 - List<ICogRegion>
 - CogGraphicCollection containing ICogRegions

These outputs are acquired from the image processing section and auto-populate within the drop-down menu.

6.2.3 Display Options

Display options in the Tray tool allow users to specify which graphics to display within the tray.



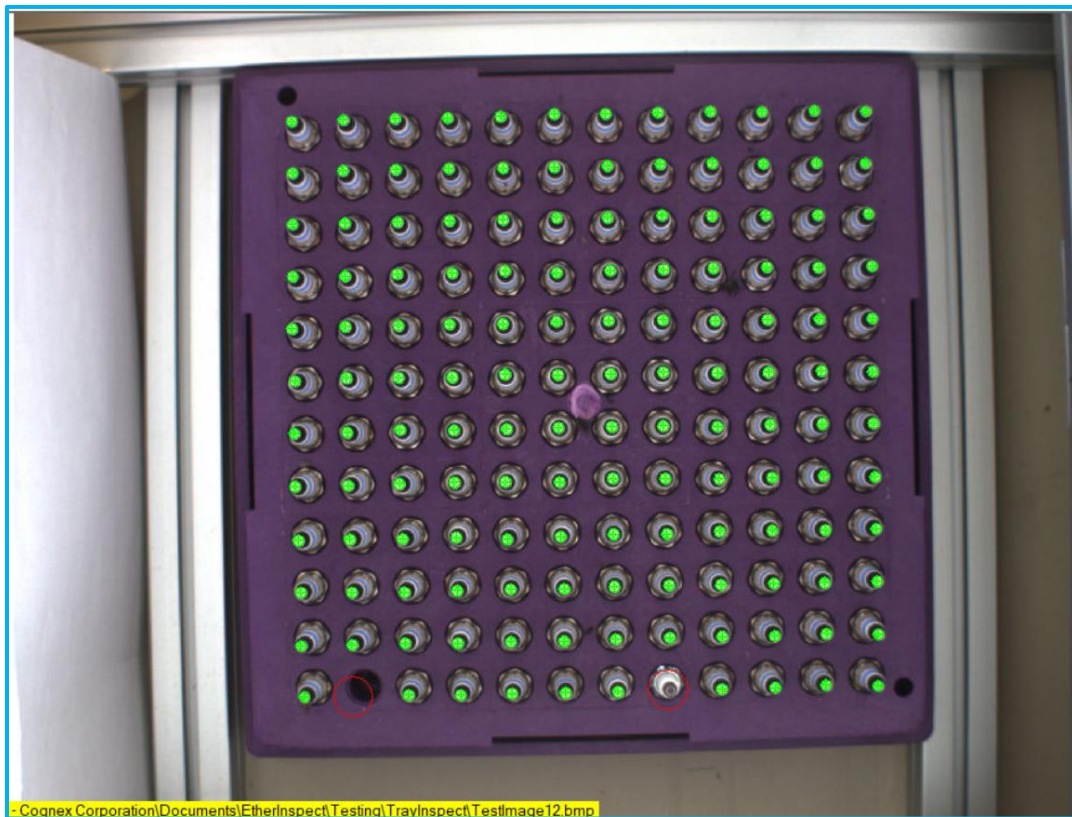
Display options include:

- Normal – Show default display settings
- All – Show graphics for all pockets
- Labels – Add labels to individual pockets
- Pass Only – Show graphics for images that pass inspection
- Fail Only – Show graphics for images that fail inspection
- Draw All Once – Displays the graphics with the given settings

6.3 Inspection of Pockets

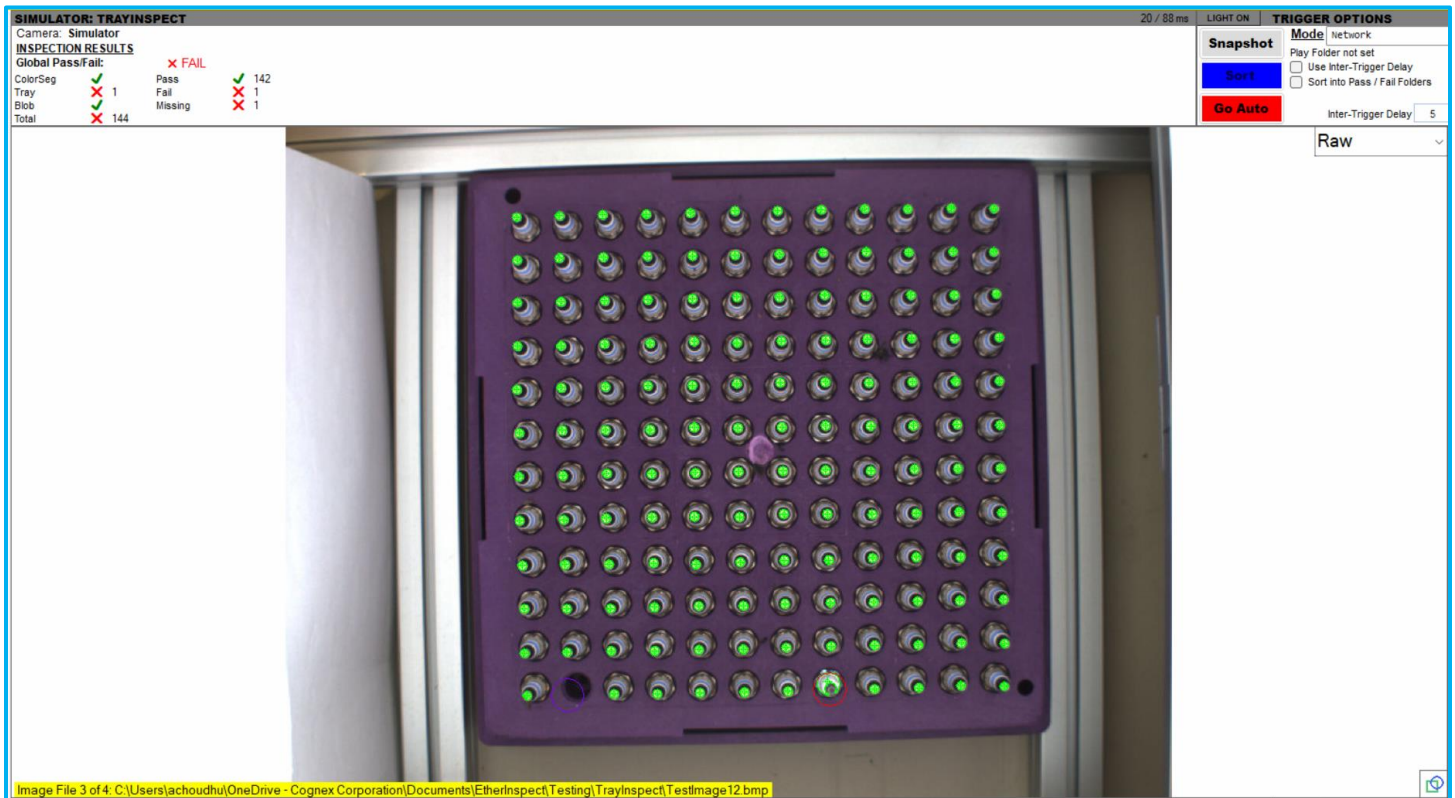
Each tray is split into an array of pockets. These pockets are regions within the tray that have the same regions shape/size and are equidistant from one another. Adding and modifying tools in this section will apply to all pockets.

In the example below, a blob tool is used to find a missing/different colored spark plug in the pockets. Any pockets that fail inspection in this section are either marked as missing or are marked as a different type than the target part.



6.4 Check for Part Present

Part present check can use a series of VisionPro and EtherInspect tools to differentiate missing parts from parts of a different type. In this case, there is a purple spark plug in the tray on occasion which is of a different type than the rest of the spark plugs. Part present check will determine whether this type of spark plug is present or if the part is missing entirely. In the image below, a blob tool is used to find white pixels in the pocket which represents the out section of the spark plug. If a pocket is marked with a red circle in this situation, then it is of a different type (fail). If the pocket is marked with a purple circle, then it is missing.



As you can see in the inspection results, there is one pocket that failed inspection (different type) and one pocket that is marked as “missing”.

6.5 TrevistaViDi

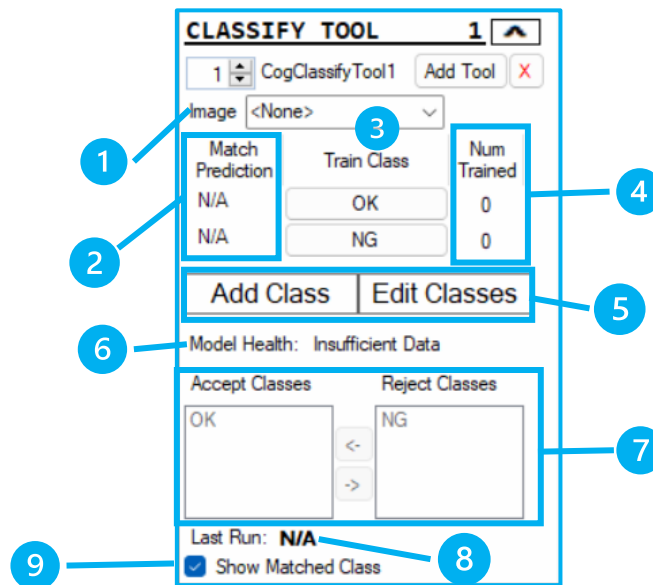
TrevistaViDi contains an option to load a VPD/ViDi workspace within each section of the tray inspection (Image Processing, Inspection of Pocket, Check for Part Present). When loading a TrevistaViDi job with multiple ViDi workspace in each of the sections, it may take up to a minute for the job to display, depending on the size of the deep learning model.

7 Edge Learning

The Edge Learning template is a standard EtherInspect template with the addition of the **CogClassify** and **CogSegment** tools. These tools are a part of a subset of artificial intelligence called **Edge Learning**. Edge Learning occurs when processing of an image or part occurs on-device or “at the edge” of where the data originates. This requires less time and fewer images when training a model compared to other AI-based solutions.

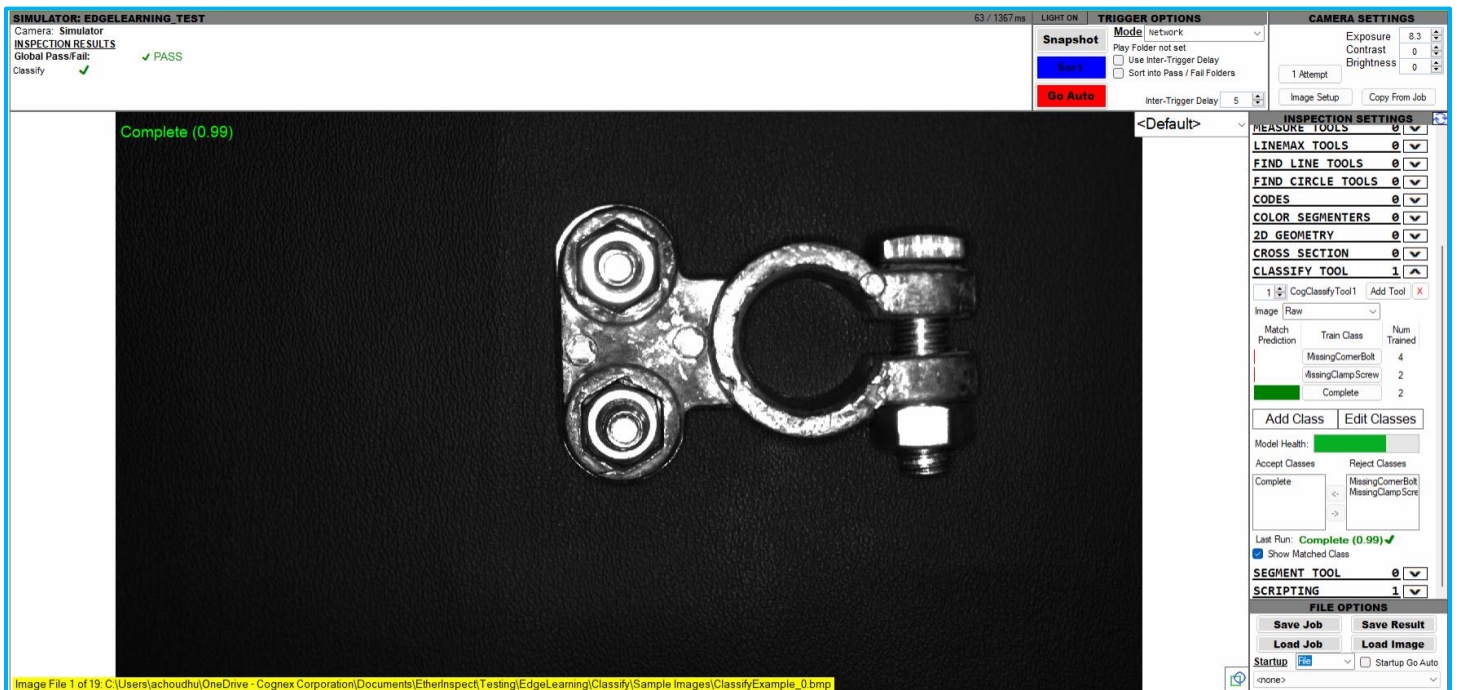
7.1 Classify Tool

The **Classify** tool is used to identify what a part within an image and associate it with a particular class. Each classification has a confidence score which tells you the likelihood that a part fits within a particular class.



Number	Name	Description
1	Input Image	Image to run with classify tool
2	Match Prediction	Shows confidence score for each class
3	Train Class	Allows user to assign classes based on the image provided
4	Number Trained	Number of images trained with target class
5	Add/Edit Class	Allows users to add and edit classes for inspection
6	Model Health	Overall reliability/stability of EL model
7	Class Assignment	Assigns classes to either pass inspection when detected or fail when detected
8	Last Run	Shows class with highest confidence score from last run
9	Show/Hide Matched Class	Show/Hide class with highest confidence score in top left of image display

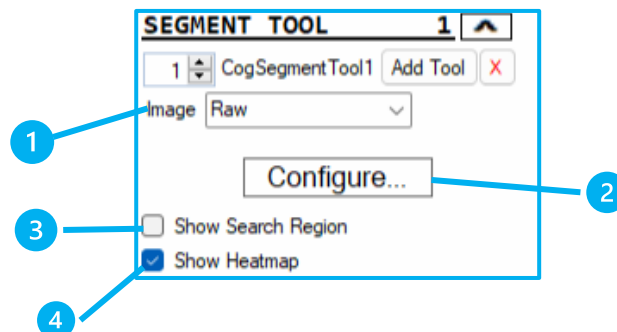
After acquiring an image, you can click on the matching class in the **Train Class** section to train the image on the target class. This will update the match predictions and model health. Whenever an image is mislabeled or the model prediction is ambiguous, train the image on the target class to update and improve the model health. Be sure to maintain a balanced number of trained images for each class.



In the above example, there are three classes: **MissingCornerBolt**, **MissingClampBolt**, **Complete**. The model health is displayed below showing that it is fairly reliable. The match predictions showcase the confidence score for each class in the form of a bar. The **Complete** class is marked as an accept class, indicating that if a part is marked as “Complete”, it passes inspection. The remaining classes are marked as reject classes which fail inspection when detected.

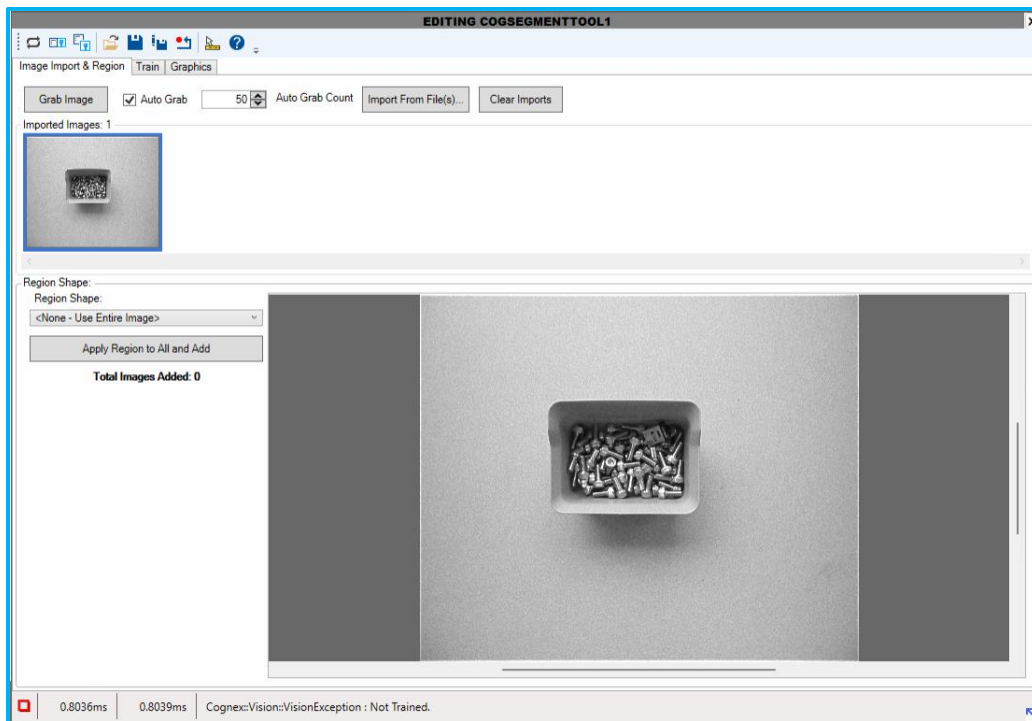
7.2 Segment Tool

The **segment** tool is used to generate a heatmap around a region that contains a target part marked by the user. The marked part is surrounded by a “Segment” when labeling the image indicating to the model what the target part looks like.



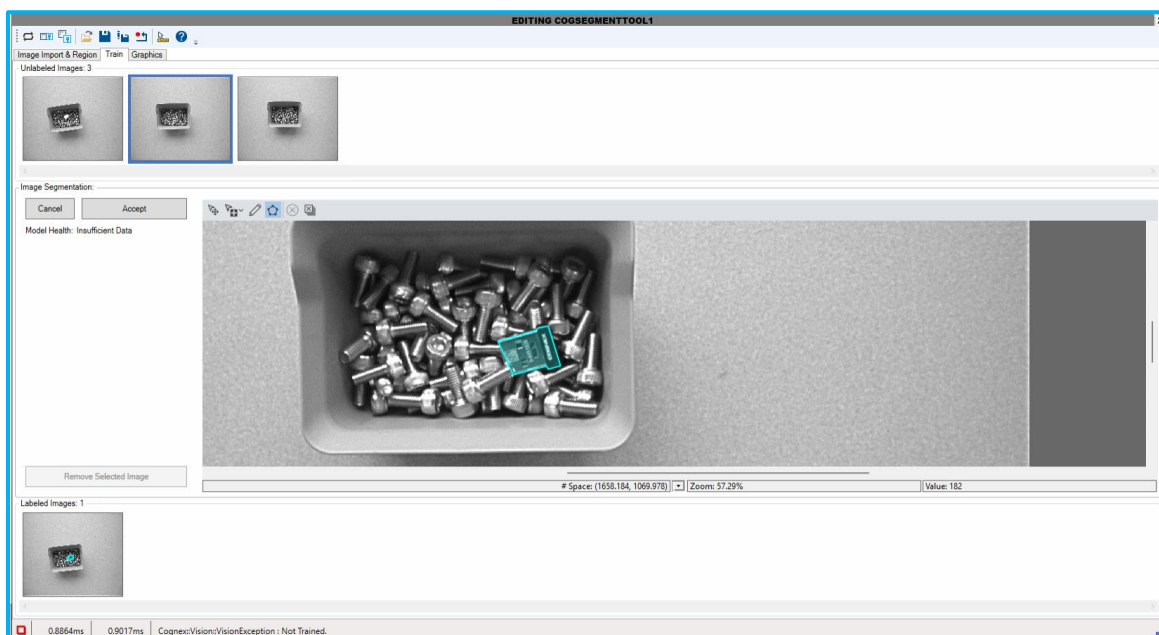
Number	Name	Description
1	Input Image	Image source to run segment tool
2	Configure	Opens edit control for CogSegmentTool
3	Show Search Region	Displays the search region for the segment tool when enabled
4	Show Heatmap	Displays the heatmap of the resulting segment tool when enabled

To acquire an image with the segment tool, click snapshot or load an image into the display in EtherInspect. From here, open the toolblock by clicking the **Configure** button and click **Grab Image**. To automatically acquire an image and feed it into the segment tool, enable the setting **Auto Grab**. Now, for each new image acquired, the image will be added to the segment tool.



Click **Apply Region to All and Add** to add the image to the training dataset. Be sure to grab a couple images prior to running training.

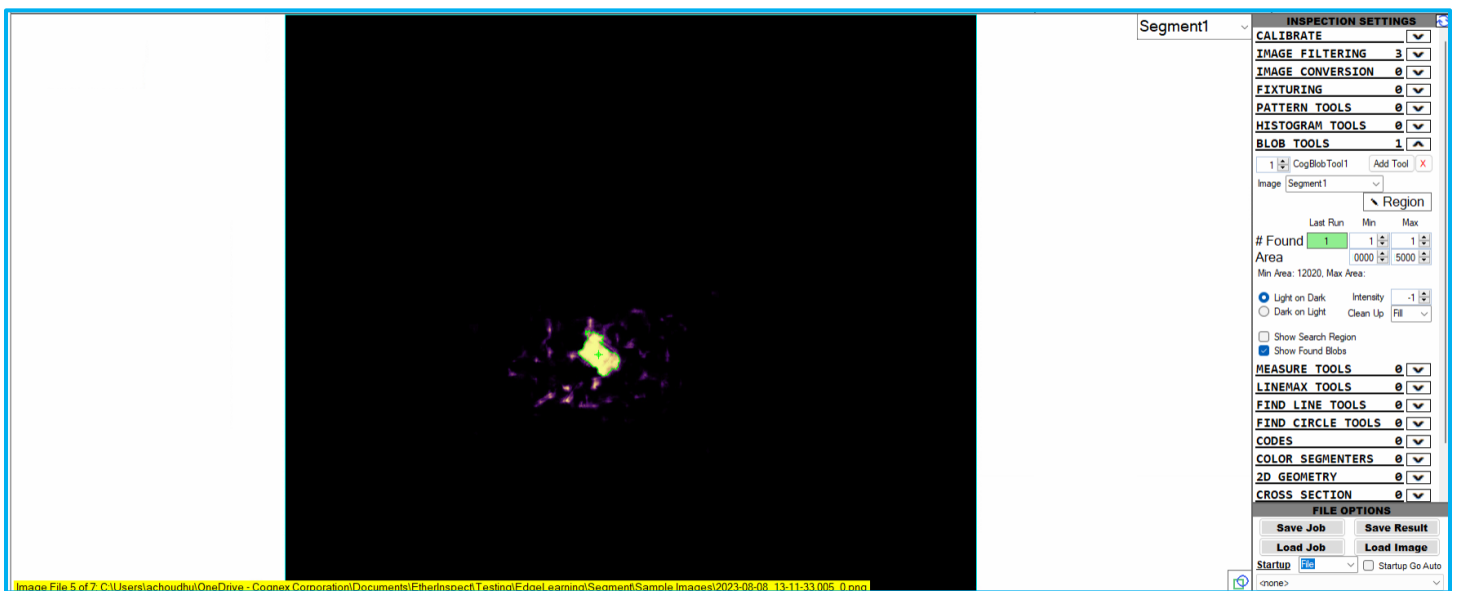
In the training tab, select an image and click **Label** to start labeling the target region. Two tools are provided to label the image: **Pencil** tool and **Polygon** tool. The pencil tool is used to freely draw around the region to highlight it. Polygon tool is used to set points to define the region within a shape. Once the labeling has been completed, click **Accept** to accept the oncoming changes and train the model with the initial image. Do this for each image you acquired until your model health is at a good score and the heatmap is depicted accurately.



Once training is complete, a heatmap will display over the image highlighting where the target part is.



An additional filter will be created with just the heatmap and the image hidden. This can be followed up with a blob tool to indicate where the heatmap region is highlighting to find the part.



8 Trevista

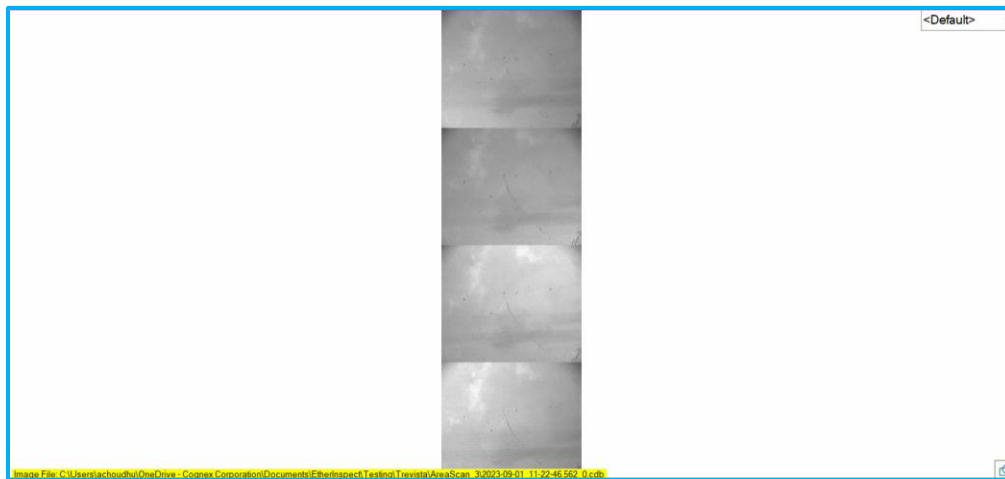
The **Trevista** template is a standard EtherInspect template with the newly featured **Trevista** tool. This template is used alongside the **Trevista CI Dome** for surface defect detection. Included in the template is the Trevista Launch wizard from VisionPro under **Image Setup** that allows the user to configure their camera with the included light controller. For more info on setting up the hardware and configuring VisionPro to acquire images, please refer to the following guides:

- [Area Scan](#)
- [Line Scan](#)

When setting up Trevista, the following steps are needed to acquire an image and begin surface detection:

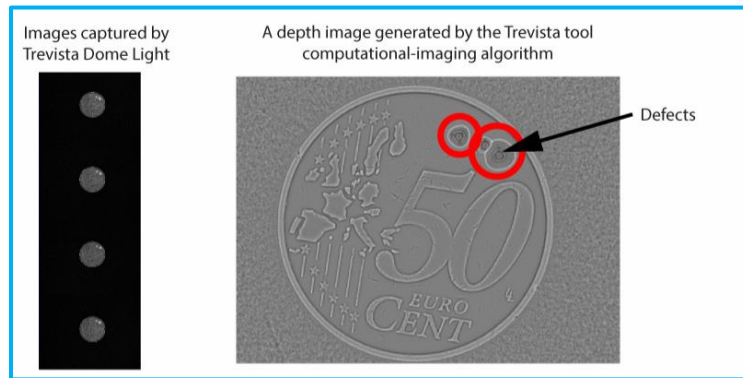
1. Create job with **Trevista** template
2. Click **Image Setup** and click **Launch Trevista Startup Wizard**
3. Upon opening the wizard, wait for the Device Image source to be detected
 - a. This will take a moment, but the camera will be automatically detected with the camera type set as either Area Scan or Line Scan
4. Set Trigger mode to manual
5. Click the play button in the top left of the image display in the wizard to test acquisition
6. Adjust exposure and contrast under camera settings
7. Set the width and height of the camera settings accordingly
8. Exit the wizard
9. Click **Snapshot** to acquire an image

An interlaced CogImage8Grey image will be displayed with 4 different shading methods from each quadrant. Each shading image is generated by lighting up specific quadrants within the dome to make certain features pop out.



8.1 Trevista Tool

The trevista tool uses computational imaging on an CogImage8Interlaced image from the Trevista CI Dome to highlight features in an image that raw cameras cannot display. The image below showcases defects found when using the trevista tool.



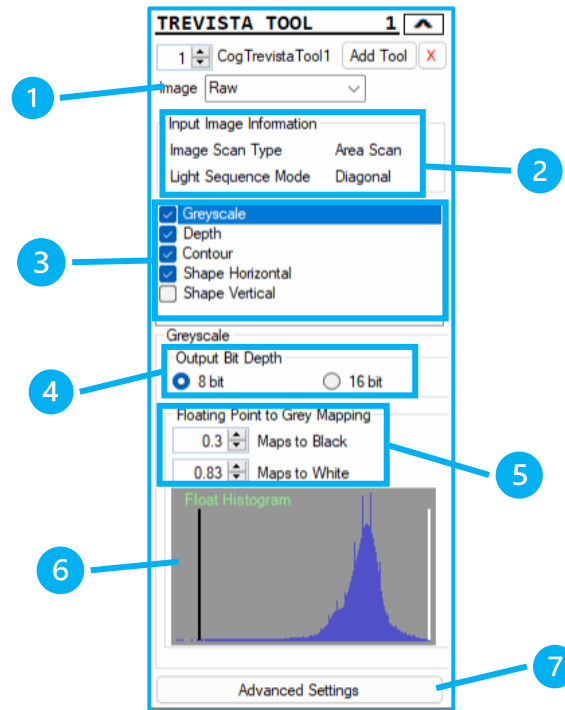
To better understand how the trevista tool works, it is important to understand how the Trevista CI dome is set up. The kit consists of a Trevista Dome Light with a mounted AreaScan/LineScan camera. Within this dome is a series of programmable lights ran by a light controller that comes with the dome.

Trevista Dome Light



Trevista Controller



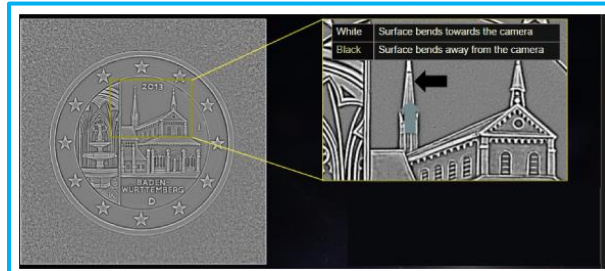


Number	Name	Description
1	Input Image	Image source for Trevista tool
2	Input Image Information	Displays image scan type (Area/Line Scan) and Light Sequence of image
3	Tool Result Images	Settings for result images. Can enable certain result outputs while disabling others. Clicking on a result image output will display the settings for that image underneath
4	Output Bit Depth	Determines whether to output image as CogImage8Grey or CogImage16Grey
5	Floating Point to Grey Mapping	Maps floating point numbers to grayscale pixel values in output image
6	Float Histogram	Displays distribution of floating point pixel values and mapping limits for white and black greyscale values. Used to adjust mappings
7	Advanced Settings	Opens advanced tab in Trevista tool ToolBlock

8.1.1 Tool Result Images

Various light sources from the dome are used to detect defects based on the locations in which the light hits the surface. This allows the trevista tool to generate the following image types:

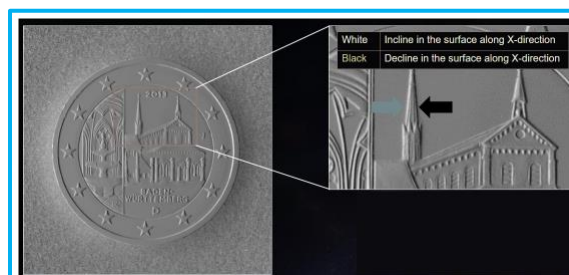
- Contour
 - Evaluates the curvature of a surface to spot small, local defects



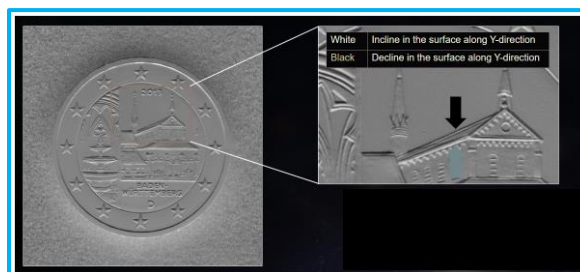
- Depth
 - Allows application to find surface defects



- Greyscale
 - Factors out shape of the surface to enhance the texture of the surface. Used to find discoloration and contamination
- Shape Horizontal
 - Highlights defects along the x-axis of your image



- Shape Vertical
 - Highlights defects along the y-axis of your image



The **Depth** tool image has special settings to modify the method that the image is computed.

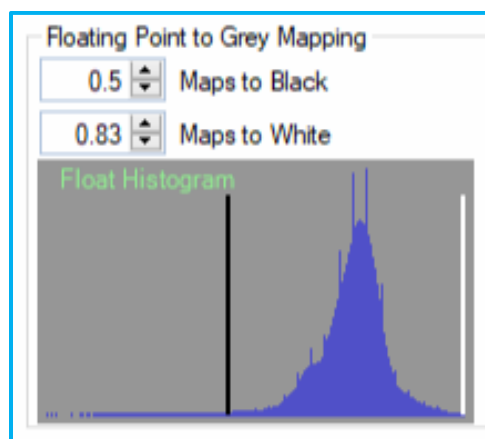
- Depth mode
 - Modifies the method that the surfaces are computed
 - Local – Computes relative heights of surface areas that lie near each other in the image
 - Global – Computes relative heights of surface areas that lie far away from each other in the image
 - Cognex recommends Global mode for surfaces with large flat areas that are not perpendicular to the optical axis of your lens
 - Local requires certain procedures and processing time will increase
- Depth level
 - Specifies depth-calculation level of the image
 - Global – Sets the depth level to a number that is approximately the size (in pixels) of the entire surface over which you wish to compute the depth
 - Local – Sets the depth level to a number that is bigger than the size (in pixels) of the local surface patch over which you wish to compute the depth

8.1.2 Floating Point to Greyscale Mapping

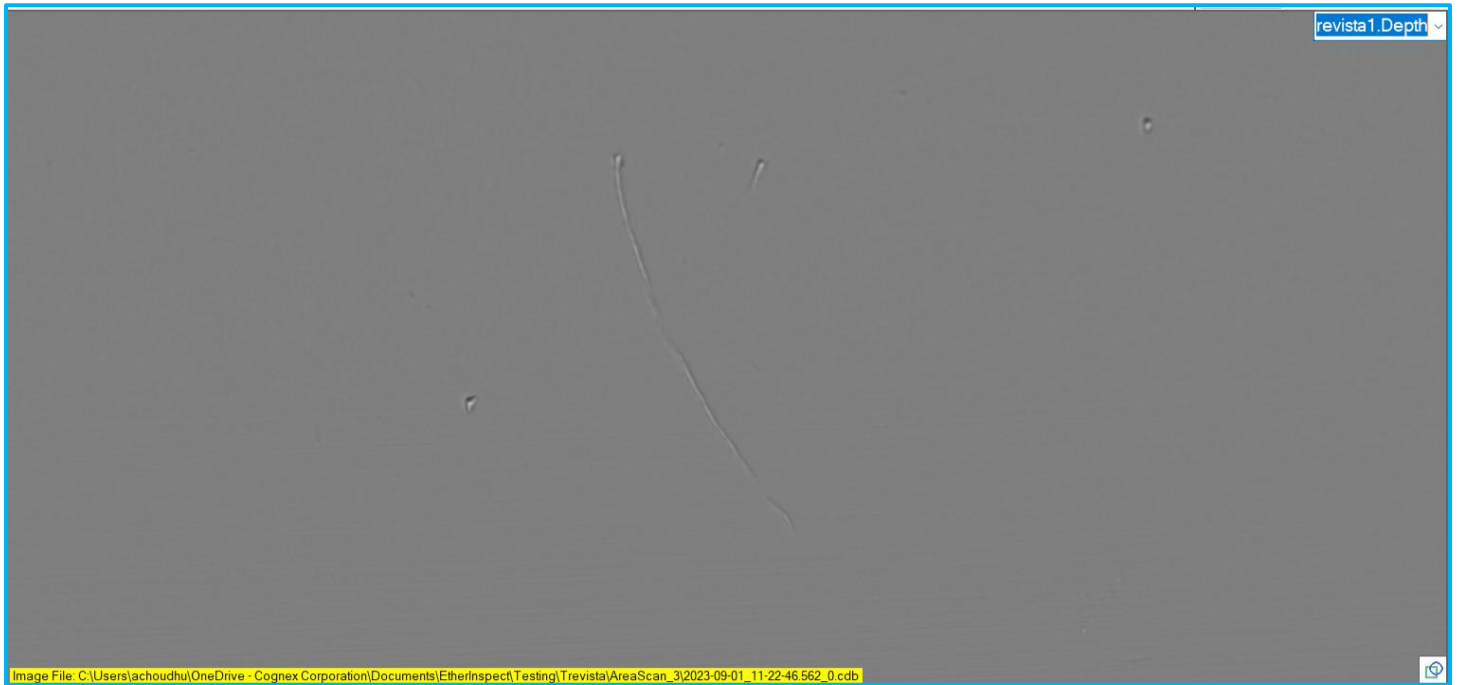
The **trevista** tool uses a computational imaging algorithm to produce a set of pixel values initially stored as floating point numbers. The tool then maps these floating point numbers to greyscale pixel value to display the output image as greyscale. There are two mappings to keep in mind when setting up your output image:

- Map to Black
 - Maps any region of the image with a matching floating point number to black
- Map to White
 - Maps any region of the image with a matching floating point number to white, the maximum pixel value:
 - 255 for CogImage8Grey
 - 65535 for CogImage16Grey
- Once the mappings are set, the tool sets any floating-point values between these two limits linearly to intermediate shades of grey. Values closer to the **Map to Black** value will have darker shades and values closer to **Map to White** will receive lighter shades

The **Floating Histogram** shows the distribution of the floating-point values in the image. This can be used to better adjust the mappings based on the distribution of the pixels.



Once the mappings are configured, you can view the image output by changing the filter to **<TrevistaToolName>.<ResultImage>** (Example: **Trevista1.Depth**)

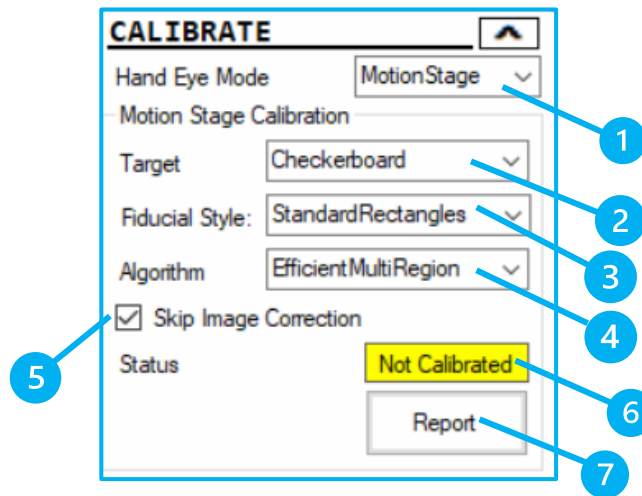


9 Alignment

The alignment template is used for applications involving aligning parts and pieces for placement using an industrial robot. An example of this would be battery cell stacking during the production of batteries for electric vehicles. There are two tools specific to the alignment template. These tools are:

- Alignment Finder
- Alignment Compute

These tools utilize calibrated robot coordinates to find the part to fixture the robot's movement off and to compute the translation from robot coordinates to part coordinates. For calibration, the template uses an advanced Hand-Eye 2D calibration tool. This tool has a special mode referred to as "Motion Stage". Motion stage uses a series of points along a robots motion path for calibration.



Number	Name	Description
1	Hand Eye Mode	Operation Mode for Hand-Eye Calibration
2	Target	Specify calibration target for calibration
3	Fiducial Style	Specify reference points on calibration target
4	Calibration Algorithm	Algorithm to use when calibrating robot
5	Skip Image Correction	If enabled, tool will not accommodate for distorted images. Saves calibration time but is less accurate.
6	Calibration Status	State of calibration process
7	Report	Displays text report of calibration performance, including a table of all of the calibrated points and the overall quality of the calibration process.

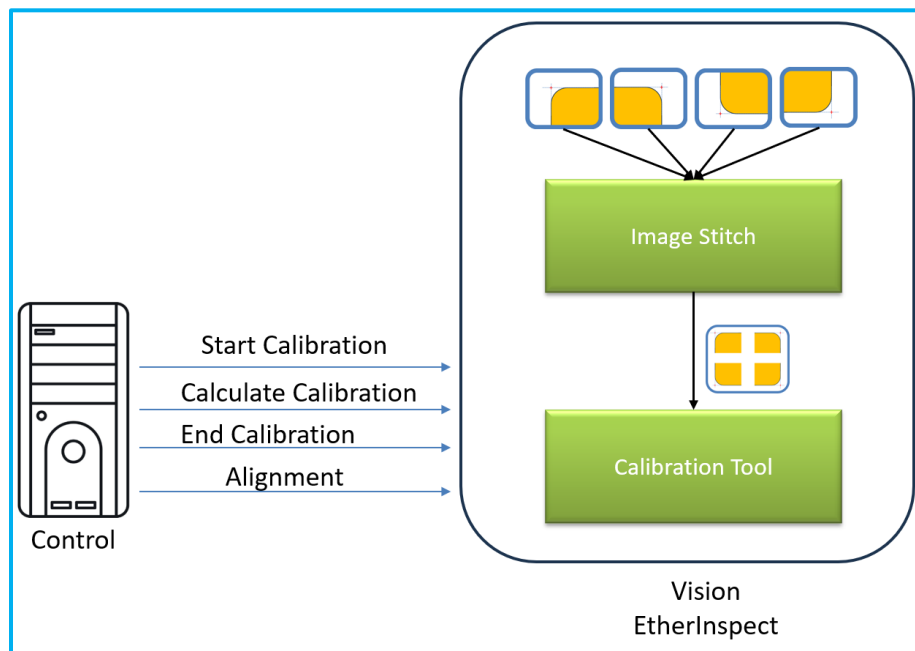
Motion stage is selected as the default operation mode for the Hand-Eye 2D tool in the alignment template.

9.1 Checkerboard Calibration

Checkerboard calibration works similarly to the standard hand-eye tool calibration used in other templates. This method utilizes a checkerboard plate placed in front of the camera. With the camera attached to the robot, the robot can be set to move to different poses to acquire images throughout the calibration process. The hand-eye tool then takes these points and computes the translation from image coordinates to robot coordinates. To run checkerboard calibration, the following workflow is used:

1. Set target to **Checkerboard**
2. Set **Fiducial Style** based on the target reference points on the calibration plate you wish to use.
 - a. **StandardRectangle** is recommended but data matrices can be used as well if your checkerboard plate contains that feature.
3. Set **Algorithm** to specify which calibration algorithm to use when computing a calibration using the multiple points:
 - a. **Standard** – Standard calibration algorithm
 - b. **Exhaustive** – Takes additional time to find and calibrate using fiducial points but is more accurate
 - c. **ExhaustiveMultiRegion** – Takes additional time to find and calibrate using fiducial points in multiple regions of image but is more accurate
 - d. **Efficient** – Finds and calibrates using fiducial points in least amount of time but is less accurate
 - e. **EfficientMultiRegion** – Finds and calibrates using fiducial points in multiple regions in least amount of time but is less accurate
4. If your application is time-critical, you can reduce the amount of time for calibration by skipping image correction. Image correction is the process of finding distortions within an acquired image and adjusting the calibration accordingly. Skipping image correction can save time during calibration but results in less accuracy if your image has distorted regions.

Once all the above steps are met, you can calibrate the system using EtherInspect's TCP interface (refer to section 22.2.23 for information on hand-eye 2D commands). The following diagram showcases the workflow when completing the calibration process using the TCP interface:



The above image uses a hybrid camera which stitches 4 separate images into one to be fed into the calibration tool. To start off, a “**STARTC**” command is sent via the TCP interface to EtherInspect to start the Calibration process. From here, the **HECAL** command is used to acquire an image from the target camera and register the point with its respective robot coordinates. When using this command, you will need to provide the following parameters:

- Target Camera
- Point index (Starts from 1 and goes up to 99)
- X robot coordinate
- Y robot coordinate
- Theta robot coordinate

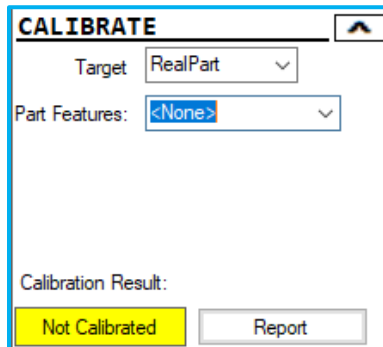
Repeat this step for every point you wish to register. The more points you provide, the more accurate the calibration will be. Once this is done, you can end the calibration process and begin computing the calibration using the **ENDC** command.

9.2 Real Part Calibration

If you are unable to use a checkerboard pattern calibration plate as a target, you can use this option to run calibration off a target part. Prior to running Real Part calibration, you will need to add and set up an **Alignment Finder** tool. This tool is used to find several edges and corners on a target part as reference points for the calibration. The workflow is as follows:

1. Set target to **RealPart**
2. Add **Alignment Finder** tool
3. Set features using target part and Alignment Finder tool
4. Set **Part Features** to output of Alignment finder tool

From here, use the TCP interface to run the calibration process as shown in section 9.1 above. You will need to register multiple points on the target part to run calibration. Once the calibration process ends, all points will be factored into the computation to adjust the coordinate system.



9.3 Calibration Report

Once calibration is complete, a report will be generated which provides a summary of the calibration process. There are several pieces of information provided in this report including:

- General calibration information
- Camera Pose table
- Calibrated points table
- Camera Calibration Residuals table

Camera Pose

This table showcases the cameras pose (X, Y, theta) in relation to the Robot's coordinate system. The table contains two categories of data:

- **Camera Pose in Home2D**
 - Camera pose in robot coordinate system
- Overall residuals
 - Showcases variations in camera pose compared to calibrated robot coordinates

There are several pieces of data provided here:

- **Index** – Index of Camera
- **PeISize** – Pixel Size (mm)
- **X** – X coordinate of camera in robot coordinate system (pixels)
- **Y** – Y coordinate of camera in robot coordinate system (pixels)
- **Theta** – Theta angle of camera in robot coordinate system (deg)
- **RMS** – Root mean square error of camera pose (pixels)
- **Max** – Maximum Root Mean Square of camera pose (pixels)
- **Quality** – Quality of camera calibration

Camera		Camera Pose in Home2D (Home2DFromCamera2D)			Home2D		Overall Residuals Raw2D		Quality
Index	PeISize	X	Y	T degs	RMS	Max	RMS	Max	
0	0.134	149.926	109.962	89.997	0.039	0.104	0.29	0.78	Excellent

Calibrated Points

This table showcases the list of points used for calibration. This table is split into 3 categories:

- **Uncorrected Home2D From Stage2D**
 - User provided robot coordinates from Motion Stage process before calibration
- **Corrected Home2D From Stage2D**
 - Adjusted Robot coordinates from motion stage process after calibration
- **Estimated Home2DFromStage2D**
 - Estimated Robot coordinates from motion stage process after calibration with real-life distortions factored in

Each entry includes X (mm), Y (mm), theta (degrees).

Pose Index	Uncorrected			Corrected			Estimated		
	Home2DFromStage2D			Home2DFromStage2D			Home2DFromStage2D		
	X	Y	T degs	X	Y	T degs	X	Y	T degs
0	-10.000	-10.000	0.000	-10.001	-10.001	0.000	-10.000	-10.000	0.000
1	0.000	-10.000	0.000	-0.001	-10.001	0.000	-0.004	-10.000	0.000
2	10.000	-10.000	0.000	9.999	-10.001	0.000	10.001	-10.001	0.000
3	10.000	0.000	0.000	10.000	0.000	0.000	10.001	-0.001	0.000
4	0.000	0.000	0.000	0.000	0.000	0.000	-0.002	-0.001	0.000
5	-10.000	0.000	0.000	-10.000	0.000	0.000	-9.999	0.000	0.000
6	-10.000	10.000	0.000	-9.999	10.001	0.000	-9.998	10.001	0.000
7	0.000	10.000	0.000	0.001	10.001	0.000	-0.003	10.001	0.000
8	10.000	10.000	0.000	10.001	10.001	0.000	10.002	10.000	0.000
9	0.000	0.000	-3.000	0.000	0.000	-3.000	0.000	0.002	-3.000
10	0.000	0.000	-1.000	0.000	0.000	-1.000	0.001	-0.001	-1.000
11	0.000	0.000	1.000	0.000	0.000	1.000	0.002	0.001	1.000
12	0.000	0.000	3.000	0.000	0.000	3.000	0.001	0.001	3.000

Calibration Residuals

This table includes camera pose variation information and accuracy for each point from the target camera used during calibration. This information includes:

- Pose Index
- Plate2D – Calibration Plate coordinates
 - RMS
 - Max
- Raw2D – Image coordinates
 - RMS
 - Max

Pose Index	Camera 0 Calibration Residuals					
	Plate2D		Raw2D			
	RMS	Max	RMS	Max		
0	0.048	0.096	0.36	0.71		
1	0.045	0.086	0.34	0.64		
2	0.046	0.091	0.34	0.68		
3	0.045	0.084	0.34	0.63		
4	0.044	0.087	0.33	0.65		
5	0.047	0.103	0.35	0.77		
6	0.046	0.096	0.34	0.72		
7	0.043	0.085	0.32	0.64		
8	0.044	0.082	0.33	0.61		
9	0.011	0.031	0.08	0.23		
10	0.014	0.032	0.10	0.24		
11	0.014	0.031	0.10	0.23		
12	0.012	0.032	0.09	0.24		

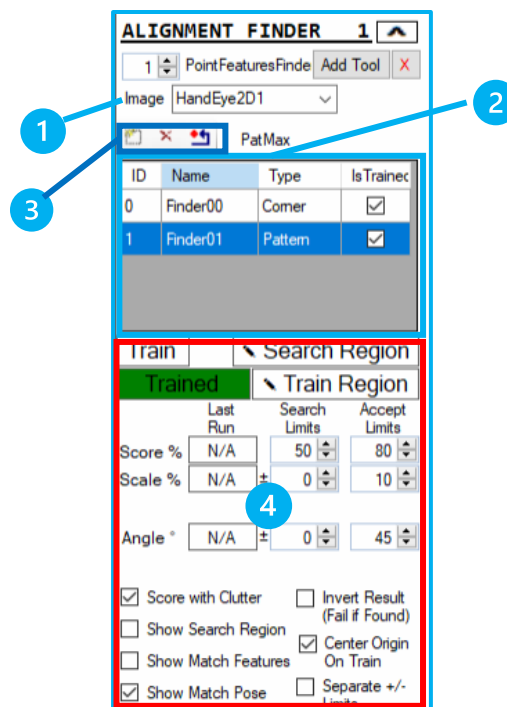
9.4 Alignment Finder

The alignment finder tool is used to detect the corner points and find features within a part to be used later in the Alignment Compute Tool. These features serve as fixture point to find changes in the target position during alignment.

There are two feature finder types to keep note of:

- Pattern
 - Uses VisionPro PatMax tool to find unique pattern on part
- Corner
 - Initially uses VisionPro PatMax tool to locate part, then uses two VisionPro FindLine tools to locate the intersection of those two lines. This intersection is referred to as a corner.

Each feature located using the Alignment finder tool can be inputted into the Alignment Compute tool to track its positioning throughout each image. When the part is misaligned or shifted out of its target position, the tool will fail and provide delta values indicating the change in (X, Y, Theta).



Number	Name	Description
1	Input Image	Image source for Alignment Finder tool. Use HandEye2D1 for calibrated space and Raw for uncalibrated space
2	Feature Finder List	Contains a list of feature finders (Pattern/Corner) with the status and name of each
3	Toolbar	Add/Delete/Reset Feature Finders
4	Sub-tool Region	Sub-tools controls used as part of each feature finder type will be shown here depending on which feature finder type is selected



Add Feature Finder – Used to add Feature finder. Can specify whether to use **Pattern** or **Corner**



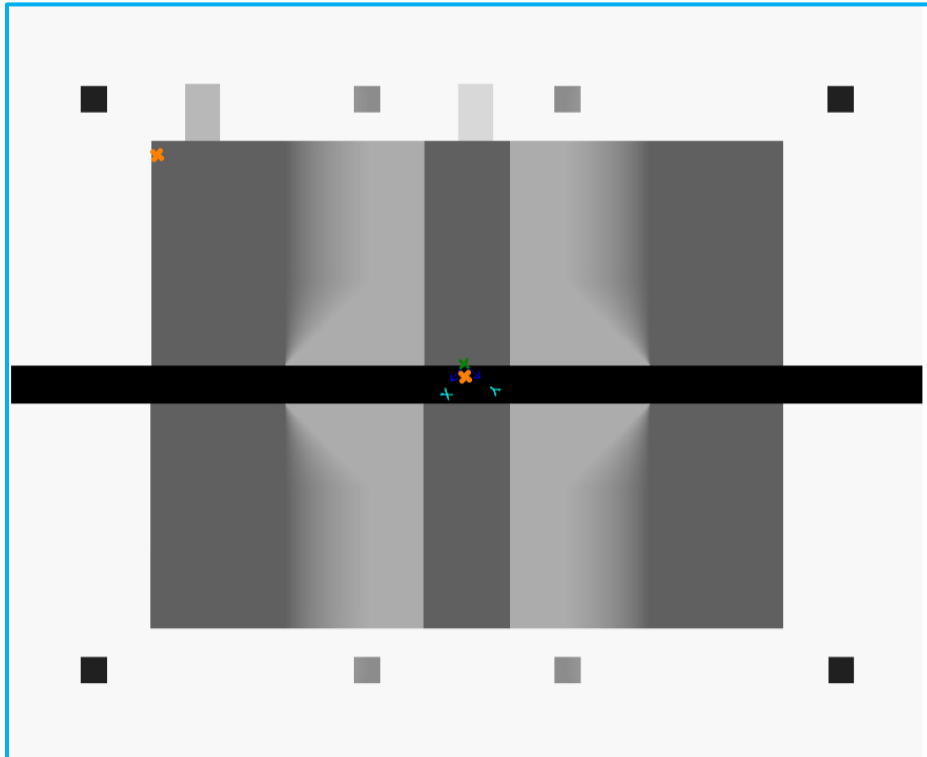
Delete Feature Finder – Used to remove Feature finder from list.



Reset Feature Finders – Used to remove all Feature finders from list

9.4.1 Pattern Feature Finder

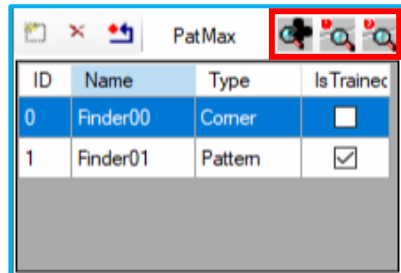
The pattern feature finder type uses VisionPro's PatMax tool to generate a feature and finds the center of that pattern. The original center point of the pattern is marked as an orange crosshair. The original top-left corner of the pattern is also marked with an orange crosshair. This will serve as references to determine whether a part is displaced. When a pattern type feature finder is added, a PatMax tool control will be displayed underneath the list of feature finders. Once the pattern is trained, you can access the **HandEye2D1** filter to access the graphics for the feature finder.



9.4.2 Corner Feature Finder

The corner feature finder type is set by using a VisionPro PatMax tool to locate the part and then uses two VisionPro FindLine tool to find the intersection of those two lines. This is used to find the corners of the part. Each corner located with this feature finder type is marked with a green crosshair.

Once the corner type is selected and added to the list of feature finders, three buttons will appear in the top right of the table.



ID	Name	Type	Is Trained
0	Finder00	Corner	☐
1	Finder01	Pattern	☒



PatMax – Accesses PatMax sub-tool within feature finder



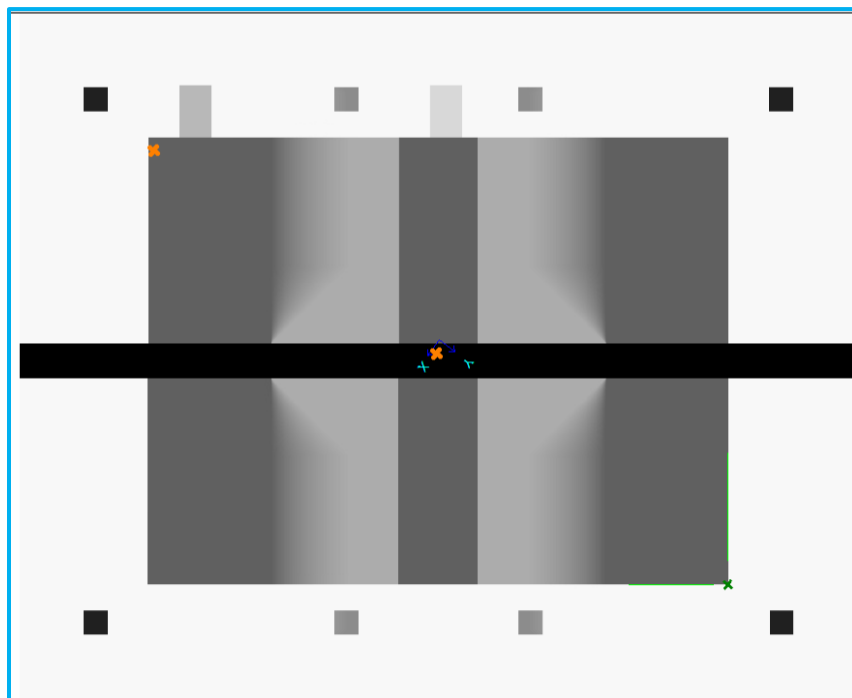
FindLine1 – Accesses first FindLine tool within feature finder



FindLine2 – Accesses second FindLine tool within feature finder

When setting up the corner feature finder, the following steps should be followed:

1. Train PatMax tool on target part to locate
2. Set first FindLine tool to one side of part
3. Set second FindLine tool to adjacent side of part
4. Run inspection on main image



9.5 Alignment Compute

The alignment compute tool uses features from the Alignment Finder tool and the calibration from the advanced HandEye2D tool to train the tool on the target positions of the part. If the tool output is 0 under the delta values, the tool will result in a PASS. Whenever a part is displaced or misaligned, the tool will output non-zero values under the X, Y, Theta section of the tool and the result will be a FAIL. This non-zero value is referred to as the delta and it is split into two sections:

- Relative – Difference in position between target image coordinate and new image coordinate
- Target – Difference in position between target part coordinate and new part coordinate

The screenshot shows the 'ALIGNMENT COMPUTE 1' window. It includes a dropdown for 'Image' (HandEye2D1), a dropdown for 'Calibration Data' (HandEye2D1.Hand), and a dropdown for 'Pattern Source' (PointFeaturesFinde). There are 'Train' and 'Trained' buttons. Below these is a table showing 'Relative' and 'Target' values for X, Y, and Theta. At the bottom, there are checkboxes for 'Enable Compensate' and 'Invert Compensate Direction', and a section for 'Compensate Value' with input fields for X, Y, and Theta.

	Relative	Target	
X	-0.1978	-0.1978	mm
Y	1.24414	1.24414	mm
Theta	-0.0159	-0.0159	deg

Compensate Value		
X	0	mm
Y	0	mm
Theta	0	deg

Number	Name	Description
1	Input Image	Image source for Alignment Compute tool
2	Calibration Data	Calibration data to use for delta calculations
3	Pattern Source	Features inputs. Must have Alignment Finder tool defined before setting features
4	Train	Train tool on feature and calibration data
5	Delta Values	Difference between current value and target value. Split into relative and target for X, Y, and theta. If 0, then no misalignment has been found.
6	Enable Compensate	Enable compensate values for delta values
7	Invert Compensate Direction	Invert signs for compensate values
8	Compensate Values	Provides offset for final result to accommodate for factors such as moving parts, laminations, etc.