

Zebra Aurora[™] Vision

Aurora Vision Studio 5.7

User Interface

Created: 7/6/2026

Product version: 5.7.1.80289

Table of content:

- Complexity Levels
- Finding Filters
- Connecting and Configuring Filters
- Creating Macrofilters
- Creating Models for Template Matching
- Preparing Rectification Transform Map
- Creating Text Segmentation Models
- Creating Golden Template Models
- Creating Models for Golden Template
- Creating Text Recognition Models
- Auto-tuning Datacode Parameters
- Analysing Filter Performance
- Seeing More in the Diagnostic Mode
- Deploying Programs with the Runtime Application
- Performing General Calculations
- Managing Projects with Project Explorer
- Keyboard Shortcuts
- Working with 3D data
- Creating Deep Learning Model
- Managing Workspaces
- Using Filmstrip Control

Complexity Levels

Introduction

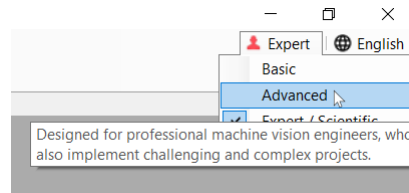
Aurora Vision Studio has three levels of feature complexity. At lower levels we hide advanced features and ones that may be confusing.

Available Levels

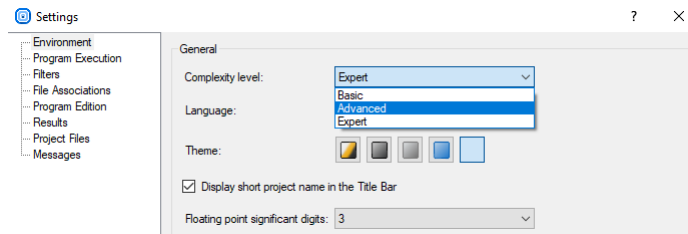
- **Basic** – Designed for production engineers who want to quickly build simple machine vision projects without devoting much time to learning the full capabilities of the software.
- **Advanced** – Designed for professional machine vision engineers who also implement challenging and complex projects.
- **Expert / Scientific** – Gives access to experimental and scientific filters (tools), which are not recommended for typical machine vision applications, but might be useful for research purposes.

Changing Complexity Level

Complexity Level can be changed at any time. It can be done by clicking on the level name in the upper-right corner of Aurora Vision Studio:



Complexity Level can also be changed in the application settings:

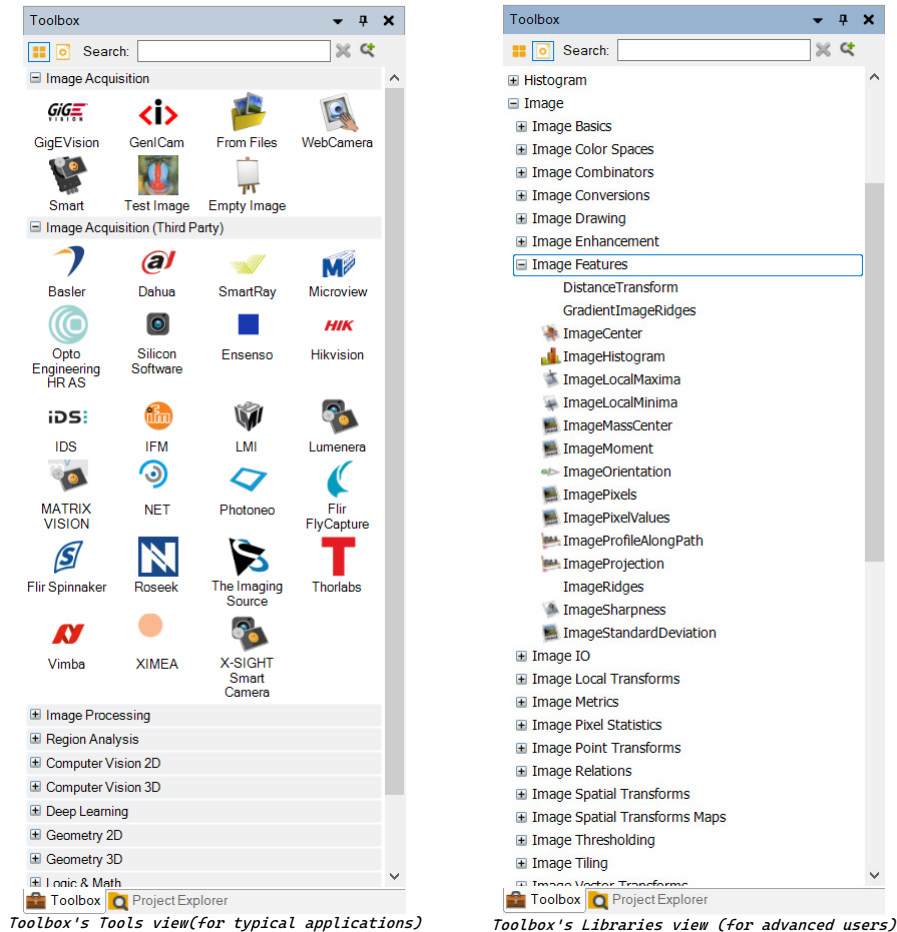


Finding Filters

Introduction

There are many hundreds of ready-for-use [filters](#) (tools) in Aurora Vision Studio implementing common image processing algorithms, planar geometry, specialized machine vision tools as well as things like basic arithmetics or operating system functions. On the one hand, it means that you get instant access to results of tens of thousands of programmers' work hours, but it also means that there are quite a lot of various libraries and filter categories that you need to know. This article advises you how to cope with that multitude and how to find filters that you need.

The most important thing to know is that there are two different catalogs of filters, designed for different types of users:



The Toolbox's Tools view is designed for use in typical machine vision applications. It is task-oriented, most filters are grouped into tools and they are supported with intuitive illustrations. This makes it easy to find the filters you need the most. It does not, however, contain all the advanced filters, which might be required in more challenging applications. To access the complete list of filters, you should use the Toolbox's Libraries view. This catalog is organized in libraries, categories and subcategories. Due to its comprehensiveness, it usually takes more time to find what you need, but there is also an advanced text-based search engine, which is very useful if you can guess a part of the filter name.

Toolbox

Sections

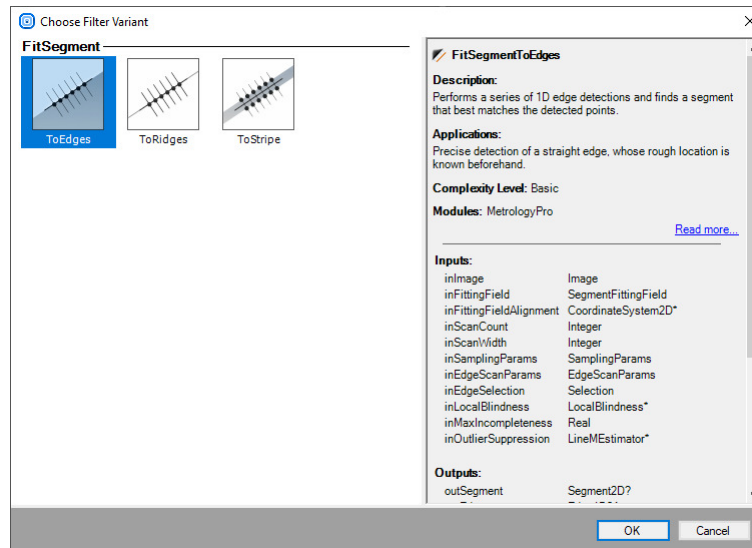
When you use the Toolbox's Tools view, the general idea is that you will most probably start with a filter (tool) from the first section, Image Acquisition, and then follow with filters from consecutive sections:

1. **Image Acquisition**
 - Acquiring images from cameras, frame grabbers or files.
2. **Image Acquisition (Third Party)**
 - Acquiring images from third-party cameras.
3. **Image Processing**
 - Image conversions, enhancements, transformations etc.
4. **Region Analysis**
 - Operations on pixel sets representing foreground objects (blobs).
5. **Computer Vision 2D**
 - Specialized tools for high-level image analysis and measurements.
6. **Computer Vision 3D**
 - Specialized tools for analysis of 3D point clouds.
7. **Deep Learning**
 - Self-learning tools based on deep neural networks.
8. **Geometry 2D**
 - Filters for constructing, transforming and analysing primitives of 2D geometry.
9. **Geometry 3D**
 - Filters for constructing, transforming and analysing primitives of 3D geometry.
10. **Logic & Math**
 - Numerical calculations, arrays and conditional data processing.
11. **Program Structure**
 - Category contains the basic program structure elements.
12. **File System**
 - Filters for working with files on disk.
13. **Program I/O**
 - Filters for communication with external devices.

Choosing Filter from Tools

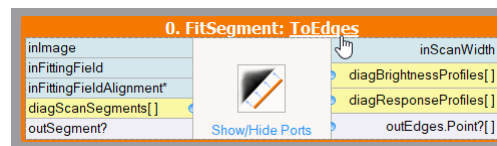
A tool is a collection of related filters. The process of selecting a filter from the Toolbox thus consists of two steps:

1. First you select a tool in the Toolbox, e.g. "Fit Shape".
2. Then you select a filter from the tool in the "Choose Filter of Group" window.



Choosing a filter from a tool (Toolbox).

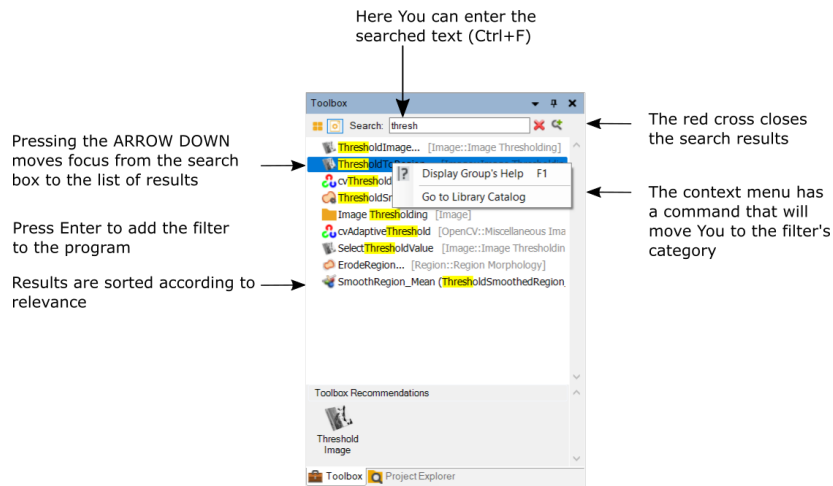
This dialog can have several sections (e.g. "Fit Segment", "Fit Circle" etc.)



Changing filter variant.

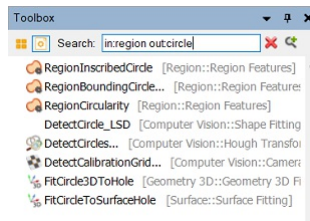
The Search Box

If you know a part of the name of the filter (tool) that you need, or if you can guess it, the Filter Search Box will make your work much more efficient. Just enter the searched text there and you will get a list of filters with most relevant matches at the top:



The Search Box and search results in the Toolbox's Libraries view.

If you cannot guess the filter name, but you know what you expect on the inputs and outputs, you can use special queries with "in:" and "out:" operators as the image below depicts:



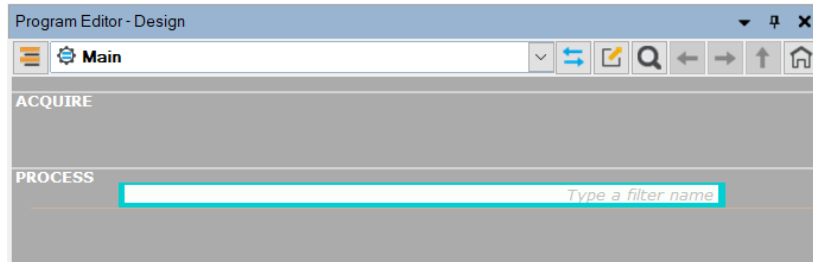
Advanced search with expected inputs and outputs.

As a matter of fact, some advanced users of Aurora Vision Studio stop browsing the categories and just type the filter names in the Search Box to have them quickly added to the program.

Program Editor

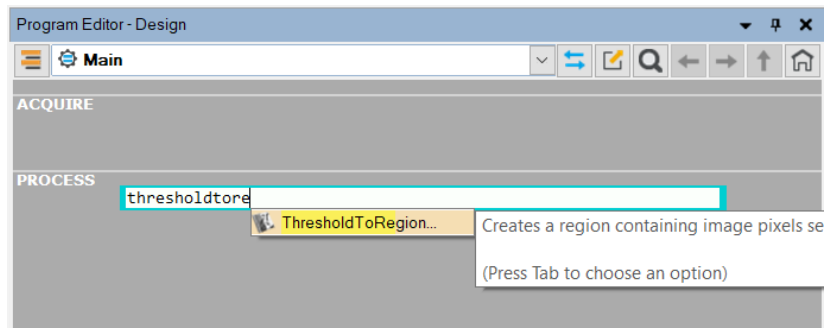
Ctrl+Space / Ctrl+T

When you know the name of a filter, which you would like to add into your program, you can use a keyboard shortcut **Ctrl+Space** or **Ctrl+T** to find it straightaway in the Program Editor, without having to open the Toolbox's Libraries view. After applying this shortcut, you are prompted to type a filter name:



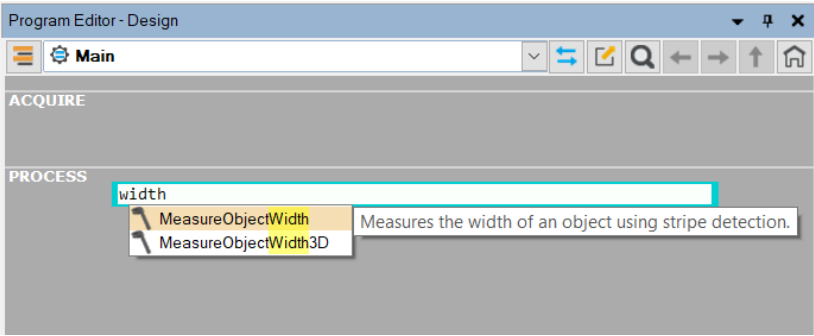
Advanced search using CTRL+SPACE shortcut.

Let us assume that you need to perform simple thresholding on your input image. You already know that there is a **ThresholdToRegion** filter in our library, which you have utilized many times so far. Instead of time-consuming searching in either Toolbox's Tools view or Libraries view, you can quickly access the desired function by typing its name:



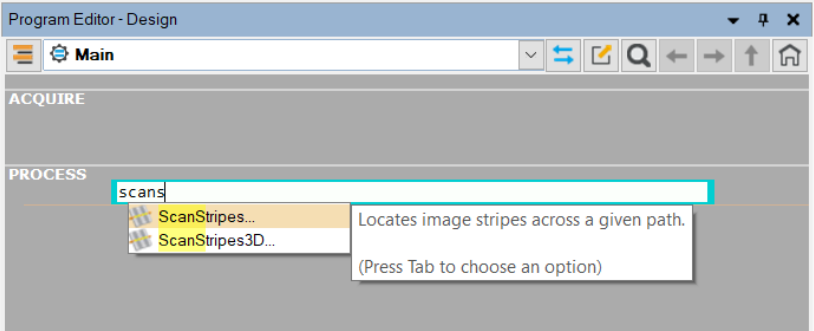
Browsing for a particular filter.

If you do not remember well the name of a filter, you can track it as well by typing only a part of it:



Searching a filter without its full name.

The search engine in the Program Editor also allows you to display the description of a filter in case you want to make sure or remind yourself what the filter is for:

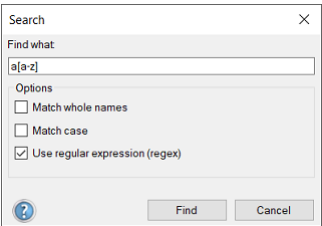


The description of a filter.

Search Window

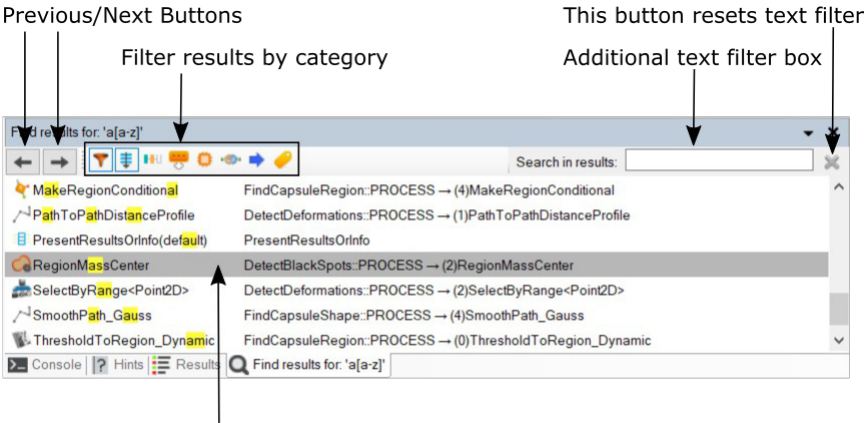
Ctrl+F

Creating a large program in Aurora Vision Studio may require finding elements in it's structure. Using Search Window is the best way to find Filters (tools), Macrofilters, Variants and Global parameters which already exist in the project. To open the search window find a Magnifier button in Program Editor or press Ctrl+F. In the search window insert the name of an element you want to find or a part you remember. Press 'Find' button and results of search will appear in the new window. You may also pick some search options like using case sensitivity, matching whole names or regular expression which can narrow your search.



Browsing for elements in project.

In Search Results window you may select element on the list to indicate an object in project. It is possible to filter results with buttons on panel on the top of the window. Additionally if there are too many results they can be filtered with keyword given in the box on top-right.



This is selected item with filter name and location as details description on right
Managing search results.

Rules

Entering a search phrase also allows to pre-filter data with some keywords. When your project is large it might be useful to use Rules in the search phrase. Here are some examples of using Rules in Search Window.

Name	Syntax	Example	Constraint
Parent	parent:<MacrofilterName>	rectangle parent:Initialize	Must have an instance in Macrofilter <MacrofilterName>.
Input	in:<Name>	load in:File	Must contain an input which either name or type contains phrase <Name>.
Output	out:<Name>	load out:Integer	Must contain an output which either name, type or Data Source Label contains phrase <Name>.

Further information

For more details on the topic, please refer to the video tutorial on our YouTube channel:

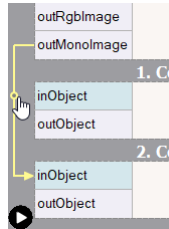
Connecting and Configuring Filters

After a filter is added to the program, it has to be configured. This consists of setting its inputs to appropriate constant values in the Properties window or connecting them with outputs of other filters. It is also possible to make an input linked with an external AVDATA file, connected with [HMI](#) elements or with [Global Parameters](#).

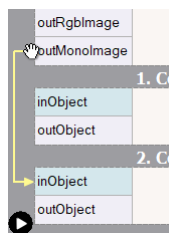
Connecting with Other Filters

Filters receive data from other filters through connections. To create a connection, drag an output of a filter located below in the Program Editor (upward connections are not allowed). When you drag an output – possible destinations will be highlighted. It is basic way to create program's work-flow.

For convenience, new connections can also be created by forking existing connections (drag from the vertical part of the existing connection):



Moreover, it is possible to reconnect an existing connection to another output or input port by dragging the connection's head (near the destination port) or the connection's first horizontal segment (near the source port).



Another way to create connections between filters is by using only the Properties window. When you click the plug () icon in the right-most column, you get a list with possible connections for this input. When the input is already connected the plug icon is solid and you can change the connection to another one.

Setting Basic Properties

Most of the filters have several parameters which you can set in the Properties window as shown on the picture below. It is very important to go through these parameters in order to get desired results for your specific application. To start, first select a filter in the Program Editor window.

This button expands a composite data (a structure or a list) into separate elements

This button opens a list of possible connections for this input

This button opens a specialized data editor

These are primitive properties than can be edited directly

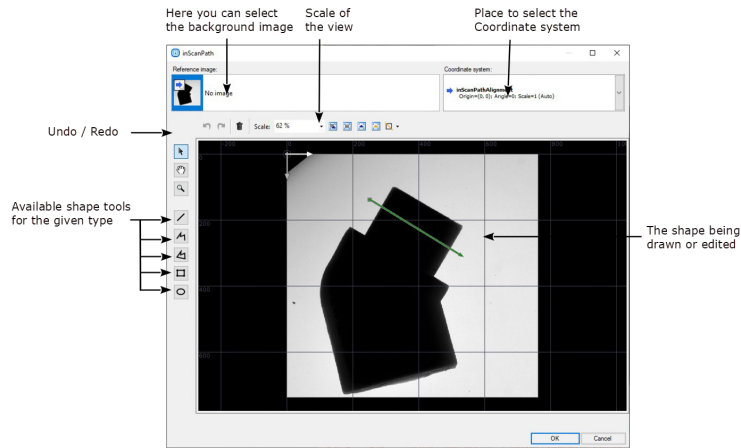
This checkbox switches between proper and empty (automatic) data

Name	Value
Filter	SingleColor
inImage	(0).outMonolm...
inPath	Empty
inPathAlignment	Auto
inColor	{0.000;0.000;...
inDrawingStyle	{HighQuality;1.00...
DrawingMode	HighQuality
Opacity	1.000
Thickness	1.000
Filled	False
PointShape	Auto
PointSize	1.000
inForceRgb	True

Note: After clicking on the header of the properties table it is possible to choose additional columns.

Editing Geometrical Primitives

To edit geometrical data, such as line segments, circle, paths or regions, click the three dots button () in the Properties window at the input port you want to set or modify. A window similar to the one below will appear. The first thing you will usually need to do, when you open this window for the first time, is to select the background image from the list at the top of the window. This will set a context for your geometric data.



Editing of a path in a context of an image.

Tips:

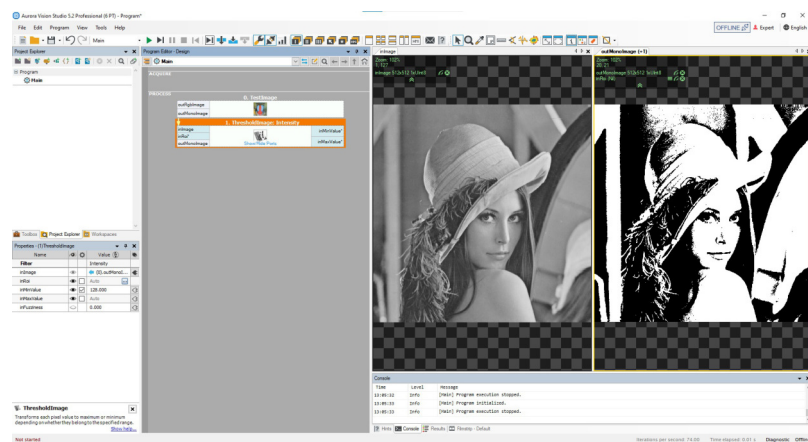
- Select a point and use its context menu to inspect or set the numeric coordinates.
- Use the mouse wheel to zoom the view without changing the tool.
- Hold 3rd mouse button and drag to move the view without changing the tool.
- Hold *Ctrl* to limit the segment angles to the multitudes of 15 degrees.

Testing Parameters in Real Time

One of the greatest features of Aurora Vision Studio is its orientation on rapid development of algorithms. This encompasses the ability to instantly see how different values of parameters affect the results. Due to the dynamic nature of this feature it cannot be presented in a static picture, so please follow the instructions:

1. Create a simple program with a **TestImage** filter connected to a **ThresholdImage** filter.
2. Put the output of the **ThresholdImage** to a data preview.
3. Click **Iterate Current Macro** to execute the program to the last filter, but without ending it as the whole (the execution state will be: *Paused*).
4. Select the **ThresholdImage** filter in the Program Editor and change its **inMinValue** input in the Properties window.

Now, you will be able to see in real time how changing the value affects the result.



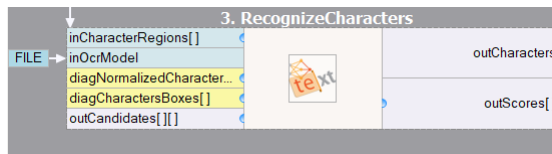
Re-executing a filter after a change of a parameter.

Remarks:

- Please note, that although being extremely useful, this is a "dirty" feature and may sometimes lead to inconsistencies in the program state. In case of problems, stop the program and run it again.
- This feature works only when the filter has already been executed and the program is *Paused*, but NOT *Stopped*.
- By default re-executed is the entire macrofilter. By unchecking the "Global Rerun Mode" setting the re-execution can be limited to a single filter. This can be useful when there are long-lasting computations in the current macrofilter.
- It is not possible to re-execute i/o filters, loop accumulators or loop generators (because this would lead to undesirable side effects). These filters are skipped when the entire macrofilter is getting re-executed.
- When re-executing a nested instance of another macrofilter, the previews of its internal data are NOT updated.
- Re-executing some filters, especially macrofilters, can take much time. Use the Stop command (Shift+F5) to break it, when necessary.
- If you set an improper value and cause a **Domain Error**, the program will stop and it will have to be started again to use the re-execution feature.
- The filter parameters can also be modified during continuous program execution (F5).

Linking or Loading Data From a File

Sometimes data values that have to be used on an input of a filter are stored in an .avdata file, that has been created as a result of some Aurora Vision Studio program (for example creating a custom OCR model). It is possible to load such data to the program with the **LoadObject** filter, but most often it is more convenient and more effective to link the input port directly to the file. To create such a link choose **Link from AVDATA File...** from the context menu of the input (click with the right mouse button on an input port of a filter).

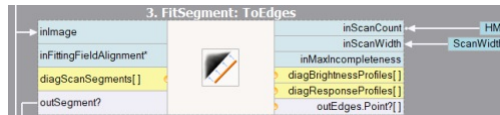


Input data linked from an AVDATA file.

It is also possible to load, not link, data from an .avdata file. This is done with the **Import from AVDATA File...** command in the context menu, which copies the data and makes them part of the current project.

Connecting HMI and Global Parameters

It is also possible to connect filter inputs from outputs of HMI elements and from **Global Parameters**. Both get displayed as rectangular labels at the sides of the Program Editor as can be seen on the image below:



A filter with connections from HMI and from a Global Parameter.

For further details please refer to the documentation on the specific topics:

- [HMI Designer](#)
- [Global Parameters](#)

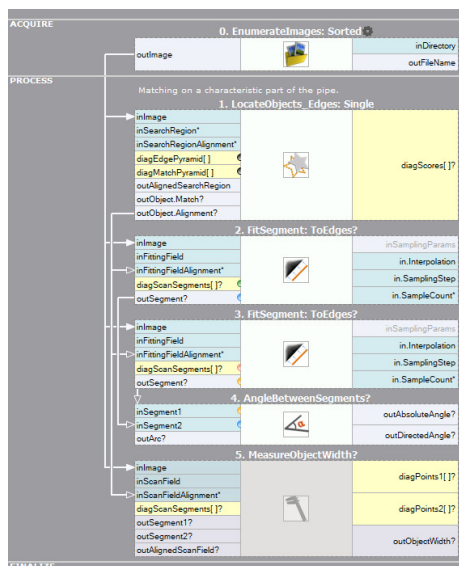
Writable and Readable Global Parameters

It is possible to read and write the value of global parameter by using **ReadParameter** and **WriteParameter** filters. This approach allows users to change global parameter values accordingly to their needs at any point of an algorithm. A few practical applications of this feature are listed below:

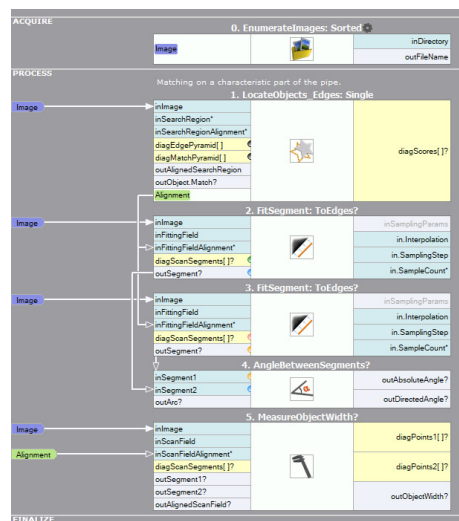
- Managing recipes depending on a signal from PLC,
- Storing data transferred between nested macrofilters,
- Setting global flags.

Labeling Connections

Connections are easier to follow than variables as long as there are not too many of them. When your program becomes complicated, with many intersecting connections, its readability may become reduced. To solve this problem Aurora Vision Studio provides a way to replace some connections with labels. You can right-click on a connection, select "Label All Connections..." - when there is more than one connection or "Label Connection..." when only one connection is present. Then set the name of a label that will be displayed instead. The labels are similar to variables known from textual programming languages - they are named and can be more easily followed if the connected filters are far away from each other. Here is an example:



A program with some long connections becomes not easy to analyze.



Long connections are replaced with labels for better readability.

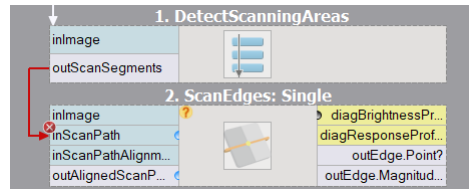
Remarks:

- Labeled connections can be brought back (unlabeled) by using the "Un-label This Connection" or "Un-label All Connections" commands available in the context menu of a label.
- Please note, that when your program becomes complicated, the first thing you should consider is reducing its complexity by refactoring the macrofilter hierarchy or rethinking the overall structure. Labeling connections is only a way to visualize the program in a more convenient way and does not make its structure any simpler. It is the user's responsibility to keep it well organized anyway.

- Aurora Vision Studio enforces that all connections between filters are clearly visualized, even if making them implicit would make programming easier in typical machine vision applications. This stems from our design philosophy that assumes that: (1) it is wrong to hide something that the user has to think about anyway, (2) the user should be able to understand all the details of a macrofilter looking at a static screen image.
- When the amount of connections becomes large despite good program structure you should also consider creating User Structures that may be used for merging multiple connections into one. (Do NOT use global parameters for that purpose).

Invalid Connections

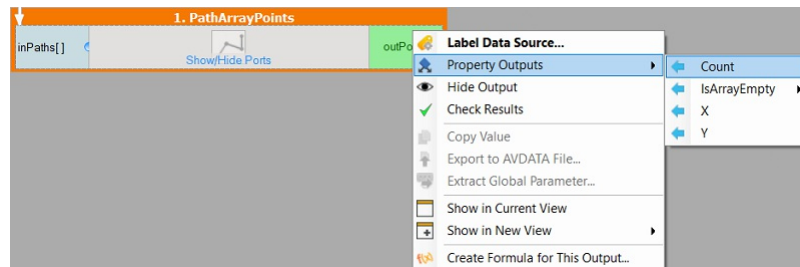
As types of ports in macrofilters and formulas may be changed after connections with other filters have been created, sometimes an existing connection may become invalid. Such an invalid connection will be marked with red line and white cross in the Program Editor:



Invalid connections do not allow to run the program. There are two ways to fix it: replace the invalid connections or revise the types of the connected ports.

Property Outputs

In addition to available filter's outputs, it is possible to get much more information out of a filter. Many data types are represented as structures containing fields, e.g. [Point2D](#) consists of "X" and "Y" fields of [Real](#) data type. Although these fields are not available as standard outputs, a user can easily add them as additional filter outputs – we call them "Property Outputs". That way, they are directly available for creating connections with inputs of other filters.



Accessing fields of [Point2DArray](#) type.

Additional Property Outputs

Some of the properties are calculated based on the current state of the output (for example, checking whether the array is empty or counting its elements), whereas other are derived directly from the internal structure of the type (for example, "X" and "Y" fields composing [Point2D](#) structure). The types providing such properties are listed in the table below.

Structure name	Property outputs
Bool	Not
ByteBuffer	Size IsByteBufferEmpty
Histogram	Size
Matrix	IsMatrixEmpty
Path	Points Size Length IsPathEmpty
Profile	Size IsProfileEmpty
Region	Area IsRegionEmpty
Segment2D	Length Direction Center
String	Length IsStringEmpty
Vector2D	Length Direction
Vector3D	Length

There are also special types, which cannot exist independently. They are used for wrapping outputs, which may not be produced under some conditions ([Conditional](#)) or optional inputs ([Optional](#)), or else for keeping a set of data of specified type ([Array](#)). Property outputs of such types are listed in the table below.

Type name	Property outputs
Array	Count IsArrayEmpty
ArrayArray (array of arrays)	Count IsArrayEmpty IsNestedArrayEmpty
Conditional	IsNil
Optional	IsNil

All of the above-mentioned property outputs are specific for the type. However, ports of [Array](#), [Optional](#) or [Conditional](#) type may have more property outputs, depending on the wrapped type. For example, if the output of a filter is an array of regions, this port will have Count, IsArrayEmpty (both resulting from the array form of the port), IsRegionEmpty and Area (both resulting from the type of the objects held in the array – in this case, regions) as property outputs. However, if the output of a filter is an array of objects without any property outputs (e.g. [Integer](#)), only outputs resulting from the array form of the port (Count and IsArrayEmpty) will be available.

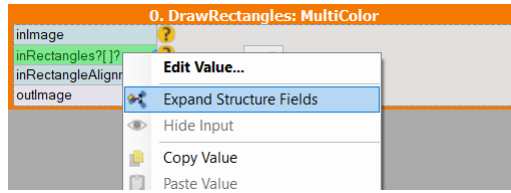
Tip: Avoid using basic filters like [Not](#), [ArraySize](#) which will have bigger performance impact than additional property outputs.

What Do *IsNil* and *IsEmpty* Mean:

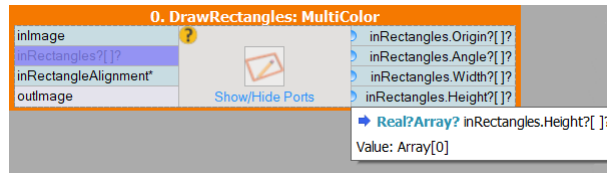
These property outputs return a bool value depending on the content of the data. If the data is *Nil*, what usually means that no object was detected, the property output *IsNil* is set to *True* otherwise it remains *False*. The *IsEmpty* property has a similar use. The only difference is that it reacts if the inputted object is *Empty*. In most common cases one use it for conditional execution of macrofilters or as an input date to formulas. You can read how to deal with *Nil* and *Empty* objects in an article about [conditional execution](#)

Expanded Input Structures

Analogously to expanding output properties, it is also possible to expand input structures. This works only for basic structures which allow access to all their fields (e.g. *Point2D*, but not *Image*). Please note, that the expanded input structure is replaced by the fields it consists of (unlike by adding property outputs, where the structure itself remains available).



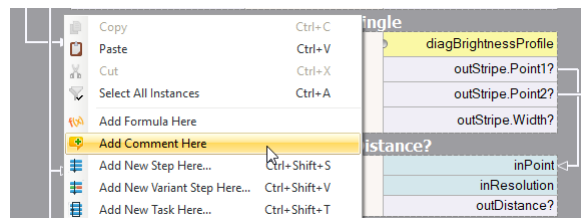
Expanding filter input of *Rectangle2D?Array?* type.



Result of expanding input of *Rectangle2D?Array?* type.

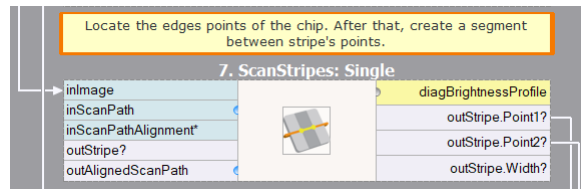
Comment Blocks

Comments are very important part of every computer program. They keep it clear and easy to understand during development and further maintenance. There are two ways to make a comment in Aurora Vision Studio. One way is to add a special comment block. Another option is adding a comment directly in the filter. To add a new comment block to program click the right mouse button on the Program's Editor background and select the "Add Comment Here" option like on the image below.



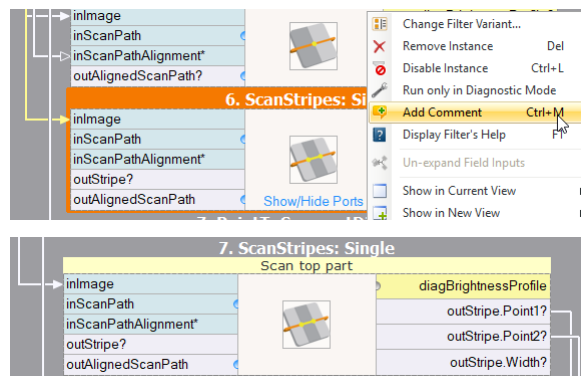
Adding a new comment.

Comment block can be very useful for describing the details of the algorithm.



Comments can be used for describing program steps.

But when you need just a simple tip or short remark, use a "Add Comment" after mouse right click on the filter:



New comment directly in the filter.

Creating Macrofilters

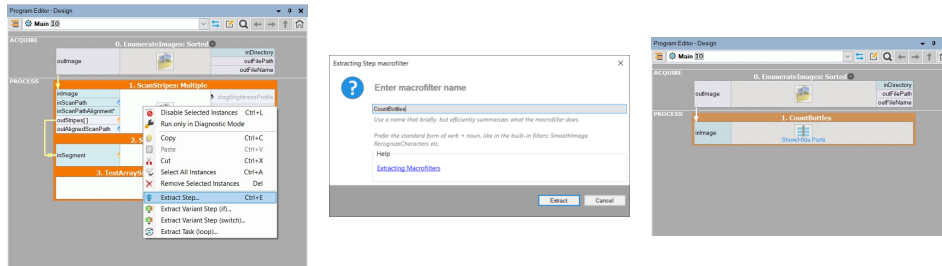
Introduction

Macrofilters play an important role in developing bigger programs. They are subprograms providing a means to isolate sequences of filters and re-use them in other places of the program. After creating a macrofilter you can create its instances. From outside an instance looks like a regular filter, but by double-clicking on it you navigate inside and you can see its internal structure.

When you create a new project, it contains exactly one macrofilter, named "Main". This is the top level macrofilter from which the program execution will start. Further macrofilters can be added by extracting them from existing filters or by creating them in the Project Explorer window.

Extracting Macrofilters (The Quick Way)

The most straightforward way of creating a macrofilter is by extracting it from several filters contained in an existing macrofilter. This is done by selecting several filters in the Program Editor and choosing the *Extract Step...* command from the context menu, as depicted in the picture below:



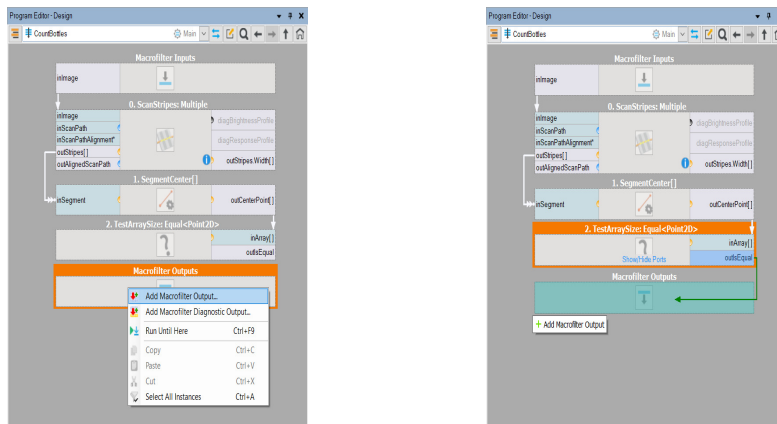
Extracting a macrofilter from three existing filters.

After completing this operation, a new macrofilter definition is created and the previously selected filters are replaced with an instance of this macrofilter. Now, additional inputs and outputs of the new macrofilter can be created by dragging and dropping connections over the new macrofilter instance.

Remark: The "Extract Task (loop)..." command creates a new Task macrofilter which should be used only in special cases. Execution of the Task macrofilter is more complex than execution of the Step macrofilter. Usage of the Task macrofilter for reducing macrofilter complexity may be inappropriate. For more details please read section about [Task macrofilters](#).

Defining the Interface

Being inside of a macrofilter other than "Main" you can see two special blocks: the *Macrofilter Inputs* and the *Macrofilter Outputs*. The context menus of these blocks allow to add new inputs or outputs. Another method of adding them is by dragging connections and dropping them over these blocks.



Adding a new output using the context menu of the Macrofilter Outputs block.

Adding a new output by dragging and dropping a connection.

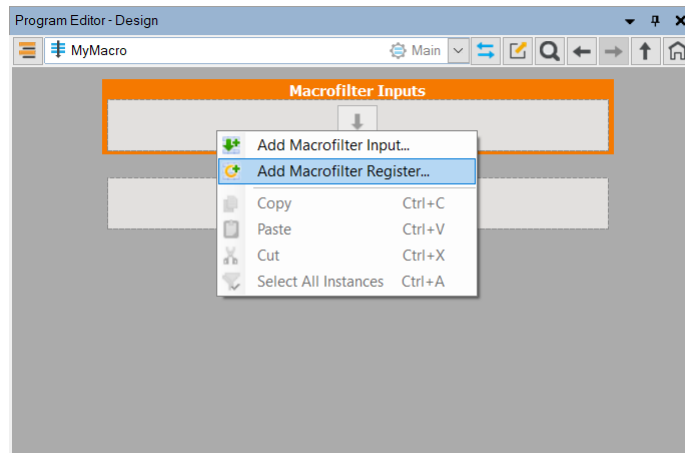
Before the new port is created you need to provide at least its name and type:

Defining a port of a macrofilter.

NOTE: Names of macrofilter inputs and outputs should be clear and meaningful. Names of inputs always start with "in", while names of outputs always start with "out".

Adding Registers

The context menus of macrofilter input and output blocks also contain a command for adding [macrofilter registers](#), *Add Macrofilter Register...*. This is an advanced feature.

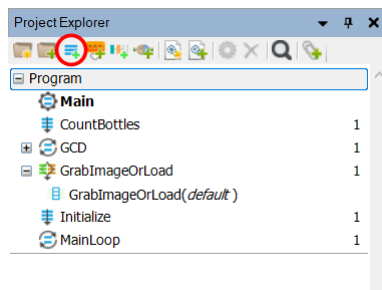


Creating Macrofilters in the Project Explorer

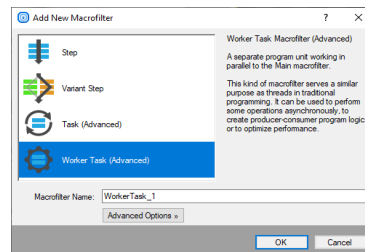
All macrofilter definitions that are contained in the current project can be displayed in the [Project Explorer](#) window (by default it is behind the Toolbox on a tab page). Using this control, the user can create or edit macrofilters, but it also acts as a filter catalog from which instances can be created by dragging and dropping the items into the Program Editor.

To create a new macrofilter in the Project Explorer click the *Create New Macrofilter...* button. A new window will appear allowing you to select the name and [structure](#) of the new macrofilter.

Please note, that due to their special use *Worker Tasks* can only be created in the Project Explorer. For more information on *Worker Tasks* please refer to [Macrofilters](#) article.



Adding Macrofilters in the Project Explorer.



Selecting the name and the structure of a new macrofilter.

Copying Macrofilters

When copying macrofilters in the Program Editor window, you can only create their new instances: modifying one instance will inevitably affect all the others. In order to duplicate the definitions (create brand-new copies) of old macrofilters as independent entities that you will be able to modify separately, you need to do it inside the Project Explorer window.

Copying macrofilters in the Project Explorer - creating new definitions.

Copying macrofilters in the Program Editor - creating new instances.

Please keep in mind, that regardless of whether you copy macrofilters in the Program Editor or the Project Explorer, with other macrofilters nested inside, no new definitions of nested macrofilters will be created.

1. When copying the parent macrofilter (first nest level) in the Program Editor, the number of instances of nested (child) macrofilters will not change.
2. When copying the parent macrofilter (first nest level) in the Project Explorer, new instances of nested (child) macrofilters will be created.

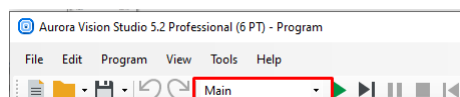
If you want to create a new copy of the whole macrofilter tree (copy of all definitions) you will have to copy each macrofilter separately, starting with the most nested one.

The StartUp Program

It is sometimes useful to have several programs in a single project. The most common scenarios are:

- When some more complex data, e.g. custom OCR models or calibrated reference images, need to be prepared in a separate program.
- When it is convenient to have multiple versions of a single program customized for different installations.
- When the highest reliability is important and sets of automated tests need to be performed on recorded images.

In Aurora Vision Studio a program can be any [Worker Task](#) macrofilter. The list of all macrofilters fulfilling this criterion can be seen in the StartUp Macrofilter combo box on the Application Toolbar. This control makes it possible to choose the program that will be run. The related macrofilter will be displayed in the Project Explorer with the bold font.



The StartUp Program Combo Box.

Macrofilter Guidelines

Macrofilters organize big projects, but it is responsibility of the user to make this organization clean and effective. When a project grows, especially under the pressure of time, it is easy to forget this and create programs that are difficult to understand and maintain. To avoid this, please follow the following guidelines:

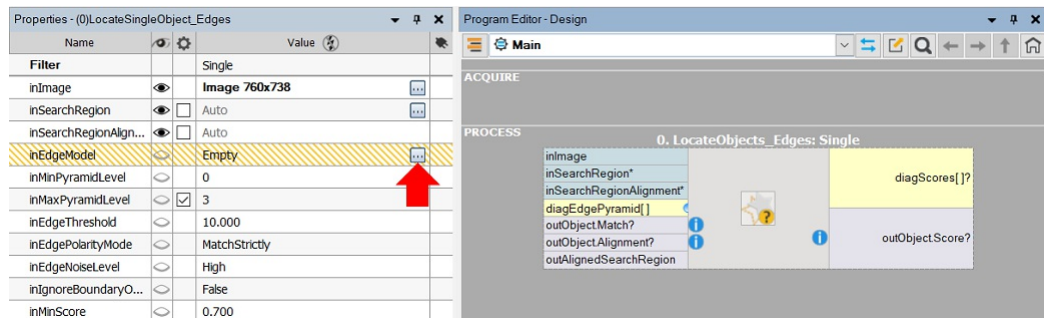
1. Macrofilters should not be too big. For program clarity they should typically consist of 3-15 filters. If you macrofilters have more than 20, it is certainly going to be cause trouble.
2. Each macrofilter should have a clear and single function well reflected in its name. It is recommended to create macrofilters as tools that could possibly be used in many different projects.
3. Macrofilter names do not affect the program execution, but they are very important for effective maintenance. Choose precise and unambiguous English names. Follow the *verb + noun* naming convention whenever possible and avoid abbreviations (this applies also to names of macrofilter ports).
4. Do not mix data analysis with things like communication or visualization in a single macrofilter. These should be separated. If data visualization is needed for HMI, compute results in one macrofilter and then prepare the visualization in another. This will make the data flow more clear.
5. During project development it is very common that the initial program structure becomes inappropriate for what has been added during development, or that it becomes too complicated and unclear. This is when you should consider REFACTORING, i.e. redesigning the program structure, boundaries of macrofilters and revising any complicated sequences of filters. We strongly recommend considering refactoring as a routine part of the job.

Creating Models for Template Matching

Introduction

Template Matching tools are very often used as one of the first steps in industrial inspection applications. The goal is to detect the location of an object. Before this can be done the user has to create a model representing the expected object's shape or structure. To make this step straightforward, Aurora Vision Studio provides an easy user interface. We call it a "GUI for Template Matching".

The GUI for Template Matching is an editor for values of two types: **EdgeModel** and **GrayModel**. This means that to open it the user has to select a template matching filter in the program and then click on the  button at the **inGrayModel** or **inEdgeModel** input in the Properties window:



Opening GUI for Template Matching.

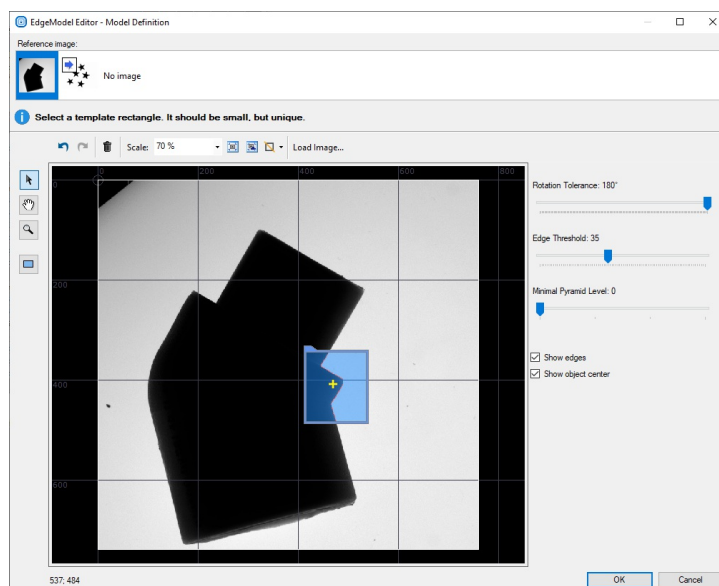
In majority of industrial applications Template Matching algorithms implemented in Aurora Vision Studio easily detect the location of an object using default parameters. However, there are cases where it is needed to tune some of them, mainly in order to achieve higher reliability. It turned out that there is a group of parameters, which is used more often than others. Therefore GUI for Template Matching is available in two variants: Basic and Expert. This is related to **Complexity Levels** available in Aurora Vision Studio.

Creating a Model

Basic

The Basic window contains the following elements:

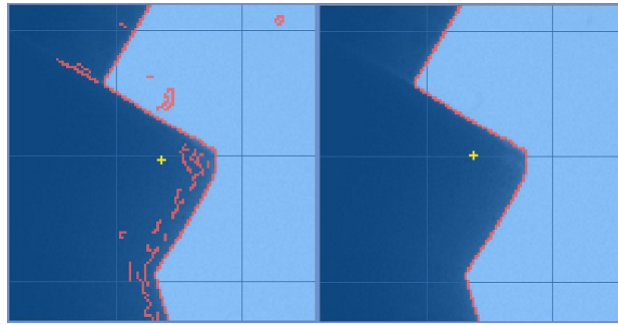
1. At the top: a list of possible template images
2. Below: a simple toolbar that also contains a button for loading a template image from a file
3. On the left: a tool for selecting a rectangular template region
4. On the right: track-bars for setting parameters and some options related to the view
5. In the center: an area for selecting the template region in the context of the selected template image



Basic GUI for Template Matching window

To create a template matching model you need to:

1. Choose a template image from the "Reference image" list.
2. Select a rectangular template region using  drawing tool. For maximum performance this rectangle should be as small as possible.
3. Edge-based matching only: Set the **Edge Threshold** parameter, which should be set to value that results in the best quality of the edges. **Edge Threshold** determines the minimum strength (gradient's magnitude) of pixels that can be considered as an edge.



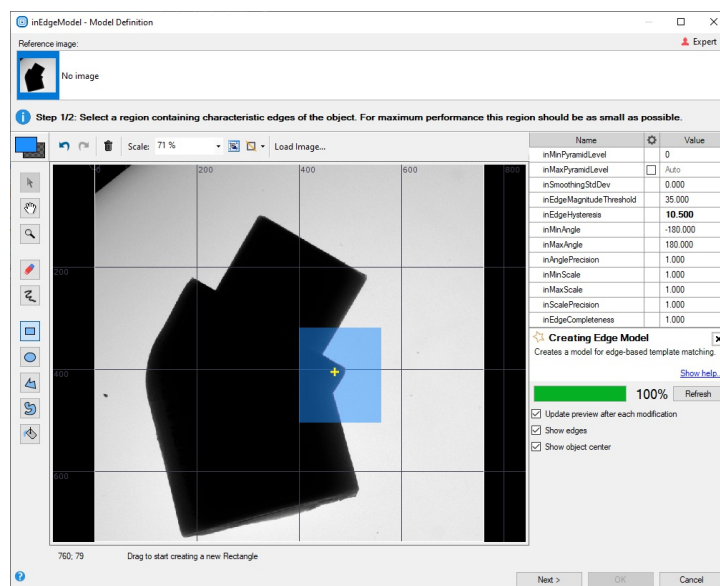
Low quality edges (Edge Threshold = 8) and high quality edges (Edge Threshold = 30)

4. Set the **Rotation Tolerance** in range from 0° to 360° . This parameter determines the maximum expected object rotations. Please note that the smaller the **Rotation Tolerance**, the faster the matching.
5. Set the **Minimal Pyramid Level** parameter which determines the lowest pyramid level used to validate candidates who were found on the higher levels. What is worth mentioning is that setting this parameter to a value greater than 0 may speed up the computation significantly, however, the accuracy of the matching can be reduced. More detailed information about Image Pyramid is provided in [Template Matching](#) document in our [Machine Vision Guide](#).

Expert

The Expert window contains the following elements:

1. At the top: a list of possible template images
2. Below: a simple toolbar that also contains a button for loading a template image from a file
3. On the left: a tool for selecting a template region of any shape
4. On the right: parameters of the model, their description and some options related to the view
5. In the center: an area for selecting the template region in the context of the selected template image

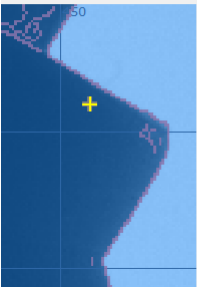


GUI for Template Matching window

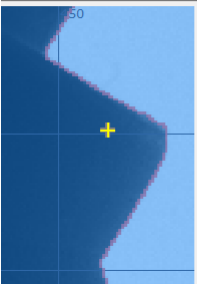
To create a template matching model you need to:

1. Choose a template image from the "Reference image" list.
2. Mark a template region using drawing tools (please note, that the button above the tools switches the current color, i.e. you can both draw or erase shapes). For maximum performance this region should be as small as possible.

- Edge-base matching only: Set the **inSmoothingStdDev**, **inEdgeMagnitudeThreshold** and **inEdgeHysteresis** parameters to values that result in the best quality of the found edges. It is advisable to first set **inEdgeHysteresis** to zero, then choose a value for **inEdgeMagnitudeThreshold** that assures that all edges have some parts detected and then increase **inEdgeHysteresis**. Small noise might be removed by change the value of parameter **inSmoothingStdDev**. For example:

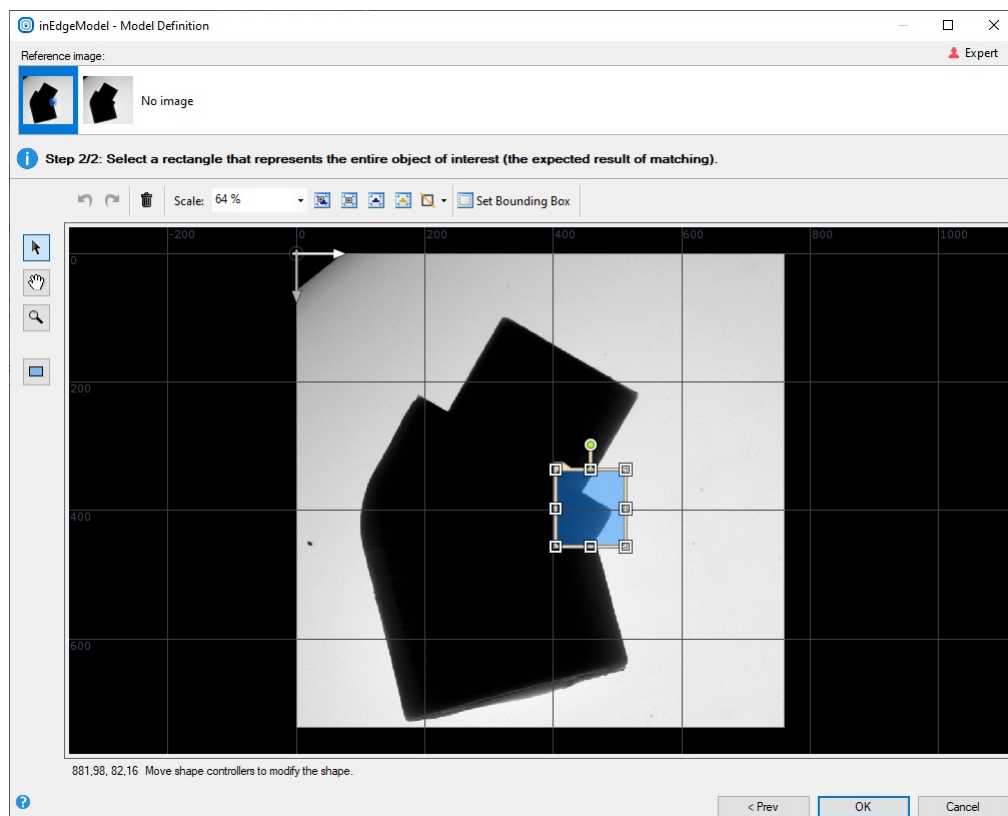
	Name		Value
	inMinPyramidLevel		0
	inMaxPyramidLevel	☐ Auto	
	inSmoothingStdDev		0.000
	inEdgeMagnitudeThreshold		13.000
	inEdgeHysteresis		15.000
	inMinAngle		-180.000
	inMaxAngle		180.000
	inAnglePrecision		1.000
	inMinScale		1.000
	inMaxScale		1.000
	inScalePrecision		1.000
	inEdgeCompleteness		1.000

Low quality edges

	Name		Value
	inMinPyramidLevel		0
	inMaxPyramidLevel	☐ Auto	
	inSmoothingStdDev		0.000
	inEdgeMagnitudeThreshold		30.000
	inEdgeHysteresis		15.000
	inMinAngle		-180.000
	inMaxAngle		180.000
	inAnglePrecision		1.000
	inMinScale		1.000
	inMaxScale		1.000
	inScalePrecision		1.000
	inEdgeCompleteness		1.000

High quality edges

- Set the **inMinAngle** and **inMaxAngle** parameters accordingly to the expected range of object rotations (the smaller the range, the faster the matching).
- Set the **inAnglePrecision** to a value lower than 1.0 if you prefer to lower the angular precision for the benefit of speed.
- Set the **inMinScale**, **inMaxScale** and **inScalePrecision** to appropriate value if you need to detect objects in different scale. You should be aware of longer detection time when you detect objects in scale.
- Click the "Refresh" button or check "Update preview after each modification" to review the results.
- Click the "Next >" button to select template rectangle that represents the entire object of interest (the expected result of matching)



Template rectangle selection

- Click "OK" to close the window and generate the model.

Performing Template Matching

When the model is ready, performing template matching in an application is straightforward – after connecting the filters and setting the matching parameters, the results are on the outputs of the [LocateSingleObject_Edges1](#) filter (or similar):

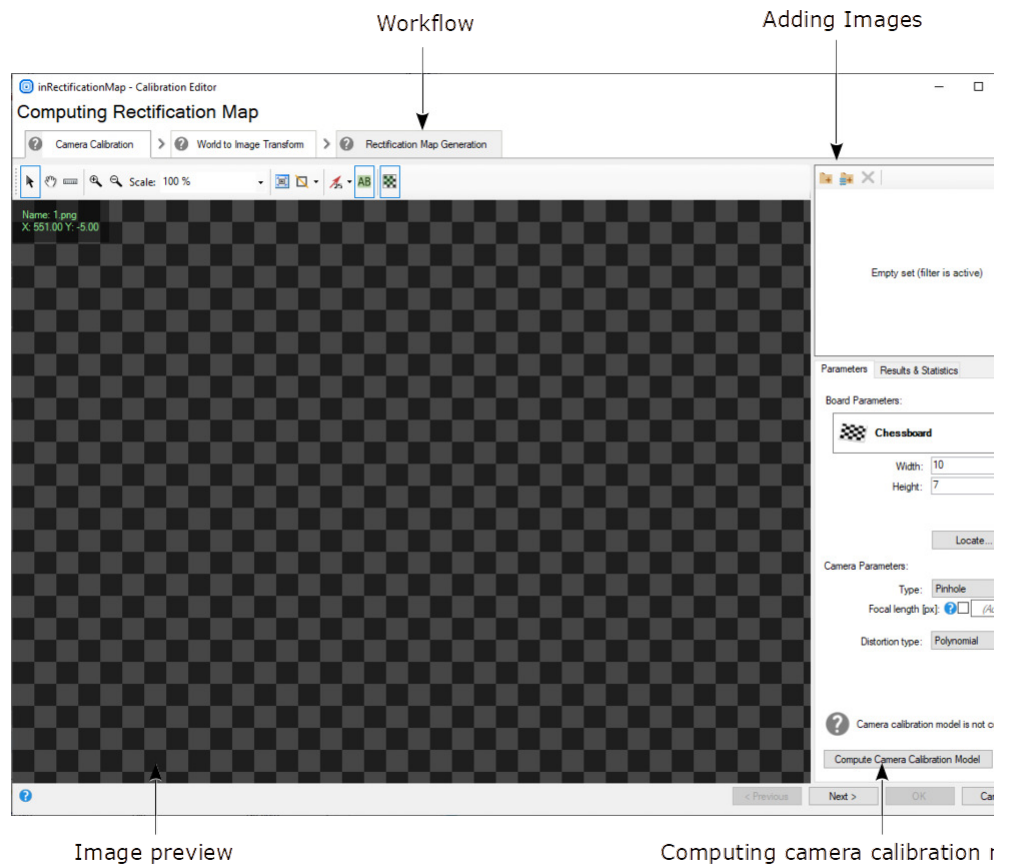
Preparing Rectification Transform Map

Overview

Before you read this article, make sure you are acquainted with [Camera Calibration and World Coordinates](#).

If you want to perform camera calibration in Aurora Vision Studio, there are two basic ways to do that. You can either use a calibration editor (plugin) or use filters (manual configuring parameters and connecting outputs with corresponding inputs). Both approaches provide you with the very same results, but the latter way allows you to control intermediate steps and outputs if they are relevant to you for some reason.

This guide will focus mainly on the first approach and show how to perform calibration with the editor step by step. To perform image rectification, the [RectifyImage](#) filter should be applied. When you click on the **inRectificationMap** input, the Calibration Editor will be displayed:



The overview of the Calibration Editor.

As you can see the editor consists of three pages:

1. **Camera Calibration** - in which camera lens parameters are computed.
2. **World to Image Transform** - in which perspective and transform converting real-world points to image points are computed.
3. **Rectification Map Generator** - in which fast pixel transform is computed to get a rectified image.

Each of the pages will be individually described, but as it has been already noted, each of the pages in the Calibration Editor invokes corresponding calibration filters. The general overview is in the table below:

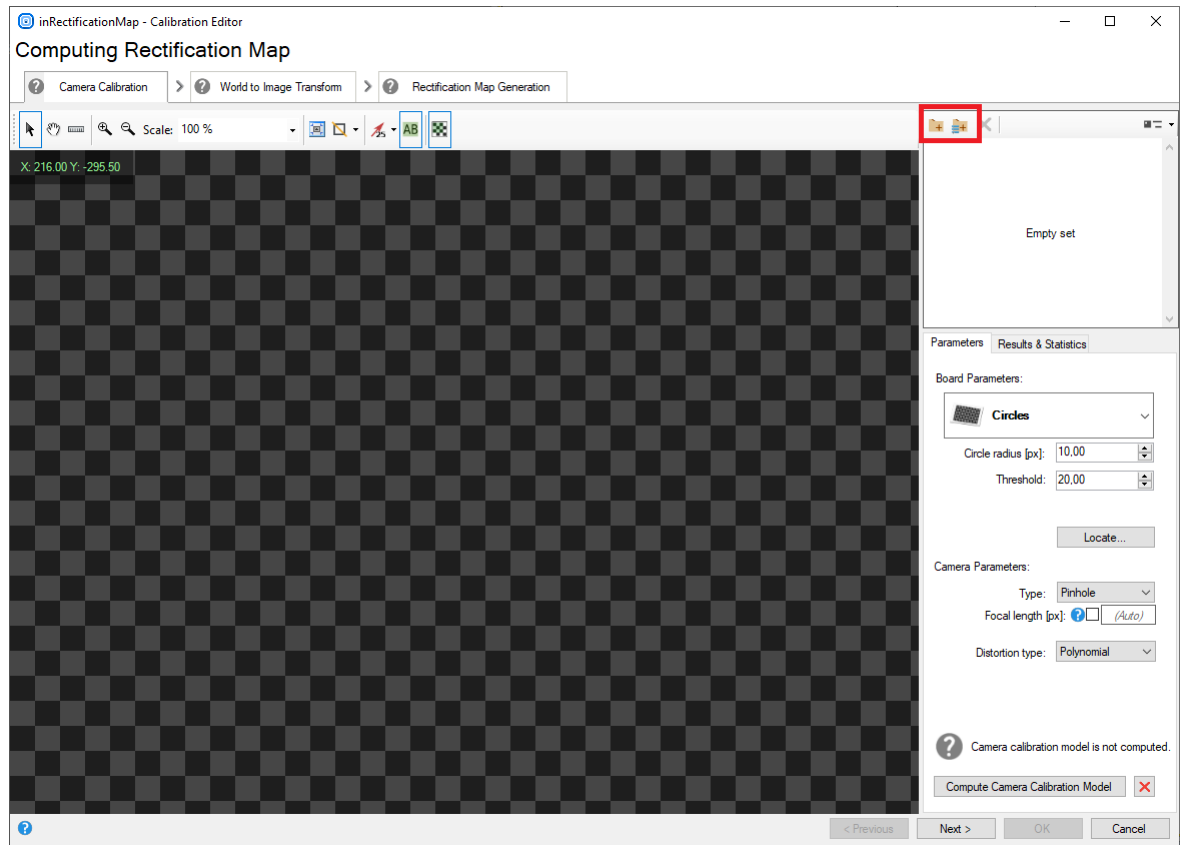
Page name	Corresponding filters
Camera Calibration	CalibrateCamera_Pinhole CalibrateCamera_Telecentric
World to Image Transform	CalibrateWorldPlane_Default CalibrateWorldPlane_Labeled CalibrateWorldPlane_Manual CalibrateWorldPlane_Multigrid
Rectification Map Generation	CreateRectificationMap_Advanced CreateRectificationMap_PixelUnits CreateRectificationMap_WorldUnits

Camera Calibration Page

On the first page of the Calibration Editor you have to provide images of calibration boards, using the buttons marked in red.

The purpose of this stage is to estimate intrinsic camera parameters. They do not depend on the

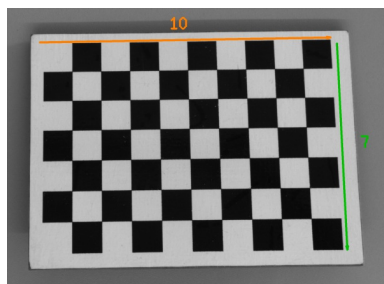
transformation between the camera and its external world, therefore this step can be done before mounting the camera at the target place.



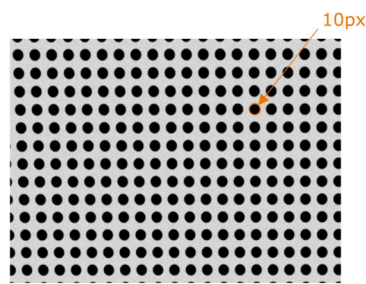
Camera Calibration Page - adding images.

Next, you need to specify board and camera parameters. There are two possible options to choose from, depending on what kind of pattern is used:

- **Chessboard** - where you have to define two parameters of the board: width and height, which correspond to the number of squares in horizontal and vertical directions, respectively.
- **Circles** - where you have to define a single circle's radius and threshold value.



If you are using chessboard pattern, you have to count squares in both dimensions. In this example, the width is 10 and the height is 7.



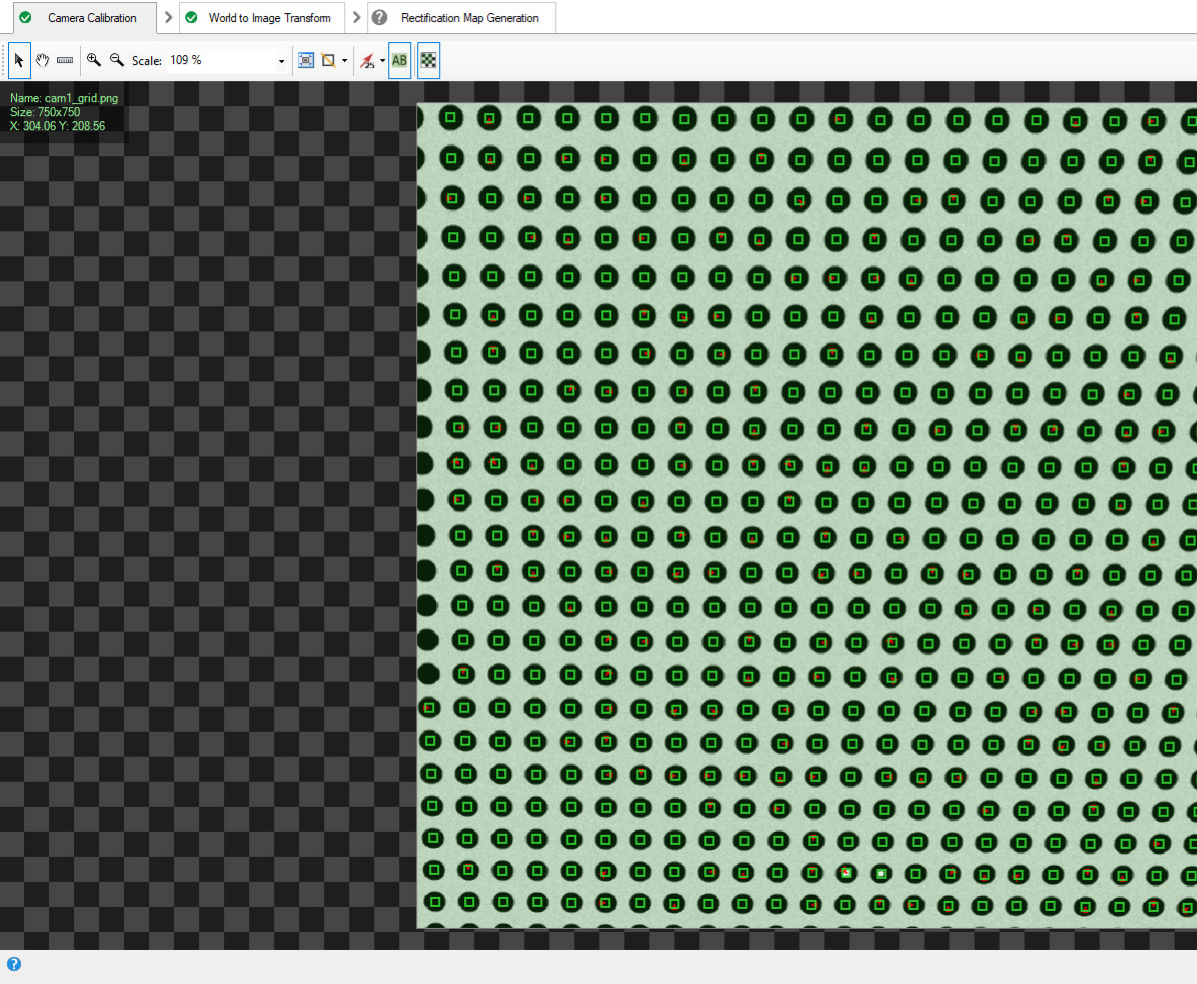
If you are using circle pattern, you have to measure radius of any circle. In this example, it is about 10px. Note: it is important to use a symmetric board as shown in the image. Asymmetric boards are currently not supported.

Once they are set, you should adjust camera type according to an applied camera, which can be either **pinhole** (which uses a standard perspective projection) or **telecentric** (uses an orthographic projection).

A few distortion model types are supported. The simplest - divisional - supports most use cases and has predictable behavior even when calibration data is sparse. Higher order models can be more accurate, however, they need a much larger dataset of high quality calibration points, and are usually needed for achieving high levels of positional accuracy across the whole image - order of magnitude below 0.1 px. Of course this is only a rule of thumb, as each lens is different and there are exceptions.

Results & Statistics tab informs you about results of the calibration, especially about reprojection error, which corresponds with reprojection vectors shown in red on the preview in the picture below:

Computing Rectification Map

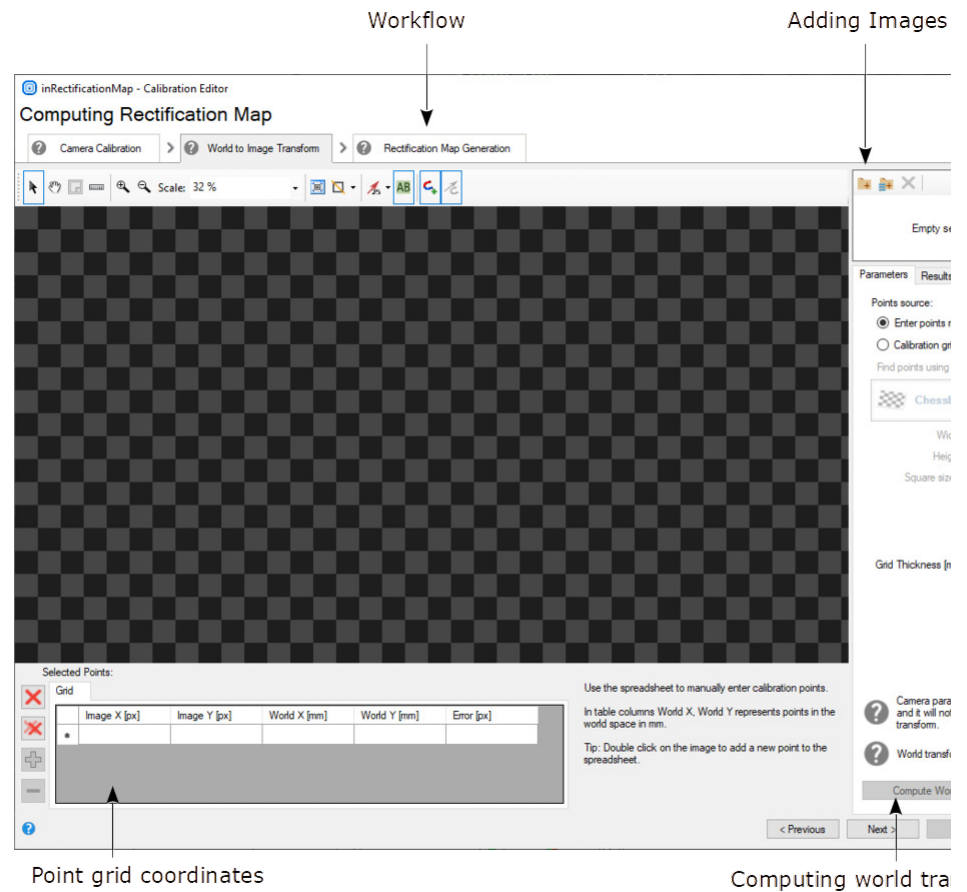


They indicate the direction towards each located point should be moved in order to deliver an immaculate grid. Additionally, there are values of Root Mean Square and Maximal error. The tab also displays value and standard deviation for each computed camera and lens distortion parameter.

For more details, please refer to the corresponding group of filters for this page in the [table](#) above.

World to Image Transform Page

Points added to the second page are used to find transformation between real-world and image points.



World to Image Transform - overview.

First of all, in this step, you need to add images of the calibration board as well (using the very same buttons as in the previous step). Please note that in this step they have to be taken from a fixed angle in contrast to the previous step, where all of images could be taken from different angles.

In the next step you should indicate points source in either of two ways:

- By **entering points manually**
- By using **calibration grid**

Points can be found automatically using the calibration board as in the previous step.

After successful points location on the calibration board, you should use the spreadsheet to manually enter coordinates in world plane:

Computing Rectification Map

Camera Calibration

World to Image Transform

Rectification Map Generation

Scale: 104 %

AB

Name: cam1_grid.png
Size: 750x750
X: 428.78 Y: 195.40

Selected Points:

Grid 1.1

	Image X [px]	Image Y [px]	World X [mm]	World Y [mm]	Error [px]
▶	30.65	13.41	1.00	0.00	
+	66.06	13.57	2.00	0.00	
	101.54	13.73	3.00	0.00	
-	136.99	13.94	4.00	0.00	

Use the spreadsheet to manually enter coordinates in world plane.
Empty points will be calculated automatically based on entered points before computing. If there aren't any entered points, c

In table columns World X, World Y represents points in the world space in mm.

World to Image Transform - modifying points in the spreadsheet.

Points with unspecified world coordinates will have them calculated automatically based on points with specified coordinates. If there are no world coordinates entered, then algorithm will assume some default world point locations, what might produce false results. At least two grid point world coordinates are needed to uniquely determine the world plane position, rotation and spacing.

inRectificationMap - Calibration Editor

Computing Rectification Map

Camera Calibration

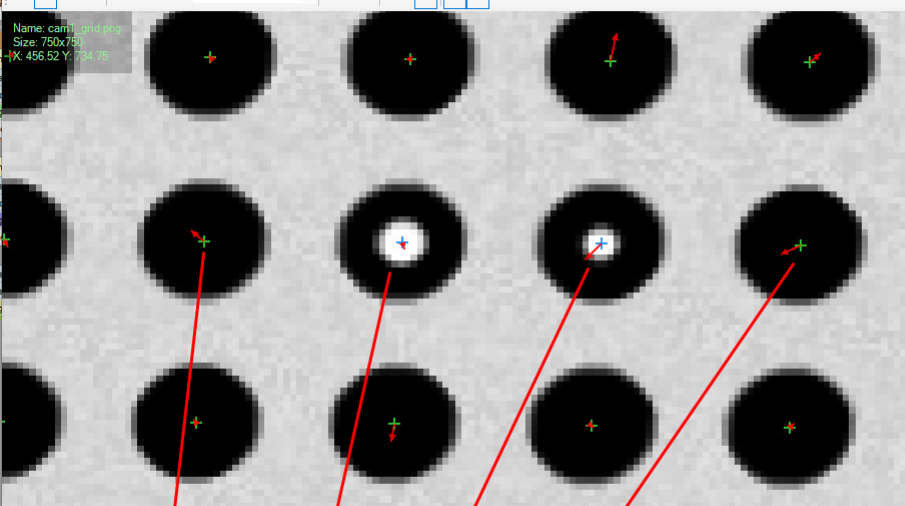
World to Image Transform

Rectification Map Generation

Scale: 635 %

AB

Name: cam1_grid.png
Size: 750x750
X: 456.52 Y: 734.75



Selected Points:

Grid 1.1

	Image X [px]	Image Y [px]	World X [mm]	World Y [mm]	Error [px]
▶	468.77	428.06	40.00	-180.00	0.07
	501.75	428.30	60.00	-180.00	0.04
	534.70	428.55	80.00	-180.00	0.06
	567.84	428.80	100.00	-180.00	0.07

Use the spreadsheet to manually enter coordinates in world plane.
Empty points will be calculated automatically based on entered points before computing. If there aren't any entered points, default points locations will be accepted.

In table columns World X, World Y represents points in the world space in mm.

cam1_grid.png

media/image_stitching

Parameters

Results and Statistics

Total Reprojection Error

Image (RMS): 0.08 [px]
World (RMS): 0.08 [mm]

Grids Statistics

Computed Grid Spacing: 20.00 [mm]

[Grid 1.1]
Rotation: 0.00 [deg]
Translation:
X: 0.00 [mm]
Y: 0.00 [mm]
Max Repr. Error: 0.19 [px]

Homography Matrix

1.560.010.85390.66
0.011.63700.11
0.000.001.00

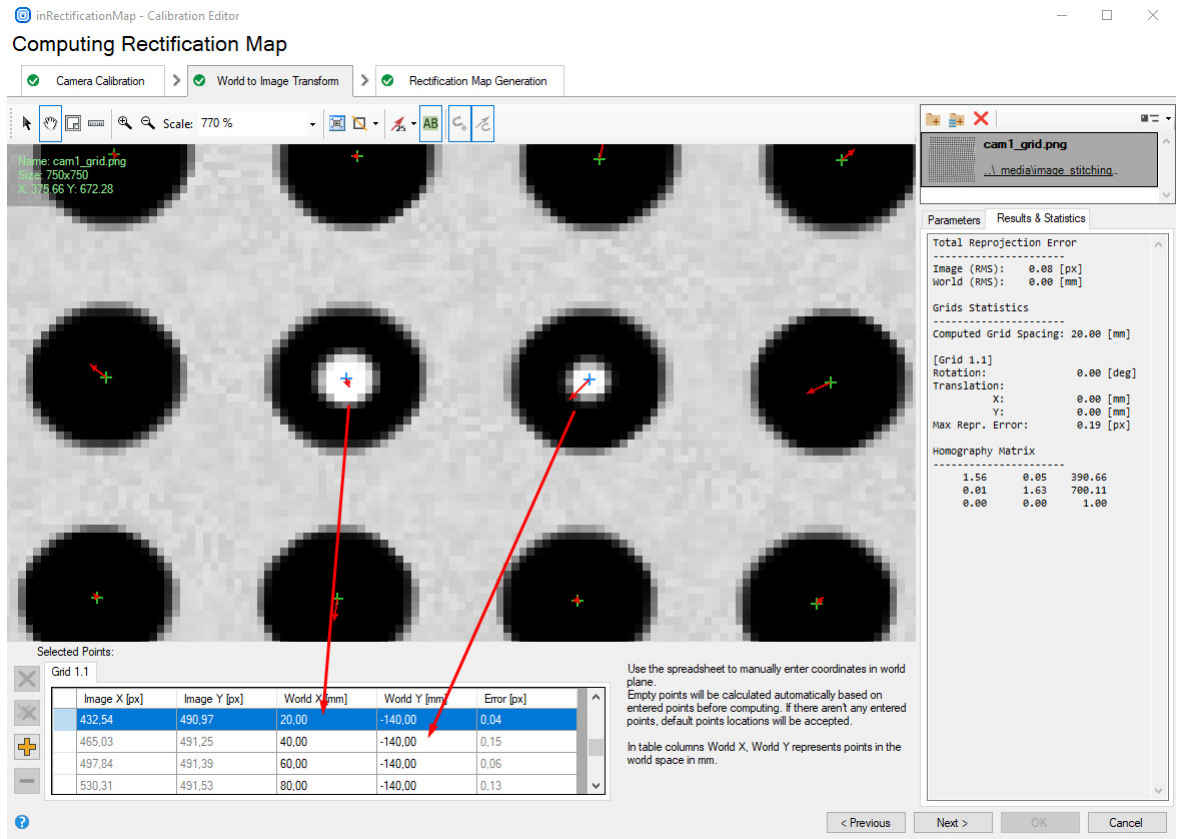
< Previous

Next >

OK

Cancel

World to Image Transform - automatic grid spacing assumed.



World to Image Transform - grid spacing calculated from world point coordinates given by the user.

Arrows indicate which points from the calibration grid relate to corresponding rows in the spreadsheet. As you can see each row consists of coordinates in the image plane (given in **pixels**), coordinates in the world plane (given in **mm**), and error (in **pixels**), which means how much a point is deviated from its model location. In this case reprojection vectors, which are marked as small, red arrows, also indicate the deviation from the model.

Colors of points have their own meaning:

- **Green Point** - the point has been computed automatically.
- **Orange Point** - the point which has been selected.
- **Blue Point** - the point has been adjusted manually.

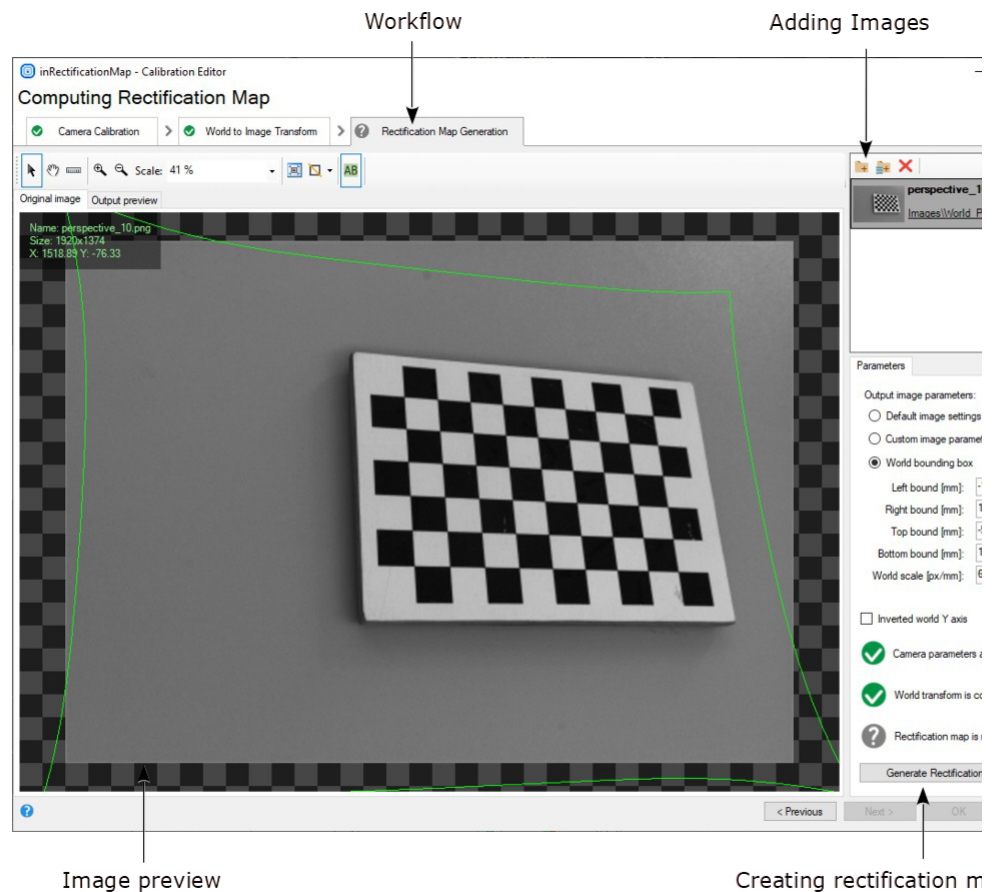
The **Results & Statistics** tab shows information about computed errors for both image and world coordinates. The output reprojection errors are useful tool for assessing the feasibility of computed solution. There are two errors in the plugin: Image (RMS) and World (RMS). The first one denotes how inaccurate the evaluation of perspective is. The latter reflects inaccuracy of labeling of grid 2D world coordinate system. They are independent, however they both influence quality of the solution, so their values should remain low.

For the details, please refer to the corresponding group of filters for this page in the [table](#) above.

Rectification Map Generator Page

Last page is used to set parameters of an image after the rectification.

First of all, you need to load images of the calibration board in the same way as in previous steps.



Rectification Map Generation - rectifying the image.

Next, you have to choose one of three basic options of the output image:

- **Default image settings**
- **Custom image parameters**
- **World bounding box**

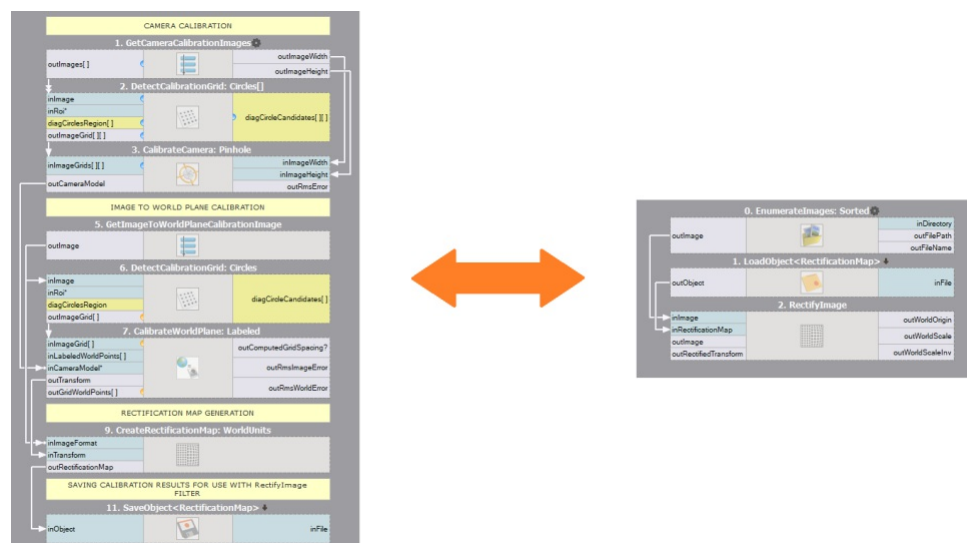
The green frame on the preview informs you about the size and borders of the output image, and depending on what option you have chosen, you can set different parameters. Using *custom parameters* or *world bounding box* will provide you with the same rectification map, but what makes the difference is that you are working in different domains - in the first case you are operating in the image coordinates (given in pixels), whereas in the other one you are operating in the world coordinates (given in millimeters).

When you are done, you can click on the **Generate Rectification Map** button and assess the rectified image displayed on the preview.

For the details, please refer to the corresponding group of filters for this page in the [table](#) above.

Relation between the Calibration Editor and filters

The relation between both approaches could be presented in a form of the below graphics:



*The left side presents which filters are necessary to generate a rectification map and how to save it using **SaveObject**. The right side presents how to load the rectification map using **LoadObject** and passing it to the **RectifyImage** filter.*

This is just an exemplary set of filters which might be applied, but using specific filters depend on the calibration board and other parameters relevant to a case.

Further readings

- [Calibration-related list of filters in Aurora Vision Studio](#)

Creating Text Segmentation Models

Please be informed that this article is related to the traditional OCR method. Nowadays, we strongly recommend using Deep Learning OCR tools, which are much faster and more efficient than the traditional ones in many cases. You can find more information about the Deep Learning tools [here](#).

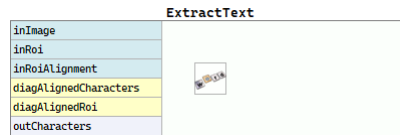
The graphical editor for text segmentation performs two operations:

1. **Thresholding** an image with one of several different methods to get a single foreground region corresponding to all characters.
2. **Splitting** the foreground region into an array of regions corresponding to individual characters.

Details about using OCR filters can be found in [Machine Vision Guide: Optical Character Recognition - traditional method](#).

To configure text extraction please perform the following steps:

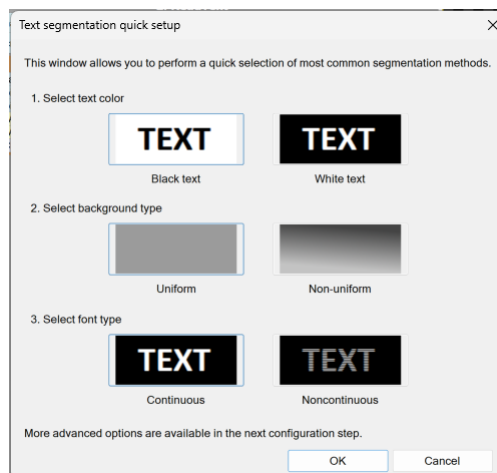
1. Add an [ExtractText](#) filter to the program.



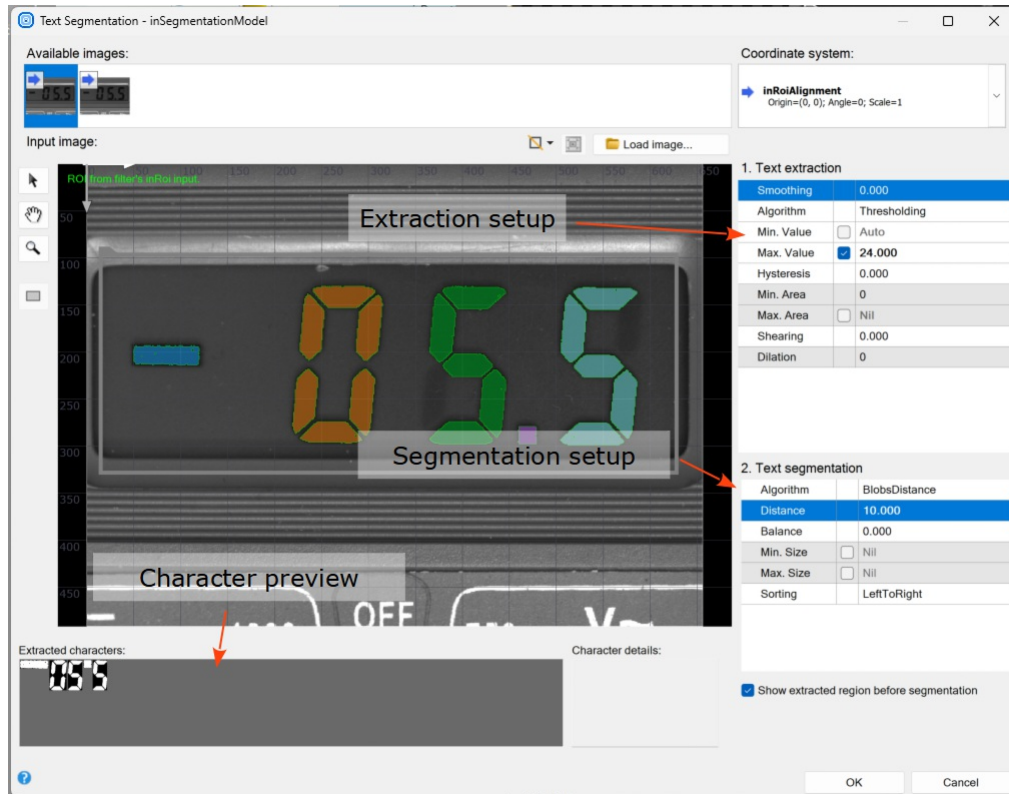
2. Set the region of interest on the inRoi input. This step is necessary before performing next steps. The image below shows how the ROI was selected in an example application:



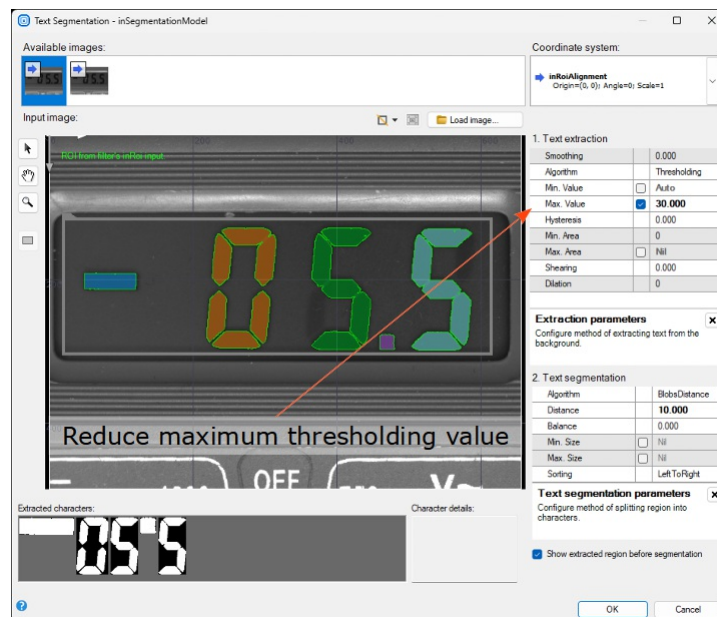
3. Click on the "..." button at the **inSegmentationModel** input to enter the graphical editor.
4. When entering first time, complete the quick setup by selecting most common settings. In this example a black non-continuous text should be extracted from a uniform background. Configuration was set to meet these requirements.



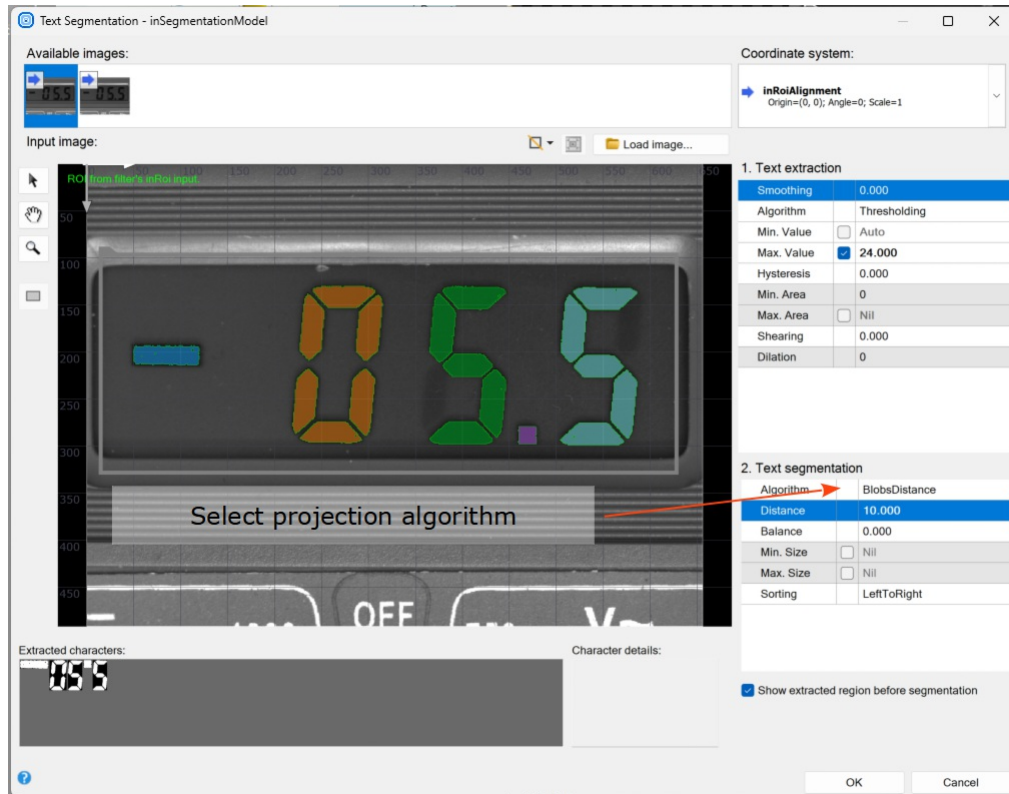
5. After the quick setup the graphical editor starts with some parameter set. Adjust the pre-configured parameters to get best results.



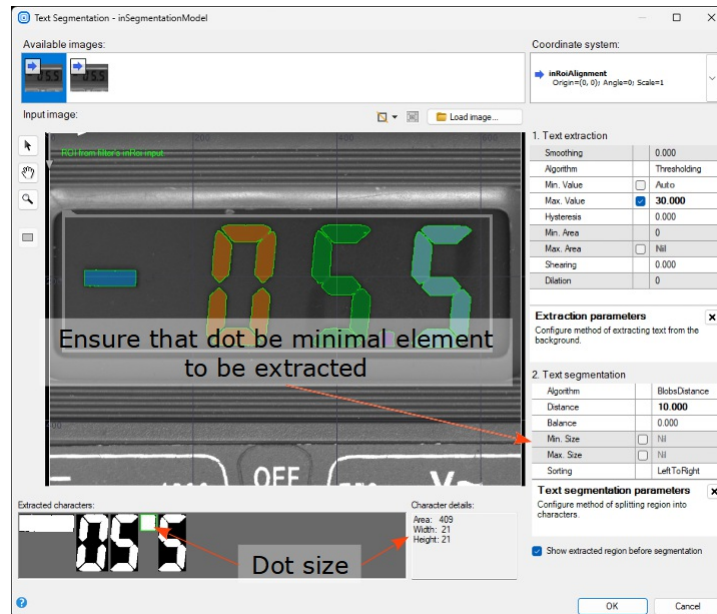
6. Configure a character extraction algorithm. In this case thresholding value is too high and must be reduced.



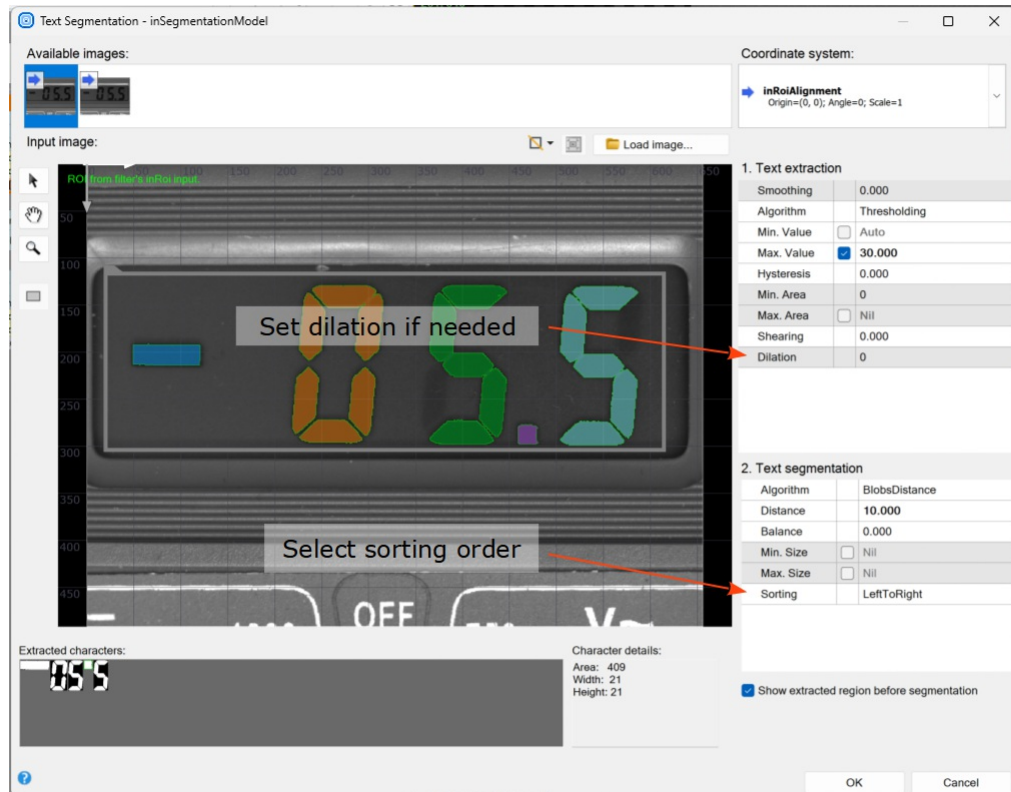
7. Select a character segmentation algorithm.



8. Set the minimal and the maximal size of a character. The editor shows the character dimensions when the character is selected in the list below.



9. Select a character sorting order, character deskewing (shearing) and image dilation.

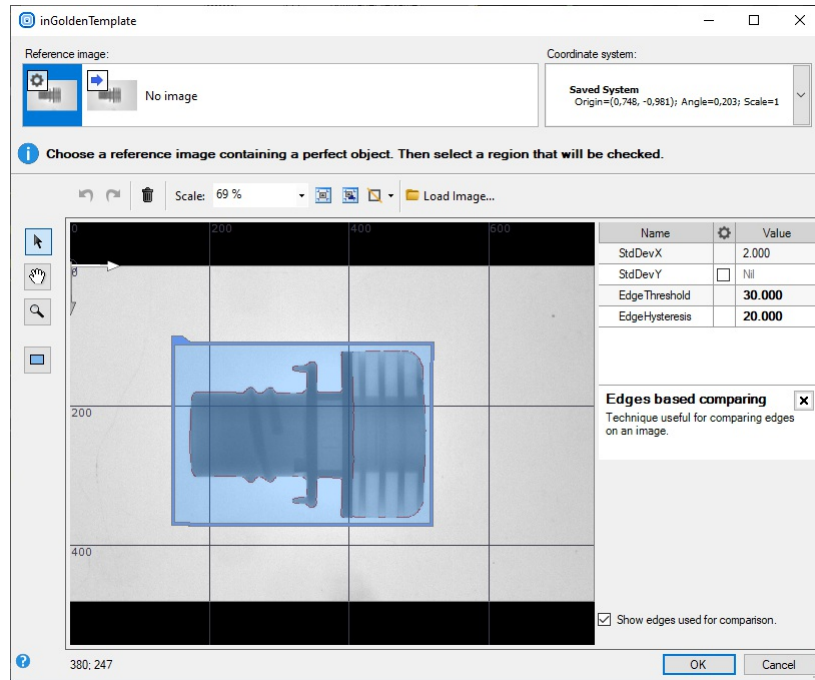


10. Check results using available images.



Creating Golden Template Models

Golden template technique is the most powerful method for finding objects' defects. Editor presented below is available in filters [CompareGoldenTemplate_Intensity](#) and in [CompareGoldenTemplate_Edges](#).



To create golden template, select template region and configure its parameters.

Remarks:

- To reduce computation time try to select only necessary part of an object,
- For comparing both edges and surface use two [CompareGoldenTemplate](#) filters,
- To create mode programmatically use filter [CreateGoldenTemplate](#).

Creating Models for Golden Template

Introduction

Golden Template is an image comparison technique. It is based on the pixel-to-pixel comparison but uses multiple images and advanced algorithms to create a multi-image model. It is useful for finding general defects of objects that have fixed shape. In order to simplify the process of model creation a "GUI for Golden Template 2" was created. It is possible to access it by clicking on the  button at the inModel input in the Properties window:

Opening GUI for Golden Template 2

Image Preparation

To create a golden template model you need at least three same-sized images representing the same object. The object should be placed in the same way in all the images. Otherwise, the final model may not be accurate enough.

The first step is to prepare the images. To be sure that the object is always precisely positioned and have the same size – in both the model and the program – you can use the sequence of filters presented below:

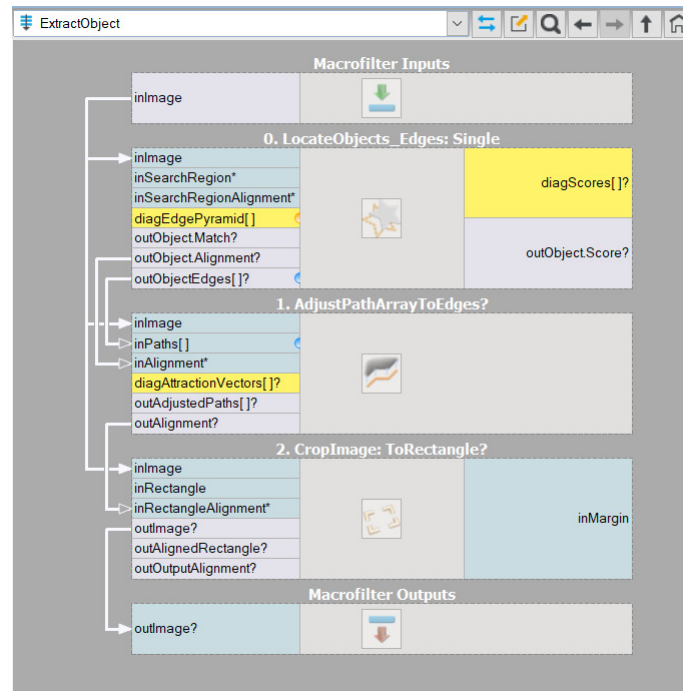


Image preparation process

Following steps were performed there:

- [LocateSingleObject_Edges1](#) filter was used to create a robust model able to locate the logo and remember its alignment
- [AdjustPathArrayToEdges](#) allows you to improve the template matching results
- [CropImageToRectangle](#) gives evenly-cut images of the object. You should specify the **inRectangle** input manually.

If you have at least three images prepared, you can proceed to the next stage.

Model Creation

Add [CreateGoldenTemplate2](#) filter to the program and click **inModel** in the properties window, as shown in the image at the beginning of the article. After that the following window will appear:

Golden Template Editor

Firstly, add images to the editor. It is possible to either use the drag and drop function or load images from the directory by clicking on the  icon. The images will be used to create the Golden Template model, so they should present samples without defects.

After loading all the images, draw a mask that represents the object of interest on one image. It applies to all the other images that you loaded. That is why it was necessary to properly position the objects in the images. If no mask is detected the warning will appear. If the object covers most of the field of view, please feel free to mark the whole image.

Now you can check the preview with the "Show average of all images" and adjust the training parameters:

- **Downscale** – resizing the image dividing by the value. It greatly speeds up the computing in exchange for the ability to spot pixel-size defects
- **MaximalDisplacement** – possible error in object positioning, high values may impair detection of small defects, especially near edges
- **LargeDefectSize** – expected diameter of largest, extensive defects
- **BrightnessAugmentation** – allows for additional brightness deviation
- **NoiseAugmentation** – allows for additional noise presence in the images
- **SmoothingAugmentation** – allows for additional smoothing in the images, uses gaussian smoothing with specified standard deviation

Clicking **OK** button will start the training. The editor will close itself afterwards.

The model is created and loaded in the [CreateGoldenTemplate2](#) filter. If you prepare the inspection images in the same way as during model creation (by positioning and cropping them) everything should work properly.

Creating Text Recognition Models

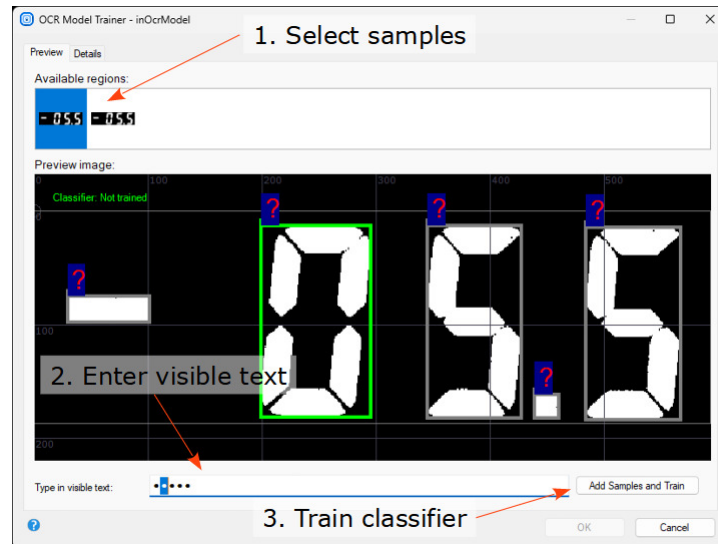
Please be informed that this article is related to the traditional OCR method. Nowadays, we strongly recommend using Deep Learning OCR tools, which are much faster and more efficient than the traditional ones in many cases. You can find more information about the Deep Learning tools [here](#).

Text recognition editor creates an OCR model for getting text from regions. More details about the traditional OCR technique can be found in [Machine Vision Guide: Optical Character Recognition](#).

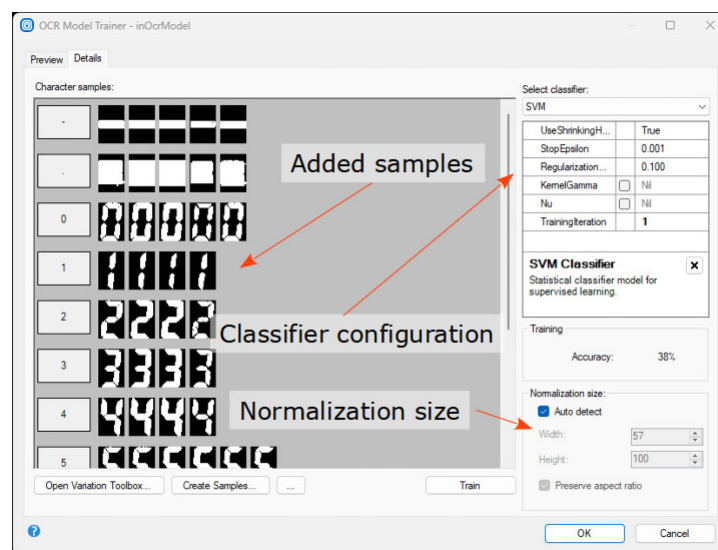
To create an OCR model a set of characters should be collected. If recognition score is low after training based on real samples then artificial character variations can be created.

Creation of a model consists of following steps:

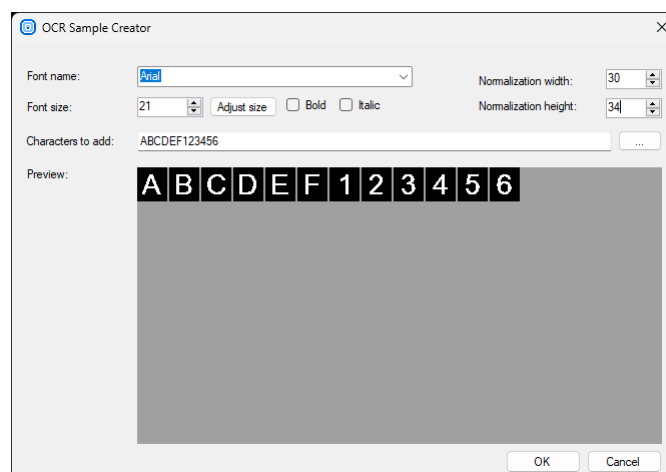
1. **Collecting real samples** - after opening the editor characters are visible and can be added to a training set.



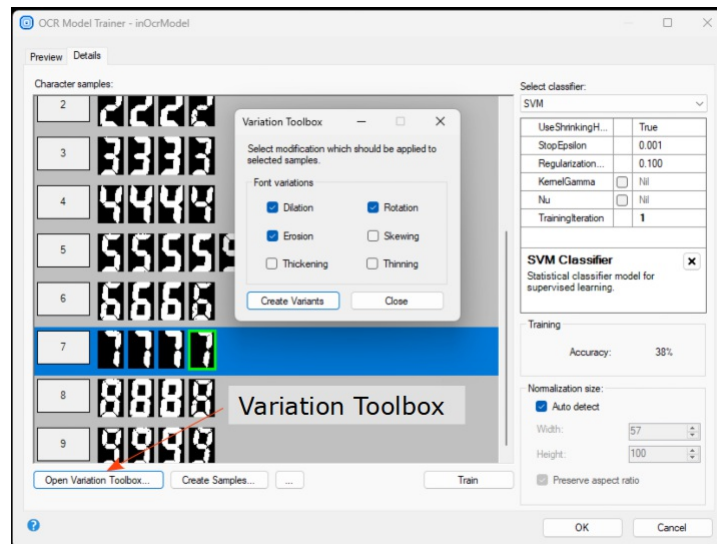
After the training character samples can be viewed in the details tab:



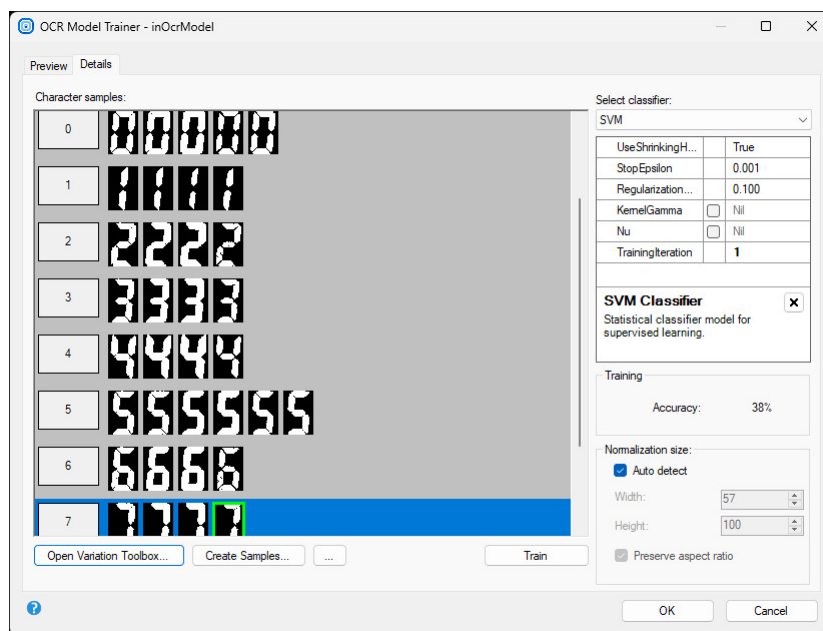
2. **Creating artificial samples** - when no samples are available user can create a training set using systems fonts.



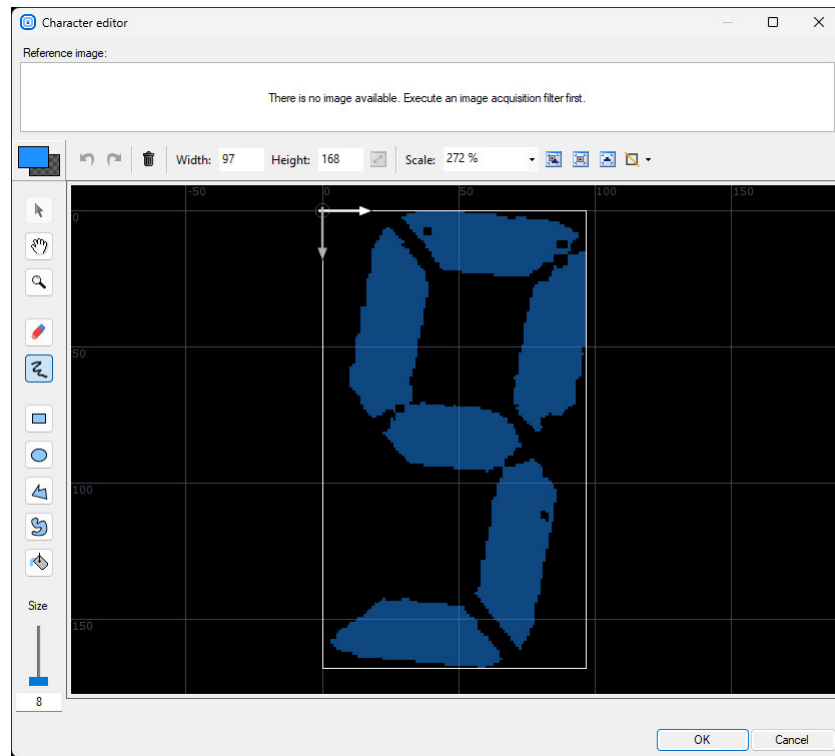
3. **Creating character variations** in case when no more samples are available and the training result is not fine the editor can modify existing samples to create a new set.



The training set after adding new samples variations:



4. **Editing samples** – in case when gathered samples contain noises, or its quality is low, user can edit them manually. The image below show how to edit a character '8' to get character '9'.



Note:

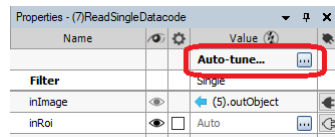
- Each training character should have this same number of samples.
- In cases when some characters are very similar number of samples can be increased to improve classification.
- Character samples can be stored in an external directory to perform experiments on them.

Auto-tuning Datacode Parameters

Introduction

Auto-tune Editor can assist you in determining the optimal parameters for the [ReadDataCodes](#) filter. By using one or more provided images, it attempts to read as many codes as possible by testing different parameter combinations. It also optimizes the reading score and execution time.

To open the Editor, double-click on the [ReadDataCodes](#) filter or click on **Auto-tune...** in the Properties panel. Note that inputs to be modified by Auto-tune cannot be connected.



Opening Auto-tune Editor

Basic Operation

Auto-tune Editor overview

Parameters for datacode reading from the inputs **inDetectorParams**, **inMatrixCodeParams** and **inLinearCodeParams** are loaded into the editor and can be seen on the right side. Additionally, the **inMatrixSymbolologies** and **inLinearSymbolologies** input values are also loaded.

First, select an image from the program to be used (you may need to run the program first). After selecting an image, the editor tries to read the codes on the image using the current parameters. Codes that can be read are highlighted on the image.

After pressing the **Auto-tune** button, the optimization process begins. The algorithm initially attempts to read as many codes as possible (at least the minimum defined number). Then, the parameters are refined to improve the scores of the read codes until at least the minimum defined score is achieved. Finally, the parameters are optimized to achieve the smallest possible execution time while still reading the required number of codes and maintaining an acceptable score.

During optimization, the current best parameters are updated live and displayed on the right. Read codes are displayed on the image. The progress of optimization regarding score and execution time can be seen on the charts.

If the results are satisfactory, the user can accept them by clicking **OK**. The optimized parameters are then written to the filter inputs.

If something goes wrong during optimization, you can start from scratch by clicking the **Reset** button. The results are then cleared, and the parameters are returned to their default values.

Conservative Optimization

If the user clicks the **Extend only** button instead of **Auto-tune**, optimization is performed in conservative mode. This mode is designed to make modifications to the reading parameters that will not prevent the filter from reading codes that were previously readable. For example, the parameter **MaxRowCount** in regular auto-tune can be increased or decreased, but in conservative mode, it can only be increased.

This mode is intended for tweaking parameters that already work in most cases. For example, it can help read a new difficult code while continuing to read existing codes.

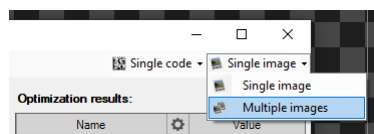
Single and Multiple Codes Mode

By default, the editor starts in **single code** mode, meaning it can read only one code per image. To read more codes, change the variant of the [ReadDataCodes](#) filter to **Multiple** and reopen the editor.

Note that in **multiple codes** mode, additional filter inputs are loaded into the editor (and written back when closing the editor). The input **inMaxCodeCount** determines how many codes can be present in one image at most. The input **inAllowMultipleScales** can be useful when there are codes of different sizes in one image.

Single and Multiple Images Mode

Similarly, the editor begins in **single image** mode. For simplicity, the user can select a single image from those used in the current program. To assist auto-tune in choosing better parameters, the user can switch to **multiple images** mode to provide more images. Use the menu in the top-right corner to switch.



Switching to multiple images mode

Before starting auto-tune you will need to load the images into the list on the left. All of them will be considered during optimization process.

Editor in multiple images mode

Multiple codes mode and **multiple images** mode can be used at the same time.

Auto-tune Parameters

Auto-tune requires some parameters to be set manually:

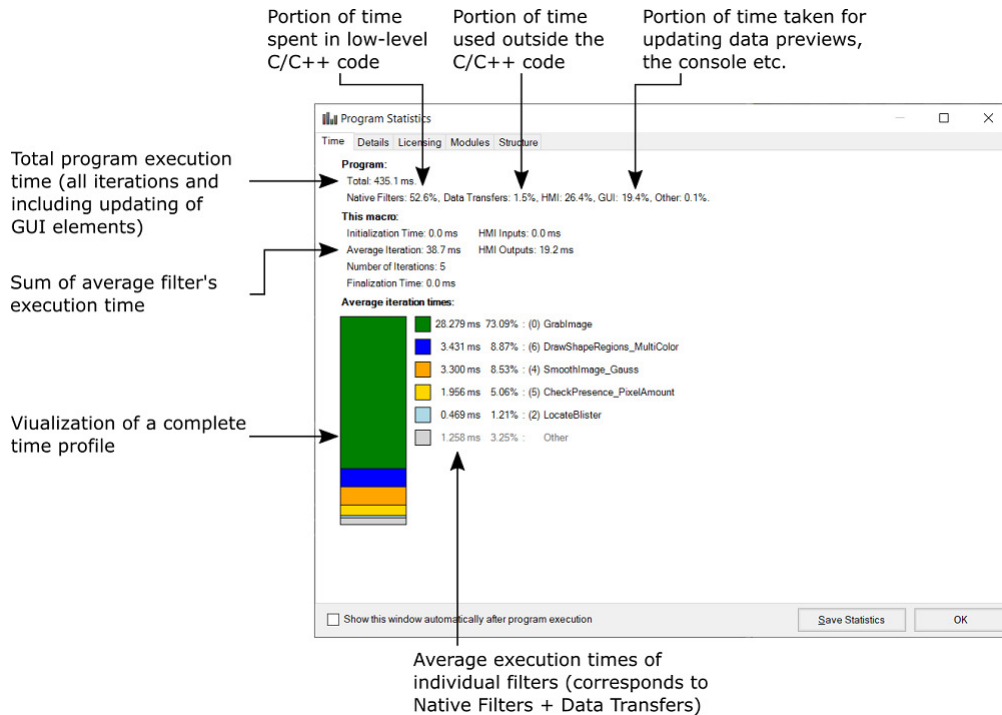
- **Minimum score:** The algorithm will attempt to improve the score up to this value, and during execution time optimization, it will not allow the score to fall below this value.
- **Detection method:** If set to a value other than Auto, the DetectionMethod parameter will consistently be set to this value. This can be used to prevent auto-tune from altering it.
- **Time budget:** Specifies the time to spend on optimization (in seconds). Alternatively, you can set a large value and then click **Stop** to manually finish auto-tune.
- **Minimal number of codes to be read in all images:** The algorithm will initially attempt to use parameters that allow reading at least this many codes in all images combined. Setting this value too high may cause optimization failure. Conversely, setting it lower than the actual number of codes may result in not all codes being read, but the score and execution time may improve.
- ***Maximal number of codes in one image:** If set too low, the code reader may stop reading before finding all codes in one image, potentially preventing auto-tune from achieving the minimum required number of codes across all images.

- ***Allow multiple scales:** If set to false, the code reader may fail to read two codes when their sizes differ significantly.
- **Read codes after parameters or image changed:** After a relevant code reading parameter is changed, the editor will read codes again on the current image so the user can observe the effect. However, if the codes are difficult and reading takes too much time, you can disable this for a better user experience.
- ***Types of code:** Limits which types of codes are read. **If a code type is unchecked here, it will never be read.**

Parameters marked with * are linked to filter inputs (they are loaded when opening the editor and written back when accepting changes). They are also passed to the code reader inside auto-tune (which is essentially another [ReadDataCodes](#) filter).

Analysing Filter Performance

The Program Statistics window contains information about the time profile of the selected macrofilter. This is very important as real vision algorithms need to be very fast. However, before starting [program optimization](#) we must know what needs to be optimized, so that later we optimize the right thing. Here is how the required information is presented:



As can be seen on the above illustration, program execution time is affected by several different factors. The most important is the "Native Filters" time, which corresponds to the core data processing tools. Additional time is consumed by "Data Transfers", which is related to everything that happens on connections between filters – this encompasses automatic conversions as well as packing and unpacking arrays on array and singleton connections. Another statistic called "Other" is related to the virtual machine that executes the program. If its value is significant, then [C++ code generation](#) might be worth considering. The last element, "GUI", corresponds to visualization of data and program execution progress. You can expect that this part is related only to the development environment and can possibly be reduced down to zero in the runtime environment.

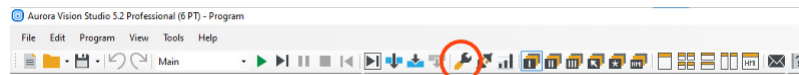
Remarks:

- In practice, performance statistics may vary significantly in consecutive program executions. It is advisable to run the program several times and check if the statistics are coherent. It might also be useful to add the [EnumerateIntegers](#) filter to your program to force a loop and collect performance statistics not from one, but from many program iterations.
- Turn off the [diagnostic mode](#) when testing performance.
- Data Preview Panels, animations in the Program Editor and even the Console Window can affect performance in Aurora Vision Studio. Choose *Program » Previews Update Mode » Disable Visualization* to test performance with minimal influence of the graphical environment. (Do not be surprised however that nothing is visible then).
- Even with all windows closed there are some background threads that affect performance. Performance may still be higher when you run the program with the Executor (runtime) application.
- Please note that the first program iteration might be slower. This is due to the fact that in the first iteration memory buffers are allocated, filters are initialized, communication with external devices is established etc.

See also: [Optimizing Image Analysis for Speed](#).

Seeing More in the Diagnostic Mode

Programs can be executed in two different modes: *Diagnostic* and *Non-Diagnostic*. The difference between them is in the computation of values on the diagnostic outputs. Values of this kind of outputs are computed only in the *Diagnostic* mode. They can be helpful in debugging programs but are not necessary in its final version. In the *Non-Diagnostic* mode, execution is faster because no diagnostic values are computed.



The *Diagnostic Mode* switch.

The execution mode can be easily changed in Aurora Vision Studio using a button on the Application Toolbar. Outside of Aurora Vision Studio, programs are always executed in the *Non-Diagnostic* mode to provide the highest performance.

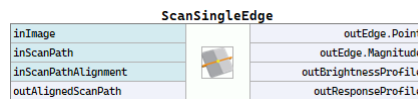
Diagnostic Filter Instances

Filters that have inputs connected to diagnostic outputs of some filters above them, are said to be diagnostic too. They are executed only when the program runs in the *Diagnostic* mode.

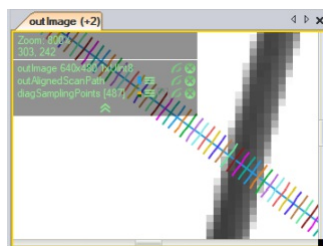
Example

The [ScanSingleEdge](#) filter has three diagnostic outputs:

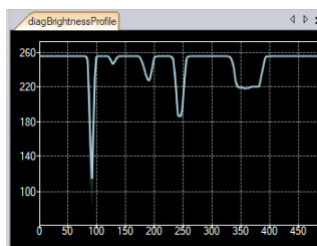
- **diagBrightnessProfile** is the profile of image brightness sampled along the scan path.
- **diagResponseProfile** is the profile derivative after preprocessing.
- **diagSamplingPoints** visualizes the points on the input image from where the brightness samples were taken.



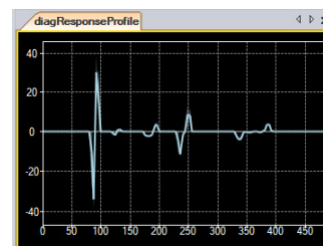
The *ScanSingleEdge* filter.



Input image with the scan path and the diagnostic sampling points.



The brightness profile.



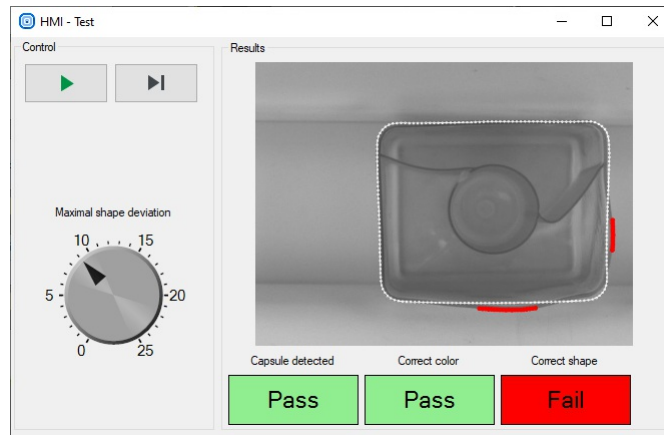
The response profile.

Deploying Programs with the Runtime Application

Introduction

Aurora Vision Executor is a lightweight application that can run programs created with Aurora Vision Studio. The GUI controls that appear in this application are the ones that have been created with the [HMI Designer](#). The end user can manipulate the controls to adjust parameters and can see the results, but he is not able to change the project.

Aurora Vision Executor application is installed with the Aurora Vision Studio Runtime package. It can be used on computers without the full development license. Only a runtime license is required. What is more, programs executed in Aurora Vision Executor usually run significantly faster, because there is no overhead of the advanced program control and visualization features of the graphical environment of Aurora Vision Studio.



The screen of Aurora Vision Executor.

Usage

Open a project from a file and use standard buttons to control the program execution. A file can also be started using the Windows Explorer context menu command *Run*, which is default for computers with Aurora Vision Studio Runtime and no Aurora Vision Studio installed.

Please note, that Aurora Vision Executor can only run projects created in exactly the same version of Aurora Vision Studio. This limitation is introduced on purpose – little changes between versions of Studio may affect program compatibility. After any upgrade your application should first be loaded and re-saved with Aurora Vision Studio as it then runs some backward compatibility checks and adjustments that are not available in Executor.

Console mode

It is possible to run Aurora Vision Executor in the console mode. To do so, the `--console` argument is needed to be passed. Note, that this mode makes the `--program` argument required so the application will know which program to run at startup.

Aurora Vision Executor is able to open a [named pipe](#) where its log will be written into. This is possible with `--log-pipe` argument which accepts a pipe name to be opened. One may then connect to the pipe and process Aurora Vision Executor log live. This can be easily done e.g. in C#:

```
var logPipe = new NamedPipeClientStream(".", "myProjectPipe", PipeDirection.In);
logPipe.Connect();

byte[] buffer = new byte[1024];
int count = 0;
while (logPipe.IsConnected && (count = logPipe.Read(buffer, 0, 1024)) > 0)
{
    Console.WriteLine(Encoding.UTF8.GetString(buffer, 0, count));
}
```

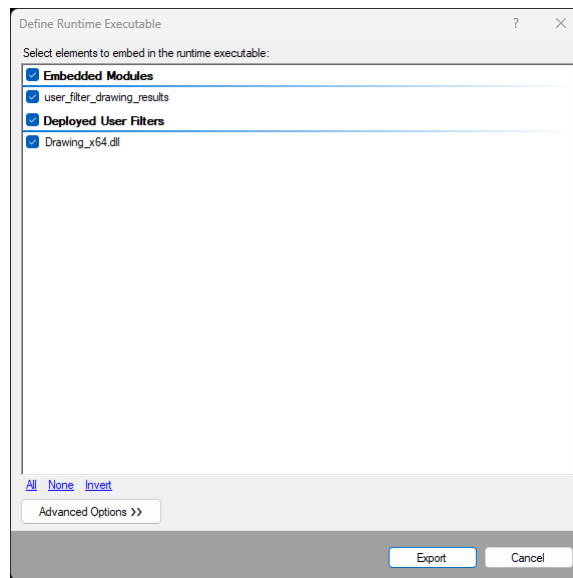
Available Aurora Vision Executor arguments are as follows:

- `--program`
 - Path to the program to be loaded
- `--log-level`
 - Sets the logged information level
- `--console`
 - Runs the application in the console mode
- `--auto-close`
 - Automatically closes the application when program is finished. Meaningful only in console mode.
- `--language`
 - Specifies the language code to use as the user interface language.
- `--attach`
 - Attaches application process to the calling process console.
- `--log-pipe`
 - Creates a named pipe which will be populated with log entries during application lifetime. Meaningful only in console mode.
- `--help`
 - Displays help

Runtime Executables

Aurora Vision Executor can open `.avproj` files, the same as Aurora Vision Studio, however it is better to use `.avexe` files here. Firstly one can have a single binary executable file for the runtime environment. Secondly this file is encrypted so that nobody is able to look at project details. To create one, open a project in Aurora Vision Studio and use *File » Export to Runtime Executable...* This will produce an `.avexe` file that can be executed directly from the Windows Explorer.

If Aurora Vision project contains any User Filter libraries, it is crucial to put their `*.dll` files into the appropriate directory when running in Aurora Vision Executor. This is when exporting to `.avexe` file might also be a handy option. While defining the `.avexe` contents, it is possible to select all the User Filters libraries that the exported project depends on. Selected libraries are deployed then to the same directory as generated `.avexe` file and the `.avexe` itself is set to use all User Filter libraries from its directory.



Defining the Runtime Executable.

In case there are any other dependencies, e.g. exposed by used User Filter libraries, one can add them into the Aurora Vision project as an attachment in [Project Explorer](#) and also deploy with .avexe file during export.

Using a Project ID to protect an application

The **Project ID** is an additional security parameter available in *Advanced Options* that allows a specific Runtime Executable to be tied to a designated Runtime license. At the customer's request, the Aurora Vision Team can generate a unique Project ID key and embed it into the license. The same Project ID must then be entered during the Runtime Executable export process to connect the application with that specific license.

If the Project ID field is empty, any valid Aurora Vision Runtime license will be able to run the .avexe file.

Obtaining a license with a Project ID requires contacting the Aurora Vision Team. The process is as follows:

1. Request the Project ID:
 - For an existing license: The [license file](#) should be sent to the support team to be updated.
 - For a new license: A comment should be added to the order requesting that a Project ID be included.
2. Receive the Project ID: The Aurora Vision Team will generate the Project ID and send the customer the ID number, along with either the updated license file or a new USB key with the Project ID embedded.
3. Use the Project ID: When exporting the application to a Runtime Executable, the provided Project ID is entered in the *Advanced Options* menu.

Important Considerations:

- The Project ID is assigned to the Runtime Executable version of the application, not the corresponding project files (.avproj).
- The .avexe file with an assigned Project ID automatically checks for license validation upon initialization.
- While a development license can open any application project, it cannot run a Runtime Executable that has been protected with a Project ID. Attempting to do so will result in a license error message.

Trick: Configuration File as a Module Not Exported to .avexe

It is often convenient to have an configuration file separated from the executable so that various parameters can be adjusted for a particular installation (but not made available to the end user). This can be easily implemented with a simple programming idiom:

1. Use [global parameters](#) for values that might require adjusting.
2. Place the global parameters in a separate module (through the Project Explorer window).
3. Exclude the module when exporting the .avexe file.
4. In the runtime environment copy the .avexe file and the module (as a separate file) with global parameters.
5. Open the config module in the Notepad to edit it when needed.

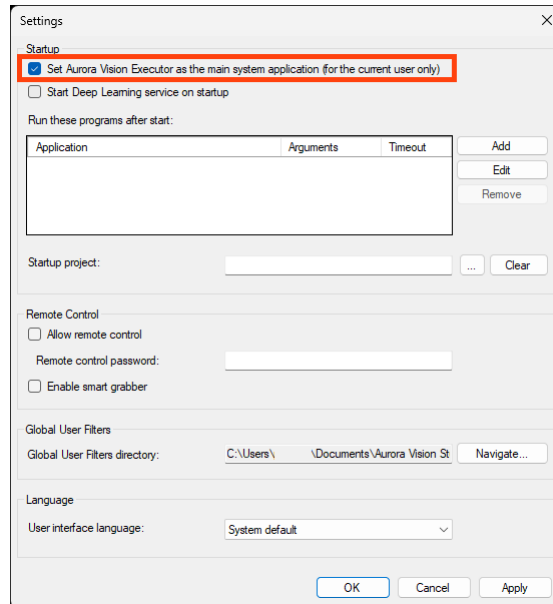
Other Runtime Options

Please note that Aurora Vision Executor is only one of several options for creating end-user's applications. Other available options are:

- [.NET Macrofilter Interface Generator](#) – generates a native .NET assembly (a DLL file) and makes it possible to invoke macrofilters created in Aurora Vision Studio as simple class methods from a .NET project. Internally the execution engine of Aurora Vision Studio is used, so modifying the related macrofilters does not require to re-compile the .NET solution. The HMI can be implemented with WinForms, WPF or similar technologies.
- [C++ Code Generator](#) – generates native C++ code (a CPP file) that is based on Aurora Vision Library C++. This code can be integrated with bigger C++ projects and the HMI can be implemented with Qt, MFC or similar libraries. Each time you modify the program in Studio, the C++ code has to be re-generated and re-compiled.
- Aurora Vision Console – allows to easily run created Studio applications (.avproj / .avexe files) on Windows or Linux based systems. Such programs can be controlled through created WebHMI panel.

Set Aurora Vision Executor as the "system shell"

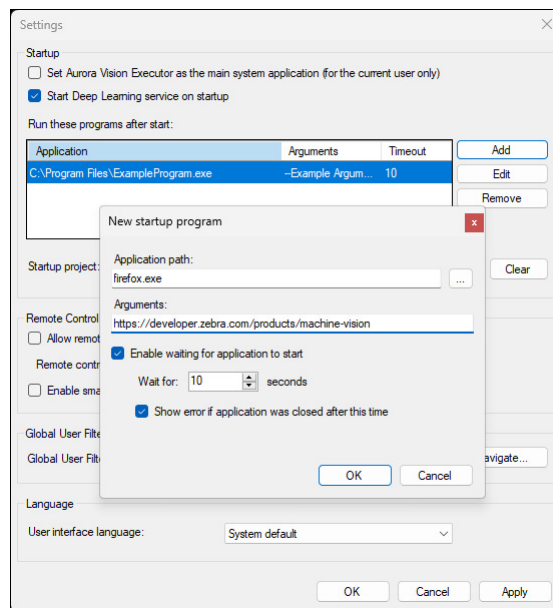
On Windows systems it is possible to set Aurora Vision Executor as the "system shell", thus removing Desktop, Menu Start etc. completely. To do this go to Settings in Aurora Vision Executor and the Startup section. Mark the *Set Aurora Vision Executor as the main system application (for the current user only)*. Please be informed that this option requires administrator privileges.



Set Aurora Vision Executor as the "system shell"

Startup Applications

It is possible to run any process before starting a program in Aurora Vision Executor. Go to Settings in Aurora Vision Executor and the Startup section. To define a new startup program select the *Add* button on the right. In a *New startup program* dialog box you need to specify the application path (obligatory) and arguments (optional). It is similar to typing the application name and command-line arguments in the *Run* dialog box of the Windows Start menu. The added program will appear in the list. All added programs will start each time you run Aurora Vision Executor.



Defining Startup Applications

Startup Project

It is possible to choose the project the Aurora Vision Executor should run after the startup. Go to Settings in Aurora Vision Executor and the Startup section. To define the startup project select the ... button on the right. In an *Open* dialog box you need to specify the startup project path (obligatory). The added project's path will appear in the box. It will start each time you run Aurora Vision Executor.

Settings

Startup

☐ Set Aurora Vision Executor as the main system application (for the current user only)

☐ Start Deep Learning service on startup

Run these programs after start:

Application	Arguments	Timeout

Add

Edit

Remove

Startup project: ... Clear

Remote Control

☐ Allow remote control

Remote control password:

☐ Enable smart grabber

Global User Filters

Global User Filters directory: C:\Users\...Documents\Aurora Vision St... Navigate...

Language

User interface language: System default

OK Cancel Apply

Defining Startup Project

If Deep Learning Service is installed you can choose to run it on start by selecting 'Start Deep Learning service on startup'.

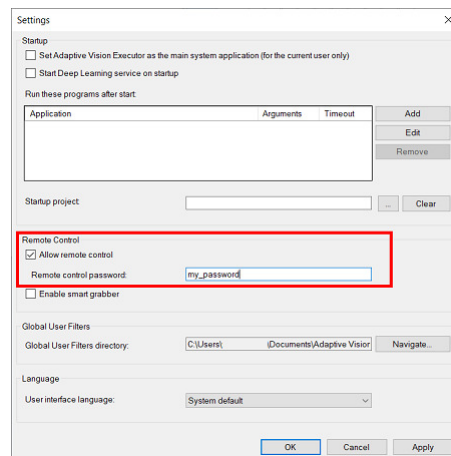
Runtime Executable Log Files

Runtime executables keep separate logs from Aurora Vision Studio, located in the C:\Users\%USERNAME%\AppData\Local\Aurora Vision\Aurora Vision Studio 5.7 Runtime\LOGS directory. These logs contain all the messages that appear in the runtime application console and are often vital for program debugging and catching errors. Also, please note, that even if the runtime console is disabled in the AVExecutor Behaviour section of the HMICanvas parameters, and thus invisible to the user, its log files are still being created and gathered in the above mentioned directory.

Remote Access to the Runtime Application

Introduction

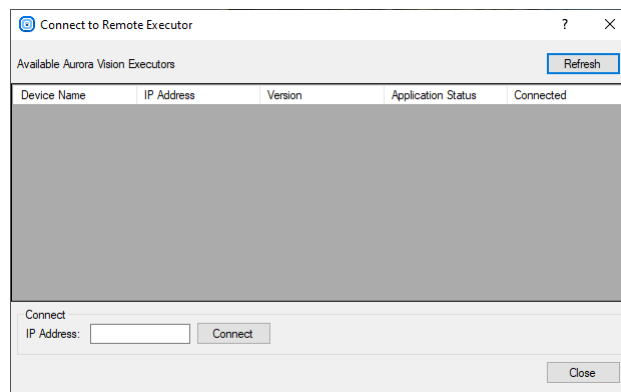
It is possible to use Aurora Vision Studio to control an Aurora Vision Executor running on a remote computer, if only the two computers are in the same Ethernet subnet. To enable remote access in Aurora Vision Executor, the option "Allow remote control" should be enabled. For security reasons, it is also possible to protect connection with a password:



Enable remote executor.

Usage

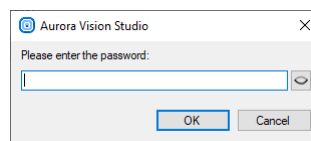
In Aurora Vision Studio the list of available remote systems is accessible in the Connect to Remote Executor window which opens up after choosing the File > Connect to Remote Executor menu command:



Browse remote systems.

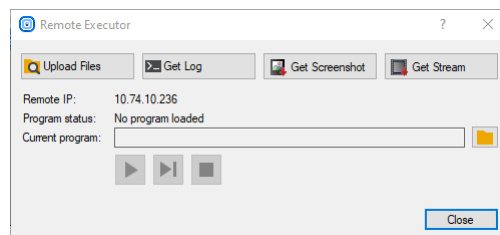
If an expected remote system is not visible on the list, please verify that: (1) it is configured for the same local area network, and (2) it can access Ethernet through its firewall (in the first place verify that in the Windows' Control Panel, Windows Firewall section, the option "Notify me when Firewall blocks a new program" is enabled). In some local networks information about communication type and ports may be useful to assign application access through firewall. In this case please use UDP ports 6342 and 6343 or TCP port 6342.

When connecting to a selected device, the program will ask for the password (if it has been set):



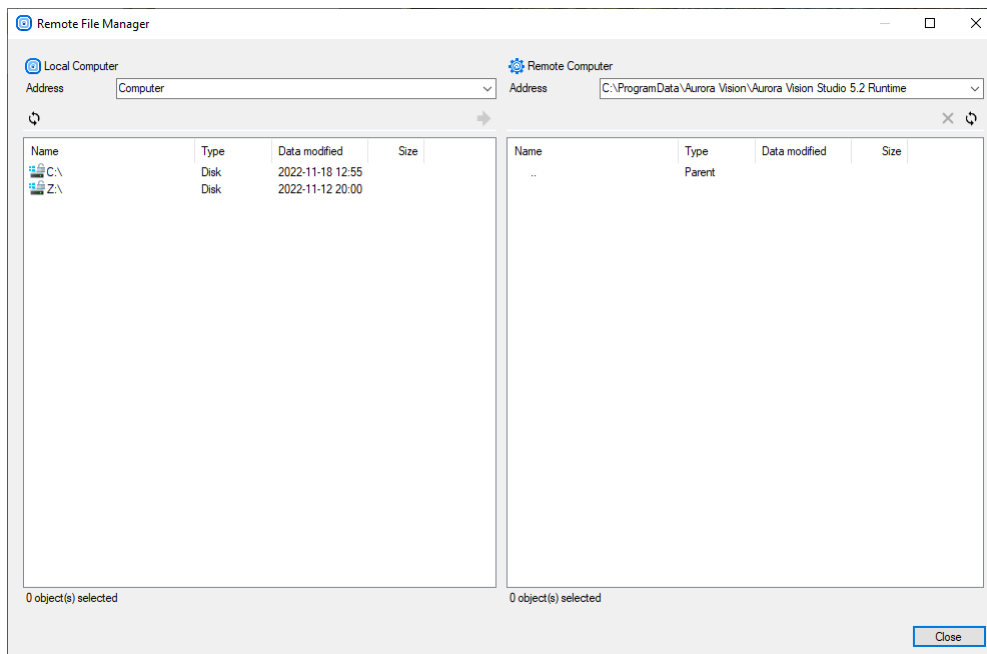
Password protection window.

After successful connection, management of the selected executor becomes possible. Now you are able to do several actions: Upload Files, Control Program and Get Diagnostics.



Remote executor window.

Upload Files:

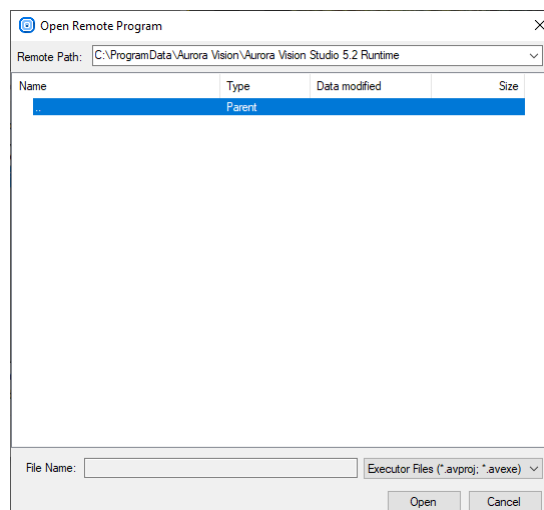


Upload files window.

To Manage Files on remote executor, click Upload Files button. In this window it is possible to send program files to the remote executor. By default all files are stored in the directory <CommonApplicationData>\Aurora Vision\Aurora Vision Executor.

Program Control:

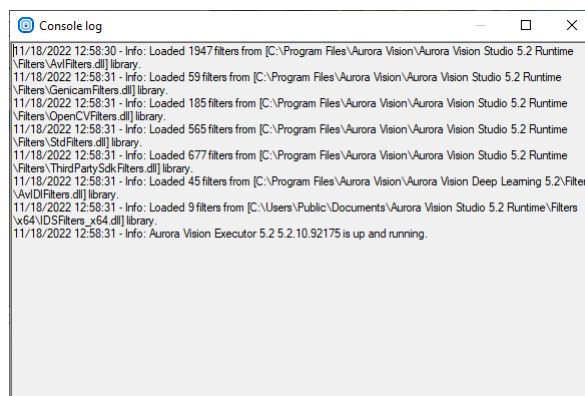
In the Remote executor window you are able to control currently executed program. In the middle of the window we can see the current program status and the path to the currently loaded program. Next there are four buttons controlling program execution. Finally there is an option to open another program.



Remote file dialog window.

Diagnostics:

In the Remote Executor window there is available a few diagnostic tools. Here, it is possible to preview the execution log and a screen preview from the currently connected executor. All diagnostic options are located at the top of the Remote executor window.



Log preview window.

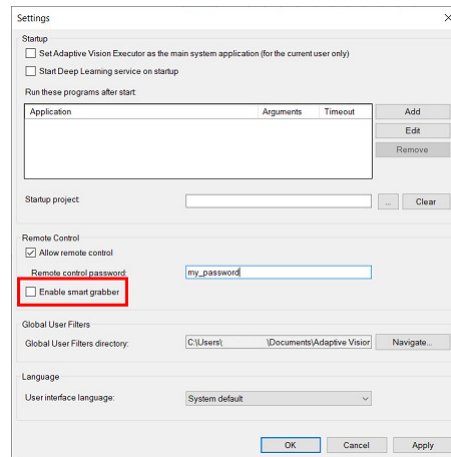
Remote Image Acquisition

Introduction

It is also possible to get images from a connected device using the remote executor. This method offers an easy way to acquire images using a different image protocols. Remote image acquisition filters enable to run this same code on the smart cameras (client side) as good as on developer computers without any additional modifications.

Usage

To enable remote image acquisition it is necessary to stop the executor program and enable the "Enable smart grabber" option:



Enable Remote Image Acquisition.

In Aurora Vision Studio it is possible to use the filters from the category [Camera Support\Smart](#) to get images from a device connected to the remote Executor system.

Currently the available image acquisition protocols are:

- **AvSMART** – access to AvSMART,
- **GenICam** – access to GenICam complaint devices,
- **Roseek** – access to Roseek cameras,
- **SynView**– access to NET GmbH cameras,
- **WebCamera**– access to DirectX complaint image sources like web cameras or frame grabbers.

To grab images using one of these protocols it is necessary to set **inIpAddress** in the filter input. The IP Address can be checked in the Connect to Remote Executor window when "Allow remote control" option is enabled in the Executor.

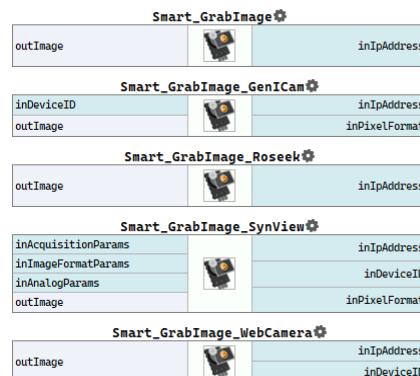
Properties - (0)Smart_GrabImage_SynView		
Name	Value	
Filter	SynView	
inIpAddress	10.74.15.227	
inDeviceID	Auto	
inPixelFormat	Mono8	
inAcquisitionParams	{Auto;Auto;Auto;...}	
inImageFormatPara...	{Auto;Auto;Auto;...}	
inAnalogParams	{Auto;Auto;Auto}	

Smart_GrabImage_SynView filter properties.

If system detect that the provided IP address describes local machine the **Smart_GrabImage** filters will perform all operation on local computer. So it is no need to perform any changes during transferring project from developers machine to the client side device.

If the program is deployed to a device with another IP address then the input **inIpAddress** should be changed to the IP of the Executor or left empty. If the input **inIpAddress** is empty then the image is always grabbed using an appropriate local image acquisition protocol.

Available filters for the remote image acquisition:



If **inDeviceID** is set to *NULL* the first found device will be used.

Performing General Calculations

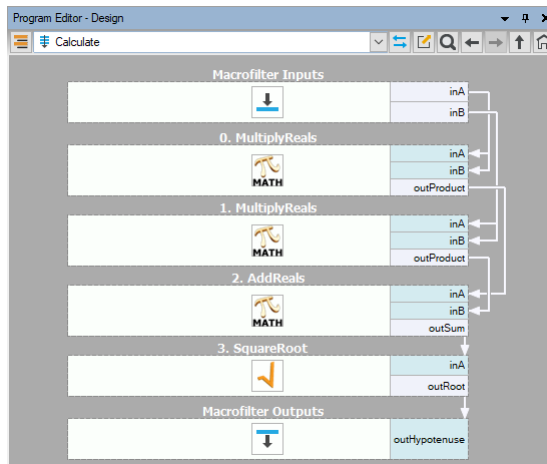
Introduction

Apart from using image processing or computer vision tools, most often it is also necessary to perform some general calculations on numeric values or geometrical coordinates. There are two ways to do this in Aurora Vision Studio:

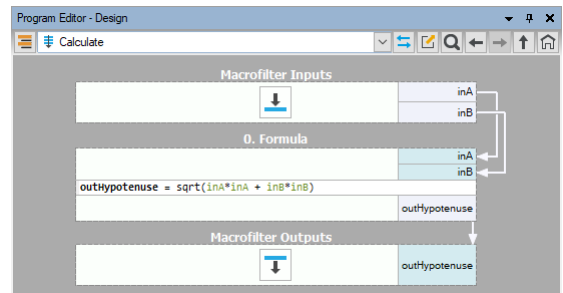
1. Using filters from the Standard Library.
2. Using special [formula blocks](#).

Example

Let us assume that we need to compute the hypotenuse $\sqrt{a^2 + b^2}$. Here are the two possible solutions:



Calculations using standard filters.



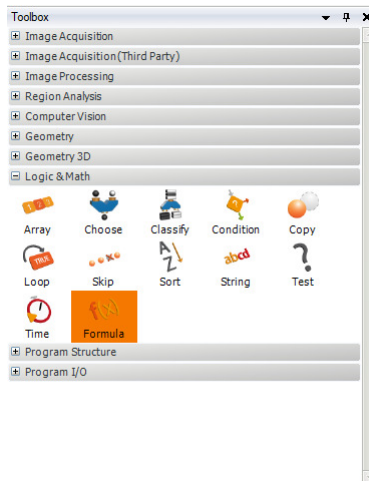
Calculations using formula blocks.

The second approach, using formula blocks, is the most recommended. Data flow programming is just not well suited for numerical calculations and standard formulas, which can be defined directly in formula blocks, are much easier to read and understand. You can also think of this feature as an ability to embed little spreadsheets into your machine vision algorithms.

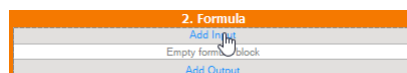
Creating Formula Blocks

To create a formula block:

1. Drag and drop the [Formula](#) tool from the Toolbox (*Logic & Math* section) to the Program Editor:

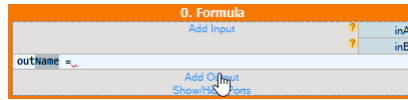


2. Add inputs and outputs using the context menu or by clicking *Add Input* and *Add Output* links:

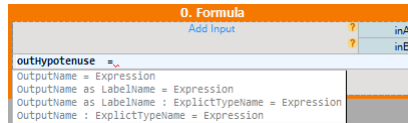


3. For each input define the name and the type:

4. In order to create an output, first press **Add Output** button. It will create a new output with template name: *outName*:

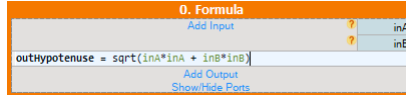


Replace the highlighted *Name* with meaningful name. While typing a Tooltip with available options should appear:

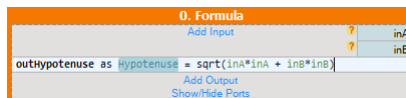


Example usage of each option:

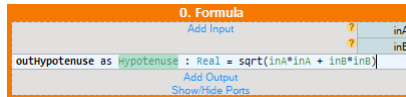
- After the name type equal sign = and the expression. There is no need to define a type of output. The type will be assigned automatically.



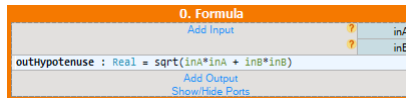
- After the name add *as* followed by *LabelName*. This label will later appear as a formula output.



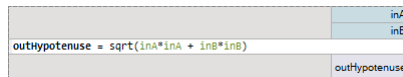
- After the name add *label* as described in the previous point. Then add colon : and directly specify the output type:



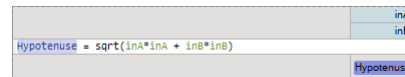
- In the last option output type is defined directly. After the name add colon : and directly specify the output type:



Final formula should look like this:

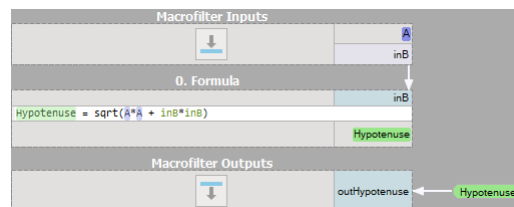


Without output label.



With output label.

Not only the output data of the formulas can be labeled but also inputs and outputs of other filters outside of the formula. This way you can perform a direct call to another labeled data inside a formula without explicitly defining new formula inputs. In the image below the input *A* of the macrofilter is labeled (highlighted in violet color) and a direct call of this label (*A*) is performed inside the formula block. Later the output *Hypotenuse* is connected via a label with the macrofilter output, instead of linking it directly using an outgoing arrow connection.



Remarks

- Existing formulas can also be edited directly in a formula block in the Program Editor.
- It is also possible to create new inputs and outputs by dragging and dropping connections onto a formula block.
- Formula blocks containing incorrect formulas are marked with red background. Programs containing such filters cannot be run.
- When defining a formula for an output it is possible to use other outputs, provided that they are defined earlier. The order can be changed through the outputs context menu.
- For efficiency reasons it is advisable not to use "heavy" objects in formulas, such as images or regions.

Syntax and Semantics

For complete information about the syntax and semantics please refer to the [Formulas](#) article in the Programming Reference.

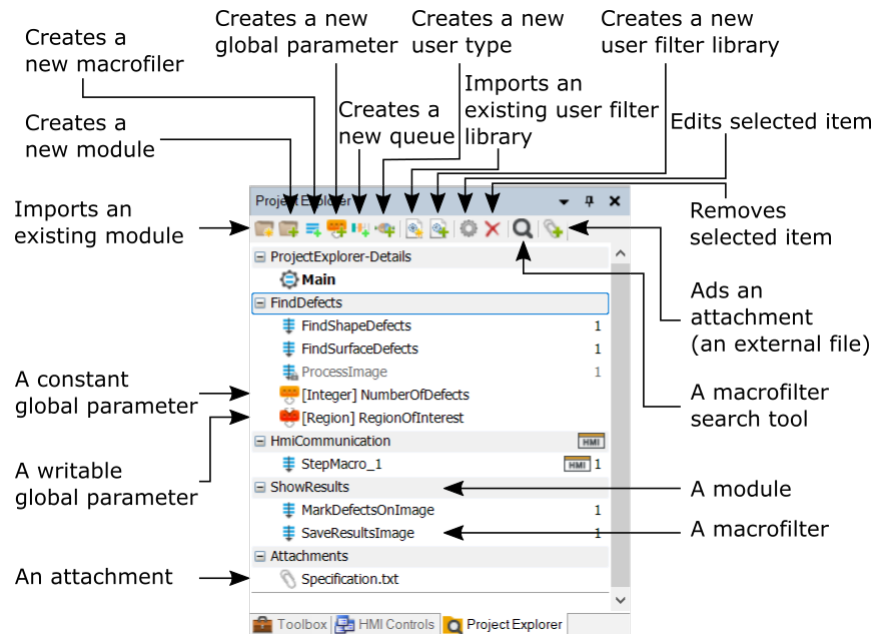
Managing Projects with Project Explorer

Introduction

Project Explorer is a window displaying elements which are contained in the currently opened project:

- Modules
 - Macrofilters (definitions)
 - Global Parameters
 - User Types
- User Filter libraries
- Attachments

Its main goal is to provide a single place to browse, add, remove, rename or open the items, which are grouped into categories in the same way as the filters in the Filter Catalog. A category may correspond to a standard module or to a module of a User Filter library. There is also one special category, Attachments, which appears when the user adds an external file to the project (it might be for example a text document with a piece of documentation).



Modules in the Project Explorer.

Opening Macrofilters

As described in [Running and Analysing Programs](#), there are two ways of navigating through the existing [macrofilters](#). One of them is with the Project Explorer window, which displays *definitions* of macrofilters, not the instances. After double-clicking on a macrofilter in the Project Explorer, however, a macrofilter instance is opened in the Program Editor. As one macrofilter definition can have zero, one or many instances. Some special rules apply to which of the instances it is:

- If possible, the most recently executed instance is opened.
- If no instance has been executed yet, the most recently created one is opened.
- If there are no instances at all the "ghost instance" is presented, which allows editing the macrofilter, but will never have any data on the output ports.

Macrofilter Counter

Macrofilter counter shows how many times a given macrofilter is used in the program.

ADVANCED NOTE: If a macrofilter X is used in a macrofilter Y and there are multiple instances of the macrofilter Y, we still consider macrofilter X being used once. Number of uses is not the same as number of instances.

Global Parameters

If some value is used many times in several different places of a program then it should be turned into a global parameter. Otherwise, consecutive changes to the value will require the error-prone manual process of finding and changing all the occurrences. It is also advisable to use global parameters to clearly distinguish the most important values from the project specification – for example the expected dimensions and tolerances. This will make a program much easier to maintain in future.

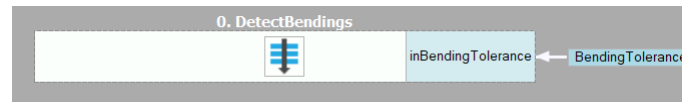
In Aurora Vision Studio global parameters belong to specific modules and are managed in the Project Explorer. To create one, click the *Create New Global Parameter...* button and then a dialog box will appear where you will provide the name, the type and the value of the new item. After a global parameter is created it can be dragged-and-dropped on filter inputs and appropriate connections will be created with a visual label displaying the name of the parameter.

The 'Creating Global Parameter' dialog box contains the following fields:

- Name:** BendingTolerance
- Type:** Real
- Module:** HandleInspection
- Access:** Public
- Initial Value:** 26.500

Buttons: OK, Cancel

Creating a global parameter.



Global parameter used in a program.

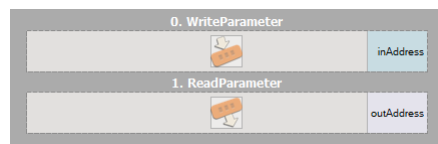
Global parameters contained in a project can also be edited in the Properties window after being selected in the Project Explorer or in the Program Editor.

Since version 4.11 it is possible to create and manage global parameters with dedicated filters: WriteParameter and ReadParameter. They are available in the Program Structure category of the Toolbox.

While adding a new input, "Select Global Parameter" window is displayed:

The 'Select Global Parameter' dialog box contains a dropdown menu labeled 'Global Parameter:' and buttons 'OK' and 'Cancel'.

You can select an already created global parameter or create a new one.



Global parameter "inAddress" created within the filter WriteParameter can be read with the ReadParameter filter.

Thanks to these filters you can easily read or write the values of global parameters anywhere in your algorithm. In order to facilitate development the icon of the global parameter has different appearance depending on whether it is overwritten somewhere in the program. The color of the icon will be red then so that you will know that this value may change during the execution of your application.

To see how Global Parameters work in practice, check out our official example: [HMI Handling Events](#).

Remarks:

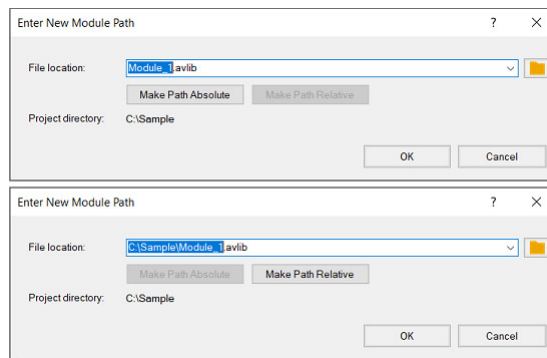
- Connected filters are not re-executed after the global parameter is changed. This is due to the fact, that many filters in different parts of the program can be connected to one global parameters. Re-executing all of them could cause unexpected non-local program state changes and thus is forbidden.
- Do NOT use writable global parameters unless you really must. In most cases data should be passed between filters with explicit connections, even if there are a lot of them. Writable global parameters should be used only for some very specific tasks, most notably for configuration parameters that may be dynamically loaded during program execution and for high level program statistics that may be manipulated through the HMI (like the last defect time).

Modules

When a project grows above 10-20 [macrofilters](#) it might be appropriate to divide it into several separate modules, each of which would correspond to some logical part. It is advisable to create separate modules for things like i/o communication, configuration management or for automated unit testing. Macrofilters and global variables will be then grouped in a logical way and it will be easier to browse them.

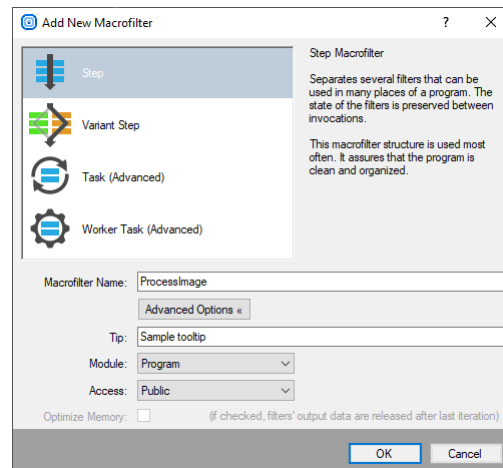
Modules are also sometimes called "libraries of macrofilters". This is because they provide a means to develop sets of common user's tools that can be used in many different projects. This might be very handy for users who specialize in specific market areas and who find some standard tasks appearing again and again.

To create a module, click the *Create New module...* button and then a dialog box will appear. In there you specify the location and name of the module. The path may be absolute or relative. Modules are saved with extension **.avlib**. Saving and updating the module files happens when the containing program is saved.

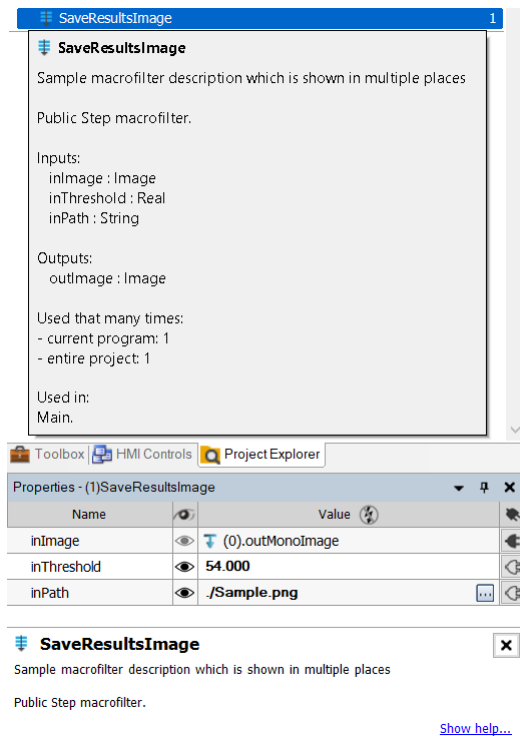


Module creation windows for both absolute and relative paths.

After creating a module you may move already existing macrofilters into the module by dragging them onto the module in Project Explorer. Clicking the *Create New Macrofilter...* button allows you to create a new macrofilter with the given name. During that you can access *Advanced options* by clicking the appropriate button. There you can specify the parent module and access to the macrofilter. Access can be either *private* or *public*. Private macrofilters cannot be directly used outside its module. You can also provide a tooltip (description) for the macrofilter. This will show up in the properties section of an instance and when hovered over in the Project Explorer. Tooltip should describe the macrofilter's purpose and be concise. They are especially important when creating an

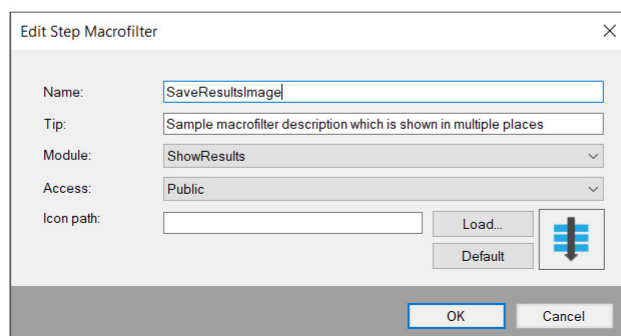
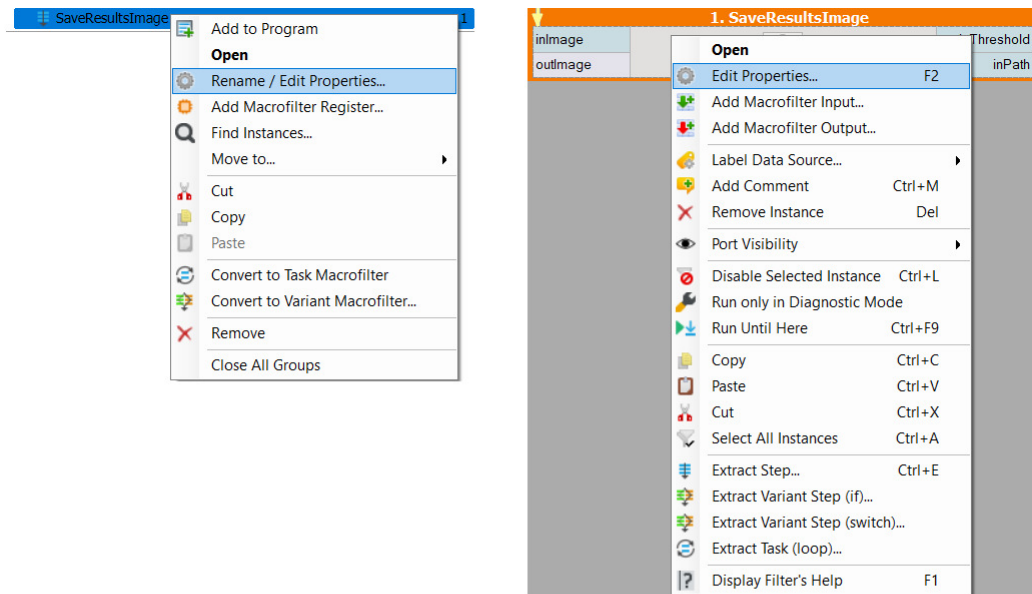


Macrofilter creation with advanced options.



View showing a sample tooltip.

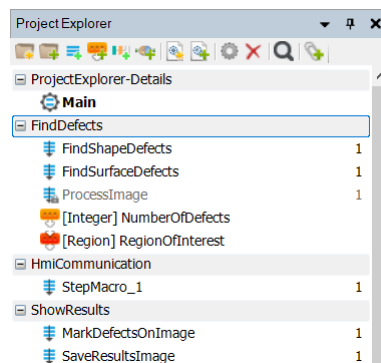
Right-clicking a macrofilter and selecting *Edit properties...* allows modifying its properties, including access, after creation.



Ways to access the editing window. The option to change the icon is highlighted

The image below shows a structure of an example program. The macrofilters have been grouped into two Modules. Module *FindDefects* has macrofilters related to defect detection and a global parameter used by the macrofilters. Notice how the *ProcessImage* macrofilter is grayed out. It indicates it is a *private* macrofilter (here it is used by *FindShapeDefects*). *ProcessImage* cannot be used outside its module.

The other module has macrofilters related to showing and storing the results of the inspection.



Example program structure with macrofilters grouped into modules.

Here are some guidelines on how to use modules in such situations:

- Create a separate module for each set of related, standard macrofilters.
- Give each module a unique and clear name.
- Use the English language and follow the same naming conventions as in the native filters.
- Create the common macrofilters in such a way, that they do not have to be modified between different projects and only the values of their parameters have to be adapted.
- If some of the macrofilters are intended as implementation only (not to be used from other modules), mark them as *private*.

It is important to note that modules containing filters interfacing with the HMI should not be shared between programs. Every filter port connected to the HMI has a unique identifier. Those identifiers vary between programs - ports of the same filter in different programs will have different identifiers.

Generally it is a good practice to create a separate module for all things related to the HMI. That way every other module can be shared between programs without any problems.

Importing Modules

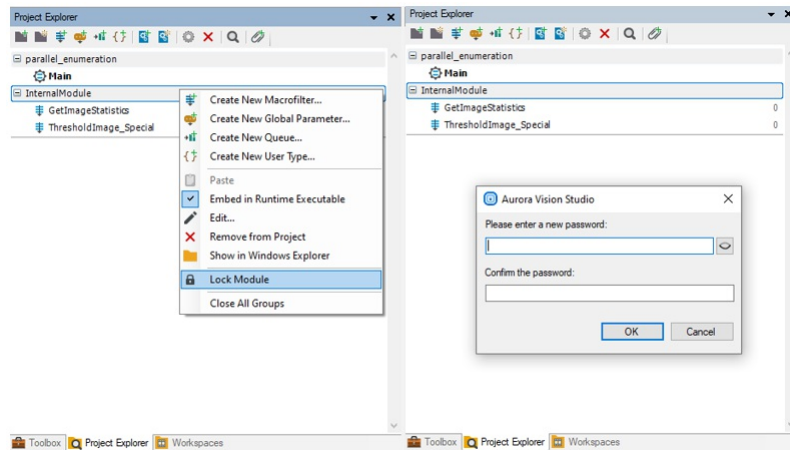
If you want to use a module which had been created before click the *Import Existing Module...* icon. This will open a window in which you can select modules to add. Now the path to the module will be linked to the project. Similarly to creating a new module you can choose whether the path to it will be relative or absolute.

Remember that modules are separate files and as such can be modified externally. This is especially important with modules which are shared between multiple projects at the same time.

See also: [Trick: INI File as a Module Not Exported to AVEXE.](#)

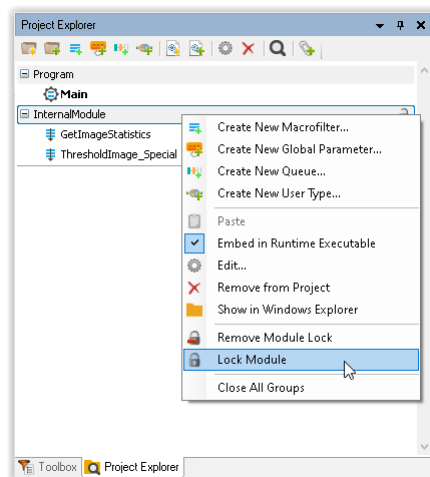
Locking Modules

Sometimes users would like to hide the contents of some of their macrofilters to protect them against unauthorized access. They can do this by placing them inside a module and locking it. To do this, right-click the module in the Project Explorer and select the Lock Module option. The user will then be prompted to provide a password for this specific module.

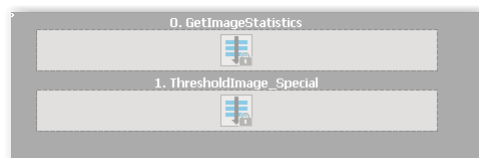


Adding lock to the module.

After creating a password, the user should see an open lock icon next to the module name in the Project Explorer. You can work with this module exactly the same way as with any other module. What distinguishes it from other modules is that it can be locked, which will make checking implementation of macrofilters included inside not possible. Such modules will be similar to filters in this matter. This operation will also obfuscate the avlib file.



Locking module in project explorer.



Macrofilters from locked modules in the program.

It is important to lock the module every time you finish working on it. In order to unlock a module, you need to select the Unlock Module option in the Project Explorer and enter the correct password. To completely remove module locking feature from a selected module, use the Remove Module Lock option.

Keyboard Shortcuts

Introduction

Many of Aurora Vision Studio actions can be invoked with keyboard shortcuts. Most of them have default shortcuts – such as copying with `Ctrl+C`, pasting with `Ctrl+V`, finding elements with `Ctrl+F`, saving with `Ctrl+S`, navigating with arrow keys etc.

In some controls – e.g. formula editor – there are present commonly used keyboard shortcuts for text editors: navigating through whole words with `Ctrl+Left/Right Arrow`, selecting consecutive letters/lines (`Shift+Arrow keys`), selecting whole words (`Ctrl+Shift+Left/Right Arrow`), increasing indents with `Tab`, decreasing them with `Shift+Tab` etc.

In this article you can find listed all of the less-known shortcuts.

Table of Contents

- [1. Program Editor](#)
- [2. HMI Designer](#)
- [3. Properties Control](#)
- [4. 3D View](#)
- [5. Deep Learning Editors](#)

Shortcuts Table

Command	Shortcut
Program Editor Back to Top	
Run program	<code>F5</code>
Run program with Aurora Vision Executor	<code>Ctrl+F5</code>
Run program until selected point	<code>Ctrl+F9</code>
Stop program	<code>Shift+F5</code>
Pause program at current executing step	<code>Ctrl+Alt+Pause</code>
Iterate program	<code>F6</code>
Iterate current macrofilter	<code>Ctrl+F10</code>
Step over	<code>F10</code>
Step into	<code>F11</code>
Step out	<code>Shift+F11</code>
Insert new filter instance	<code>Ctrl+T</code> <code>Ctrl+Space</code>
Load recent project	<code>Alt+Number</code>
Rename currently open macrofilter	<code>F2</code>
View program statistics	<code>F8</code>
Toggles breakpoint for currently selected filter and macrofilter outputs block	<code>F9</code>
Copy element	<code>Ctrl+Insert</code> <code>Ctrl+C</code>
Paste element	<code>Shift+Insert</code> <code>Ctrl+V</code>
Change currently selected macrofilter	<code>Ctrl+Number</code>
Navigate to home macrofilter	<code>Alt+Home</code>
Navigate to parent macrofilter	<code>Shift+Enter</code>
Open next macrofilter in history	<code>Alt+Right Arrow</code>
Open previous macrofilter in history	<code>Alt+Left Arrow</code>
Extract step from selected filter/filters	<code>Ctrl+E</code>
Add new Formula instance to current program	<code>Ctrl+Shift+E</code>
Add new Comment instance to current program	<code>Ctrl+Shift+K</code>
Create step macrofilter and add instance	<code>Ctrl+Shift+S</code>

Create task macrofilter and add instance	Ctrl+Shift+T
Create variant macrofilter and add instance	Ctrl+Shift+V
Remove currently opened macrofilter with all its instances	Ctrl+Shift+R
Undock currently opened macrofilter	Ctrl+U
Enable/Disable selected filter instance/instances	Ctrl+L
Add new/Edit existing filter comment	Ctrl+M
Copy currently moving instance	Hold Ctrl while moving instance
Navigate and select elements instances	Shift+Up/Down/Home/End
Move selected filters	Alt+Up Alt+Down
HMI Designer Back to Top	
Move currently selected control	Arrows
Move currently selected control to the edge of its container	Ctrl+Arrows
Resize currently selected control	Shift+Arrows
Resize currently selected control to the edge of its container	Ctrl+Shift+Arrows
Properties Control Back to Top	
Hide/Show selected property	Ctrl+H
Restore default value of selected property	Ctrl+D
Enable/Disable port in current macrofilter variant	Ctrl+E
Set/Unset optional value	Ctrl+P
Enable/Disable port in current macrofilter variant	Ctrl+E
Change scale on property value slider from 1 to 10	Hold Shift while modifying value with slider
Change scale on property value slider from 1 to 0.1	Hold Ctrl while modifying value with slider
Change scale on property value slider from 1 to 0.01	Hold Ctrl+Shift while modifying value with slider
Insert a new line into the text	Shift+Enter
3D View Back to Top	
Show bounding box	B
Show grid	G
Increase point size	]
Decrease point size	[
Move view up	Q Up Arrow
Move view down	Z Down Arrow
Move view left	A Left Arrow
Move view right	D Up Arrow
Zoom in	W Page Up
Zoom out	S Page Down
Decrease rotation angle	Hold CTRL while rotating view

Deep Learning Editors	
Back to Top	
Change current class	0 - 9
Move one image up	Page Up
Move one image down	Page Down
Save current state of the model	Ctrl+S
Automatic training	Alt+F2
Generate the report	Alt+F3
Deep Learning Editors - Anomalies Detection only	
Back to Top	
Change the current image's annotation	Space
Label image as good and go to the next one	G
Label image as bad and go to the next one	B
Deep Learning Editors - Object Classification only	
Back to Top	
Remove selected ROI	Delete
Deep Learning Editors - Instance Segmentation only	
Back to Top	
Remove single instance	Delete
Add new instance	Space
Split the instances into separate blobs	S
Deep Learning Editors - Point Location only	
Back to Top	
Remove selected point	Delete

Working with 3D data

Introduction

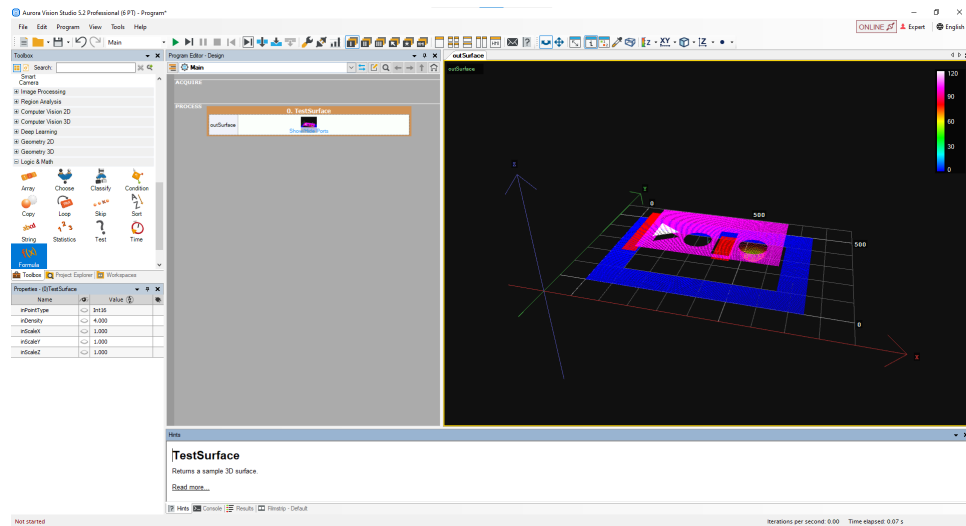
This article summarizes how to work with 3D data previews. Aurora Vision Studio allows you to work not only with images or conventional data types, but also with point clouds of following types:

- [Surface](#)
- [Point3DGrid](#)
- [Point3DArray](#)

Please note that the term *surface* may be used in this article to denote a point cloud of [Surface](#) type.

Toolbar

The toolbar appears the moment a point cloud is previewed:



The location of 3D tools in Aurora Vision Studio.



3D tools available in the toolbar.

Button (Tool)	Description of the tool
Rotate	This button permits to rotate a point cloud around the rotation center point.
Pan	This button permits to move the rotation center point.
Resetting view	This button permits to revert changes made by rotating or panning a point cloud.
Probe Point Coordinates	This button permits to obtain information about coordinates of a point.
Bounding Boxes	This button permits to display the bounding box of a point cloud.
Coloring Mode	This button permits to switch between different coloring modes which are divided into 4 categories: • Solid (whole point cloud is of a single color) • Along X-axis (greater values along the X-axis are colored according to the colors' scale bar in the top right corner of the preview) • Along Y-axis (greater values along the Y-axis are colored according to the colors' scale bar in the top right corner of the preview) • Along Z-axis (greater values along the Z-axis are colored according to the colors' scale bar in the top right corner of the preview)
Grid Mode	This button permits to display a grid plane to your liking. There are also 4 modes available: • None (no grid) • XY (the grid is displayed in XY plane) • XZ (the grid is displayed in XZ plane) • YZ (the grid is displayed in YZ plane)
View Projection Mode	This button permits to display a point cloud in 4 different modes: • Perspective (default mode) • Up (overhead view – showing a point cloud from above – looking down at XY plane) • Back (view from the back – looking at XZ plane) • Left (side view – looking at YZ plane)
World Orientation	This button permits to display a coordinate system in 4 different modes: • Right Handed Z Up • Right Handed Y Up • Left Handed Z Up • Left Handed Y Up
Point size	This button permits to control the size of a single point within a point cloud. There are 5 possible sizes: • Very small • Small • Medium • Large • Very large You can also change point size by using keys [] and [].

Using a computer mouse

Aurora Vision Studio also allows you to control a 3D preview with your computer mouse. You can perform some of the above features to analyze the point cloud without a necessity to find and switch between buttons in the toolbar.



If you click **Left Mouse Button** on a preview, it works as if you used the Rotate button, so you can rotate a point cloud around a fixed center point until the button is released.



If you click **Right Mouse Button** on a point cloud, a dropdown list with all features described in the previous section appears.



If you use **Mouse Wheel** on a point cloud, you can zoom it in and out.

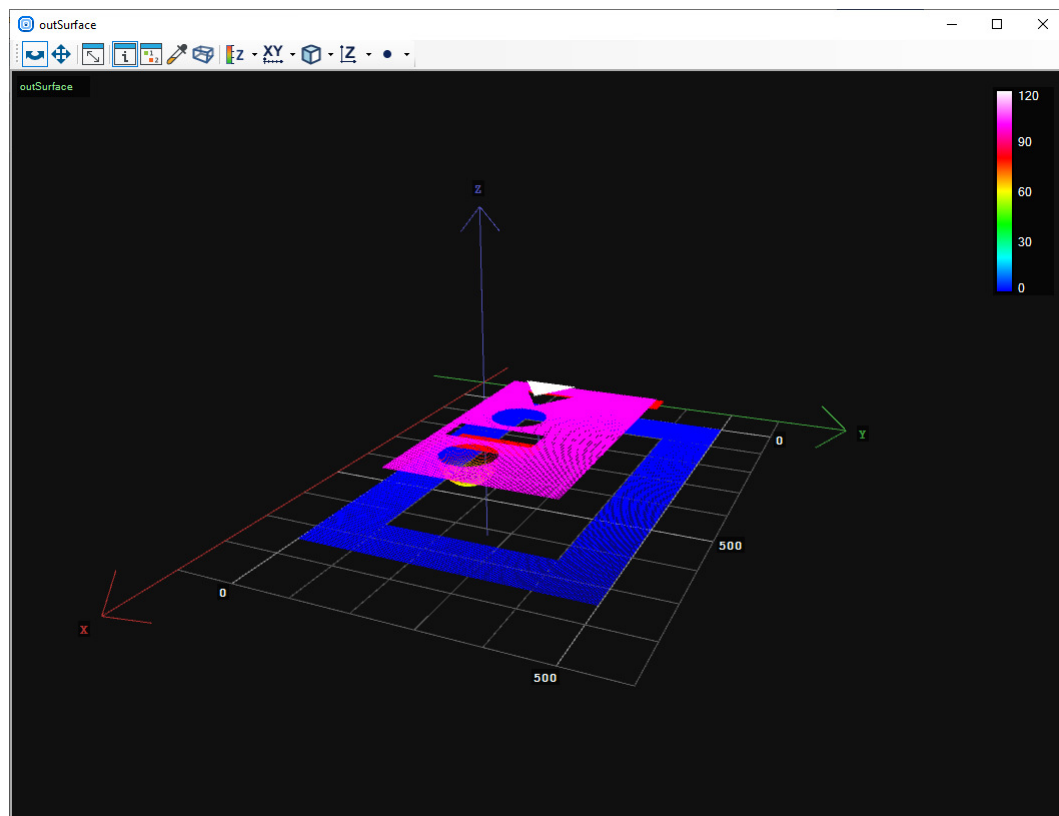


If you press **Mouse Wheel** on a point cloud, you can move the rotation center point of the coordinate system (works like the "Pan" button).

Preview

When you drag and drop the output, which is of 3D data type, and you have already previewed some images, string or real values, please note that it can be only previewed in a separate view. It is not possible to display an image and a point cloud in the very same view.

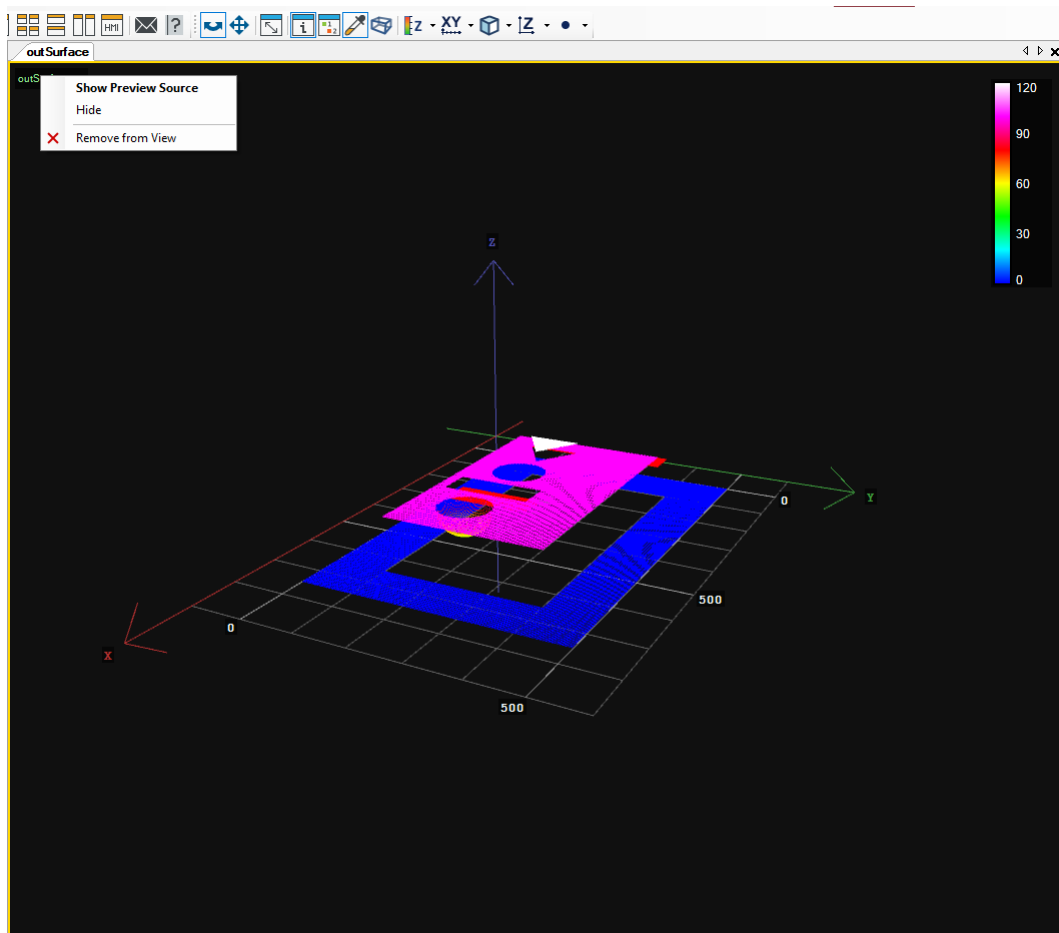
While analyzing 3D data, please pay attention to the colors' scale bar in the top right corner of the preview:



Colors' scale bar.

It might be helpful if you want to estimate the coordinate value based on color.

You can also hide or remove surfaces from a view. To do it, you have to right-click on the information about a preview in the top left corner of the view:



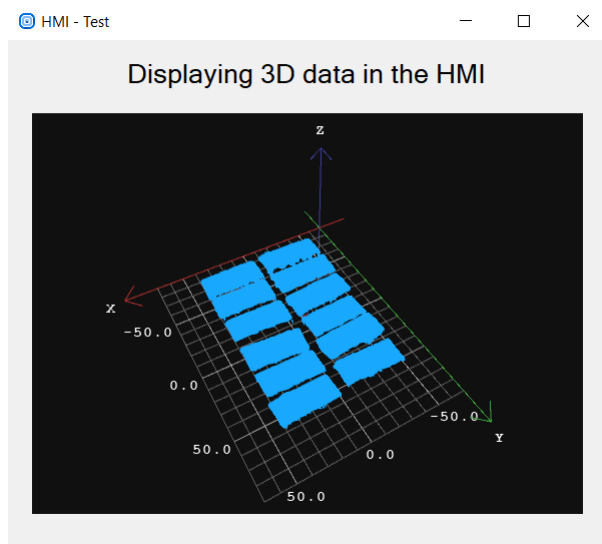
Dropdown list with possible actions on previews.

As it is shown in the above image, you can:

- Show Preview Source – it will indicate the filter in the Program Editor, of which the output has been previewed
- Hide – it will hide only indicated surface without removing the preview
- Remove from View – it will delete the preview, leaving an empty view

HMI

3D data can be displayed in the HMI as well – using the [View3DBox](#) control:



View3DBox HMI control – a point cloud is displayed in the same manner as in the preview.

It is worth mentioning that in this HMI control you can also rotate and change the rotation center of the point cloud. For more details about designing and using HMI, please refer to [HMI Controls](#).

Creating Deep Learning Model

Note: The following article concerns the functionalities related to another product: **Deep Learning Add-on**. More information are available [here](#).

Contents:

1. [Introduction](#)
2. [Workflow](#)
3. [Detecting anomalies 2](#)
4. [Detecting features](#)
5. [Classifying objects](#)
6. [Segmenting instances \(deprecated\)](#)
7. [Locating points](#)
8. [Locating objects](#)
9. [Tips and tricks](#)

Introduction

Deep Learning editors are dedicated graphical user interfaces for **Deep Learning Model** objects (which represent training results). Each time a user opens such an editor, they are able to add or remove images, adjust parameters, and perform new training.

Since version 4.10, it is also possible to open the Deep Learning Editor as a stand-alone application, which is especially useful for re-training models with new images in a production environment.

Since version 5.4, the Aurora Vision Deep Learning tool-chain has transitioned to a completely new training engine. Because of that, some of the Deep Learning tools have been deprecated: **Segmenting instances** and **Detecting anomalies 1**. They are trainable in version 5.3 and older versions only. In more recent releases, the models can only be inferred.

Warning: .TIFF images cannot be used with Aurora Vision Deep Learning tools.

Requirements:

- A Deep Learning license is required to use Deep Learning editors and filters.
- The Deep Learning Service must be up and running to perform model training.

Warning: Some of the Deep Learning algorithms, e.g. Instance Segmentation (deprecated), require a lot of memory, therefore we highly recommend using the GPU version of Aurora Vision Deep Learning.

The currently available Deep Learning tools include:

1. **Anomaly Detection** – for detecting unexpected object variations; trained with sample images marked simply as good or bad.
2. **Feature Detection** – for detecting regions of defects (such as surface scratches) or features (such as vessels on medical images); trained with sample images accompanied by precisely marked ground-truth regions.
3. **Object Classification** – for identifying the name or the class of the most prominent object in an input image; trained with sample images accompanied by the expected class labels.
4. **Instance Segmentation (deprecated)** – for simultaneously locating, segmenting, and classifying multiple objects in an image; trained with sample images accompanied by precisely marked regions of each individual object.
5. **Point Location** – for locating and classifying multiple key points; trained with sample images accompanied by marked points indicating expected classes.
6. **Read Characters** – for locating and classifying multiple characters; this tool uses a pretrained model and cannot be manually retrained by a user, so it is not elaborated on further in this entry.
7. **Text Location** – for locating text; this tool uses a pretrained model and cannot be manually retrained by a user, so it is not elaborated on further in this entry.
8. **Object Location** – for locating and classifying multiple objects; trained with sample images accompanied by marked bounding rectangles indicating expected classes.

Technical details about these tools are available in the [Machine Vision Guide: Deep Learning](#), while this article focuses on the training graphical user interface.

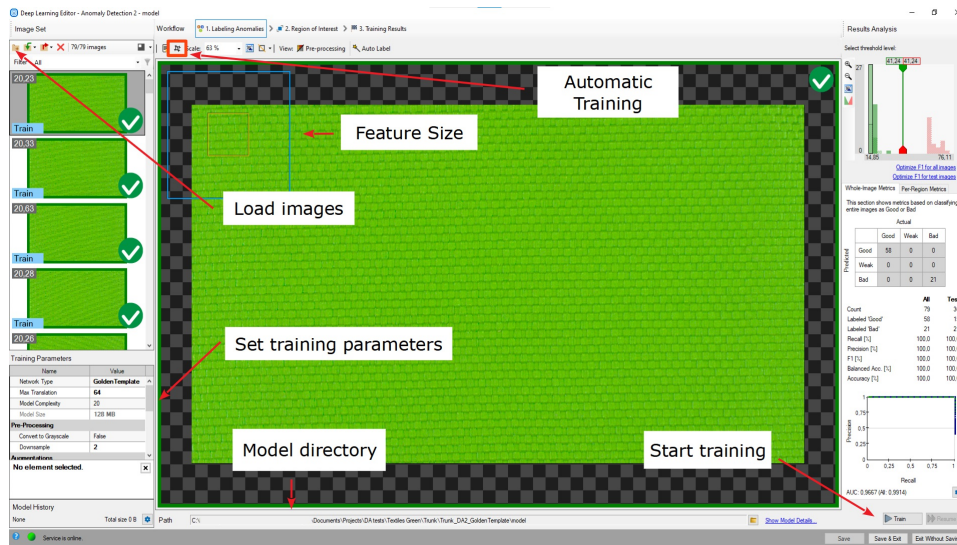
Workflow

You can open a **Deep Learning Editor** via:

- a filter in the Aurora Vision Studio:
 1. Place the relevant DL filter (e.g. [DL_DetectFeatures](#) or [DL_DetectFeatures_Deploy](#)) in the Program Editor.
 2. Go to its Properties.
 3. Click on the button next to the **inModelDirectory** or **inModelId.ModelDirectory** parameter.
- a standalone **Deep Learning Editor** application:
 1. Open a standalone **Deep Learning Editor** application (which can be found in the **Aurora Vision Studio installation folder** as "DeepLearningEditor.exe", in the Aurora Vision folder in the **Start menu**, or in the Aurora Vision Studio application in **Tools menu**).
 2. Choose whether you want to create a new model or use an existing one:
 - **Creating a new model:** Select the relevant tool for your model and press OK, then select or create a new folder where files for your model will be contained and press OK.
 - **Choosing existing model:** Navigate to the folder containing your model files – either write the path to it, click on the button next to field to browse to it, or select one of the recent paths if there are any; then press OK.

The Deep Learning model preparation process is usually split into the following steps:

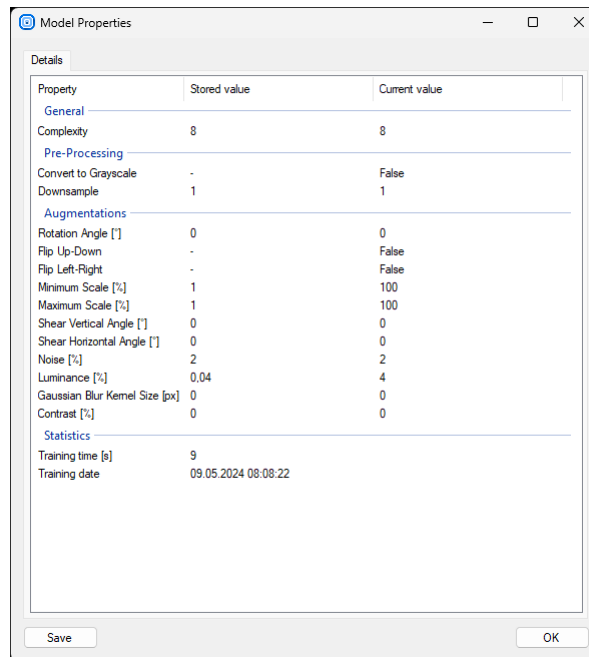
1. **Loading images** – load training images from the disk.
2. **Labeling images** – mark features or attach labels to each training image.
3. **Assigning image sets** – allocate images to the train, test, or validation set.
4. **Setting the region of interest (optional)** – select the area of the image to be analyzed.
5. **Adjusting training parameters** – select training parameters, preprocessing steps, and augmentations specific to the application at hand.
6. **Training the model and analyzing results.**



Overview of a Deep Learning Editor.

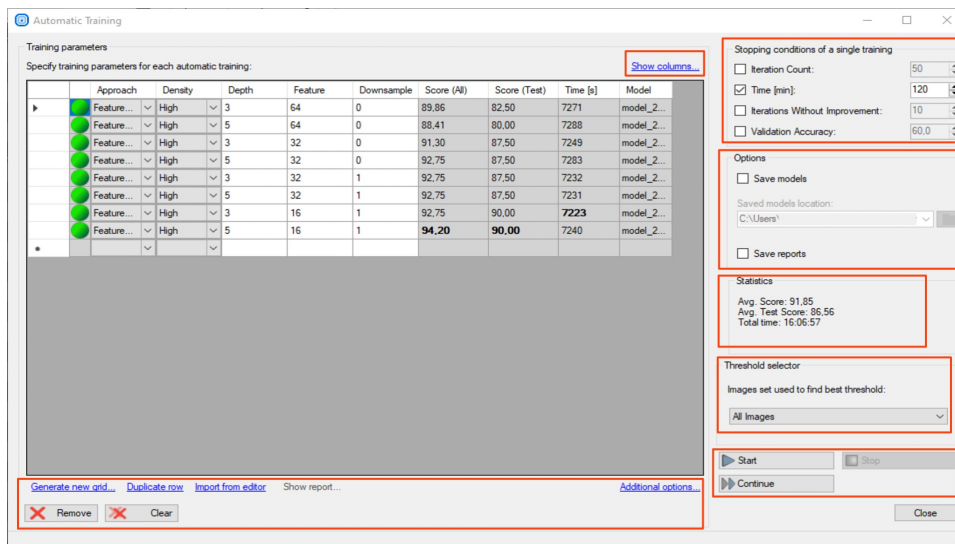
Important features:

- **Pre-processing button** – located in the top toolbar; allows you to see the changes applied to a training image, e.g., grayscale or downsampling.
- **Current Model directory** – located in the bottom toolbar; allows you to switch to a model in another directory or to simply see which model you are actually working on.
- **Show Model Details button** – located next to the previous control; allows you to display information about the current model and save it to a file.

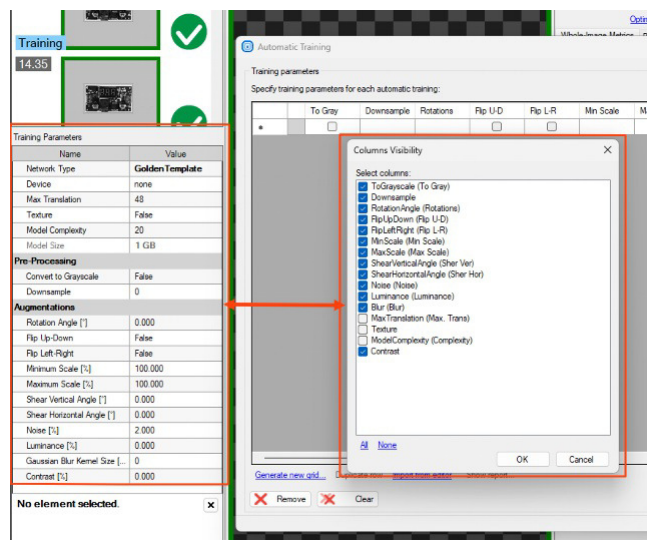


- **Train & Resume buttons** – allow you to start training or resume training in case you have changed some of training parameters (except for **Convert to Grayscale** value).
- **Saving buttons:**
 - **Save** – saves the current model in a chosen location.
 - **Save & Exit** – saves the model and then exits the Deep Learning Editor.
 - **Exit Without Saving** – exits the editor, but the model is not saved.

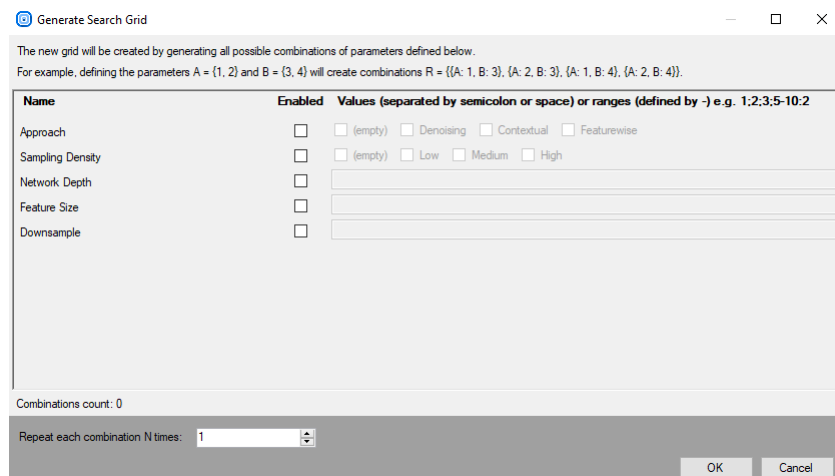
Open automatic training window button – allows you to prepare a training series for different parameters. If you are not sure, which parameter settings will give you the best result, you can prepare a combination for each value to compare the results. The test parameters can be prepared automatically with **Generate new grid** or manually entered. After setting the parameters you need to start the test. The settings and the results are shown in the grid, one row for one model.



- **Show columns** – hides / shows model parameters you will use in your test. The view is common for all deep learning tools. To create an appropriate grid search, choose the parameters which are correct for the used tool (the ones you can see in Training Parameters). For **DL_DetectAnomalies2** choose the network type first to see the appropriate parameters.



- **Generate new grid** – prepares the search grid for the given parameters. Only parameters chosen in **Show columns** are available. The values should be separated with a semicolon (;).



- **Duplicate rows** – duplicates a training parameters configuration. If the parameters inside the row aren't modified, this model will use the same settings twice for the training.
- **Import from editor** – copies the training parameters from the **Editor Window** to the last search grid row.
- **Show report** – shows the report for the chosen model (the chosen row). This option is available only if you choose **Save Reports** before starting the training session.
- **Additional options:**
 - **Export grid to CSV file** – exports the grid of training parameters to a CSV file.
 - **Import grid from CSV file** – imports the grid of training parameters from a CSV file.
- **Remove** – removes a chosen training configuration.
- **Clear** – clears the whole search grid.

- **Stopping conditions of a single training** – determines when a single training stops.
 - Iteration Count
 - Time
 - Iterations without improvement
 - Validation Accuracy
- **Options**
 - **Save models** – saves each trained model in the defined folder.
 - **Save reports** – saves a report for each trained model in the folder defined for saving models.
- **Statistics** – shows statistics for all trainings.
 - **Avg. Score** – shows average score of all trained models for all images.
 - **Avg. Test Score** – shows average score of all trained models for test images.
 - **Total time** – sums up time of each training.
- **Threshold selector** – choses for which image group the best threshold is searched.
 - **All Images**
 - **Test Images**
- **Start** – starts the training series with the first defined configuration.
- **Stop** – stops the training series.
- **Continue** – continues the stopped training series with the next configuration of the parameters.

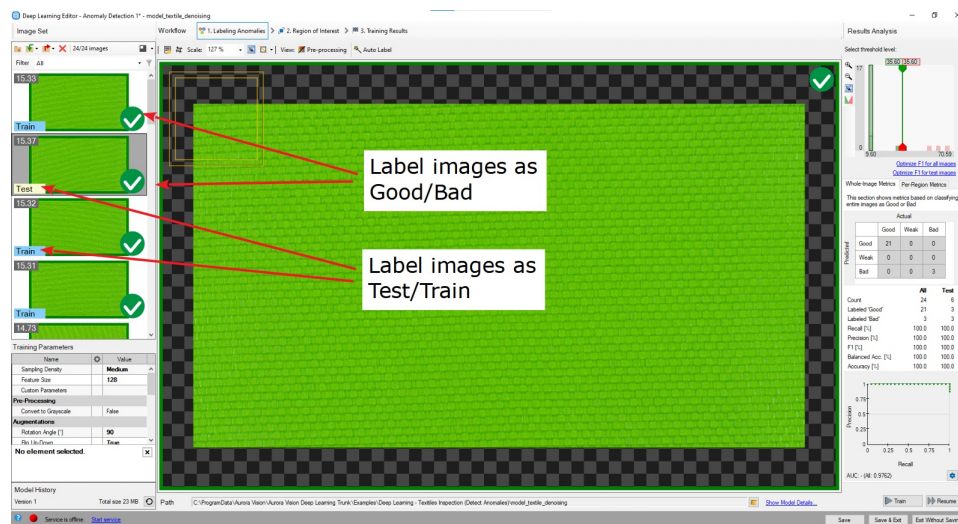
Detecting anomalies 2 (classification-based approach)

In this tool, the user only needs to mark which images contain correct cases (good) or incorrect ones (bad). Training is performed on good images only. Inference can later be performed on both good and bad images to determine a suitable threshold.

[DL_DetectAnomalies2](#) performs one-class classification of each part of the input image.

1. Marking Good and Bad samples and dividing them into Test and Training.

Click on a question mark icon to label each image in the training set as **Good** or **Bad**. Green and red icons on the right side of the training images indicate to which set the image belongs. Alternatively, you can use **Add images and mark...** Then divide the images into **Train** or **Test** by clicking on the left label on an image. Remember that all bad samples should be marked as **Test**.



Labeled images in Deep Learning Editor.

2. Configuring augmentations

It is usually recommended to add some additional sample augmentations, especially when the training set is small. For example, the user can add additional variations in pixel intensity to prepare the model for varying lighting conditions on the production line. Refer to "Augmentation" section for detailed description of parameters: [Deep Learning – Augmentation](#).

3. Reducing region of interest

Reduce the region of interest to focus only on the important part of the image. Reducing the region of interest will speed up both training and inference.

It is possible to define multiple regions of interest on a single image. Each new region of interest is treated as a separate sample based on the original image where it was marked.

By default, the first defined region of interest is applied to all images within the **Image Set**. Its position can be freely modified for each image individually, but the size is synchronized. If you adjust the size of the region on one image, all other regions will automatically resize to match. Each image must have at least one region of interest.

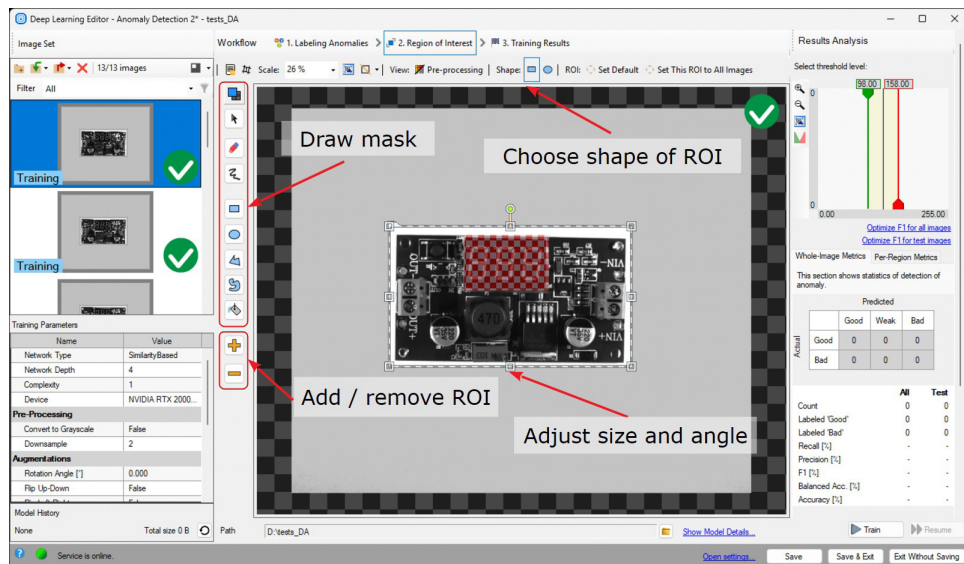
Regions of interest can be rectangular or ellipse. All of them have the same chosen shape. They can be added and deleted using the buttons in the toolbox to the left of the image preview.

To exclude a part of the image from analysis within a region of interest, you can draw a mask inside it. Please note the following about masks:

- The mask is the same for every region of interest.
- It automatically scales to fit the size of the region of interest.
- It cannot be adjusted individually for each region.

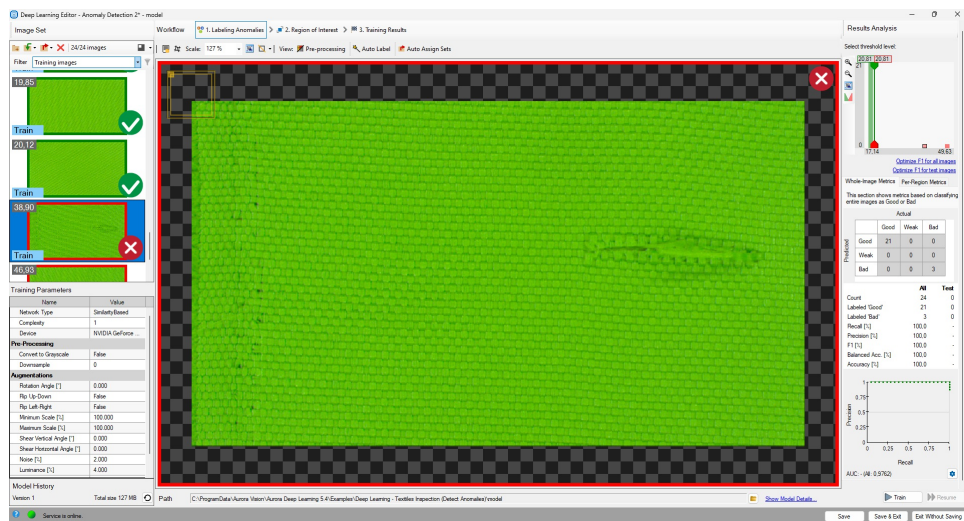
As a result, this mask is applied automatically during model execution, and a user has no control over its size or shape in the Program Editor.

Masks can be added and deleted using the drawing tools in the toolbox to the left of the image preview.



4. Setting training parameters

- **Model Complexity** – defines the size of the resulting model.
- **Max Translation** – determines the spatial tolerance of the model. It's available only for the **Golden Template** approach.



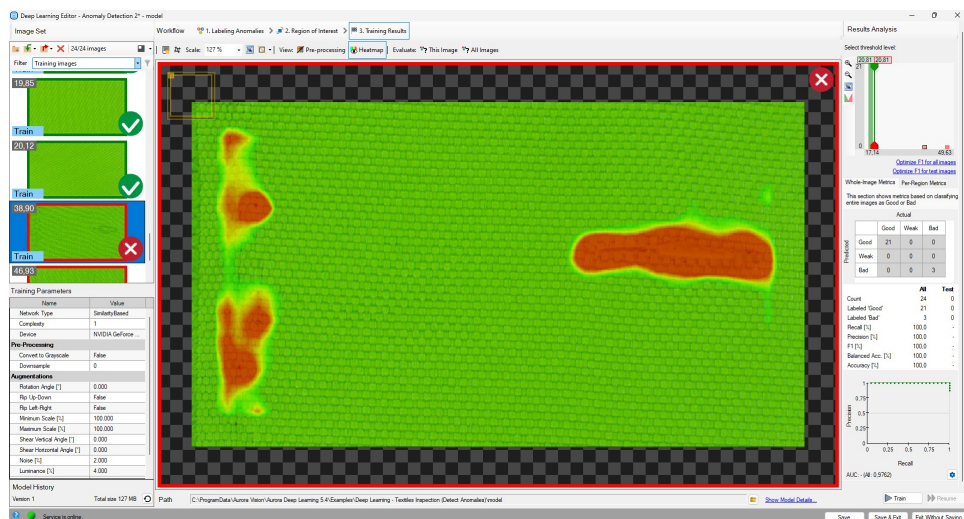
Results are marked as rectangles of either of two colors: green (classified as good) or red (classified as bad).

For more details, read [Deep Learning - Setting parameters](#).

5. Analyzing results

The window shows a histogram of sample scores and a heatmap of found defects. The right column contains a histogram of scores computed for each image in the training set. Additional statistics are displayed below the histogram.

To evaluate the trained model, the **Evaluate: This Image** or **Evaluate: All Images** buttons can be used. It can be useful after adding new images to the data set or after changing the area of interest.



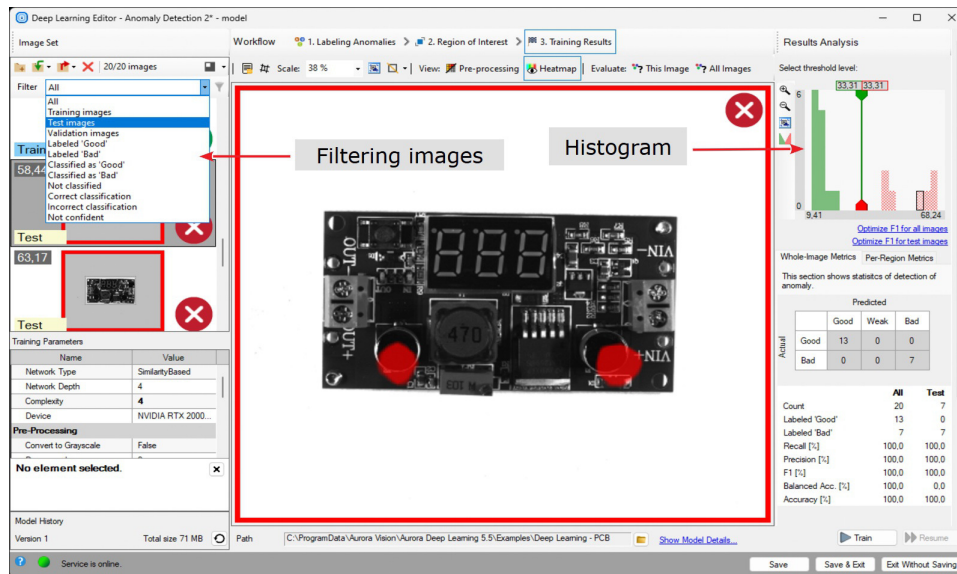
Heatmap indicates the most possible locations of defects.

After training, two border values are computed:

1. Maximum good sample score (T1) – all values from 0 to T1 are marked as Good.
2. Minimum bad sample score (T2) – all values greater than T2 are marked as Bad.

All scores between T1 and T2 are marked as "Low quality". Results in this range are uncertain and may not be correct. Filters contain an additional output **outIsConfident** which determines the values which are not in the T1-T2 range.

After evaluation, additional filtering options may be used in the list of training images.



Filtering the images in the training set.

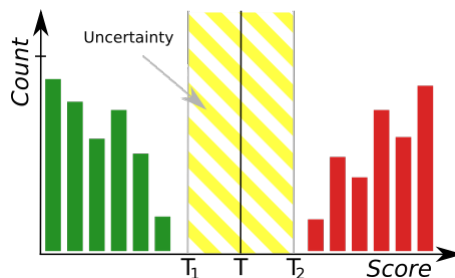
Interactive histogram tool

DetectAnomalies filters measure deviation of samples from the normal image appearances learned during the training phase. If the deviation exceeds a given threshold, the image is marked as anomalous. The suggested threshold is automatically calculated after the training phase, but it can also be adjusted by the user in the Deep Learning Editor using an interactive histogram tool described below.

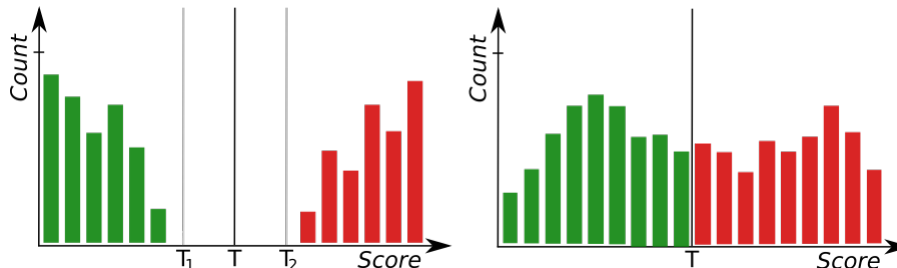
After the training phase, scores are calculated for every training sample and are presented in the form of a histogram; good samples are marked with green, and bad samples with red bars. In an ideal case, the scores for good samples should be all lower than for bad samples, and the threshold should be automatically calculated to give the optimal accuracy of the model. However, the groups may sometimes overlap because of:

1. incorrectly labeled samples,
2. bad training parameters,
3. ambiguous definition of the expected defects,
4. high variability of the samples appearance or environmental conditions.

In order to achieve more robust threshold, it is recommended to perform training with a large number of samples from both groups. If the number of samples is limited, our software makes it possible to manually set the uncertainty area with additional thresholds (the information about the confidence of the model can then be obtained from the hidden outIsConfident filter output).



The histogram tool where green bars represent correct samples and red bars represent anomalous samples. T marks the main threshold and T1, T2 define the area of uncertainty.



Left: a histogram presenting well-separated groups indicating a good accuracy of the model. Right: a poor accuracy of the model.

Detecting features (segmentation)

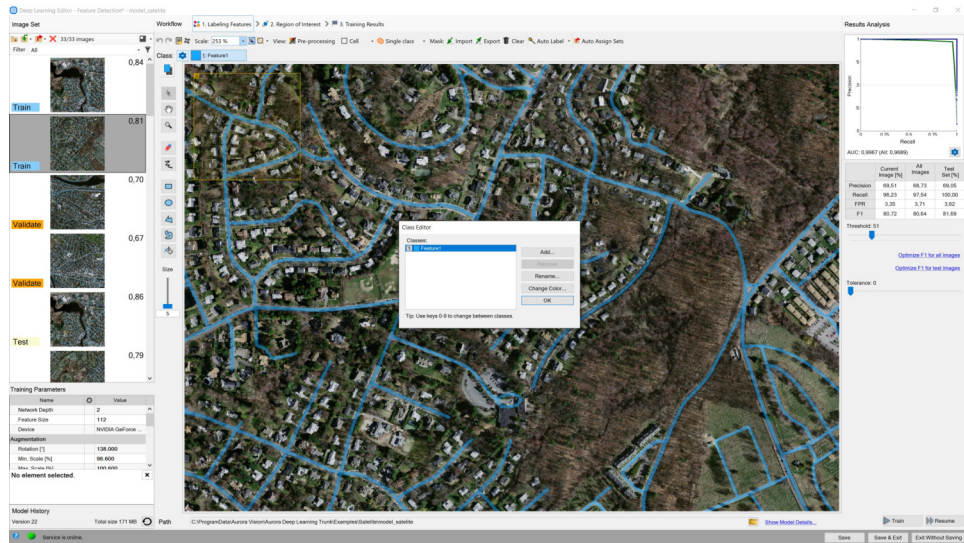
In this tool, the user has to define each feature class and then mark features on each image in the training set. This technique is used to find object defects like scratches or color changes, and for detecting image parts trained on a selected pattern.

1. Defining feature classes (Marking class)

First, the user has to define classes of defects. Generally, they should be features that the user would like to detect on images. Multiple different classes can be defined, but it is not recommended to use more than a few.

The Class editor is available under the sprocket wheel icon in the top bar.

To manage classes, the **Add**, **Remove** or **Rename** buttons can be used. To customize appearance, the color of each class can be changed using the **Change Color** button.



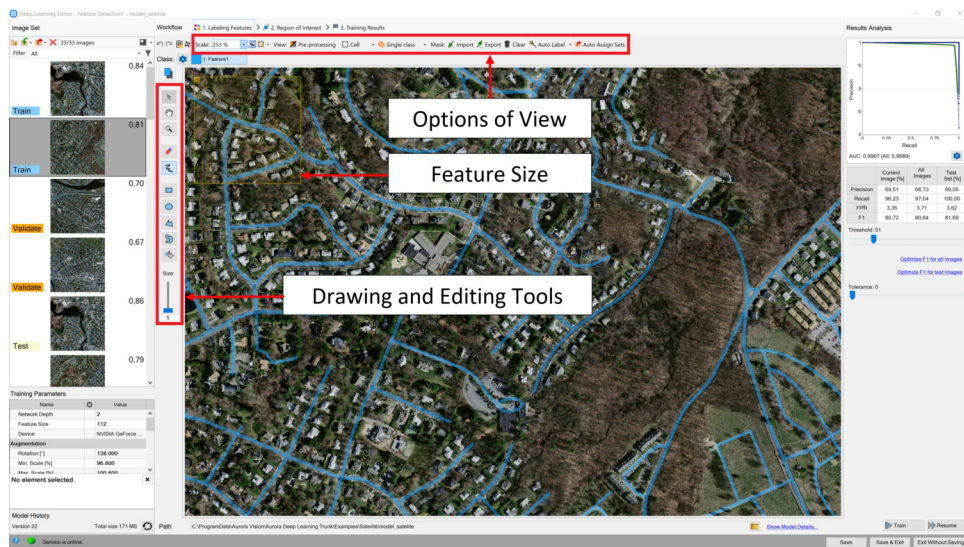
In this tool, it is possible to define more classes of defects.

The current class for editing is displayed on the left, the user can select a different class by clicking.

Use the drawing tool to mark features on the input images. Tools such as **Brush** or **Rectangle** can be used for selecting features.

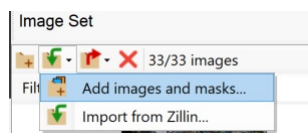
In addition, class masks can be imported from external files. There are buttons for **Import** and **Export** of created classes so that the user can create an image of a mask automatically prior to a Deep Learning model.

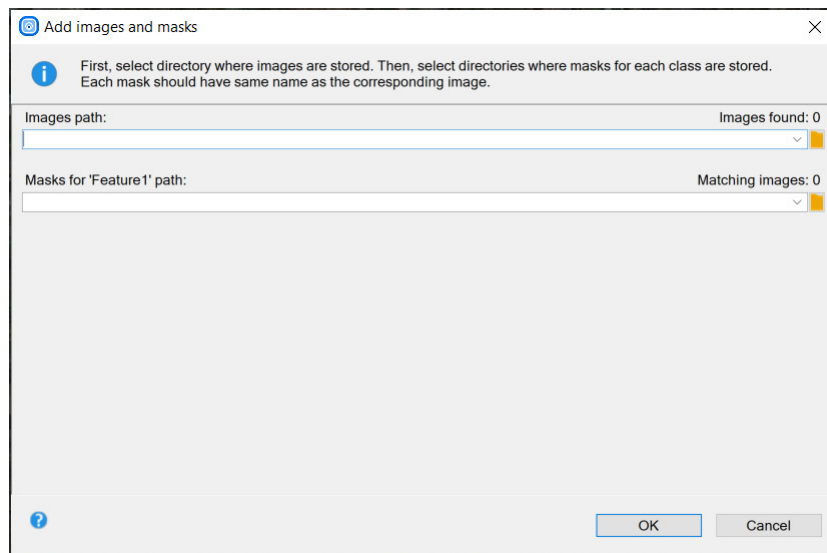
The image mask should have the same size as the selected image in the input set. When importing an image mask, all non-black pixels will be included in the current mask.



The most important features of the tool.

The user can also load multiple images and masks at the same time, using **Add images and masks** button.





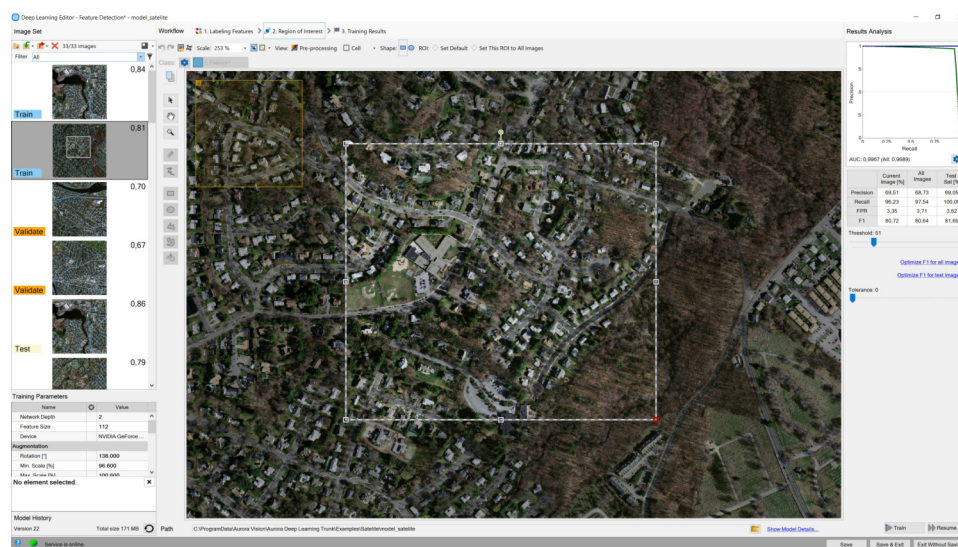
Selecting path to images and masks.

The directory containing input images should be selected first. Then, directories for each feature class can be selected below. Images and masks are matched automatically using their file names. For example, let us assume that "images" directory contains images 001.png, 002.png, 003.png; "mask_class1" directory contains 001.png, 002.png, 003.png; and "mask_class2" directory contains 001.png, 002.png, 003.png. Then "images\001.png" image will be loaded together with "mask_class1\001.png" and "mask_class2\001.png" masks.

2. Reducing region of interest

The user can reduce the input image size to speed up the training process. In many cases, the number of features on an image is very large, and most of them are the same. In such cases, the region of interest can also be reduced.

On the top bar, there are tools for applying the current ROI to all images, as well as for resetting the ROI.



Setting ROI.

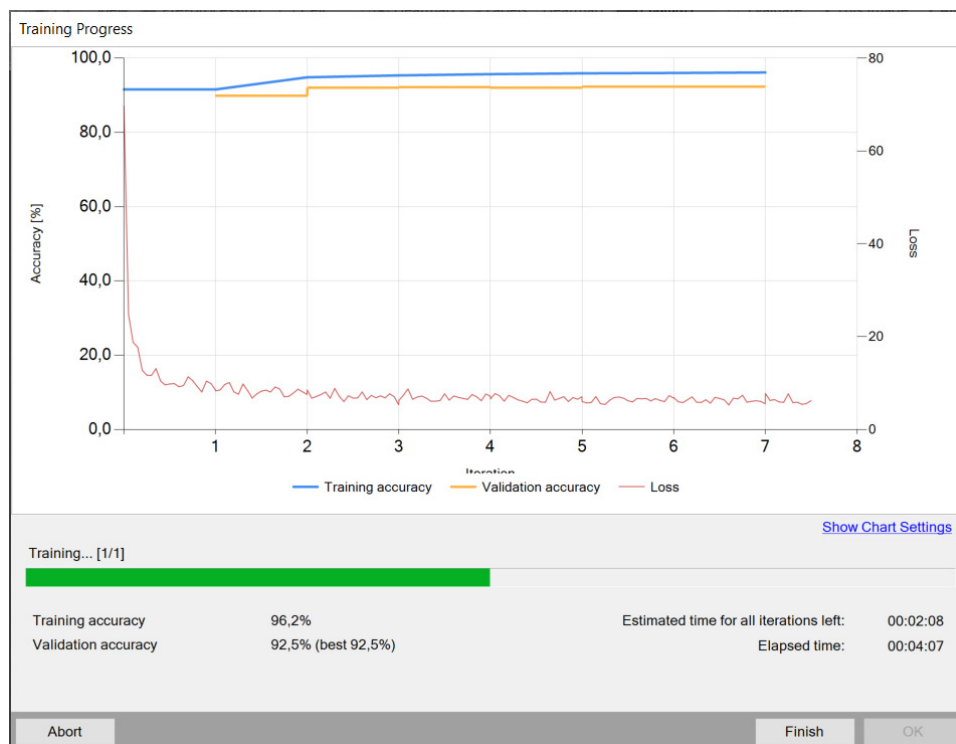
3. Setting training parameters

- **Network depth** – chooses one of several predefined network architectures varying in their complexity. For bigger and more complex image patterns, a higher depth might be necessary.
- **Feature size** – the size of an image part that will be analyzed with one pass through the neural network. It should be significantly bigger than any item of interest, but not too big, – as the bigger the feature size, the more difficult and time consuming the training process is.
- **Device** – selects the computing device (e.g., CPU or GPU) for the training. If multiple GPUs are available, a specific one can be chosen from the list of available devices.
- **Stopping conditions** – define when the training process should stop.

For more details, please refer to [Deep Learning – Setting parameters](#) and [Deep Learning – Augmentation](#).

4. Model training

During training, three main series are visible: training accuracy, validation accuracy, and loss. The training and validation charts should have a similar, increasing pattern. The loss chart should decrease.



The training process chart.

5. Result analysis

Image scores (heatmaps) are displayed in red after using the model for evaluation of the image. These scores indicate the association of a specific part of the image to the currently selected feature class.

Due to the lack of context on the image border, correctly detecting objects at the image edges is problematic. Therefore, the heatmaps returned by the network focus on the image content beyond the edges without analysing the data located on the image border. The thickness of the skipped frame depends directly on the patch size parameter and ranges from 6 to 12 pixels. When the **inRoi** is applied, the border is removed from the selected image region.

Evaluate: This Image and **Evaluate: All Images** buttons can be used to classify images. It can be useful after adding new images to the data set or after changing the area of interest.

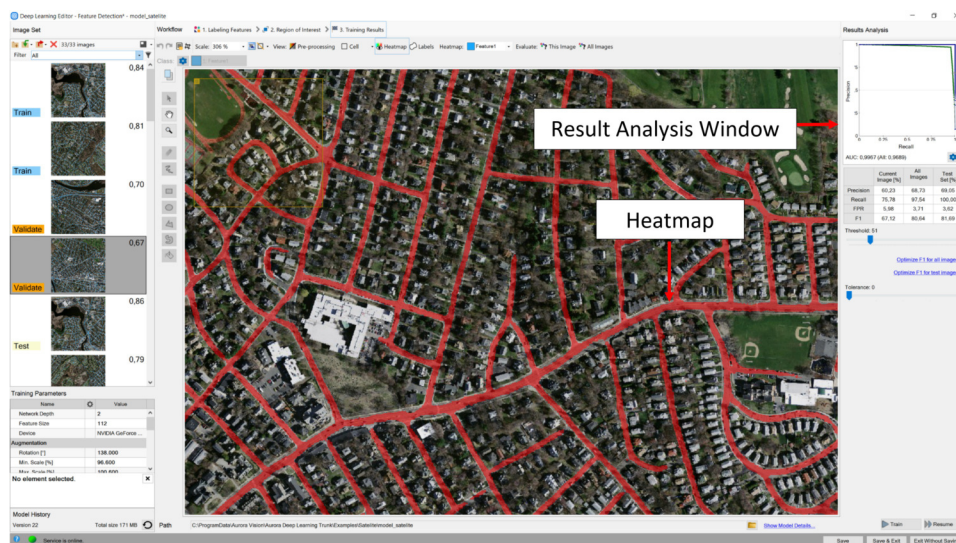


Image after classification.

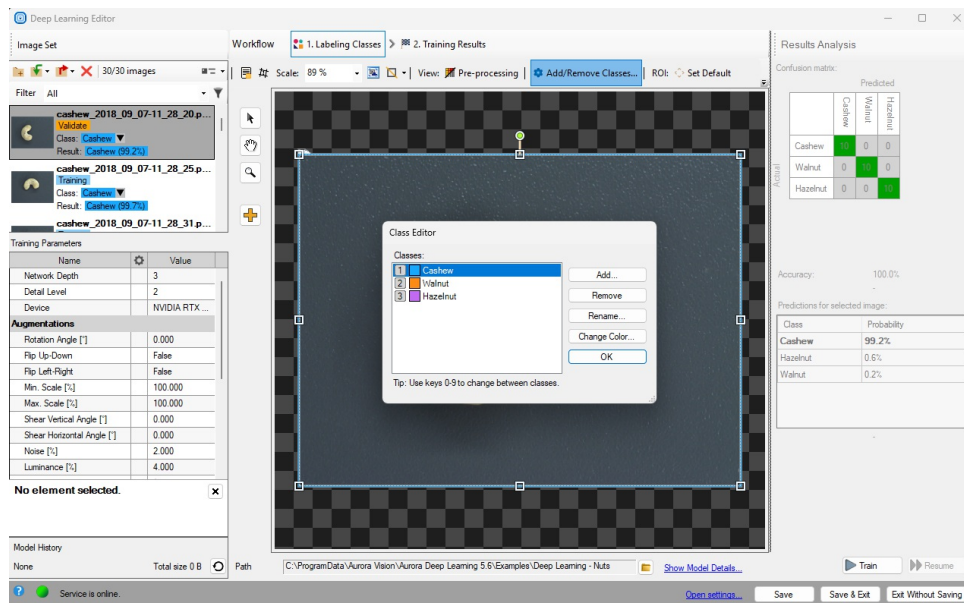
In the top left corner of the editor, a green rectangle visualizes the selected feature size.

Classifying objects

In this tool, the user only has to label images with respect to a desired number of classes. Theoretically, the number of classes that a user can create is infinite, but please note that you are limited by the amount of data your GPU can process. Labeled images will allow the user to train a model and determine features which will be used to evaluate new samples and assign them to the proper classes.

1. Editing number of classes

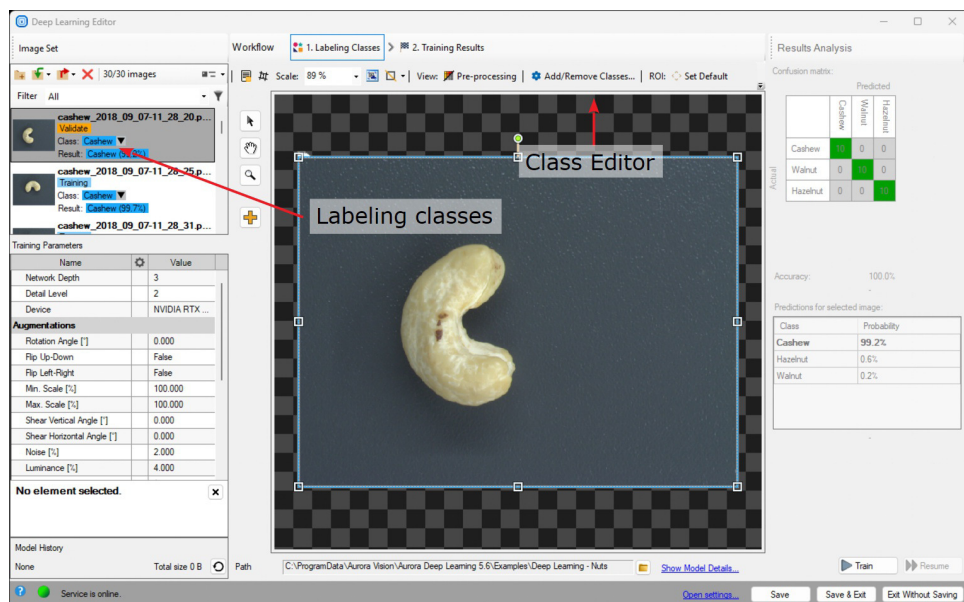
By default, two classes are defined. If the problem is more complex than that, the user can edit classes and define more if needed. Once the user is ready with the definition of classes, images can be labeled.



Using Class Editor.

2. Labeling samples

Labeling of samples is possible after adding training images. Each image has a corresponding drop-down list which allows for assigning a specific class. It is possible to assign a single class to multiple images by selecting desired images in the Deep Learning Editor.



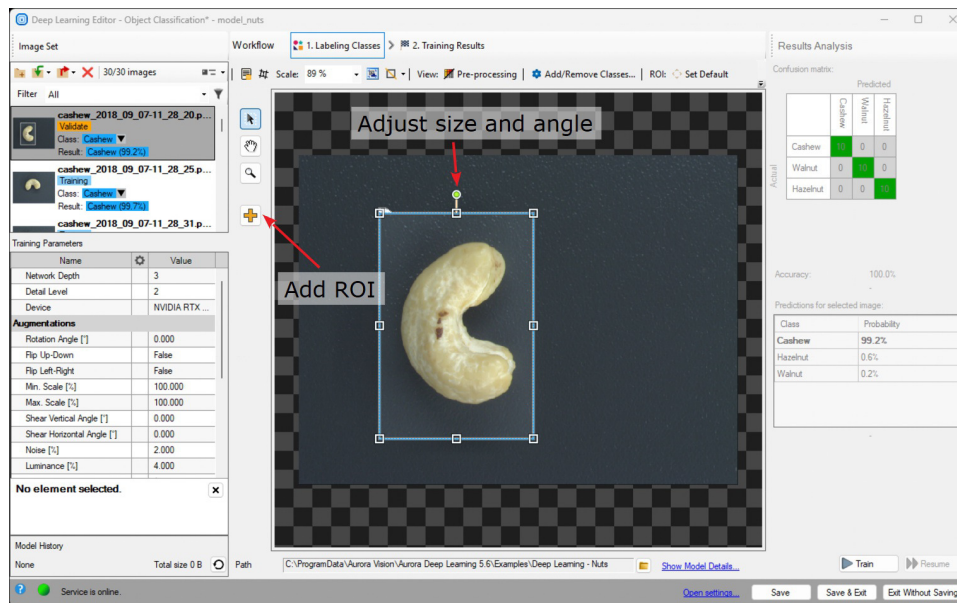
Labeling images with classes.

3. Setting region of interest

Reduce the region of interest to focus only on the important part of the image. Reducing the region of interest will speed up both training and classification. By default, the region of interest contains the whole image.

It is possible to define multiple regions of interest on a single image. Each new region of interest is treated as a separate sample based on the original image where it was marked and can be assigned to a different class.

To get the best classification results, use the same region of interest for training and classification.



Changed region of interest.

4. Setting training parameters

- **Network depth** – predefined network architecture parameter. For more complex problems, a higher depth might be necessary.
- **Detail level** – level of detail needed for a particular classification task. For the majority of cases, the default value of 1 is appropriate, but if images of different classes are distinguishable only by small features, increasing the value of this parameter may improve classification results.
- **Device** – selects the computing device (e.g., CPU or GPU) for the training. If multiple GPUs are available, a specific one can be chosen from the list of available devices.
- **Stopping conditions** – define when the training process should stop.

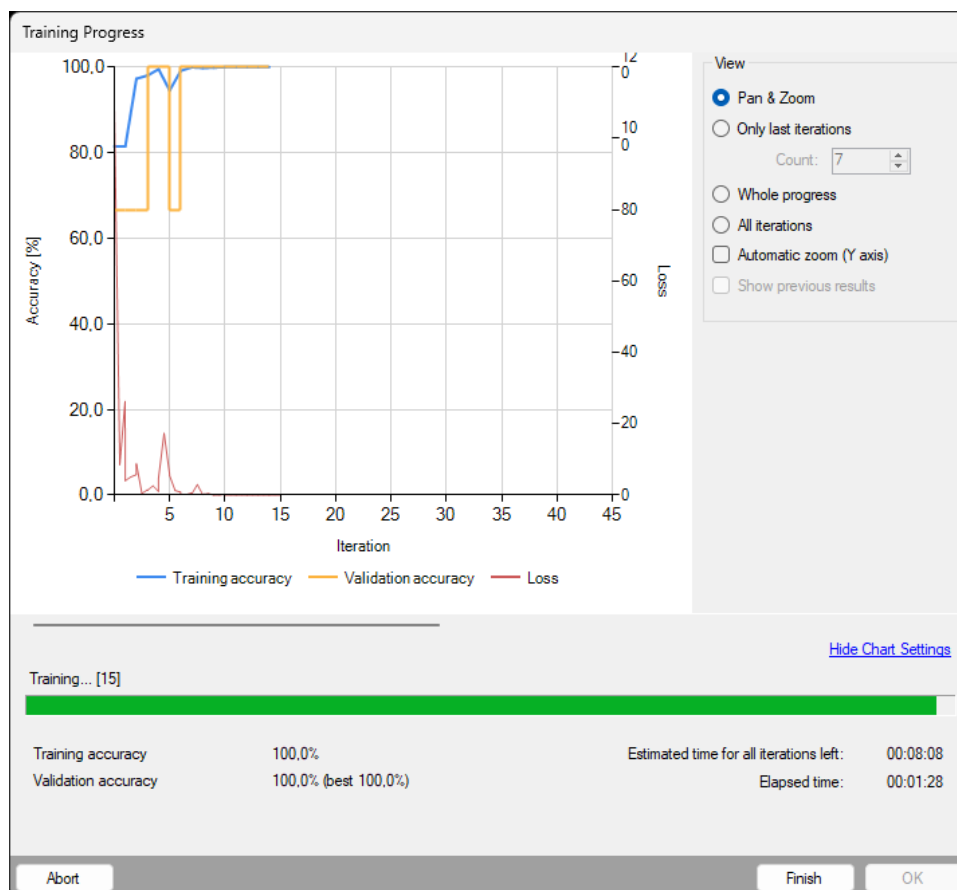
For more details, please refer to [Deep Learning – Setting parameters](#) and [Deep Learning – Augmentation](#).

5. Performing training

During training, three main series are visible: training accuracy, validation accuracy, and loss. The training and validation charts should have a similar, increasing pattern. The loss chart should decrease.

More detailed information is displayed below the chart:

- current training statistics (training and validation accuracy),
- number of processed samples (depends on the number of images),
- elapsed time.



Training object classification model.

The training process can take a couple of minutes or even longer. It can be manually finished if needed.

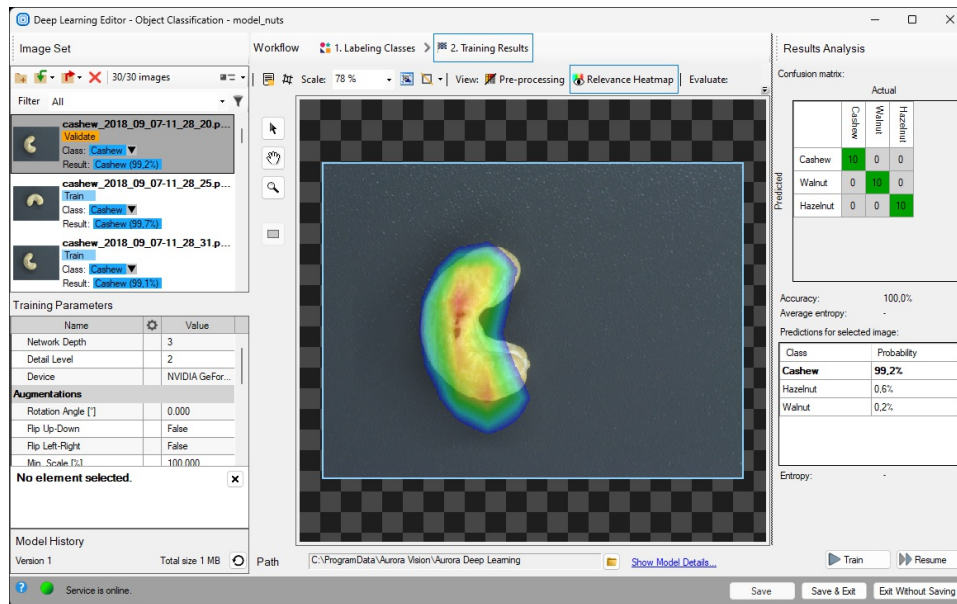
The final result of one training is one of the partial models that achieved the highest validation accuracy (not necessarily the last one). Consecutive training attempts will prompt the user whether to save a new model or keep the old one.

6. Analyzing results

The window shows a confusion matrix which indicates how well the training samples have been classified.

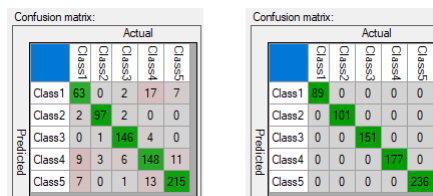
The image view contains a heatmap which indicates which part of the image contributed the most to the classification result.

Evaluate: This Image and **Evaluate: All Images** buttons can be used to classify training images. It can be useful after adding new images to the data set or after changing the area of interest.



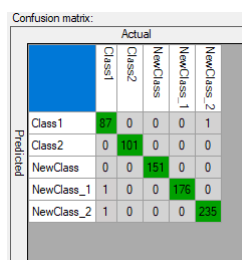
Confusion matrix and class assignment after the training.

Sometimes it is hard to guess the right parameters in the first attempt. The picture below shows a confusion matrix that indicates inaccurate classification during the training (left).



Confusion matrices for model that needs more training (left) and for model well trained (right).

It is possible that the confusion matrix indicates that the trained model is not 100% accurate with respect to training samples (numbers assigned exclusively on the main diagonal represent 100% accuracy). The user needs to properly analyze this data, and use it to their advantage.



Confusion matrix indicating good generalization.

Too many erroneous classifications indicate poor training. A few of them may indicate that the model is properly focused on generalization rather than exact matching to training samples (possible overfitting). Good generalization can be achieved if images used for training are varied (even among a single class). If the provided data is not varied within classes (user expects exact matching), and still some images are classified outside the main diagonal after the training, the user can:

- increase the network depth,
- prolong training by increasing the number of iterations,
- increase the amount of data used for training,
- use augmentation,
- increase the detail level parameter.

Segmenting instances (deprecated)

Warning: The **Instance Segmentation** model is trainable in 5.3 and older versions only. In more recent releases, the models can be only inferred.

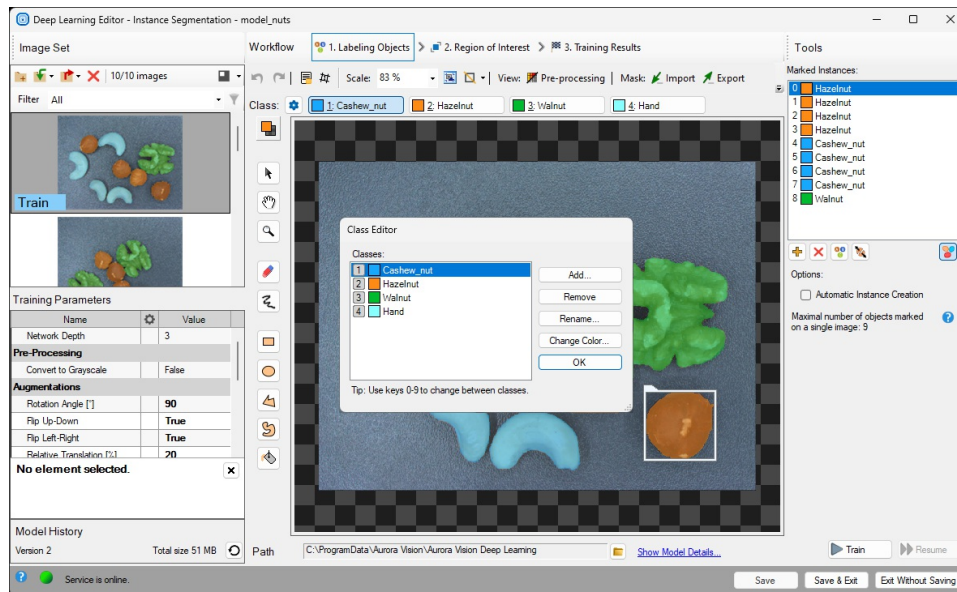
In this tool, a user needs to draw regions (masks) corresponding to the objects in the scene and specify their classes. These images and masks are used to train a model which then in turn is used to locate, segment, and classify objects in the input images.

1. Defining object classes

First, a user needs to define classes of objects that the model will be trained on and that later it will be used to detect. Instance segmentation model can deal with single as well as multiple classes of objects.

Class editor is available under the Class Editor button.

To manage classes, the **Add**, **Remove** or **Rename** buttons can be used. To customize appearance, the color of each class can be changed using the Change Color button.



Using Class Editor.

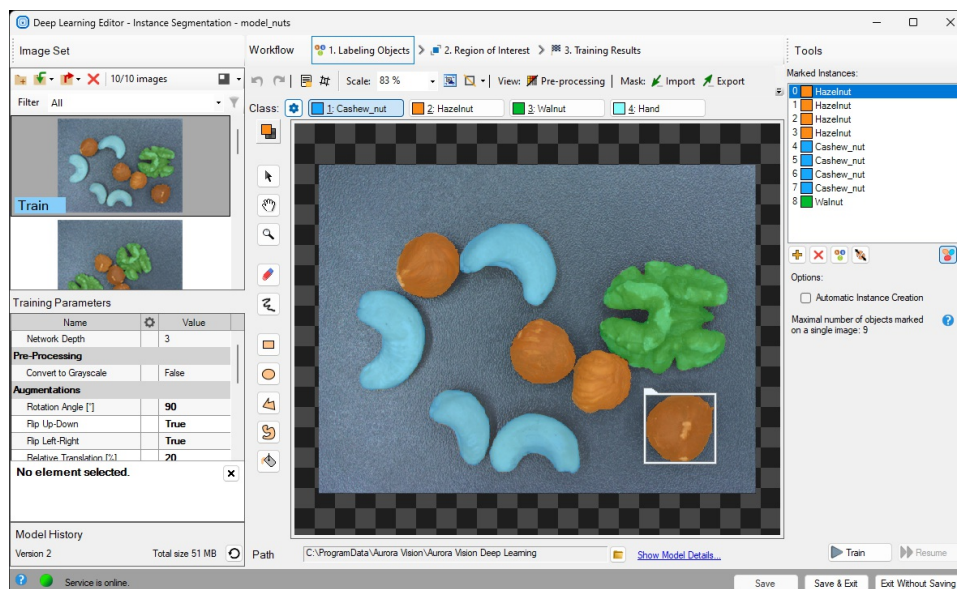
2. Labeling objects

After adding training images and defining classes, a user needs to draw regions (masks) to mark objects in images.

To mark an object, the user needs to select a proper class in the Current Class drop-down menu and click the Add Instance button (green plus). Alternatively, for convenience of labeling, it is possible to apply **Automatic Instance Creation** which allows a user to draw quickly masks on multiple objects in the image without having to add a new instance every time.

Use the drawing tool to mark objects on the input images. Multiple tools such as brush and shapes can be used to draw object masks. Masks are the same color as previously defined for the selected classes.

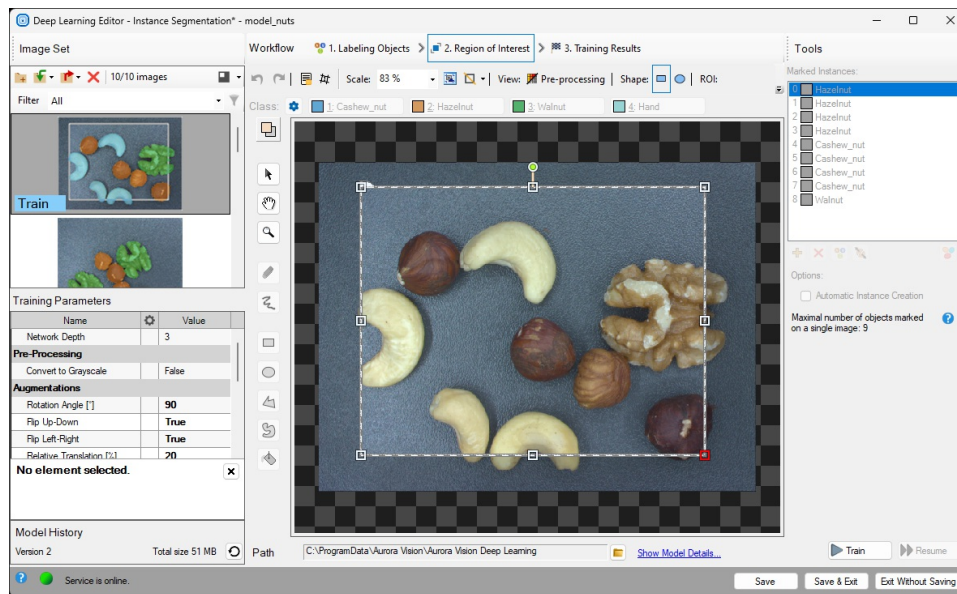
The Marked Instances list in the top left corner displays a list of defined objects for the current image. If an object does not have a corresponding mask created in the image, it is marked as "(empty)". When an object is selected, a bounding box is displayed around its mask in the drawing area. A selected object can be modified in terms of a class (Change Class button) as well as a mask (by simply drawing new parts or erasing existing ones). The **Remove Instance** button (red cross) allows you to completely remove a selected object.



Labeling objects.

3. Reducing region of interest

Reduce the region of interest to focus only on the important part of the image. By default, region of interest contains the whole image.



Changing region of interest.

4. Setting training parameters

- **Network depth** – predefined network architecture parameter. For more complex problems, a higher depth might be necessary.
- **Device** – selects the computing device (e.g., CPU or GPU) for the training. If multiple GPUs are available, a specific one can be chosen from the list of available devices.
- **Stopping conditions** – define when the training process should stop.

For more details, read [Deep Learning – Setting parameters](#).

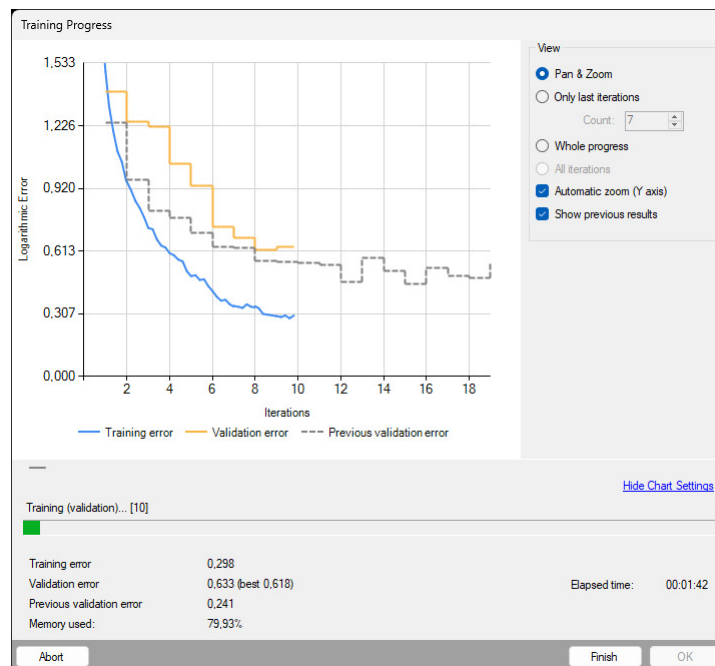
Details regarding augmentation parameters: [Deep Learning – Augmentation](#)

5. Performing training

During training, two main series are visible: training error and validation error. Both charts should have a similar pattern. If a training session was run before, the third series with previous validation error is also displayed.

More detailed information is displayed below the chart:

- current iteration number,
- current training statistics (training and validation error),
- number of processed samples,
- elapsed time.



Training instance segmentation model.

Training may be a long process. During this time, training can be stopped. If no model is present (first training attempt), the model with best validation accuracy will be saved. Consecutive training attempts will prompt a user whether to replace the old model.

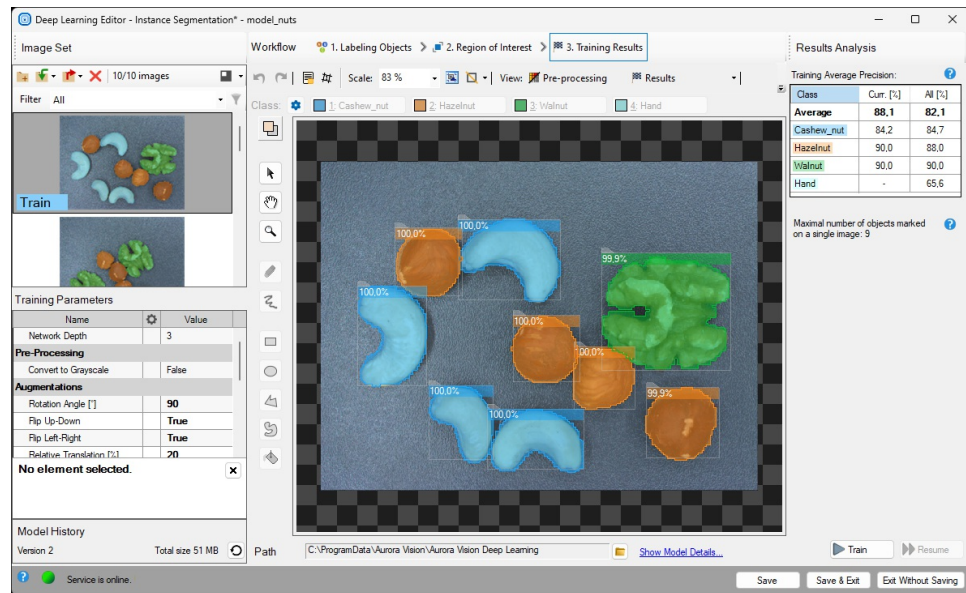
6. Analyzing results

The window shows the results of instance segmentation. Detected objects are displayed on top of the images. Each detection consists of the following data:

- class (identified by a color),
- bounding box,
- model-generated instance mask,

- confidence score.

Evaluate: This Image and **Evaluate: All Images** buttons can be used to perform instance segmentation on the provided images. It can be useful after adding new images to the data set or after changing the area of interest.



Instance segmentation results visualized after the training.

Instance segmentation is a complex task, therefore, it is highly recommended to use data augmentations to improve the network's ability to generalize learned information. If results are still not satisfactory, the following standard methods can be used to improve model performance:

- providing more training data,
- increasing the number of training iterations,
- increasing the network depth.

Locating points

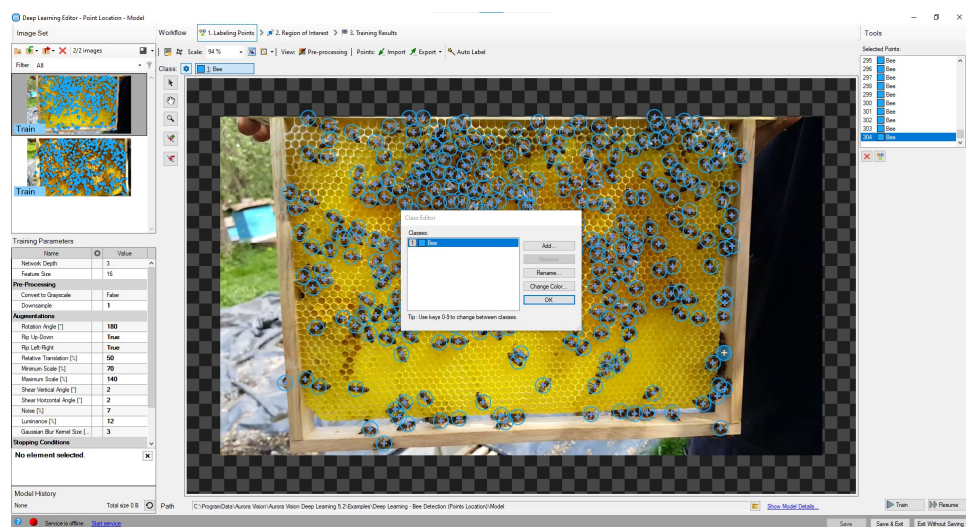
In this tool, the user defines classes and marks key points in the image. This data is used to train a model which then is used to locate and classify key points in images.

1. Defining classes

First, a user needs to define classes of key points that the model will be trained on and later used to detect. Point location model can deal with single as well as multiple classes of key points. If the size of your classes differs greatly, it is better to prepare two, or more, different models than one.

Class editor is available under the Class Editor button.

To manage classes, the **Add**, **Remove**, or **Rename** buttons can be used. The color of each class can be changed using the **Change Color** button.



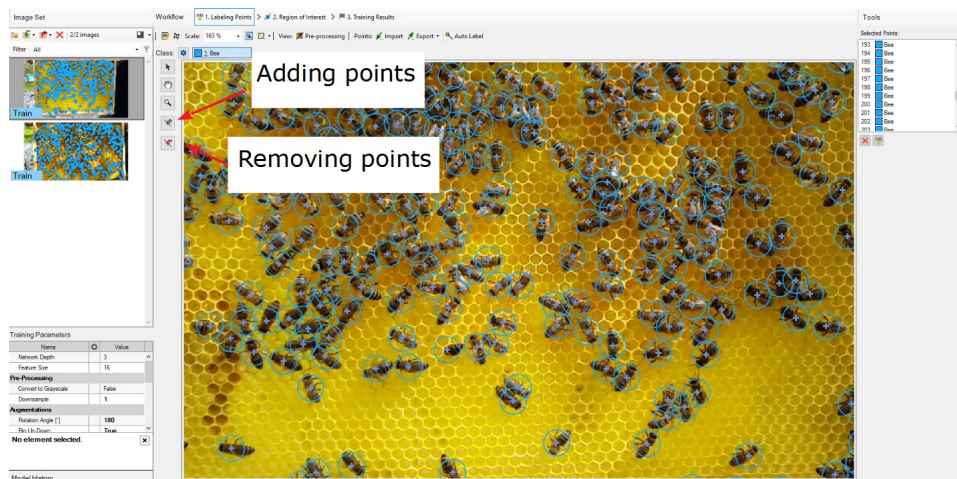
Using Class Editor.

2. Marking key points

After adding training images and defining classes, a user needs to mark points in images.

To mark an object, a user needs to select a proper class in the Current Class drop-down menu and click the Add Point button. Points have the same color as previously defined for the selected class.

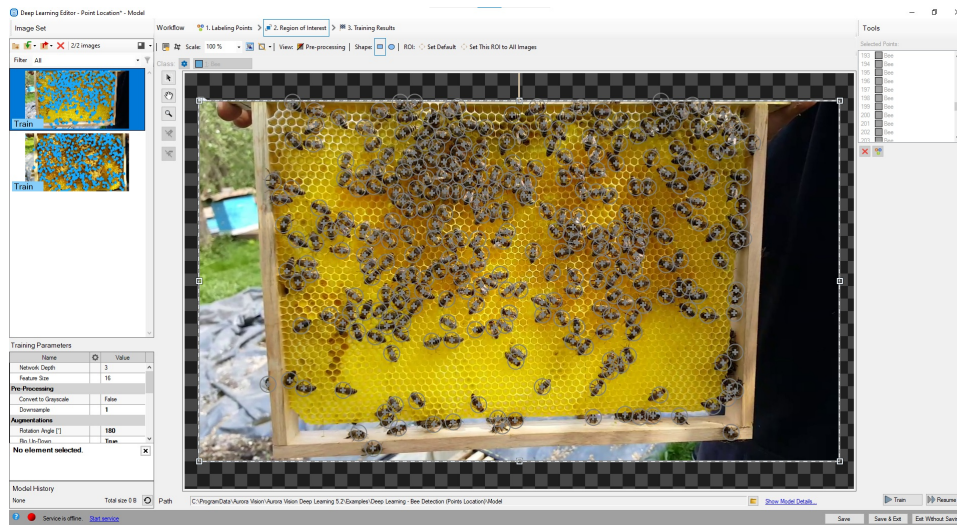
The Selected Points list in the top right corner displays a list of defined points for the current image. A point can be selected either from the list, or directly on the image area. A selected point can be moved, removed (**Remove Point** button), or has its class changed (**Change Class** button).



Marking points.

3. Reducing region of interest

Reduce the region of interest to focus only on the important part of the image and to speed up the training process. By default, the region of interest contains the whole image.



Changing region of interest.

4. Setting training parameters

- **Network depth** – predefined network architecture parameter. For more complex problems, a higher depth might be necessary.
- **Feature size** – the size of a small object or of a characteristic part. If images contain objects of different scales, it is recommended to use feature size slightly larger than the average object size, although it may require experimenting with different values to achieve optimal results.
- **Device** – selects the computing device (e.g., CPU or GPU) for the training. If multiple GPUs are available, a specific one can be chosen from the list of available devices.
- **Stopping conditions** – define when the training process should stop.

For more details, read [Deep Learning – Setting parameters](#).

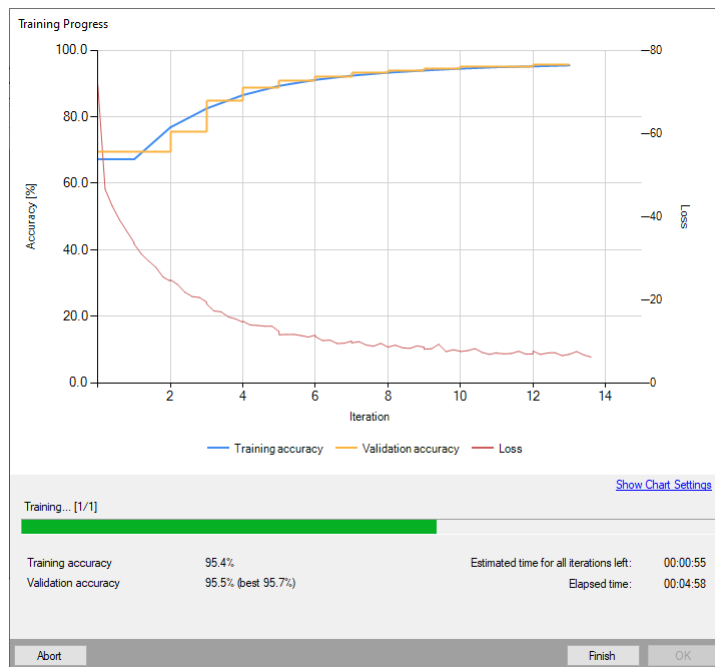
Details regarding augmentation parameters: [Deep Learning – Augmentation](#)

5. Performing training

During training, three main series are visible: training accuracy, validation accuracy, and loss. The training and validation charts should have a similar, increasing pattern. The loss chart should decrease.

More detailed information is displayed below the chart:

- current iteration number,
- current training statistics (training and validation accuracy),
- number of processed samples,
- elapsed time.



Training point location model.

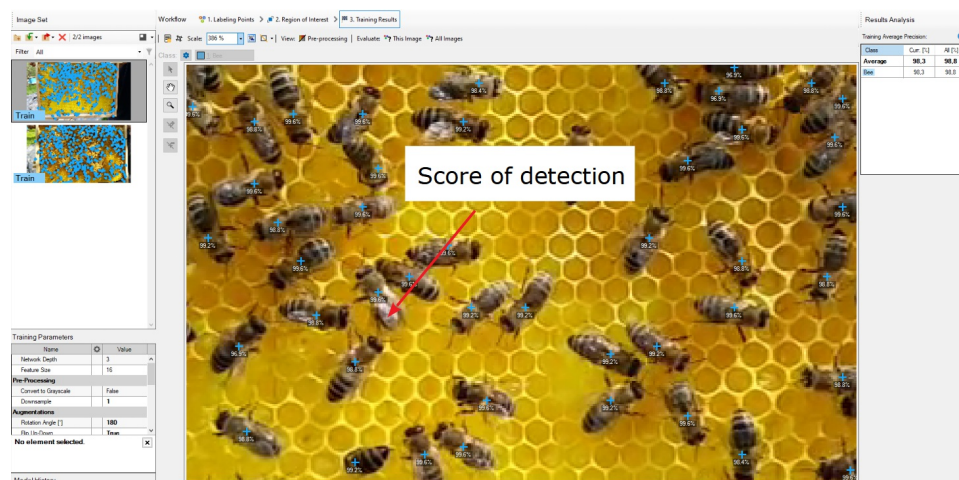
Training may be a long process. During this time, training can be stopped. If no model is present (first training attempt), the model with best validation accuracy will be saved. Consecutive training attempts will prompt the user whether to replace the old model.

6. Analyzing results

The window shows the results of point location. Detected points are displayed on top of the images. Each detection consists of the following data:

- visualized point coordinates,
- class (identified by a color),
- confidence score.

Evaluate: This Image and **Evaluate: All Images** buttons can be used to perform point location on the provided images. It may be useful after adding new training or test images to the data set or after changing the area of interest.



Point location results visualized after the training.

It is highly recommended to use data augmentations (appropriate to the task) to improve the network's ability to generalize learned information. If results are still not satisfactory, the following standard methods can be used to improve model performance:

- changing the feature size,
- providing more training data,
- increasing the number of training iterations,
- increasing the network depth.

Locating objects

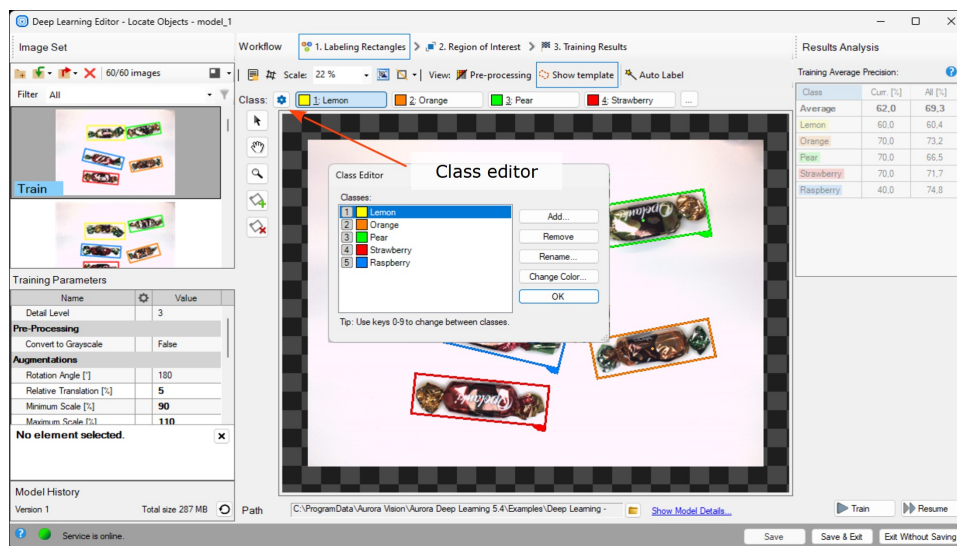
In this tool, a user needs to draw rectangles bounding the objects in the scene and specify their classes. These images and rectangles are used to train a model to locate and classify objects in the input images. This tool doesn't require from a user to mark the objects as precisely as it is required for segmenting instances.

1. Defining classes

First, a user needs to define classes of objects that the model will be trained on and later used to detect. Object location model can deal with single as well as multiple classes of objects.

The class editor is available under the **Class Editor** button.

To manage classes, the **Add**, **Remove**, or **Rename** buttons can be used. The color of each class can be changed using the **Change Color** button.



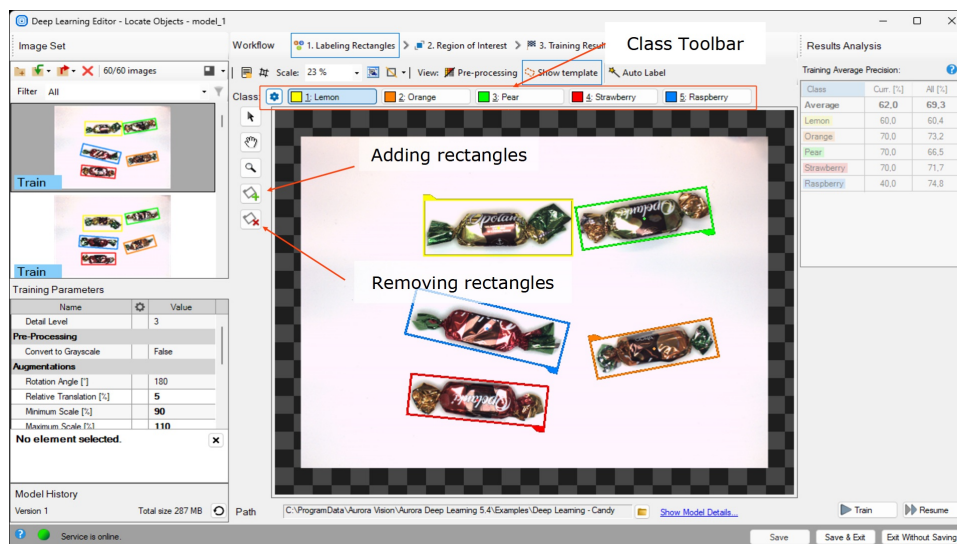
Using Class Editor.

2. Marking bounding rectangles

After adding training images and defining classes, a user needs to mark rectangles in images.

To mark an object, a user needs to click on a proper class from the Class Toolbar and click the **Creating Rectangle** button. Rectangles have the same color as previously defined for the selected class.

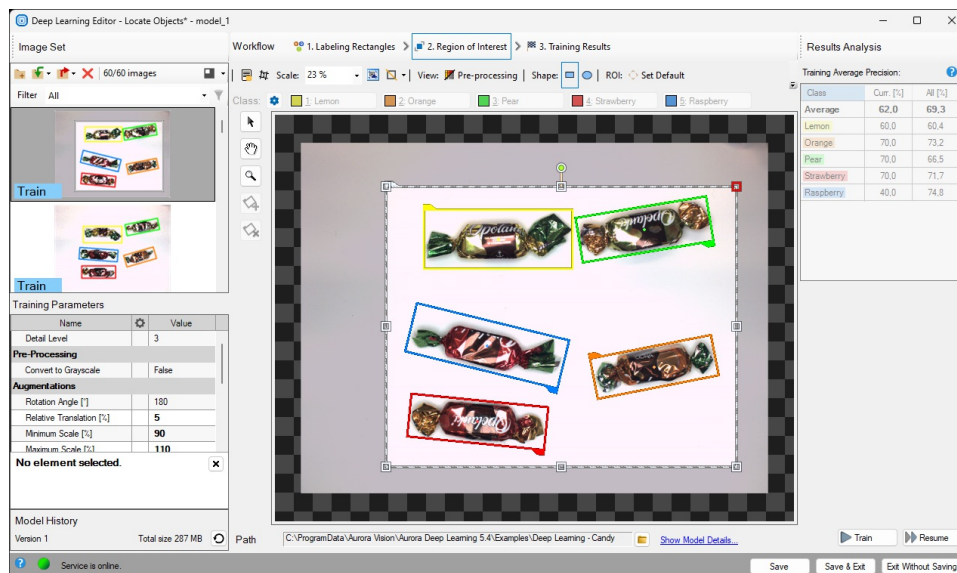
A rectangle can be selected directly on the image area. A selected rectangle can be moved, rotated, and resized to fit it to the object, removed (**Remove Region** button) or has its class changed (**Right-click on the rectangle » Change Class** button).



Marking rectangles.

3. Reducing region of interest

Reduce the region of interest to focus only on the important part of the image and to speed up the training process. By default, the region of interest contains the whole image.



Changing region of interest.

4. Setting training parameters

- **Detail level** – level of detail needed for a particular classification task (only for L01 approach). For the majority of cases, the default value of 3 is appropriate, but if images of different classes are distinguishable only by small features, increasing the value of this parameter may improve classification results.
- **Localization Mode** – defines how algorithm determines object orientation (only for L02 approach). Simpler algorithm will be faster.
- **Postprocessing parameters** – remove false positives from results (only for L02 approach).
- **Stopping conditions** – define when the training process should stop.

For more details, read [Deep Learning – Setting parameters](#).

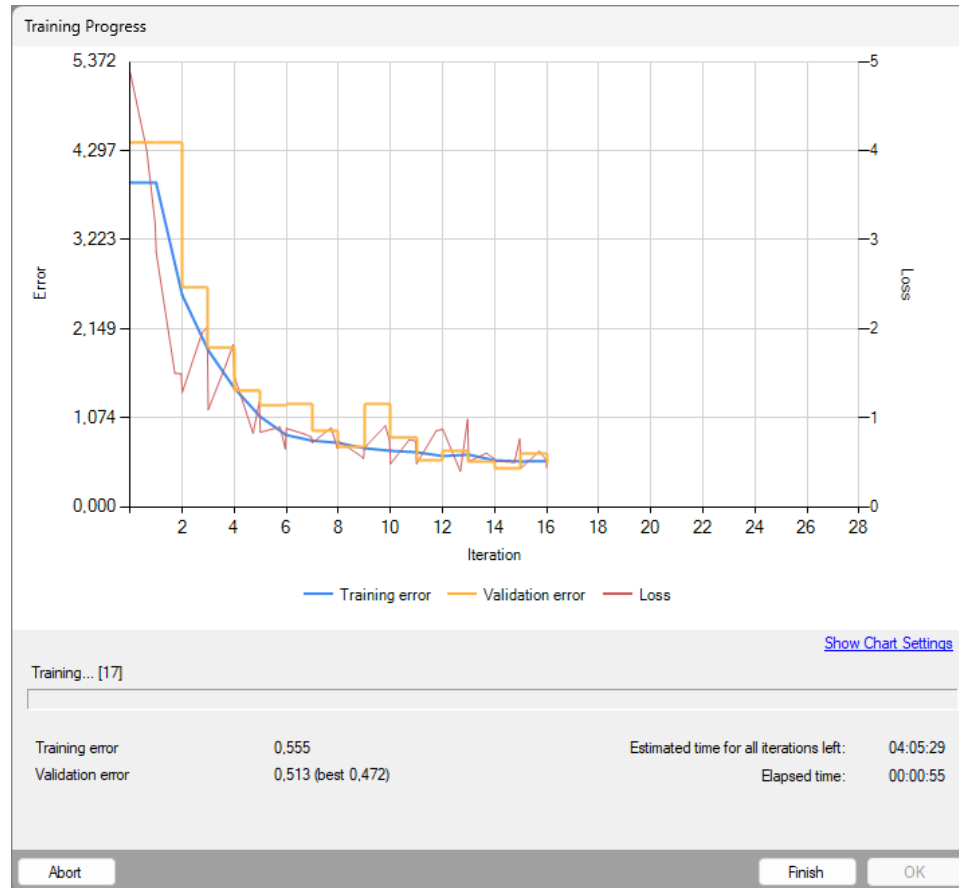
Details regarding augmentation parameters: [Deep Learning – Augmentation](#).

5. Performing training

During training, three main series are visible: training error, validation error, and loss. All charts should have a similar decreasing pattern.

More detailed information is displayed below the chart:

- current iteration number,
- current training statistics (training and validation error),
- number of processed samples,
- elapsed time.



Training object location model.

Training may be a long process. During this time, training can be stopped. If no model is present (first training attempt), the model with best validation accuracy will be saved. Consecutive training attempts will prompt the user whether to replace the old model.

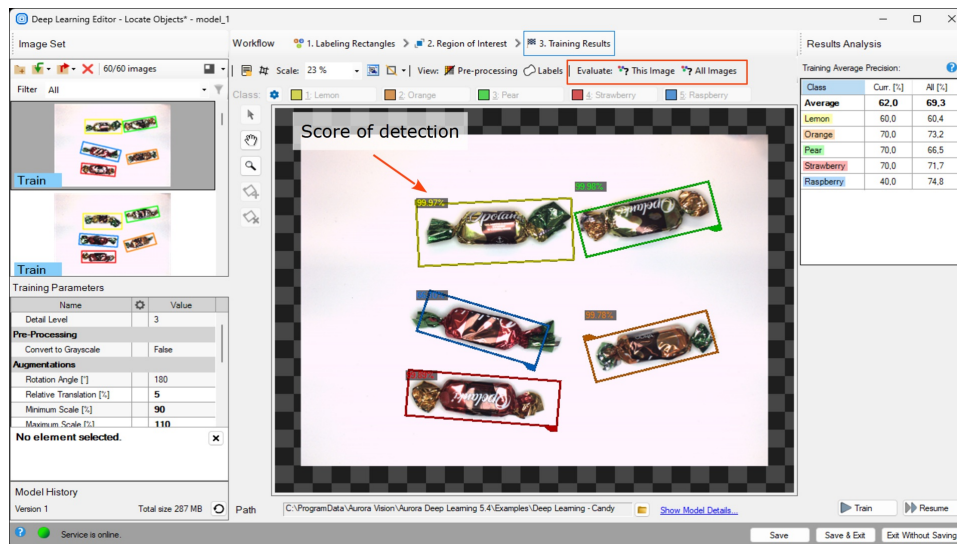
Warning: Training L01 models can be memory-intensive, particularly for complex models. Ensure sufficient system memory is available before beginning the training process.

6. Analyzing results

The window shows the results of object location. Detected rectangles are displayed around the objects. Each detection consists of the following data:

- visualized rectangle (object) coordinates,
- class (identified by a color),
- confidence score.

Evaluate: This Image and **Evaluate: All Images** buttons can be used to perform object location on the provided images. It may be useful after adding new training or test images to the data set or after changing the area of interest.

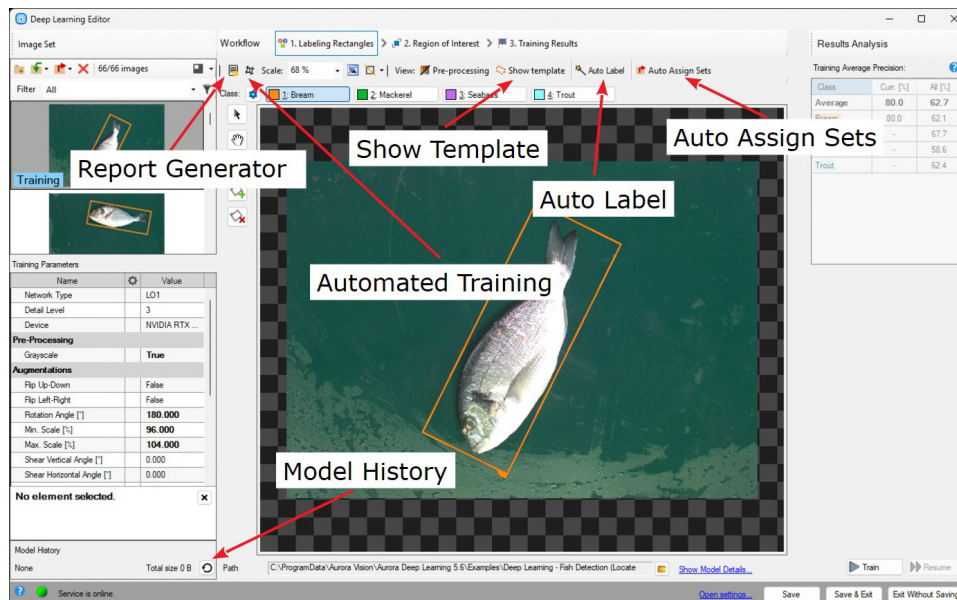


Object location results visualized after the training.

It is highly recommended to use data augmentations (appropriate to the task) to improve the network's ability to generalize learned information. If results are still not satisfactory the following standard methods can be used to improve model performance:

- changing the detail level,
- providing more training data,
- increasing the number of training iterations or extending the duration of the training.

Tips and tricks



Location of useful functions

Auto label

It is a helpful feature when operating on bigger datasets, when a lot of labeling is necessary. First, you need to properly label at least 2 images (10 is recommended) and train the model with them. If the results are satisfactory, you can add more images to your training set, select them all, and press **Auto Label** in the Deep Learning Editor.

For **DL_DetectAnomalies2** models, the operations need to be performed only once. For the other models, the operations need to be performed for each created class. Remember to always check if the masks are properly selected and correct any mistakes before continuing the training.

Auto assign sets

As of version 5.4, some of the tools may require Validation set for the training. If you are not sure which images should be picked, you can press the **Auto Assign Sets** button. That way, from your training set, some validation images will be chosen randomly.

Show template

This option assists in marking objects in **DL_LocateObjects** models. After marking the first object, its outline will appear as a template when marking subsequent objects of the same class on the same image.

Model history

It is possible to access previously trained models. You can do so by pressing the looped arrow icon in the bottom left corner of the Deep Learning Editor. If you right click on a selected model, you will be able to select it instead of the current one. From that level, it is also possible to remove the models you do not need.

Report generator

It is possible to generate a report of currently used model and view it, as long as the path to the images remains unchanged. To do so, simply press the report button in the editor and specify the location that you want it to be saved to.

Automated Training

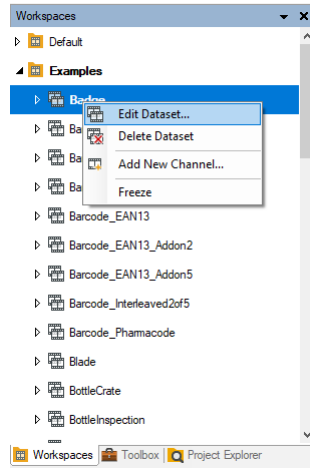
For the tutorial on this tool, refer to the [Automated Training](#) section in this article.

See also:

- [Machine Vision Guide: Deep Learning](#) – Deep Learning technique overview,
- [Deep Learning – Getting Started](#) – installation and configuration of Aurora Vision Deep Learning.

Managing Workspaces

Workspace window enables user a convenient way to store datasets grouped by this same category or purpose. For example single workspace may be created for a single project like "Part Inspection" and each included dataset may represent inspection from different day or images of the different part types.



Example Workspace window with examples dataset.

Using Filmstrip Control

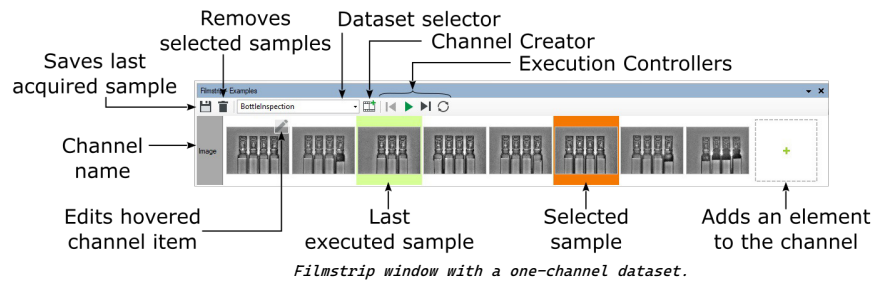
1. [Toolbox](#)
 2. [Setting Basic Properties](#)
 3. [Editing Geometrical Primitives](#)
 4. [Testing Parameters in Real Time](#)
 5. [Linking or Loading Data From a File](#)
 6. [Labeling Connections](#)
 7. [Invalid Connections](#)
 8. [Property Outputs](#)
 1. [Additional Property Outputs](#)
 9. [Expanded Input Structures](#)
 10. [Comment Blocks](#)
 11. [Extracting Macrofilters \(The Quick Way\)](#)
 12. [Creating Macrofilters in the Project Explorer](#)
 1. [Using a Project ID to protect an application](#)
 13. [Trick: Configuration File as a Module Not Exported to .avexe](#)
 14. [Macrofilter Counter](#)
 15. [Global Parameters](#)
 16. [Modules](#)
 17. [Importing Modules](#)
 18. [Locking Modules](#)
 19. [Table of Contents](#)
- [Introduction](#)
 - [Workflow](#)
 - [Detecting anomalies 1 \(deprecated\)](#)
 - [Detecting anomalies 2 \(classification-based approach\)](#)
1. [4. Setting training parameters](#)
- [Detecting features \(segmentation\)](#)
 - [Classifying objects](#)
 - [Segmenting instances \(deprecated\)](#)
 - [Locating points](#)
 - [Locating objects](#)

- [Tips and tricks](#)

1. [Overview](#)
2. [Common Tasks](#)
3. [See Also](#)

Overview

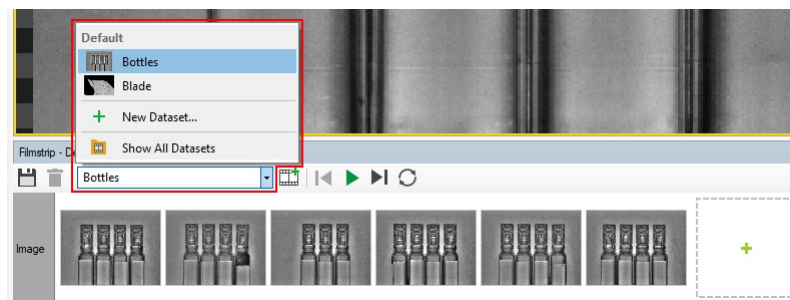
Filmstrip control is a powerful tool for controlling the execution of the program in the [Offline Mode](#) where the Filmstrip data is accessed with the [ReadFilmstrip](#) filters and through the [bound outputs](#) of the [Online-Only](#) filters.



The data presented in the Filmstrip control is arranged in the grid layout, where the rows represents the Channels and the columns represents the [Samples](#).

Additionally, the control enables most common operations over the current workspace, i.e., adding Datasets and Channels.

Changing the current dataset is as easy as selecting one from the datasets combo box:



To keep the program consistent, the to be selected dataset must contain channels with the same names as channels bound in the current Worker Task.

Common Tasks

- Dragging a channel from the Filmstrip control onto the Program Editor empty area inserts the [ReadFilmstrip](#) filter assigned to that channel,
- Dragging a channel from the Filmstrip control onto the filter instance output binds the output with that channel, as long as:
 - The filter is the [Online-Only](#) filter,
 - The filter is in the ACQUIRE section of the Worker Task,
 - The output data type is the same as the channel data type.
- Dragging files onto the Filmstrip control empty area creates a new Channel with the dragged files included.
- Dragging files onto the existing channel within the Filmstrip control appends the dragged data to that channel, if only the dragged data type match the channel type.
- Double-click on the Filmstrip sample executes one program iteration with the clicked sample. Requirements:
 - The Offline mode is active
 - There is at least one channel assigned in the [Single-Threaded application's](#) Worker Task or in the [Multi-Threaded application's](#) Primary Worker Task

See Also

1. [Managing Workspaces](#) – extensive description how to manage dataset workspaces in Aurora Vision Studio.
2. [Offline Mode](#) – the mode that enables access to the Channel data items.

Zebra
AuroraTM Vision

This article is valid for version 5.7.1

[Legal Terms of Use](#) [Privacy Policy](#)

