Zebra
Aurora™ Vision

Aurora Vision Library 5.3

Technical Issues

Created: 6/8/2023

Product version: 5.3.4.94078

Table of content:

- Interfacing with Other Libraries
- Loading Aurora Vision Studio Files (AVDATA)
- Working with GenICam GenTL Devices
- Processing Images in Worker Thread
- Troubleshooting
- Memory Leak Detection in Microsoft Visual Studio
- ATL Data Types Visualizers
- Optimizing Image Analysis for Speed
- Libraries comparison
- Deep Learning Training API

Interfacing with Other Libraries

Aurora Vision Library contains the avl::Image class which represents an image. This article describes how to create an avl::Image object with raw data acquired from cameras, and how to convert it to image structures specific to other libraries.

Aurora Vision Library provides a set of sample converters. To use it in your program you should include a specific header file which is available in Aurora Vision Library include directory (e.g. AVLConverters/AVL_OpenCV.h). The list below presents all the available converters:

- Euresys
- MFC
- MvAcquire
- OpenCV
- Pylon
- QT
- SynView

An example of using MFC converters can be found in the Aurora Vision Library directory in My Documents (Examples\MFC Examples). Below is shown also an OpenCV converter example.

Example: Converting Between avl::Image and OpenCV Mat

It is also possible to convert avl::Image to image structures from common libraries. The example code snippets below show how to convert an avl::Image object to other structures.

```
#include <opencv2/highgui/highgui.hpp>
#include <AVLConverters/AVL_OpenCV.h>

#include <AVL.h>

avl::Image inputImage, processedImage;
cv::Mat cvImage;

int thresholdValue, rotateAngle;

//image processing
void ProcessImage()
{
 avl::Image image1;
 avl::ThresholdImage(inputImage, atl::NIL, (float)thresholdValue, atl::NIL, 0.0, image1);
 avl::RotateImage(image1, (float)rotateAngle, avl::RotationSizeMode::Fit,
  avl::InterpolationMethod::Bilinear, false, processedImage);
}

// callback
void on_trackbar(int, void*)
{
 ProcessImage();

 avl::AvlImageToCVMat_Linked(processedImage, cvImage);
 cv::imshow("CV Result Window", cvImage);
}

int main(void)
{
 // Load AVL image
 avl::Image monoImage, rgbImage;
 avl::TestImage(avl::TestImageId::Lena, rgbImage, monoImage);
 avl::DownsampleImage(monoImage, 1, inputImage);
 thresholdValue = 128;
 rotateAngle = 0;

 // Create OpenCV Gui
 cv::namedWindow("Settings Window", 1);
 cv::resizeWindow("Settings Window", 300, 80);
 cv::createTrackbar("Threshold", "Settings Window", &thresholdValue, 255, on_trackbar);
 cv::createTrackbar("Rotate", "Settings Window", &rotateAngle, 360, on_trackbar);

 // set trackbar
 on_trackbar(0, 0);

 cv::waitKey(0);
 return 0;
}
```

Example: avl::Image from pointer to image data

It is also possible to create an avl::Image object using a pointer to image data, without copying memory blocks. This, however, requires compatible memory representations of images and proper information about the image being created has to be provided.

The constructor shown below should be used for this operation:

```
Image::Image(int width, int height, int pitch, PlainType::Type type, int depth, void* data,
 atl::Optional&lt; const avl::Region&amp; &gt; inRoi = atl::NIL);
```

Please note that all of the XxxToXxx_Linked functions do not copy data and the user has to take care of freeing such data. See also the usage example in OpenCV converter above. Functions AvlImageToCVMat_Linked and CVMatToAvlImage_Linked do not copy data.

Displaying Images Directly on WinAPI/MFC Device Context (HDC)

For convenience, there is also a function that directly displays an image on a WinAPI device context (HDC). This function is defined in the header "AVLConverters/AVL_Winapi.h" as:

```
void DisplayImageHDC(HDC inHdc, avl::Image& inImage, float inZoomX = 1.0, float inZoomY = 1.0);
```

For sample program showing how to use this function, please refer to the official example in the "06 WinAPI tutorial" directory.

Loading Aurora Vision Studio Files (AVDATA)

Aurora Vision Studio has its own format for storing arbitrary objects - the AVDATA format. It is used for storing elements of the program (paths, regions etc.) automatically, or manually when using "Export to AVDATA file" option or the **SaveObject** and **LoadObject** generic filters.

Aurora Vision Library can load and save several types of objects in AVDATA format. This is done using dedicated functions, two corresponding for each supported type. The functions start with **Load** and **Save** and accept two parameters - a filename and an object reference - for loading or saving.

```
void LoadRegion
(
 const File& inFilename,  //:Name of the source file
 Region& outRegion  //:Deserialized output Region
);
```

```
void SaveRegion
(
 const Region& inRegion,  //:Region to be serialized
 const File& inFilename  //:Name of the target file
);
```

The supported types include:

- Region
- Profile
- Histogram
- SpatialMap
- EdgeModel
- GrayModel
- OcrMlpModel
- OcrSvmModel
- Image*

Because the **LoadImage** function is a more general mechanism for saving and loading images into common file formats (like BMP, JPG or PNG), the functions for loading and saving `avl::Image` as AVDATA are different:

```
void LoadImageObject
(
 const File& inFilename,  //:Name of the source file
 Image& outImage  //:Deserialized output Image
);
```

```
void SaveImageObject
(
 const Image& inImage,  //:Image to be serialized
 const File& inFilename  //:Name of the target file
)
```

Simple types like **Integer**, **Real** or **String** can be stored in files in textual form - by setting **inStreamMode** to *Text* when using **SaveObject** - this can be read by formatted input output in C/C++ (for example using functions from the `scanf` family).

Working with GenICam GenTL Devices

Introduction

GenICam GenTL is a standard that defines a software interface encapsulating a transport technology and that allows applications to communicate with general vision devices without prior knowledge of its communication protocol. GenTL supporting application (a GenTL consumer) is able to load a third party dynamic link library (a GenTL provider) that is a kind of a "driver" for a vision device. GenICam standard allows to overcome differences with communication protocols and technologies, and allows to handle different devices in same common way. However application still needs to be aware of differences in device capabilities and be prepared to cooperate with specific device class or device model.

Aurora Vision Library contains a built-in GenTL subsystem that helps and simplifies usage of a GenTL device in vision application. AVL GenTL subsystem helps in loading provider libraries, enumerating GenTL infrastructure, managing acquisition engine and frame buffers, converting image formats and implements GenAPI interface.

In order to be able to use a GenTL provider it needs to be properly registered (installed) in local system. Usually this task is performed by an installer supplied by a device vendor. Please note that a 32bit application requires a 32 bit provider library and a 64 bit application requires respectively a 64 bit provider library. A registered GenTL provider is characterized by a file with ".cti" extension. Path to cti library containing folder is stored in an environmental variable named "GENICAM_GENTL32_PATH" ("GENICAM_GENTL64_PATH" for 64 bit providers).

Basic Usage

Functions designed for GenTL support can be found in GenTL and GenApi categories. A basic application will first use a GenTL_OpenDevice function to open a device instance (to establish the connection) and to request a handle for further operations on the device. This handle can be than used with GenApi functions to access device specific configuration and manage them. When the device identifiers are not fully known, or can dynamically change at runtime a GenTL_FindDevices function can be first used to enumerate available GenTL devices.

To start streaming video out of configured device a GenTL_StartAcquisition function must be executed. After this sequentially upcoming images can be retrieved with GenTL_ReceiveImage or GenTL_TryReceiveImage functions. Images will be stored in an input FIFO queue. Not retrieved images (on queue overflow) will be dropped starting from the oldest one. To stop image acquisition a GenTL_StopAcquisition function should be called. Image acquisition can be stopped and than started again multiple times for same device with eventual configuration change in between (some parameters can be locked for time of image streaming).

To release the device instance its handle need to be closed with GenTL_CloseHandle function.

Advanced Usage

When more information need to be known about GenTL environment its structure can be explored using GenTL_EnumLibraries, GenTL_GetLibraryDescriptor, GenTL_EnumLibraryInterfaces, GenTL_GetInterfaceDescriptor functions.

When extended information or configuration, specific for GenTL provider or transport technology need to be accessed, following functions can be considered: GenTL_OpenLibrarySystemModuleSettings, GenTL_OpenInterfaceModuleSettings, GenTL_OpenDeviceModuleSettings, GenTL_OpenDeviceStreamModuleSettings.

Additional Requirements

When using GenTL subsystem of Aurora Vision Library a "Genicam_Kit.dll" file is required to be in range of application. This file (selected for 32/64 bit) can be found in Aurora Vision Library SDK "bin" directory.

Processing Images in Worker Thread

Introduction to the Problem

Aurora Vision Library is a C++ library, that is designed for efficient image processing in C++ applications. A typical application uses a single primary thread for the user interface and can optionally use additional worker threads for data processing without freezing the main window of the application. Images processing can be a time-consuming task, so performing it in a separate worker thread is recommended, especially for processing performed in continuous mode.

Processing images in a worker thread is asynchronous and it means that accessing the resources by the worker thread and the main thread has to be coordinated. Otherwise, both threads could access the same resource at the same time, what would lead to unpredictable data corruption. The typical resource that has to be protected to be thread-safe is the image buffer. Typically, the worker thread of the vision application has a loop. In this loop it grabs images from a camera and does some kind of processing. Images are stored in memory of a buffer as *avl::Image* data. The main thread (UI thread) presents the results of the processing and/or images from the camera. It has to be ensured that the images are not read by the UI thread and processed by the worker thread at the same time.

Please note that the GUI controls should never be accessed directly from the worker thread. To display the results of the worker thread processing in the GUI, a resource access control has to be used.

Example Application and Image Buffer Synchronization

This article does not present the rules of multithreaded programming. It only focuses on the most typical aspects of it, that can be met when writing applications with Aurora Vision Library. An example application that uses the main thread and the worker thread can be found among the examples distributed with Aurora Vision Library. It is called *MFC Simple Streaming* and the easiest way to open it is by opening *Examples* directory of Aurora Vision Library from the *Start Menu*. The application is located in *03 GigEVision tutorial* subdirectory. It is a good template for other vision applications processing images in a separate thread. It is written using *MFC*, but the basics of multithreading stay the same for all other technologies.

There are many techniques of synchronization of a shared resources access in a multithreading environment. Each of them is good as long as it protects the resources in all states that the application can be in and as long as it properly handles thrown exceptions, application closing etc.

In the example application, the main form of the application has a private field called *m_videoWorker* that represents the worker thread:

```
class ExampleDlg : public CDialog
{
private:
 (...)
 GigEVideoWorker m_videoWorker;
 (...)
}
```

The *GigEVideoWorker* class contains the image buffer:

```
class GigEVideoWorker
{
 (...)
private:
 avl::Image m_imageBuffer;
 (...)
}
```

This is the image buffer that contains the image received from the camera that needs to be protected from parallel access from worker thread and from the main thread that displays the image in the main form. The access synchronization is internally achieved using critical section and *EnterCriticalSection* and *LeaveCriticalSection* functions of the Windows operating system. When one thread calls the *GigEVideoWorker::LockResults()* function, it enters the critical section and no other thread can access the image buffer until the thread that got the lock calls *GigEVideoWorker::UnlockResults()*. When one thread enters the critical section, other threads that try to enter the critical section will be suspended (blocked) until the one leaves the critical section.

Using functions like *GigEVideoWorker::LockResults()* and *GigEVideoWorker::UnlockResults()* is a good choice for protecting the image buffer from accessing by multiple threads, but what if due to an error in the code the resource is locked but never unlocked? It can happen for example in a situation when an exception is thrown inside the critical section and the code lacks the *try/catch* statement in the function that locks and should unlock the resource. In the example application this problem has been resolved using the *RAII* programming idiom. *RAII* stands for *Resource Acquisition Is Initialization* and in short it means that the resource is acquired by creating the synchronization object and is released by destroying it. In the example application being described here, there is the class called *VideoWorkerResultsGuard*. It exclusively calls the previously mentioned *GigEVideoWorker::LockResults()* and *GigEVideoWorker::UnlockResults()* functions in constructor and destructor. The instance of this *VideoWorkerResultsGuard* class is the synchronization object. The code of the class is listed below.

```
class VideoWorkerResultsGuard
{
private:
 GigEVideoWorker& m_object;

 VideoWorkerResultsGuard( const VideoWorkerResultsGuard& ); // = delete

public:
 explicit VideoWorkerResultsGuard( GigEVideoWorker& object )
 : m_object(object)
 {
  m_object.LockResults();
 }

 ~VideoWorkerResultsGuard()
 {
  m_object.UnlockResults();
 }
};
```

It can be easily seen that when the object of *VideoWorkerResultsGuard* is created, the thread that creates it calls the *LockResults()* function and by that it enters the critical section protecting the image buffer. When the object is destroyed, the thread leaves the critical section. Please note that the destructor of every object is automatically called in C++ when the automatic variable goes out of scope. It also covers the cases, when the variable goes out of scope because of the exception thrown from within of the critical section. Using *RAII* pattern allows programmer to easily synchronize the access to shared resources from multiple threads. When a thread needs to access a shared image buffer, it has to create the *VideoWorkerResultsGuard* object and destroy it (or let it be destroyed automatically when the object goes out of scope) when the access to the image buffer is no longer needed. The example usage of this synchronization looks as follows:

```
// Retrieve the results under lock.
{
 VideoWorkerResultsGuard guard(m_videoWorker);
 (...)
 avl::AVLImageToCImage(m_videoWorker.GetLastResultData(), width, height, false, m_lastImage);
 (...)
}
```

The method *GetLastResultData()* returns the reference to the shared image buffer. It can be safely used thanks to the usage of *VideoWorkerResultsGuard* object.

Notifications about Image Ready to Display

Another issue that needs to be considered in a typical application that processes images and uses a worker thread is notifying the main thread that the image processed by the worker thread is ready to display. Such notifications can be implemented in several ways. The one that has been used in the example application is using system function *PostMessage()*. When the worker thread has the image ready for presentation, it copies it to the *m_lastResultData* buffer (this is the protected one) and posts the notification message to the main window of the application:

```
//
// TODO: Compute the result data and put them in the shared buffer (just copy the source image).
//
m_lastResultData = m_imageBuffer;

// Send notification message
if (PostMessage(m_hNotificationWindow, m_notificationMessage, 0, NULL))
{
 m_lastResultProcessed = false;
}
```

The message is received by the main (UI) thread. Once it's received, the main thread acquires the access to the shared image buffer by creating the *VideoWorkerResultsGuard* object. Then, the image can be safely displayed.

The worker thread has a flag called *m_lastResultProcessed*. The flag set to *false* indicates that the notification about image ready to display had been posted to the main thread but the main thread has not processed (displayed) the image yet. The flag is set to false just after posting the notification message. The main thread sets the flag back to *true* using *NotificationGiveFeedback()* function:

```
void GigEVideoWorker::NotificationGiveFeedback( void )
{
 VideoWorkerResultsGuard guard(*this);
 m_lastResultProcessed = true;
}
```

Once the worker thread has sent the notification message, it can acquire and perform the next frame from the camera, but there's no point in sending the next notification until the previous is performed by the UI thread. Sending the new notifications without performing the old ones could lead to cumulating them in the messages queue of the main window. This is why the worker thread of the example application checks if the previous notification message has been performed and sends the next one only if the processing of the previous is finished:

```
if (m_lastResultProcessed && NULL != m_hNotificationWindow)
{
 // Create the result in shared buffers under lock.
 VideoWorkerResultsGuard guard(*this);
 (...)
}
```

Please note that the flag is also protected by the *VideoWorkerResultsGuard* synchronization object, so the main thread cannot set it to *true* in the moment directly after the worker thread posted the notification message.

Issues of Multithreading

There are two primary issues to consider when using worker thread(s). The first one is destroying data by unsynchronized access from multiple threads and the second one is a deadlock that can appear when there are two (or more) resources to be synchronized.

Securing data integrity by the thread synchronization mechanisms has been shortly described in this article and is implemented in the example application distributed with Aurora Vision Library. As a rule of a thumb, please assume that every image that can be accessed from more then one thread should be protected by some kind of synchronization. We recommend the standard C++ *RAII* pattern as an easy to use and secure solution.

The example application described in this article contains only one resource – a critical section represented by the *VideoWorkerResultsGuard* class, but of course there may exist some applications where there is more then one resource to share. In such cases, the synchronization of the threads has to be implemented very carefully because there is a danger of deadlock that can be a result of bad implementation. If your application freezes (stops responding) and you have more then one synchronized resource, please review the synchronization code.

Troubleshooting

This article describes the most common problems that might appear when building and executing programs that use Aurora Vision Library.

Problems with Building

error LNK2019: unresolved external symbol _LoadImageA referenced in function
error C2039: 'LoadImageA' : is not a member of 'avl'

The problem is related to including the "windows.h" file. It defines a macro called *LoadImage*, which has the same name as one of the functions of Aurora Vision Library. Solution:

- Don't include both "windows.h" and "AVL.h" in a single compilation unit (cpp file).
- Use `#undef LoadImage` after including "windows.h".

error LNK1123: failure during conversion to COFF: file invalid or corrupt

If you encounter this problem, just disable the incremental linking (properties of the project | *Configuration Properties* | *Linker* | *General* | *Enable Incremental Linking*, set to *No (/INCREMENTAL:NO)*). This is a known issue of VS2010 and more information can be found on the Internet. Installing VS2010 Service Pack 1 is an alternative solution.

Exceptions Thrown in Run Time

Exception from the *avl* namespace is thrown

Aurora Vision Library uses exceptions to report errors in the run-time. All the exceptions are defined in *avl* namespace and derive from *avl::Error*. To solve the problem, add a *try/catch* statement and catch all *avl::Error* exceptions (or only selected derived type). Every *avl::Error* object has the *Message()* method which should provide you more detailed information about the problem. Remember that a good programming practice is catching C++ exceptions by a const reference.

```
try
{
    // your code here
}
catch (const atl::Error& er)
{
    cout << er.Message();
}
```

High CPU Usage When Running AVL Based Image Processing

When working with some AVL image processing functions it is possible that the reported CPU usage can reach 50~100% across all CPU cores even in situations when the actual workload does not justify that hight CPU utilization. This behavior is a side effect of a parallel processing back-end worker threads actively waiting for the next task. Although the CPU utilization is reported to be high those worker threads will not prevent other task to be executed when needed, so this behavior should not be a problem in most situations.

For situations when it is not desired this behavior can be changed (e.g. when profiling the application, performance testing or in any situation, when high CPU usage interfere with other system). To block the worker threads from idling for extended period of time the environment variable OMP_WAIT_POLICY must be set to the value PASSIVE, before the application is started:

```
set OMP_WAIT_POLICY=PASSIVE
```

This variable is checked when the DLLs are loaded, so setting it from the application code might not be effective.

Memory Leak Detection in Microsoft Visual Studio

When creating applications using Aurora Vision Library in Microsoft Visual Studio, it may be desirable to enable automated memory leak detection possible in Debug builds. The details of using this feature is described here: Finding Memory Leaks Using the CRT Library.

Some project types, notably MFC (Microsoft Foundation Classes) Windows application projects, have this mechanism enabled by default.

False Positives of Memory Leaks in AVL.dll

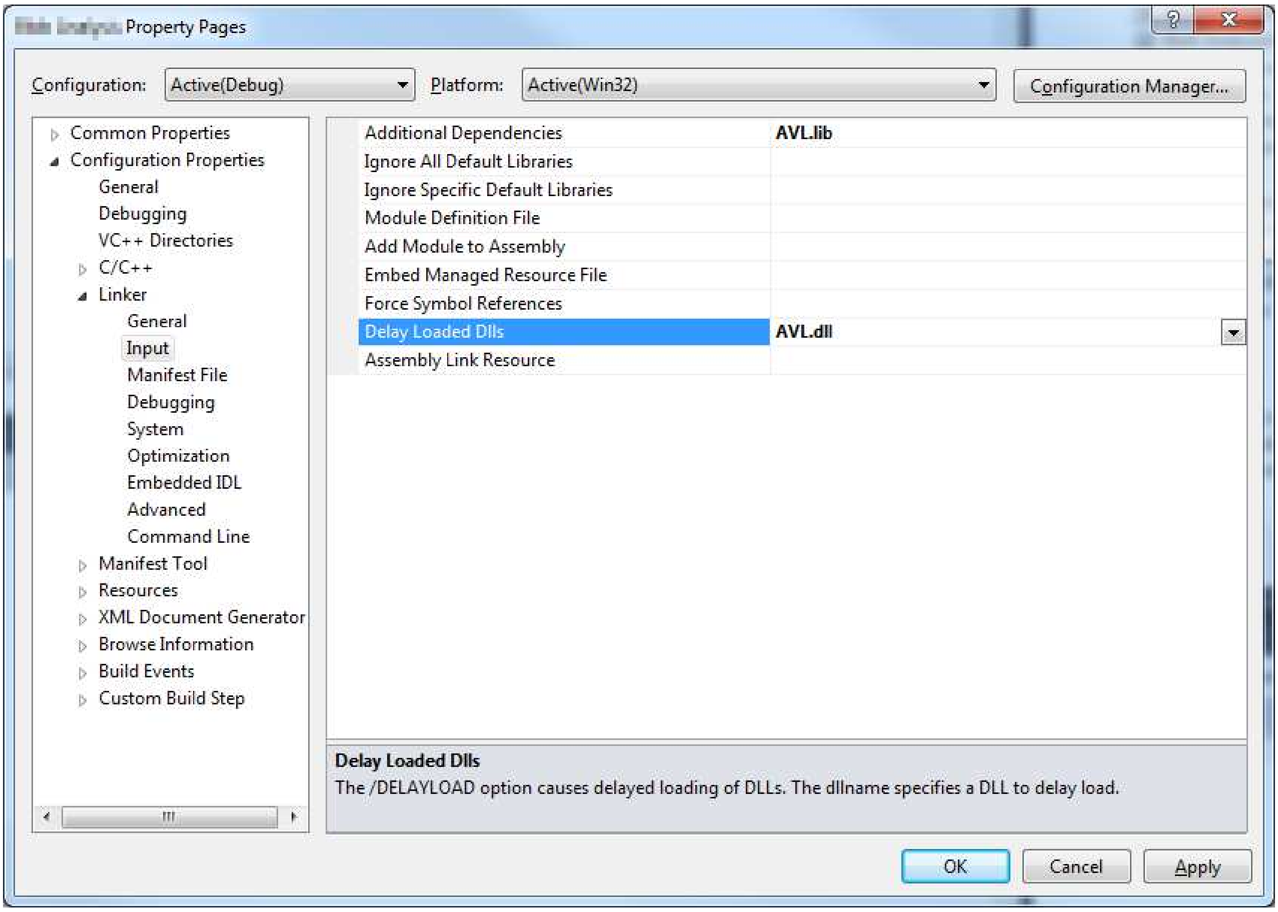
Using a default configuration, as described in Project Configuration can lead to false positives of memory leaks, which come from the AVL.dll library. The output of a finished program can look similar to the following:

```
(...)
The thread 'Win32 Thread' (0x898) has exited with code 0 (0x0).
The thread 'Win32 Thread' (0x168c) has exited with code 0 (0x0).
Detected memory leaks!
Dumping objects ->
{5573} normal block at 0x00453DB8, 8 bytes long.
 Data: <         > 01 00 00 00 00 00 00 00
{5572} normal block at 0x00453D68, 20 bytes long.
 Data: <D]NU          =E > 44 5D 4E 55 CD CD CD CD 02 00 00 00 B8 3D 45 00
{5571} normal block at 0x00453C18, 4 bytes long.
 Data: <X NU> 58 06 4E 55
(...)
```

These are not actual memory leaks, but internal resources of AVL.dll, which are not yet released when the memory leaks check is being run. Because there are many such allocated blocks reported, the actual memory leaks in your program can pass unnoticed.

Solution: Delayed Loading of AVL.dll

To avoid these false positives, AVL.dll should be configured to be delay loaded. This can be done in the Project Properties, under **Configuration Properties » Linker » Input**:



Further Consequences

With this configuration, your program will not try to load AVL.dll until it uses the first function from Aurora Vision Library. This will be also connected with license checking.

The program will stop if AVL.dll is missing: if AVL.dll was not delay loaded, this would happen at start time (the program would refuse to run). This allows the program to work without AVL.dll, and use it only when it is available. The availability of AVL.dll can be checked beforehand, using LoadLibrary or LoadLibraryEx functions.

ATL Data Types Visualizers

Data Visualizers

Data visualizers present data during the debugging session in a human-friendly form. Microsoft Visual Studio allows users to write custom visualizers for C++ data. Aurora Vision Library is shipped with a set of visualizers for the most frequently used ATL data types: *atl::String*, *atl::Array*, *atl::Conditional* and *atl::Optional*.

Visualizers are automatically installed during installation of Aurora Vision Library and are ready to use, but they are also available at *atl_visualizers* subdirectory of Aurora Vision Library installation path.

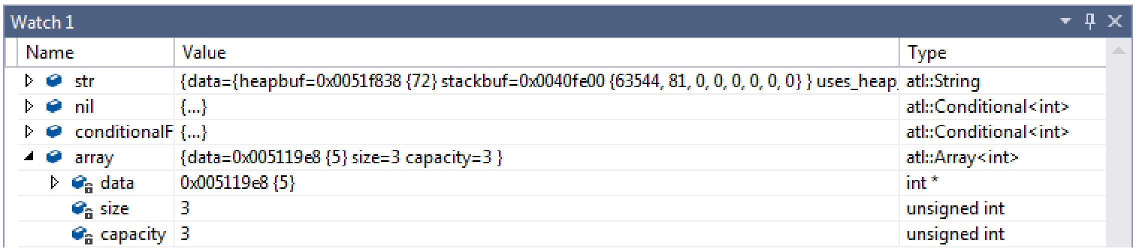
For more information about visualizers, please refer to the MSDN.

Example ATL data visualization

Please see the example variables definition below and their visualization without and with visualizers.

```
atl::String str = L"Hello world";
atl::Conditional nil = atl::NIL;
atl::Conditional conditionalFive = 5;
atl::Array array(3, 5);
```

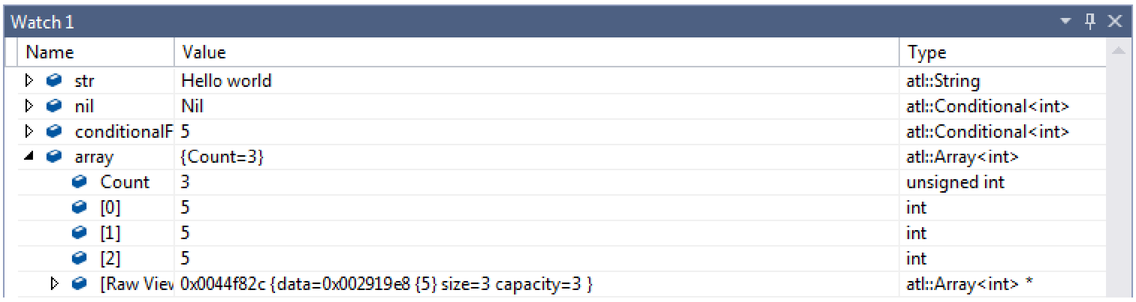
Data preview without ATL visualizers installed:



Watch 1

Name	Value	Type
str	{data={heapbuf=0x0051f838 {72} stackbuf=0x0040fe00 {63544, 81, 0, 0, 0, 0, 0, 0} } uses_heap	atl::String
nil	{...}	atl::Conditional<int>
conditionalF	{...}	atl::Conditional<int>
array	{data=0x005119e8 {5} size=3 capacity=3 }	atl::Array<int>
data	0x005119e8 {5}	int *
size	3	unsigned int
capacity	3	unsigned int

The same data presented using AVL visualizers:



Watch 1

Name	Value	Type
str	Hello world	atl::String
nil	Nil	atl::Conditional<int>
conditionalF	5	atl::Conditional<int>
array	{Count=3}	atl::Array<int>
Count	3	unsigned int
[0]	5	int
[1]	5	int
[2]	5	int
[Raw Viev	0x0044f82c {data=0x002919e8 {5} size=3 capacity=3 }	atl::Array<int> *

Image Watch extension

For Microsoft Visual Studio 2015, 2017 and 2019 an extension Image Watch is available. Image Watch allows to display images during debugging sessions in window similar to "Locals" or "Watch". To make Image Watch work correctly with *avl::Image* type, Aurora Vision Library installer provides avl::Image visualizer for Image Watch. If one have Image Watch extension and AVL installed, preview of images can be enabled by choosing "View->Other Windows->Image Watch" from Microsoft Visual Studio menu.

avl::Image description for Image Watch extension is included in *atl.natvis* file, which is stored in *atl_visualizers* folder in Aurora Vision Library installation directory. *atl.natvis* file is installed automatically during Aurora Vision Library installation.

When program is paused during debug session, all variables of type avl::Image can be displayed in Image Watch window, as shown below:

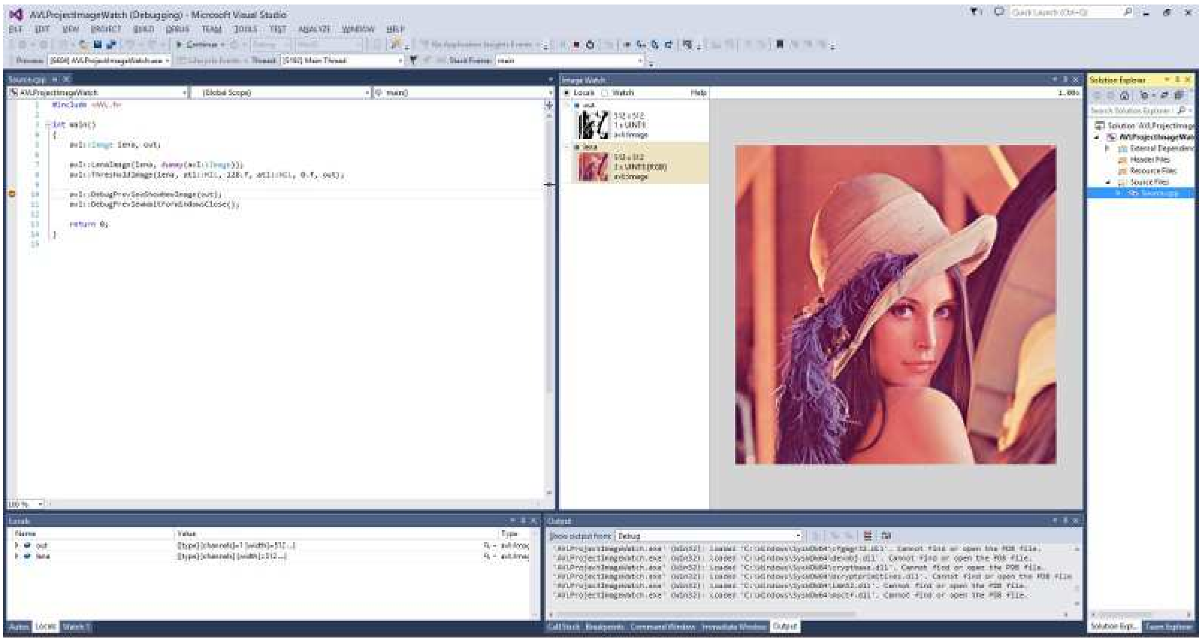


Image displayed inside Image Watch can be zoomed. When the close-up is large enough, decimal values of pixels' channel will be displayed. Hexadecimal values can be displayed instead, if appropriate option from context menu is selected.

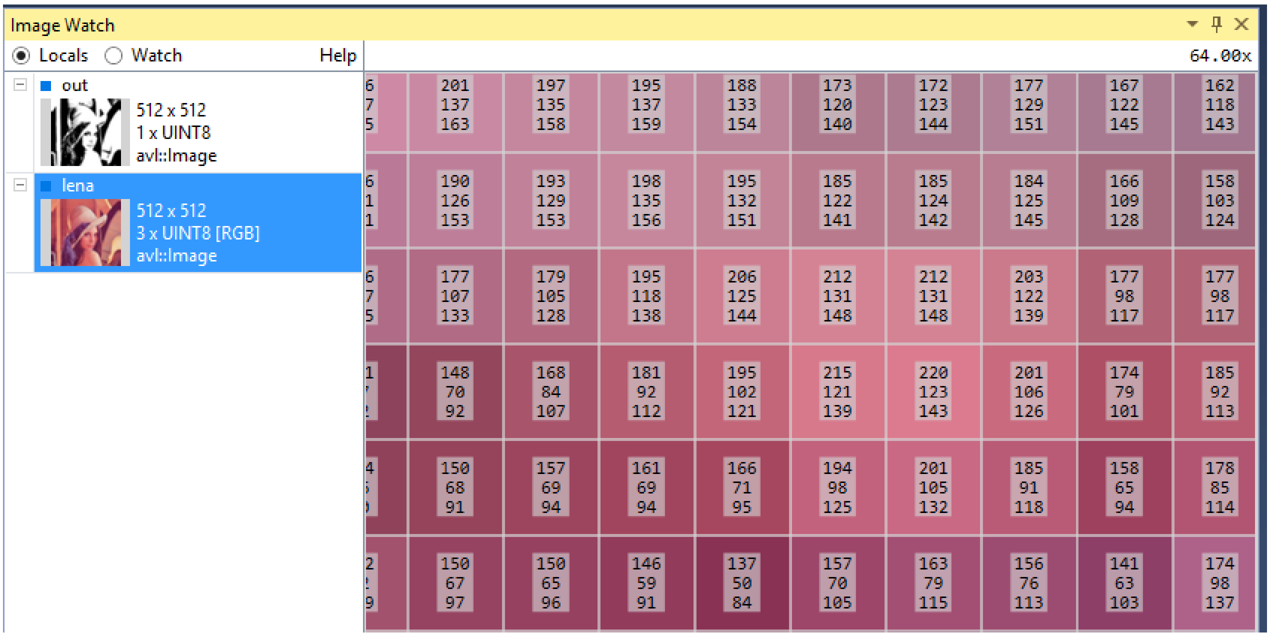


Image Watch is quite powerful tool - one can copy address of given pixel, ignore alpha channel and much more. All options are described in its documentation, which is accessible from the Image Watch site at:

- ImageWatch 2019 - for Microsoft Visual Studio 2019
- ImageWatch 2017 - for Microsoft Visual Studio 2017
- ImageWatch - for older versions of Microsoft Visual Studio

Optimizing Image Analysis for Speed

General Rules

Rule #1: Do not compute what you do not need.

- Use image resolution well fitted to the task. The higher the resolution, the slower the processing.
- Use the **inRoi** input of image processing functions to compute only the pixels that are needed in further processing steps.
- If several image processing operations occur in sequence in a confined region then it might be better to use CropImage at first.
- Do not overuse images of types other than UInt8 (8-bit).
- Do not use multi-channel images, when there is no color information being processed.
- If some computations can be done only once, move them before the main program loop, or even to a separate function.

Rule #2: Prefer simple solutions.

- Do not use Template Matching if more simple techniques as Blob Analysis or 1D Edge Detection would suffice.
- Prefer pixel-precise image analysis techniques (Region Analysis) and the *Nearest Neighbour* (instead of *Bilinear*) image interpolation.
- Consider extracting higher level information early in the program – for example it is much faster to process Regions than Images.

Rule #3: Mind the influence of the user interface.

- Note that displaying data in the user interface takes much time, regardless of the UI library used.
- Mind the Diagnostic Mode. Turn it off whenever you need to test speed. Diagnostic Mode can be turn off or on by EnableAvlDiagnosticOutputs function. One can check, if Diagnostic Mode is turned on by GetAvlDiagnosticOutputsEnabled function.
- Before optimizing the program, make sure that you know what really needs optimizing. Measure execution time or use a profiler.

Common Optimization Tips

Apart from the above general rules, there are also some common optimization tips related to specific functions and techniques. Here is a check-list:

- Template Matching: Prefer high pyramid levels, i.e. leave the **inMaxPyramidLevel** set to *atl::NIL*, or to a high value like between *4* and *6*.
- Template Matching: Prefer **inEdgePolarityMode** set not to Ignore and **inEdgeNoiseLevel** set to Low.
- Template Matching: Use as high values of the **inMinScore** input as possible.
- Template Matching: If you process high-resolution images, consider setting the **inMinPyramidLevel** to *1* or even *2*.
- Template Matching: When creating template matching models, try to limit the range of angles with the **inMinAngle** and **inMaxAngle** inputs.
- Template Matching: Consider limiting **inSearchRegion**. It might be set manually, but sometimes it also helps to use Region Analysis techniques before Template Matching.
- Do not use these functions in the main program loop: CreateEdgeModel1, CreateGrayModel, TrainOcr_MLP, TrainOcr_SVM.
- If you always transform images in the same way, consider functions from the Image Spatial Transforms Maps category instead of the ones from Image Spatial Transforms.
- Do not use image local transforms with arbitrary shaped kernels: DilateImage_AnyKernel, ErodeImage_AnyKernel, SmoothImage_Mean_AnyKernel. Consider the alternatives without the "_AnyKernel" suffix.
- SmoothImage_Median can be particularly slow. Use Gaussian or Mean smoothing instead, if possible.

Library-specific Optimizations

There are some optimization techniques that are available only in Aurora Vision Library and not in Aurora Vision Studio. These are:

In-Place Data Processing

See: In-Place Data Processing.

Re-use of Image Memory

Most image processing functions allocate memory for the output images internally. However, if the same object is provided in consecutive iterations and the dimensions of the images do not change, then the memory can be re-used without re-allocation. This is very important for the performance considerations, because re-allocation takes time which is not only significant, but also non-deterministic. Thus, it is highly advisable to move the image variable definition before the loop it is computed in:

```
// Slow code
while (...)
{
 Image image2;
 ThresholdImage(image1, atl::NIL, 128.0f, atl::NIL, 0.0f, image2);
}
```

```
// Fast code
Image image2;
while (...)
{
 ThresholdImage(image1, atl::NIL, 128.0f, atl::NIL, 0.0f, image2);
}
```

```
// Fast code (also in the first iteration)
Image image2(752, 480, PlainType::UInt8, 1, atl::NIL); // memory pre-allocation (dimensions must be known)
while (...)
{
 ThresholdImage(image1, atl::NIL, 128.0f, atl::NIL, 0.0f, image2);
}
```

Skipping Background Initialization

Almost all image processing functions of Aurora Vision Library have an optional **inRoi** parameter, which defines a region-of-interest. Outside this region the output pixels are initialized with zeros. Sometimes, when the rois are very small, the initialization might take significant time. If this is an internal operation and the consecutive operations do not read that memory, the initialization can be skipped by setting IMAGE_DIRTY_BACKGROUND flag in the output image. For example, this is how dynamic thresholding is implemented internally in AVL, where the out-of-roi pixels of the **blurred** image are not meaningful:

```
Image blurred;
blurred.AddFlags(IMAGE_DIRTY_BACKGROUND);
SmoothImage_Mean(inImage, inRoi, inSourceRoi, atl::NIL, KernelShape::Box, radiusX, radiusY, blurred);
ThresholdImage_Relative(inImage, inRoi, blurred, inMinRelativeValue, inMaxRelativeValue, inFuzziness, outMonoImage);
```

Library Initialization

Before you call any AVL function it is recommended to call the InitLibrary function first. This function is responsible for precomputing library's global data. If it is not used explicitly, it will be called within the first invocation of any other AVL function, taking some additional time.

Configuring Parallel Computing

The functions of Aurora Vision Library internally use multiple threads to utilize the full power of multi-core processors. By default they use as many threads as there are physical processors. This is the best setting for majority of applications, but in some cases another number of threads might result in faster execution. If you need maximum performance, it is advisable to experiment with the ControlParallelComputing function with both higher and lower number of threads. In particular:

- If the number of threads is **higher** than the number of physical processors, then it is possible to utilize the Hyper-Threading technology.
- If the number of threads is **lower** than the number of physical processors (e.g. 3 threads on a quad-core machine), then the system has at least one core available for background threads (like image acquisition, GUI or computations performed by other processes), which may improve its responsiveness.

Configuring Image Memory Pools

Among significant factors affecting function performance is memory allocation. Most of the functions available in Aurora Vision Library re-use their memory buffers between consecutive iterations which is highly beneficial for their performance. Some functions, however, still allocate temporary image buffers, because doing otherwise would make them less convenient in use. To overcome this limitation, there is the function ControlImageMemoryPools which can turn on a custom memory allocator for temporary images.

There is also a way to pre-allocate image memory before first iteration of the program starts. For this purpose use the InspectImageMemoryPools function at the end of the program, and – after a the program is executed – copy its **outPoolSizes** value to the input of a ChargeImageMemoryPools function executed at the beginning. In some cases this will improve performance of the first iteration of program.

Using GPGPU/OpenCL Computing

Some functions of Aurora Vision Library allow to move computations to an OpenCL capable device, like a graphics card, in order to speed up execution. After proper initialization, OpenCL processing is performed completely automatically by suitable functions without changing their use pattern. Refer to "Hardware Acceleration" section of the function documentation to find which functions support OpenCL processing and what are their requirements. Be aware that the resulting performance after switching to an OpenCL device may vary and may not always be a significant improvement relative to CPU processing. Actual performance of the functions must always be verified on the target system by proper measurements.

To use OpenCL processing in Aurora Vision Library the following is required:

- a processing device installed in the target system supporting OpenCL C language in version 1.1 or greater,
- a proper and up-to-date device driver installed in the system,
- a proper OpenCL runtime software provided by its vendor.

OpenCL processing is supported for example in the following functions: RgbToHsi, HsiToRgb, ImageCorrelationImage, DilateImage_AnyKernel.

To enable OpenCL processing in functions an AvsFilter_InitGPUProcessing function must be executed at the beginning of a program. Please refer to that function documentation for further information.

Libraries comparison

Introduction

The below table summarizes mutually corresponding functions between OpenCV, Halcon and Aurora Vision Library (AVL). Feel free to use this table when porting your program from one library to another. Please note, however, that in most cases the corresponding functions may not give the same results as their implementations are different in details. Quite often also the range of parameters exposed to the user will be different.

Most important general differences are:

- OpenCV does not have any data type for regions, so binary images must be used instead. Halcon has regions with unspecified implementation and which do not have specific outer dimensions. At the same time Aurora Vision Studio provides Region data type which is always RLE-encoded and has specific dimensions (width and height) in the same way as images have. Thus you can consider regions to be more efficiently encoded binary images.
- The representation of multi-channel images is very similar in OpenCV and in Aurora Vision Studio. It is so called interleaved representation, while Halcon focuses on planar images (each channel is represented in a separate memory fragment).
- The three libraries perform spatial image filters differently in areas close to the image borders. In OpenCV one can choose between different methods of extrapolating boundary pixels. Halcon always the boundary pixel is considered repeated beyond the image range. In Aurora Vision Studio, spatial filters are performed by considering only the pixels that are in the image range.

Image Acquisition

AVL provides dedicated support for GenICam and GigE Vision industrial standards, as well as for specific camera brands - using SDK from their manufacturers. See also See also Camera Support.

OpenCV	Halcon	AVL	Module	Comment
VideoCapture::open	grab_image_start	GenTL_StartAcquisition GigEVision_StartAcquisition	Genicam Genicam	
VideoCapture::release	close_framegrabber	GenTL_StopAcquisition GigEVision_StopAcquisition	Genicam Genicam	
VideoCapture::grab	grab_image	GenTL_ReceiveImage GigEVision_ReceiveImage	Genicam Genicam	
–	grab_image_async	GenTL_TryReceiveImage GigEVision_TryReceiveImage	Genicam Genicam	

Image Processing (Part I)

Here is a list of fundamental image transformations available in almost any library.

OpenCV	Halcon	AVL	Module	Comment
imread	read_image	LoadImage	FoundationLite	
imwrite	write_image	SaveImage	FoundationLite	
–	gen_image3	MakeImage	FoundationLite	
–	gen_image1_extern	MakeImage	FoundationLite	
–	get_grayval	GetImagePixel	FoundationLite	
addWeighted	add_image	AddImages	FoundationLite	
subtract	sub_image	SubtractImages	FoundationLite	
bitwise_and	paint_gray	ComposeImages	FoundationLite	
absdiff	abs_diff_image	DifferenceImage	FoundationLite	
divide	div_image	DivideImages	FoundationLite	
multiply	mult_image	MultiplyImages	FoundationLite	
add	scale_image	AddToImage	FoundationLite	
convertScaleAbs	scale_image	RescalePixels	FoundationLite	
cvSubRS	invert_image	NegateImage	FoundationLite	
convertScaleAbs	scale_image	MultiplyImage	FoundationLite	
max	max_image	MaximumImage	FoundationLite	
min	min_image	MinimumImage	FoundationLite	
resize, pyrDown	zoom_image_size	ResizeImage DownsampleImage	FoundationLite FoundationBasic	
resize	zoom_image_factor	ResizeImage_Relative	FoundationLite	
warpAffine	rotate_image	RotateImage	FoundationLite	
warpAffine	affine_trans_image	TranslateImage ShearImage	FoundationLite FoundationLite	
–	–	TranslatePixels	FoundationLite	
–	–	TransposeImage	FoundationLite	
–	crop_rectangle1	CropImage CropImageToRectangle	FoundationLite FoundationLite	
flip, transpose	mirror_image	MirrorImage TransposeImage	FoundationLite FoundationLite	
–	compose3	MergeChannels	FoundationLite	
–	rgb1_to_gray	AverageChannels_Weighted	FoundationLite	
–	rgb3_to_gray	AverageChannels_Weighted	FoundationLite	
–	decompose3	SplitChannels	FoundationLite	
–	access_channel	SplitChannels	FoundationLite	
cvtColor	trans_from_rgb	RgbToHsv HsvToRgb	FoundationLite FoundationLite	See also many other functions in Image Color Spaces
–	add_noise_white	AddNoiseToImage	FoundationLite	
calcHist	gray_histo	ImageHistogram	FoundationBasic	
–	draw_point	DrawPoint	FoundationLite	
rectangle	draw_rectangle1_mod	DrawRectangle	FoundationLite	
circle	draw_circle_mod	DrawCircle	FoundationLite	
–	paint_region	DrawRegions_MultiColor	FoundationLite	
–	set_draw	DrawRegion	FoundationLite	
–	–	JoinImages	FoundationLite	
–	gen_gauss_pyramid	CreateImagePyramid_Gauss	FoundationBasic	
–	–	ColorDistanceImage	FoundationPro	
–	binocular_disparity, binocular_distance	CompareImages ImageCorrelation ImageCorrelationImage	FoundationPro FoundationPro FoundationPro	
–	–	ExpaintImage_Bornemann ExpaintImage_Telea	FoundationPro FoundationPro	

Image Processing (Part II)

More advanced image processing tools, including spatial filtering and transformations.

OpenCV	Halcon	AVL	Module	Comment
cvSmooth	mean_image	SmoothImage_Mean	FoundationLite	
cvSmooth	gauss_image	SmoothImage_Gauss	FoundationLite	
–	gen_gauss_filter	SmoothImage_Gauss_Mask	FoundationLite	
–	smooth_image	SmoothImage_Deriche SmoothImage_Gauss SmoothImage_Gauss_Mask	FoundationLite FoundationLite FoundationLite	
cvSmooth	median_image	SmoothImage_Median SmoothImage_Median_Mask	FoundationLite FoundationLite	
–	midrange_image	SmoothImage_Middle	FoundationLite	
–	rank_image	SmoothImage_Quantile	FoundationLite	
–	bilateral_filter	SmoothImage_Bilateral	FoundationPro	
dilate	gray_dilation	DilateImage	FoundationLite	
erode	gray_erosion	ErodeImage	FoundationLite	
close	gray_closing_shape	CloseImage	FoundationLite	
open	gray_opening_shape	OpenImage	FoundationLite	
filter2D	convol_image	ConvolveImage	FoundationLite	
–	derivate_gauss	GradientImage	FoundationLite	
sobel	sobel_amp	GradientImage_Mask	FoundationLite	
sobel	sobel_dir	GradientImage_Mask	FoundationLite	
–	prewitt_amp	GradientImage_Mask	FoundationLite	
–	prewitt_dir	GradientImage_Mask	FoundationLite	
–	diff_of_gauss	DifferenceOfGaussians	FoundationBasic	
–	emphasize	SharpenImage	FoundationLite	
–	illuminate	NormalizeLocalContrast	FoundationBasic	
inpaint	inpainting_texture	InpaintImage	FoundationPro	
–	inpainting_ct	InpaintImage_Bornemann InpaintImage_Telea	FoundationPro FoundationPro	
threshold	–	ThresholdImage	FoundationLite	
–	threshold	ThresholdToRegion	FoundationBasic	
–	hysteresis_threshold	ThresholdImage_Hysteresis	FoundationBasic	
–	dyn_threshold	ThresholdImage_Dynamic	FoundationLite	
–	histo_to_thresh	SelectThresholdValue	FoundationBasic	
watershed	watersheds_threshold	SegmentImage_Watersheds	FoundationBasic	
normalize	equ_histo_image	EqualizeImageHistogram	FoundationLite	
LineIterator	–	ImageAlongPath ImageAlongArc	FoundationPro FoundationBasic	
–	measure_projection	ImageProfileAlongPath	FoundationPro	
–	tile_images	CutImageIntoTiles	FoundationBasic	
minMaxLoc	–	ImageMaximum	FoundationLite	
–	local_max	ImageLocalMaxima	FoundationBasic	
warpAffine	affine_trans_image	CreateAffineTransformMatrix TransformImage	FoundationBasic FoundationLite	
–	vector_angle_to_rigid	CreateCoordinateSystemFromVector CreateCoordinateSystemFromSegment	FoundationLite FoundationLite	
–	affine_trans_point_2d	AlignPoint TranslatePoint	FoundationLite FoundationLite	
–	hom_mat2d_identity	CreateIdentityMatrix	FoundationLite	
–	hom_mat2d_rotate	CreateAffineTransformMatrix	FoundationBasic	
–	hom_mat2d_translate	CreateAffineTransformMatrix	FoundationBasic	
–	affine_trans_pixel	TransformImage	FoundationLite	
linearPolar	polar_trans_image_ext	ImagePolarTransform	FoundationBasic	
linearPolar	polar_trans_image_inv	ImageInversePolarTransform	FoundationBasic	
–	projective_trans_image	TransformImage	FoundationLite	
LUT	lut_trans	LUTTransformImage	FoundationLite	
–	fill_interlace	LerpImages	FoundationLite	
meanStdDev	deviation_image	ImageStandardDeviation	FoundationBasic	
dft	fft_image	FourierTransform	FoundationPro	
dft	fft_generic	FourierTransform	FoundationPro	
–	convol_fft	FourierTransform ConvolveImage	FoundationPro FoundationLite	
–	gen_highpass	FrequencyDomain_FilterFrequencies	FoundationPro	
–	gen_lowpass	FrequencyDomain_FilterFrequencies	FoundationPro	

Region Analysis

In the OpenCV context, regions may be represented as binary image masks, but there are no true RLE-encoded regions. See also Region.

OpenCV	Halcon	AVL	Module	Comment
–	gen_circle	CreateCircleRegion	FoundationLite	
–	–	CreateCrossRegion	FoundationLite	
–	–	CreateBoxBorderRegion	FoundationLite	
–	gen_ellipse	CreateEllipseRegion	FoundationLite	
–	gen_grid_region	CreateGridRegion	FoundationLite	
–	gen_rectangle2	CreateBoxRegion	FoundationLite	
–	gen_region_line	CreateLineRegion	FoundationLite	
–	gen_region_polygon_filled	CreatePolygonRegion	FoundationLite	
–	gen_rectangle1	CreateRectangleRegion	FoundationLite	
–	gen_region_contour_xld	CreatePathRegion CreatePathBorderRegion	FoundationLite FoundationLite	
–	–	CreateRectangleBorderRegion	FoundationLite	
–	–	CreateRingRegion	FoundationLite	
–	gen_region_line	CreateSegmentRegion	FoundationLite	
–	difference	RegionDifference	FoundationLite	
–	intersection	RegionIntersection	FoundationLite	
–	union2	RegionUnion	FoundationLite	
–	symm_difference	RegionSymmetricDifference	FoundationLite	
–	complement	RegionComplement	FoundationLite	
–	mirror_region	MirrorRegion ReflectRegion	FoundationBasic FoundationBasic	
–	transpose_region	TransposeRegion	FoundationBasic	
–	dilation_rectangle1	DilateRegion	FoundationBasic	
–	dilation_circle	DilateRegion	FoundationBasic	
dilate	dilation1	DilateRegion_AnyKernel	FoundationBasic	
dilate	dilation2	DilateRegion_AnyKernel	FoundationBasic	
–	erosion_rectangle1	ErodeRegion	FoundationBasic	
–	erosion_circle	ErodeRegion	FoundationBasic	
erode	erosion1	ErodeRegion_AnyKernel	FoundationBasic	
erode	erosion2	ErodeRegion_AnyKernel	FoundationBasic	
–	–	ErodeRegion_Threshold	FoundationBasic	
–	closing	CloseRegion_AnyKernel	FoundationBasic	
–	closing_circle	CloseRegion	FoundationBasic	
–	closing_rectangle1	CloseRegion	FoundationBasic	
–	opening	OpenRegion_AnyKernel	FoundationBasic	
–	opening_rectangle1	OpenRegion	FoundationBasic	
–	opening_circle	OpenRegion	FoundationBasic	
–	expand_gray	ExpandRegions	FoundationBasic	
–	–	ThresholdSmoothedRegion_Mean	FoundationBasic	
–	–	DemarcateRegions	FoundationBasic	
findContours	morph_skeleton	SkeletonizeRegion	FoundationBasic	
–	thickening	ThickenRegion	FoundationBasic	
–	thinning	ThinRegion	FoundationBasic	
–	pruning	PruneRegion	FoundationBasic	
–	get_region_contour	RegionContours	FoundationBasic	
–	fill_up	FillRegionHoles	FoundationBasic	
–	expand_region	ExpandRegions	FoundationBasic	
–	move_region	TranslateRegion	FoundationLite	
–	affine_trans_region	ShearRegion	FoundationBasic	
–	test_equal_region	TestRegionEqualTo	FoundationBasic	
–	test_region_point	TestPointInRegion TestPointArrayInRegion	FoundationLite FoundationBasic	
–	test_subset_region	TestRegionInRegion	FoundationBasic	
–	–	TestRegionIntersectsWith	FoundationBasic	
–	–	TestRegionOnBorder	FoundationLite	
–	select_shape	VerifyRegion	FoundationLite	
–	select_shape	ClassifyRegions	FoundationBasic	
–	–	InscribeRegionInRegion	FoundationBasic	
–	select_shape	GetMaximumRegion GetMaximumRegion_OrNil	FoundationBasic FoundationBasic	
–	select_shape	GetMinimumRegion GetMinimumRegion_OrNil	FoundationBasic FoundationBasic	
–	–	GroupPathsByRegions GroupPointsByRegions GroupRegionsByRegions	FoundationBasic FoundationBasic FoundationBasic	

–	clip_region	TrimRegionToRectangle CropRegion CropRegionToRectangle CropRegionToQuadrangle	FoundationPro FoundationLite FoundationPro FoundationPro	
–	–	TrimRegionToPolygon	FoundationPro	
–	zoom_region	ResizeRegion ResizeRegion_Relative DownsampleRegion ShrinkRegionNTimes EnlargeRegionNTimes	FoundationBasic FoundationBasic FoundationBasic FoundationBasic FoundationBasic	
–	–	AlignRegion	FoundationBasic	
–	affine_trans_region	RotateRegion	FoundationBasic	
–	–	UncropRegion	FoundationLite	
–	gen_region_points	LocationsToRegion	FoundationLite	
–	get_region_points	RegionToLocations	FoundationLite	
–	–	RegionCharacteristicPoint	FoundationLite	
contourArea	area_center	RegionArea RegionMassCenter	FoundationLite FoundationLite	Utility function: area.
–	area_holes	RegionHoles	FoundationBasic	
–	circularity	RegionCircularity	FoundationBasic	
–	convexity	RegionConvexity	FoundationBasic	
–	diameter_region	RegionDiameter RegionCaliperDiameter	FoundationBasic FoundationBasic	
–	elliptic_axis	RegionEllipticAxes	FoundationBasic	
–	orientation_region	RegionOrientation	FoundationBasic	
moments	region_features	RegionMoment	FoundationBasic	
–	rectangularity	RegionRectangularity	FoundationBasic	
–	–	RegionElongation	FoundationBasic	
–	connect_and_holes	RegionNumberOfHoles	FoundationBasic	
–	–	RegionMedialAxis	FoundationBasic	
–	get_region_runs	RegionPerimeterLength	FoundationBasic	
–	–	RegionProjection	FoundationBasic	
–	–	RegionHoles_Elastic	FoundationBasic	
–	connection	SplitRegionIntoBlobs	FoundationBasic	
–	–	SplitRegionIntoComponents SplitRegionIntoExactlyNComponents	FoundationBasic FoundationBasic	
–	hit_or_miss	RegionHitAndMissTransform	FoundationBasic	
–	–	DemarcateRegions	FoundationBasic	
–	convexity	RegionConvexHull	FoundationBasic	
–	smallest_rectangle1	RegionBoundingRectangle	FoundationBasic	
–	–	RegionBoundingBox	FoundationLite	
–	smallest_circle	RegionBoundingCircle	FoundationBasic	
–	–	RegionBoundingParallelogram	FoundationPro	
–	boundary	RegionBoundaries RegionOuterBoundaries	FoundationBasic FoundationBasic	
–	inner_circle	RegionInscribedCircle	FoundationBasic	
–	–	RegionInscribedBox	FoundationBasic	
–	–	RegionInteriors	FoundationBasic	
–	–	RemoveBordersFromRegion	FoundationBasic	
–	–	RemoveRegionBlobs	FoundationBasic	
–	–	ExtractBlobs_Color ExtractBlobs_Dynamic ExtractBlobs_Intensity	FoundationBasic FoundationBasic FoundationBasic	
–	select_shape	SelectRegions	FoundationBasic	
–	sort_region	SortRegions	FoundationBasic	
–	read_region	LoadRegion	FoundationLite	
–	write_region	SaveRegion	FoundationLite	

Geometry 2D

Geometrical algorithms are essential in processing results in most computer vision applications. For the full list see Geometry 2D – AVL provides a comprehensive geometry library with many more functions than presented below. AVL also provides simple utility functions described in comments.

OpenCV	Halcon	AVL	Module	Comment
–	angle_ll	AngleBetweenLines AngleBetweenSegments	FoundationLite FoundationLite	Utility functions: angleTurn, angleDiff.
–	distance_cc_min	PathToPathDistance	FoundationPro	Utility function: distance.
–	distance_cc	PathToPathDistance PathToPathMaximumDistance	FoundationPro FoundationPro	Utility function: distance.
–	distance_lc	PathToLineDistance PathToLineDistanceProfile	FoundationBasic FoundationBasic	
–	distance_lr	RegionContours PathToPathDistance	FoundationBasic FoundationPro	
–	distance_pc	PathToPointDistance PathToPointDistanceProfile	FoundationBasic FoundationBasic	Utility function: distance.
–	distance_pl	PointToLineDistance PointToLineDistance_Oriented	FoundationLite FoundationLite	
–	distance_contours_xld	PathToPathDistanceProfile	FoundationPro	
–	distance_pp	PointToPointDistance	FoundationLite	Utility function: distance.
–	distance_pr	PathToPointDistanceProfile RegionContours	FoundationBasic FoundationBasic	
–	distance_ps	PointToSegmentDistance	FoundationLite	
–	distance_ss	SegmentToSegmentDistance	FoundationLite	
–	distance_rr_min	RegionToRegionDistance	FoundationBasic	
–	intersection_circles	CircleCircleIntersection	FoundationLite	
–	intersection_line_circle	LineCircleIntersection	FoundationLite	
–	intersection_contours_xld	PathPathIntersections	FoundationPro	
–	intersection_line_contour_xld	PathLineIntersections PathSegmentIntersections	FoundationPro FoundationPro	
–	intersection_lines	LineLineIntersection	FoundationLite	Utility function: intersection.
–	intersection_segment_line	LineSegmentIntersection	FoundationLite	Utility function: intersection.
–	intersection_ll	LineLineIntersection SegmentSegmentIntersection	FoundationLite FoundationLite	Utility function: intersection.
–	intersection_segments	SegmentSegmentIntersection	FoundationLite	Utility function: intersection.
–	projection_pl	ProjectPointOnLine	FoundationLite	Utility function: project.
–	line_orientation	LineOrientation SegmentOrientation	FoundationLite FoundationLite	Utility property: Line2D.Orientation.
–	line_position	SegmentOrientation	FoundationLite	Utility property: Line2D.Orientation.
convexHull	shape_trans	PointsConvexHull	FoundationLite	
contourArea	area_center_xld	PolygonArea	FoundationBasic	
–	draw_nurbs	CreateBicircularCurve DrawPath	FoundationBasic FoundationLite	
–	smallest_rectangle2	PathBoundingBox	FoundationLite	
–	area_center_xld	PathMassCenter	FoundationLite	
–	length_xld	PathLength	FoundationLite	
–	circularity_xld	PolygonCircularity	FoundationBasic	
–	orientation_xld	PolygonOrientation	FoundationBasic	
–	union_collinear_contours_xld	JoinAdjacentPaths	FoundationPro	
–	area_center_xld	PolygonMassCenter	FoundationBasic	
–	–	PolygonRectangularity	FoundationBasic	
–	circularity_xld	PolygonConvexity	FoundationBasic	
–	elliptic_axis_xld	PolygonEllipticAxes	FoundationBasic	
–	–	PolygonElongation	FoundationBasic	
–	smallest_circle_xld	PolygonInscribedCircle	FoundationBasic	
–	moments_xld	PolygonMoment	FoundationBasic	
–	moments_xld	PolygonMoment	FoundationBasic	
–	test_xld_point	TestPointArrayInPolygon TestPointInPolygon	FoundationBasic FoundationLite	
–	–	TestPolygonConvex	FoundationBasic	
–	–	TestPolygonInPolygon	FoundationBasic	
–	–	PolygonWithNormalizedOrientation	FoundationBasic	
–	–	FindClosestPoints	FoundationLite	

Path

Here is a list of fundamental paths transformations available in almost any library. For the full list see Path.

OpenCV	Halcon	AVL	Module	Comment
–	close_contours_xld	ClosePath	FoundationLite	
–	–	AlignPath AlignPathArray	FoundationLite FoundationLite	
–	–	FitPathToPath	FoundationPro	
–	affine_trans_contour_xld	InflatePath	FoundationPro	
–	affine_trans_contour_xld	PathAlongArc PathAlongPath	FoundationBasic FoundationPro	
–	affine_trans_contour_xld	RotatePath RotatePathArray	FoundationLite FoundationLite	
–	affine_trans_contour_xld	ReversePath	FoundationLite	
–	affine_trans_contour_xld	RescalePath RescalePathArray	FoundationLite FoundationLite	
–	affine_trans_contour_xld	ShiftPath	FoundationBasic	
–	affine_trans_contour_xld	TranslatePath TranslatePathArray	FoundationLite FoundationLite	
–	affine_trans_contour_xld	TransposePath	FoundationLite	
–	–	PathProjectionProfile	FoundationPro	
–	–	ConvertToEquidistantPath	FoundationBasic	
–	–	ExtendPath	FoundationBasic	
–	–	FindLongestSubpath	FoundationPro	
–	–	FindLongestSubpath	FoundationPro	
–	–	ReducePath	FoundationBasic	
–	–	RemovePathSelfIntersections	FoundationPro	
–	get_lines_xld	SegmentPath	FoundationPro	
–	get_contour_xld	PathArrayPoints	FoundationLite	
–	get_contour_angle_xld	PathAverageTurnAngle	FoundationPro	
–	smallest_rectangle1_xld	PathBoundingBox PathBoundingRectangle	FoundationLite FoundationBasic	
–	smallest_circle_xld	PathBoundingCircle	FoundationBasic	
–	–	PathBoundingParallelogram	FoundationPro	
–	get_contour_attrib_xld	PathCaliperDiameter	FoundationBasic	
–	diameter_xld	PathDiameter	FoundationLite	
–	get_lines_xld	PathSegments	FoundationLite	
convexHull	–	PathConvexHull	FoundationBasic	
–	–	PathEndpoints	FoundationLite	
–	test_self_intersection_xld	PathSelfIntersections	FoundationPro	
–	test_self_intersection_xld	PathSelfIntersections	FoundationPro	
–	max_parallels_xld	ConcatenatePaths	FoundationLite	
–	–	SplitPathByLine SplitPathByPath SplitPathBySegment	FoundationBasic FoundationPro FoundationBasic	
–	smooth_contours_xld	SmoothPath_Gauss SmoothPath_Mean	FoundationPro FoundationPro	
–	sort_contours_xld	SortPaths	FoundationBasic	
–	select_contours_xld	SelectClosedPaths SelectOpenPaths SelectInnerPaths SelectOuterPaths	FoundationBasic FoundationBasic FoundationPro FoundationPro	
–	select_shape_xld	ClassifyPaths GetMaximumPath GetMinimumPath	FoundationBasic FoundationBasic FoundationBasic	

Computer Vision

This section contains higher-level vision tools.

OpenCV	Halcon	AVL	Module	Comment
–	measure_pos	ScanSingleEdge ScanMultipleEdges ScanExactlyNEdges	MetrologyBasic MetrologyBasic MetrologyBasic	See also ScanSingleRidge
–	measure_pairs	ScanSingleStripe ScanMultipleStripes ScanExactlyNStripes	MetrologyBasic MetrologyBasic MetrologyBasic	
–	measure_thresh	ScanSingleEdge ScanMultipleEdges ScanExactlyNEdges	MetrologyBasic MetrologyBasic MetrologyBasic	See also ScanSingleRidge
–	edges_sub_pix	DetectEdges DetectEdges_AsPaths	FoundationLite FoundationBasic	
–	edges_color_sub_pix	DetectEdges DetectEdges_AsPaths	FoundationLite FoundationBasic	
–	find_data_code_2d	ReadSingleDataMatrixCode ReadSingleQRCode	Datacodes Datacodes	See also Datacodes
–	find_bar_code	ReadSingleBarcode	Barcodes	See also: Barcodes
–	detect_edge_segments	FitSegmentToEdges	MetrologyPro	
–	fit_line_contour_xld	FitSegmentToPoints	FoundationBasic	
–	fit_circle_contour_xld	FitCircleToPoints	FoundationBasic	
–	–	FitPathToRidges	MetrologyPro	
–	–	FitPathToStripe	MetrologyPro	
–	–	FitPathToEdges	MetrologyPro	
–	fit_circle_contour_xld	FitCircleToRidges	MetrologyPro	
–	fit_circle_contour_xld	FitCircleToEdges	MetrologyPro	
–	–	FitCircleToStripe	MetrologyPro	
–	–	FitLineToPoints_LTE FitLineToPoints_M FitLineToPoints_RANSAC FitLineToPoints_TheilSen	FoundationBasic FoundationBasic FoundationBasic FoundationBasic	
–	–	FitSegmentToPoints FitSegmentToPoints_LTE FitSegmentToPoints_RANSAC FitSegmentToPoints_TheilSen	FoundationBasic FoundationBasic FoundationBasic FoundationBasic	
matchTemplate	find_ncc_model	LocateSingleObject_NCC LocateMultipleObjects_NCC	MatchingBasic MatchingBasic	
–	gen_measure_rectangle2	CreateScanMap	MetrologyBasic	
–	find_shape_model	LocateSingleObject_Edges1 LocateMultipleObjects_Edges1	MatchingPro MatchingPro	
–	create_shape_model	CreateEdgeModel1	MatchingPro	
–	write_shape_model	SaveEdgeModel	MatchingPro	
–	read_shape_model	LoadEdgeModel	MatchingPro	
–	segment_image_mser	FindMaxStableExtremalRegions	FoundationPro	
–	auto_threshold	SegmentImage_Color	FoundationPro	
–	detect_edge_segments	SegmentImage_Edges	FoundationPro	
–	local_threshold	SegmentImage_Gray	FoundationPro	Methods are very different.
–	–	SegmentImage_Gray_Linear SegmentImage_Gray_Tiled	FoundationPro FoundationPro	
–	corner_response	DetectCorners_CornerResponse	FoundationBasic	
–	–	DetectCorners_Foerstner	FoundationBasic	
preCornerDetect	points_foerstner	DetectCorners_Foerstner	FoundationBasic	
–	photometric_stereo	PhotometricStereo_Perform PhotometricStereo_ComputeHeightMap PhotometricStereo_SurfaceCurvature	Photometric Photometric Photometric	
HoughLines2	hough_lines	DetectLines	FoundationBasic	
–	–	DetectLinePeak DetectLinePeak_Gauss	FoundationPro FoundationPro	
HoughCircles	hough_circles	DetectSingleCircle DetectMultipleCircles	FoundationBasic FoundationBasic	
–	–	DetectPaths	FoundationBasic	
–	texture_laws	LawsFilter	FoundationBasic	
–	–	LinearBinaryPattern	FoundationBasic	
–	–	LocateSinglePointPattern	FoundationPro	
–	texture_laws	LawsFilter	FoundationBasic	
–	trainf_ocr_class_svm	TrainOcr_SVM	OCR	
–	trainf_ocr_class_mlp	TrainOcr_MLP	OCR	
–	do_ocr_multi	ReadText	OCR	

Camera Calibration

The three products have quite different approach to calibration functions. OpenCV provides basic distortion correction based on (x, y) transformation maps. Commercial libraries provide highly optimized remapping based on precomputed data, but also focus on coordinate space transformations and image stitching functionalities. For details see Camera Calibration and World Coordinates.

OpenCV	Halcon	AVL	Module	Comment
calibrateCamera	camera_calibration	CalibrateCamera_Pinhole CalibrateCamera_Telecentric	Calibration Calibration	All function could use different methods.
–	find_caltab	DetectCalibrationGrid_Circles	Calibration	
–	find_marks_and_pose	GenerateCalibrationPoints	Calibration	
–	set_origin_pose	ShiftWorldPlane	Calibration	
–	gen_image_to_world_plane_map	CreateRectificationMap_Advanced	Calibration	
remap	map_image	RectifyImage	Calibration	OpenCV needs operation to extract final transformation matrix.

Machine Learning

Fundamental machine learning algorithms include K Nearest Neighbors, Support Vector Machines, Multi-Layer Perceptron and Principal Component Analysis. In the below table we provide just basic algorithms as a starting point for finding what you need.

OpenCV	Halcon	AVL	Module	Comment
CvKNearest::find_nearest	classify_image_class_knn	KNN_Classify	FoundationPro	
CvSVM::predict	classify_image_class_svm	SVM_ClassifySingle SVM_ClassifyMultiple	FoundationPro FoundationPro	
CvANN_MLP::predict	classify_image_class_mlp	MLP_Respond	FoundationPro	
–	gen_principal_comp_trans	ApplyPCATransform ReversePCATransform	FoundationPro FoundationPro	
–	–	LinearRegression LinearRegression_LTE LinearRegression_M LinearRegression_RANSAC LinearRegression_TheilSen	FoundationBasic FoundationPro FoundationPro FoundationPro FoundationBasic	
–	–	QuadraticRegression QuadraticRegression_M QuadraticRegression_RANSAC	FoundationBasic FoundationPro FoundationPro	
–	–	ClusterData_KMeans	FoundationPro	
–	–	ClusterPoints2D ClusterPoints2D_SingleLink	FoundationPro FoundationPro	

Communication

The below table contains only basic functions. Please refer to detailed documentation for more details.

OpenCV	Halcon	AVL	Module	Comment
–	open_serial	SerialPort_Config	FoundationLite	
–	read_serial	SerialPort_ReadBuffer	FoundationLite	
–	write_serial	SerialPort_WriteBuffer	FoundationLite	
–	open_socket_accept	TcpIp_Accept	FoundationLite	
–	open_socket_connect	TcpIp_Connect	FoundationLite	
–	close_socket	TcpIp_Close	FoundationLite	
–	receive_data	TcpIp_ReadBuffer	FoundationLite	
–	send_data	TcpIp_WriteBuffer	FoundationLite	
–	–	Ftp_ReceiveFile	FoundationBasic	
–	–	Ftp_ReceiveImage	FoundationBasic	
–	–	Ftp_SendFile	FoundationBasic	
–	–	Ftp_SendImage	FoundationBasic	
–	–	Http_SendRequest_GET	FoundationBasic	
–	–	Http_SendRequest_POST	FoundationBasic	
–	–	Http_DecodeURL	FoundationBasic	
–	–	Http_EncodeURL	FoundationBasic	

Deep Learning Training API

Table of contents:

1. Overview
2. Types
3. Functions
    - CreateSamples
    - Train
    - SolveTrainingSamples
    - GetWorstValidationValue
    - IsValidationBetter
    - FindBestValidation
4. Handling events

1. Overview

Whole API declaration is located in `Api.h` file (in `AVLDL_PATH5_3\include`) under `avl::DeepLearning` namespace. This namespace contains the following namespaces:

- `AnomalyDetection1_Local`, grouping together classes and functions related to training in Anomaly Detection 1 – Local mode,
- `AnomalyDetection1_Global`, analogous to the above,
- `AnomalyDetection2`, analogous to the above,
- `FeatureDetection`, analogous to the above,
- `ObjectClassification`, analogous to the above,
- `InstanceSegmentation`, analogous to the above,
- `PointLocation`, analogous to the above,
- and `Common`, grouping together classes and functions used in multiple namespaces mentioned before.

Each namespace (except `Common`) contains almost the same set of types and functions – in respect of their functionality and purpose. Naturally, types and functions in one namespace differ from their counterparts from another namespaces in respect of their signatures and declaration in code.

Example usages of Deep Learning Training API are shown in the following projects:

- *%Public%/Documents/Aurora Vision Deep Learning 5.3/02 Training - Feature Detection* - Feature Detection mode.
- *%Public%/Documents/Aurora Vision Deep Learning 5.3/03 Training - Object Classification* - Object Classification mode.
- *%Public%/Documents/Aurora Vision Deep Learning 5.3/04 Training - Instance Segmentation* - Instance Segmentation mode.
- *%Public%/Documents/Aurora Vision Deep Learning 5.3/05 Training - Detect Anomalies 1 Local* - Detect Anomalies 1 Local mode.
- *%Public%/Documents/Aurora Vision Deep Learning 5.3/06 Training - Detect Anomalies 2* - Detect Anomalies 2 mode.
- *%Public%/Documents/Aurora Vision Deep Learning 5.3/07 Training - Locate Points* - Locate Points mode.

2. Types

Namespaces except `Common`, declare classes:

- `PreprocessingConfig`, containing setters and getters for preprocessing settings (e.g. downsample).
- `AugmentationsConfig`, containing setters and getters for augmentations settings (e.g. flips, rotation).
- `TrainingConfig`, containing setters and getters for general training settings (e.g. iteration count). As well as pointers to objects of types mentioned above.
- `Sample`, containing setters and getters for one training sample (e.g. path to image file).
- `TrainingEventsHandler`, containing virtual methods used as handlers for various events happening during training process and solving training samples. This class is intended for inheriting. Handling events is described in detail in section Handling events

In addition, each class (except `TrainingEventsHandler`) has 2 more methods:

- `Create(...)` - static method intended for constructing objects of specific type. It allows setting all fields at the same time, without calling multiple setters after construction.
- `Clone()` - method for creating new objects being the exact copy of cloned one.

Due to differences between supported network types and, consequently, different sets of possible parameters, mentioned classes may differ from theirs counterparts from other namespaces. Classes except `TrainingEventsHandler`are **not** intended for being inherited by user types.

`Common`namespace contains 2 classes important for user:

- `ModelInfo`- containing information about already existing in training directory model file. Currently, only validation history is provided. It is used in `ReceivedExistingModelInfo(...)` event handler.
- `Progress`- containing data about progress in training process or solving training samples. Progress information is divided into 3 fields:
  - `Stage`- very general information describing process advancement.
  - `Phase`- more fine-grained information of advancement in current `Stage`. It contains 2 integers: total number of phases and current phase. Phases does not have take the same time to finish.
  - and optional `Step`- some `Phases`can be divided into smaller steps. In such cases, this field contains 2 integers: total number of steps in current `Phase`and number of already finished steps. Each step should take roughly the same time to finish.

  These fields are useful for creating progress bars and so on.

  Apart from that, `Progress`objects contain information about current training, validation values (if applicable) and boolean value indicating that validation has started.

3. Functions

Namespaces except `Common`, declare functions:

CreateSamples

Helper function intended for creating array of training samples from images in given directory with given parameters. Set of these additional parameters depends on mode. See documentation in source code for further explanation.

Syntax

```
atl::Array<>> CreateSamples
(
 ...,
 const atl::String& mask
)
```

Parameters

	Name	Type	Default	Description
➡	...	...	...	Sample parameters, e.g. path to directory with roi images and so on.
➡	mask	const String&	*	Mask used for filtering found files. By default, no files are filtered out. See documentation of FindFiles filter for more information

Train

Performs training process. Returns "true" if model was saved.

Syntax

```
bool Train
(
 DeepLearningConnectionState& state,
 const atl::Array<>> trainingSamples,
 const TrainingConfig& config,
 TrainingEventsHandler& eventsHandler
)
```

Parameters

	Name	Type	Description
➡	state	DeepLearningConnectionState&	Object maintaining connection with service
➡	trainingSamples	const Array<std::unique_ptr<Sample>>&	Array of training samples
➡	config	const TrainingConfig&	Training configuration
➡	eventsHandler	TrainingEventsHandler&	Training events handler. It can be object of `TrainingEventsHandler`or, more common, object of user type inherited from it

Remarks

- This function has overload without `eventsHandler`parameter. It uses object of `TrainingEventsHandler`type instead.
- In Anomaly Detection modes all training samples are automatically solved after training. In other modes, this can be done by SolveTrainingSamples. After solving each sample, `SolvedTrainingSample`event is called.

SolveTrainingSamples

Solves given training samples.

Syntax

```
void SolveTrainingSamples
(
 DeepLearningConnectionState& state,
 const atl::Array<>>& trainingSamples,
 const TrainingConfig& config,
 ...,
 TrainingEventsHandler& eventsHandler
)
```

Parameters

	Name	Type	Description
➡	state	DeepLearningConnectionState&	Object maintaining connection with service
➡	trainingSamples	const Array<std::unique_ptr<Sample>>&	Array of training samples
➡	config	const TrainingConfig&	Training configuration
➡	...	...	Additional parameters depending on mode
➡	eventsHandler	TrainingEventsHandler&	Training events handler. It can be object of `TrainingEventsHandler`or, more common, object of user type inherited from it

Remarks

- After solving each sample, `SolvedTrainingSample`event is called.
- This function does not exists in Anomaly Detection modes since solving training samples is done automatically in Train
- Unlike Train, there is no overload without `TrainingEventsHandler`object - it would be pointless as this function calls `SolvedTrainingSample`event handler and, by default, this handler does nothing.
- Additional parameters, if present, should be described in documentation for corresponding filter (e.g. AvsFilter_DL_SegmentInstances) in case of Instance Segmentation mode).

GetWorstValidationValue

Returns worst possible validation value.

Syntax

```
float GetWorstValidationValue()
```

Remarks

- It is useful for initialization validation value (e.g. in custom training events handler) and eliminates need of using special values in comparisons.
- This function does not exists in in Anomaly Detection 2 mode as it would be pointless. This is due significant differences in training process in Anomaly Detection 2 mode comparing to other modes.

IsValidationBetter

Returns "true" if `newValidationValue`is "better" than `oldValidationValue`. Comparing validation values with relational operators is strongly discouraged due the fact that in some modes lower values are better, but in other modes – otherwise.

Syntax

```
bool IsValidationBetter
(
 float oldValidationValue,
 float newValidationValue
)
```

Parameters

	Name	Type	Description
➡	oldValidationValue	float	Base validation value
➡	newValidationValue	float	Compared validation value

Remarks

- This function does not exists in in Anomaly Detection 2 mode as it would be pointless. This is due significant differences in training process in Anomaly Detection 2 mode comparing to other modes.

FindBestValidation

Returns best validation value from array.

Syntax

```
float FindBestValidation
(
 const atl::Array& validationValues
)
```

Parameters

	Name	Type	Description
➡	validationValues	const Array<float>&	Array of validation values

Remarks

- Useful for obtaining best validation value in already existing model in `ReceivedExistingModelInfo`event.
- This function does not exists in in Anomaly Detection 2 mode as it would be pointless. This is due significant differences in training process in Anomaly Detection 2 mode comparing to other modes.

4. Handling events

To communicate with user during training and solving training samples, several events are used. All event handlers are implemented as virtual methods of `TrainingEventsHandler` allowing users to write custom handlers by overriding them.
There are 5 possible events:

- `ReceivedExistingModelInfo(...)` - this handler is called after receiving information about already existing model right after training start. If no model is present at given location, handler is not called. This method takes existing model information as parameter (of type `Common::ModelInfo`) and by default does nothing.
- `ReceivedProgress(...)` - this handler is called after receiving progress information during training and solving training samples. This method takes progress information as parameter of type `Common::Progress`and have to return `true`if process should be stopped or `false`otherwise. Default handler of this event does nothing and do not interrupt process.
- `SavedModelAutomatically(...)` - after training, model file can be saved or discarded. In some cases, Deep Learning Service can make this decision automatically. This handler is called in such cases. This method contains argument indicating whether file was saved or not. Default handler does nothing.
- `SaveModel()` - in most cases, Deep Learning Service cannot determine whether model file should be saved or not. This handler is called in such cases. This method has no arguments but have to return `true`if file should be replaced or `false`otherwise. This decision can be made on the basis of data collected in `ReceivedExistingModelInfo(...)` and `ReceivedProgress(...)` event handlers. Default handler always returns `true`which results in Deep Learning Service saving model file.
- `SolvedTrainingSample(...)` - this handler is called each time after solving training sample. This happens in `Train(...)` (in both AnomalyDetection modes) or `SolveTrainingSamples(...)` (in other modes) functions. This method takes training sample (of type `Sample`) and solution results as arguments. Unlike previous methods, signature of this methods differs between various namespaces. Default handler does nothing.