adaptive-vision.com

Zebra
Aurora™ Vision

Aurora Vision Library 5.3

Getting Started

Created: 6/8/2023

Product version: 5.3.4.94078

Table of content:

SDK Installation

Requirements

Aurora Vision Library is designed to be a part of applications working under control of the Microsoft Windows operating system. Supported versions are: 7, 8 and 10, as well as the corresponding embedded editions.

To build an application using Aurora Vision Library, Microsoft Visual Studio environment is required. Supported versions are: 2015, 2017 and 2019.

Aurora Vision Library can be also used on Linux operating system with GCC compiler - for details consult Using SDK on Linux article.

Running the Installer

The installation process is required to copy the files to the proper folders and to set the environment variables used for building applications using Aurora Vision Library.

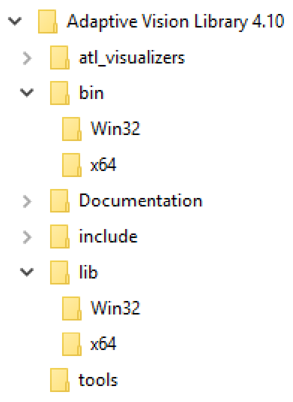
After the installation, a license for Aurora Vision Library product has to be loaded. It can be done with the *License Manager* tool available in the *Start Menu*.

To verify that the installation has been successful and the license works correctly, one can try to load, build and run example programs, which are available from the *Start Menu*.

SDK Directories

Aurora Vision Library is distributed as a set of header files (.h), dynamic (.dll) and static (.lib) libraries. The libraries (static and dynamic) are provided in versions for 32-bit and 64-bit system. The header files are common for both versions.

The picture below shows the structure of the directories containing headers and libraries included in Aurora Vision Library.



The directories (installed in the *Program Files* system folder) being a part of Aurora Vision Library are shortly described below.

- **atl_visualizers** – a directory containing the visualizers for Microsoft Visual Studio Debugger of Aurora Vision Library data types.
- **bin** – a directory containing dynamic linked library files (AVL.dll) for 32|64-bit applications. The libraries are common for all supported versions of Microsoft Visual Studio and for Debug|Release configurations. All the functions of Aurora Vision Library are included in the AVL.dll file.
- **Documentation** – a directory containing the documentation of Aurora Vision Library, including this document.
- **include** – a directory containing all header (.h) files for Aurora Vision Library. Every source code file that uses Aurora Vision Library needs the AVL.h header file (the main header file) to be included.
- **lib** – a directory containing static (.lib) libraries (AVL.lib) for 32|64-bit applications. The AVL.lib file has to be statically-linked into the program that uses Aurora Vision Library. It acts as an intermediary between the usage of Aurora Vision Library functions and the AVL.dll file. The programmer creating an application does not need to bother about DLL entry points and functions exported from the AVL.dll file. Aurora Vision Library is designed to be easy to use, so one only needs to link the AVL.lib file and can use all the functions from the AVL.dll just as easy as local functions.
- **tools** – a directory containing the *License Manager* tool helping the user to load the license for Aurora Vision Library to the developer's computer.
- **Examples** – a directory located in the *Public Documents* system folder (e.g. `C:\Users\Public\Documents\Aurora Vision Library 5.3\Examples` on Windows Vista/7) containing simple example solutions using Aurora Vision Library. The examples are a good way of learning, how to use Aurora Vision Library. They can be used as a base for more complicated programs as well. The shortcut to the Examples directory can be found in the *Start Menu* after the installation of Aurora Vision Library.

Library Architecture

Aurora Vision Library is split into four parts:

1. Aurora Vision Library - contains all functions for working with images.
2. Standard Library - contains all auxiliary functions like: file operations, XML editing or mathematical operations.
3. GenICam Library - contains all GenICam and GigEVision functions.
4. Third Party Library - contains functions of third-party hardware producers.

The usage of the library is possible only when including one of the following header files:

- AVL.h

- STD.h
- Genicam.h
- ThirdPartySdk.h

Environment and Paths

Aurora Vision Library uses the environment variable named `AVL_PATH5_3` (5_3 stands for the 5.3 version) in the building process. The variable points the directory with the headers and libraries needed in the compile time (.h files and AVL.lib) and in the run time (AVL.dll). Its value is typically set to `C:\Program Files (x86)\Aurora Vision\Aurora Vision Library 5.3`, but it can differ in other systems.

The projects using Aurora Vision Library should use the value of `AVL_PATH5_3` to resolve the locations of the header files and statically-linked `AVL.lib` file. Using an environment variable containing path makes the application source code more portable between computers. The `AVL_PATH5_3` path is typically used in the project settings of the compiler (Configuration Properties | C/C++ | General | Additional Include Directories) to find the header files, settings of the linker (Configuration Properties | Linker | General | Additional Library Directories) to find the proper version of the AVL.lib and in the configuration of Post-Build Event (Configuration Properties | Build Events | Post-Build Event | Command Line) to copy the proper version of the AVL.dll file to the output directory of the project. All the settings can be viewed in the simple example applications distributed with Aurora Vision Library.

Project Configuration

General Information

Aurora Vision Library is designed to be used as a part of C++ projects developed with Microsoft Visual Studio in versions 2015-2019.

Creating a New Project

Microsoft Visual Studio 2015, 2017 and 2019

Aurora Vision Library is provided with a project template. To create a new project using Aurora Vision Library, start Microsoft Visual Studio and choose the *File* | *New* | *Project...* command. The template called *AVL 5.3 Project* is available in the tab *Installed* | *Templates* | *Other Languages* | *Visual C++*.

Required Project Settings

All projects that use Aurora Vision Library need some specific values of the compiler and linker settings. If you want to use the Library in your existing project or you are manually configuring a new project, please apply the settings listed below:

- **Configuration Properties | General**
  - **Character Set** should be set to `Use Unicode Character Set`.
- **Configuration Properties | C/C++**
  - **General**
    - **Additional Include Directories** should contain the `$(AVL_PATH5_3)\include\` path.
- **Configuration Properties | Linker**
  - **General**
    - **Additional Library Directories** should contain the proper path to directory containing the AVL.lib file. The proper path is `$(AVL_PATH5_3)\lib\$(PlatformName)\`.
  - **Input**
    - **Additional Dependencies** should contain `AVL.lib` file.
- **Configuration Properties | Build Events**
  - **Post-Build Event**
    - **Command Line** should contain `copy "$(AVL_PATH5_3)\bin\$(PlatformName)\AVL.dll" "$(OutDir)"` call. This setting is not mandatory, but the application using Aurora Vision Library requires an access to the AVL.dll file and this is the easiest way to fulfill this requirement.

Including Headers

Every source code file that uses Aurora Vision Library needs the `#include <AVL.h>` directive. A proper path to the AVL.h file is set in the settings of the compiler (described above), so there is no need to use the full path in the directive.

Distributing Aurora Vision Library with Your Application

Once the application is ready, it is time for preparing a distribution package or an installer. There are two requirements that needs to be fulfilled:

- The final executable file of the application needs to have access to the proper version (used by *Win32* or *x64* configuration) of the AVL.dll file. Typically, the AVL.dll file should be placed in the same directory as the executable.
- The computer that the application will run on needs a valid license for the use of Aurora Vision Library product. Licenses can be managed with the License Manager application, that is installed with Aurora Vision Library Runtime package.
- A license file (*.avkey) can be also manually copied to the end user's machine without installing Aurora Vision Library Runtime. It must be placed in a subdirectory of the *AppData* system folder. The typical location for the license file is `C:\Users\%USERNAME%\AppData\Local\Aurora Vision\Licenses`. Remember that the license is valid per machine, so every computer that runs the application needs a separate license file.
- Alternatively to the (*.avkey) files we support USB Dongle licenses.

Using Library with CMake

Library ships with CMake configuration modules. It makes the project portable, and easy to compile for Windows, linux or Android. The minimum CMake version supported is 3.10 (for example shipped with Ubuntu bionic/18.04)

Quick Start

A simple template for `CMakeLists.txt` is presented below:

```
cmake_minimum_required(VERSION 3.10)
project(example)

find_package(
    AVL
    # for a specific version, uncomment the line below
    #5.3
    CONFIG
    REQUIRED
)

# copy binaries to build directory
copy_avl()

add_executable(
    # executable name
    example_exec
    # source files
    main.cpp
)

target_link_libraries(
    example_exec
    PUBLIC
    AVL
)

# install user executable
install(TARGETS example_exec)
# install ALL AVL libraries
install_avl()
```

One can also copy one of the CMake examples, and modify to your needs. For further cmake use refer to online documentation. Be aware that ubuntu 18.04 is the baseline distribution, so minimal CMake version is 3.10

Reference

package

CMake package is provided for windows installer and linux archive. Both should be usable after installation. Linux additionally ships with Android libraries. The library is only discoverable using `CONFIG` mode, so it's sensible to restrict `find_package` to that mode.

```
find_package(
    AVL
    # for a specific version, uncomment the line below
    #5.3
    CONFIG
    REQUIRED
)
```

On Android to use system installed AVL it is necessary to add `CMAKE_FIND_ROOT_PATH_BOTH` argument:

```
find_package(AVL CONFIG REQUIRED CMAKE_FIND_ROOT_PATH_BOTH)
```

Possible packages:

- AVL - full library
- AVL_Lite - lite library
- Weaver - deep learning inference library

install_avl

Install all AVL libraries when executing `make install` or `ninja install` or building `INSTALL` project in Visual Studio. It accepts a `LIB` argument to override default installation directory. It requires `find_package(AVL...)` call first.

```
find_package(AVL CONFIG REQUIRED)

install_avl()
```

By default it installs to `${CMAKE_INSTALL_PREFIX}/bin` on Windows and `${CMAKE_INSTALL_PREFIX}/lib` on Linux. When provided the LIB argument it installs to `${CMAKE_INSTALL_PREFIX}/${LIB_ARGUMENT}`

```
install_avl(LIB "avl_directory")
```

Possible variants:

- `install_avl()`
- `install_avl_lite()`
- `install_weaver()`

copy_avl

Copy all AVL libraries when compiling targets that depend on AVL to binary directory. By default it's `${CMAKE_BINARY_DIR}` or `${CMAKE_BINARY_DIR}/$<CONFIG>` on Windows. It requires `find_package(AVL...)` call first.

```
find_package(AVL CONFIG REQUIRED)

copy_avl()
```

Possible variants:

- `copy_avl()`
- `copy_avl_lite()`
- `copy_weaver()`

Using Library on Linux

Requirements

Aurora Vision Library is designed to be used with GCC compiler on Linux x86_64, embedded ARMv7-A and ARMv8-A systems. Currently `gcc` in version 5.4 is supported, and corresponding toolchains for embedded linux: `arm-linux-gnueabihf-`, `aarch64-linux-gnu-`. Custom build can be prepared upon the earlier contact with Aurora Vision team. The Aurora Vision Library is distributed as `.tar.gz` or `.tar.xz` archive. The library is compatible with Debian-like system, including - but not limited to - Ubuntu distributions.

Common prerequisites

Properly set locale on target computer is important. Non-existing locale will cause bugs and bad behavior. To list locale that exists on your computer use: `locale -a`, and currently set: `locale`. Remember that running your application as daemon (e.g. from `systemd`) may set different locale, than the one in your user terminal. Refer to your Linux distribution documentation.

To build example in simple manner, GNU Make tool and CMake is needed.

- Ubuntu 18.04/Debian 9 or newer:
  - Runtime:
    - package libc6 ≥ 2.23
    - package libudev1 ≥ 229
  - Development:
    - package g++ version ≥ 5.4
    - package make
    - package cmake version ≥ 3.10
    - `sudo apt-get install cmake make g++`
  - Examples:
    - `sudo apt-get install libgtk-3-dev libsdl-dev qtbase5-dev`
- CentOS 8/Fedora 29/OpenSUSE 15.0 or newer:
  - Runtime:
    - package glibc ≥ 2.23
    - package systemd ≥ 229
  - Development:
    - package gcc-c++ version ≥ 5.4
    - package make
    - package cmake version ≥ 3.5
    - CentOS/Fedora: `dnf install gcc-c++ make cmake`
    - OpenSUSE: `zypper install gcc-c++ make cmake`
  - Examples:
    - CentOS/Fedora: `dnf install SDL2-devel qt5-qtbase-devel gtk3-devel`
    - OpenSUSE: `zypper install libSDL2-devel libqt5-qtbase-devel gtk3-devel`
- Generic:
  - Runtime:
    - libraries libc.so.6, libpthread.so.0, libm.so.6, libdl.so.2, librt.so.1, libgcc_s.so.1 from glibc version ≥ 2.23 or compatible (i.e. musl libc)
    - library libudev.so.1 from systemd version ≥ 229

Supported input devices

Vendor	x86_64	armv7-a	armv8
ximea	✔	✔	✔
Allied Vision Vimba	✔	✔	✔
Basler Pylon	✔	✔	✔
LMI Gocator	✔	✔	✔

Installation instructions

In unpacked directory call the `install` script. In example: `sudo ./install` This command will install the library to a proper directory in opt. It will also make the library visible to CMake `find_package` command.

Compilation instructions

Directory structure

Unpacked directory consists of following entries:

- `examples/` - directory contains source files of example programs written with Aurora Vision Library
- `include/` - this directory contains library header files
- `lib/` - here the .so file with library is stored, along with any kits
- `bin/` - directory for additional binaries, like Licensing tool.
- `/README` - instruction of library usage
- `/sha512sum` - checksums for all files in archive, check with `sha512sum --quiet -c sha512sum`
- `/metadata.json` - file containing information about the optimal target system, and library version
- `/install` - installation script
- `/uninstall` - uninstall script, will be copied to installation directory, where it can be safely used

Compilation

Using CMake

CMake is the recommended way to compile on linux, see documentation Using Library with CMake

Using Makefile or your custom build system

For compiling with Aurora Vision Library please remember to:

- add the `include/` subdirectory to the compiler include directories: `-I` switch
- add the `lib/` subdirectory to the linker directories: `-L` switch
- link with Aurora Vision Library: `-lAVL`
- use `-rpath` in linker options, `LD_LIBRARY_PATH` or `LD_PRELOAD` of `libAVL.so` file.
- link with dependencies: `-lpthread -lrt -ldl`

One can consult makefile in the examples/ directory to see how to compile and link with Aurora Vision Library.

Known compilation bugs

In case of the following linker errors: (or similar)

```
/usr/bin/ld: warning: libiconv.so, needed by lib/libAVL.so, not found (try using -rpath or -rpath-link)
lib/libAVL.so: undefined reference to `libiconv'
lib/build/libAVL.so: undefined reference to `libiconv_close'
lib/build/libAVL.so: undefined reference to `libiconv_open'
```

It is a known gnu linker bug, affecting versions older than 2.28 (e.g. in Ubuntu 16.04).
To solve the problem you can:

- Try a different linker (add for linking `-fuse-ld=gold` for gold or `-fuse-ld=lld`, consult your linux distribution manual)
- Link with the missing library (for example add `-liconv`)
- Update the linker (`binutils` 2.28 or newer)

Licensing and distribution

Licensing

File based licenses are supported on all Linux platforms. Dongle licenses depend on CodeMeter runtime. Currently Codemeter runtime is available for x86_64 and ARMV7-A. To develop and debug programs written with Aurora Vision Library, Library license has to be present. To run compiled binaries linked with Aurora Vision Library, LibraryRuntime license has to be present.

One can use `license_manager` from `bin/` directory to list currently installed file or dongle licenses: `license_manager list`
Red marked licenses are invalid, for example past the license date or installed license for the wrong machine (bad ID)

File License

To obtain license:

- In a terminal, on the target machine run `license_manager --id` from `bin/` directory
- Copy the printed Computer ID
- Use that Computer ID to get a `.avkey` file from User Area on www.adaptive-vision.com website.
- Download the key to the target machine
- Install the license by **one** of the following methods:
  - Run in terminal `license_manager install downloaded_file.avkey` (*Recommended*)
  - Copy the `.avkey` file next to executable, that is using Aurora Vision Library

Dongle License

Installed CodeMeter Runtime is required, as well as proper license available on plugged in dongle.

Download runtime package from WIBU website, section "CodeMeter User Runtime for Linux".
"Driver Only" (lite) version recommended for headless (no desktop GUI) installations. ARMV7-A is available under "CodeMeter User Additional Downloads" as "Raspberry PI" version

Distribution

To distribute program with Aurora Vision Library, one have to provide license (file or dongle - depending on system used) and the `libavl.so`. To provide the `.so` file, one can install SDK on target machine, but this will provide headers etc., which may be unwanted. In such case, the library file, with any used kits should be copied to suitable system directory, or the program has to be compiled with `-rpath` and relative path to the .so file. Third option is to provide a boot script, which will set `LD_LIBRARY_PATH` or `LD_PRELOAD` with `libavl.so` location.

Program development - general advise

The most convenient way to make programs with Aurora Vision Library for Linux is to develop vision algorithm using Aurora Vision Studio on Windows and then generating C++ code. This code can be further changed or interfaced with rest of the system and tested on Windows. Then, cross-compiler can be used to prepare Linux build, which will be provided to target machine. It is easy to organize work this way, because:

- developing vision algorithm using plain C++ is hard, troublesome and error prone, but Aurora Vision Studio makes it easy,
- programs written with Aurora Vision Library on Windows can be easily debugged using Microsoft Visual Studio thanks to provided debug visualizers and the Image Watch extensions to Microsoft Visual Studio,
- cross compilation using virtualization solution, like Vagrant, is easy and fast, and does not force developer to use two systems simultaneously.

Of course, the programs can be also developed on Linux machine directly. Then a dose of work should be put into writing good `Makefile`. Debugging can be done by GDB, but we do not provide debug symbols for Aurora Vision Library.

Runtime considerations

Some architectures might impose restrictions on libavl code. In this section we present pitfalls the user should be aware of.

Homogeneous Multiprocessor/SMP

There are many identical cores. One might have a problem when cores span across multiple physical CPUs, frequent on servers. The CPU's don't share CPU cache, so when execution of thread from CPUx/COREa is moved to CPUy/COREb, cache needs to be updated. It imposes time penalty. A workaround would be to pin threads to specific cores, (set affinity) or limit execution of libavl to specific number of cores on one physical CPU.

- use `taskset` linux command to limit execution on specific cores
- use `OMP_PROC_BIND=TRUE` environment variable to bind threads to cores they started on

Heterogeneous Multiprocessor

There are different kinds of processors the code runs on. Some examples are ARM big.LITTLE architecture, (where the cores mainly differ in maximum speed), or Tegra TX2 (where the cores serve different purpose). This kind of architecture might also suffer from Homogeneous Multiprocessor problems, but might suffer from different set of problems. One have to consider the cores are designed for low power and high performance, single threaded multithreaded optimized. Use the same solutions as in previous point, just take into account what type of algorithm will be executed.

Tegra TX2

This CPU is an example of Heterogeneous Multiprocessor architecture. It comprises of 6 cores: 2 Denver2 4 Cortex-A57. Denver2 core is designed for single thread performance, while Cortex-A57 for multithreaded. One can use both, but with thread binding, so threads are executed on the cores they started on. Limiting to one type of core might be beneficial when power consumption is a factor. Remember that thread binding might bind your application to core you did not want to use. Core 0 is Cortex-A57, core 1 and 2: Denver2, and cores 3-5: Cortex-A57. Core 0 is always active.

Zebra
Aurora™ Vision