// SMART CAMERAS

Getting Started with eVision on Alecs

Introduction to eVision and Alecs

Alecs is an open smart camera, preinstalled with the eVision Web Demonstrator including eVision software libraries. You can use the eVision libraries with Open eVision Studio, a software with an intuitive GUI (graphical user interface).

With eVision libraries and Open eVision Studio, you can use a set of powerful tools for machine vision applications. Conventional operators like pattern matching, code and text reading, image filtering and transformation are available. Moreover, AI-based processing for image classification, segmentation, or localization is available. Alecs supports both C++ and Python, and eVision can take advantage of the embedded GPU.

This document shows how to prototype and build applications with eVision on the Alecs smart camera. It is a step-by-step guide, going from the prerequisites to the launch of C++ or Python programs.

Overview of eVision using the Web Demonstrator

The eVision Web Demonstrator is a pre-installed web application designed to demonstrate the usage of eVision tools without writing any code.

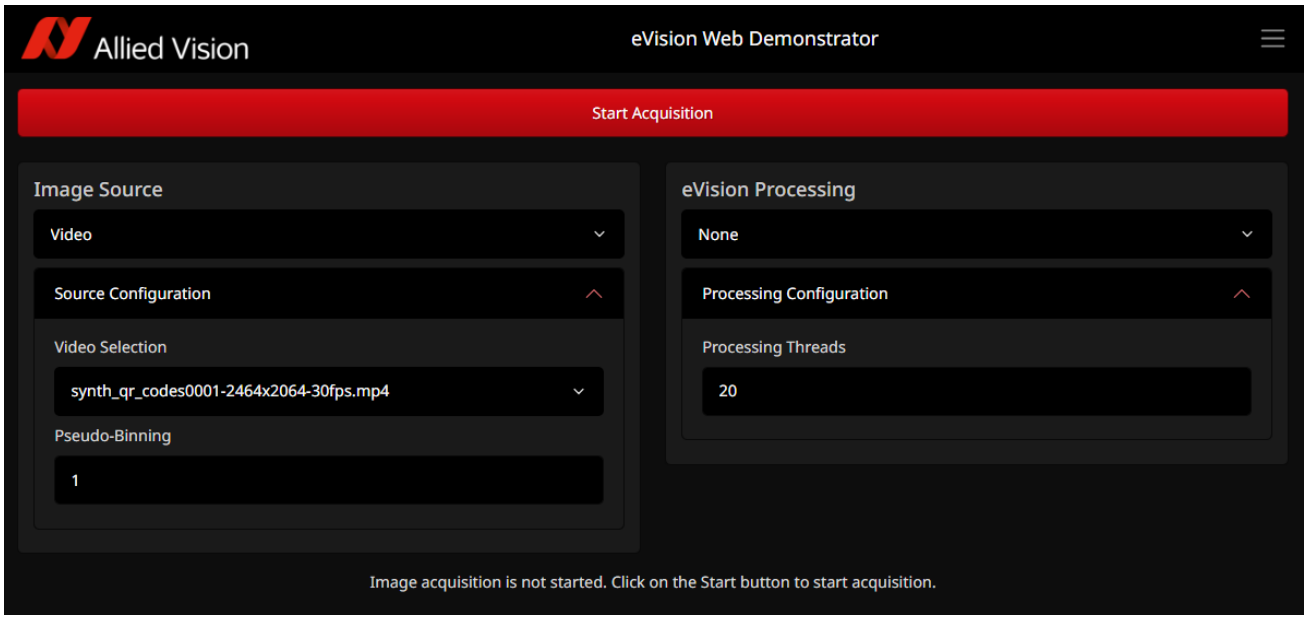


Figure 1: eVision Web Demonstrator overview.

1. Connect the camera to the same network as your PC.
2. Open the IP address of your Alecs on port 8080 (e.g. 192.168.1.10:8080) in a web browser. For more information on how to determine the IP address of your Alecs, refer to the Alecs User Guide.
3. You can test various tools such as EasyQRCode, EasyBarCode, or EasyDeepOCR directly from the interface.

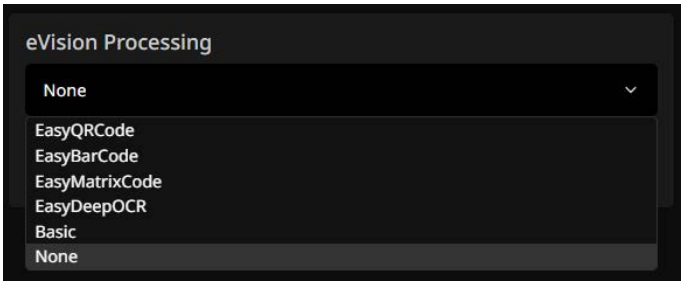


Figure 2: eVision Web Demonstrator, eVision processing tools.

Note on Licensing: The camera includes a 30-day evaluation license that activates automatically the first time you use an eVision processing tool.

Refer to Alecs User Guide, chapter “eVision Web Demonstrator”.

Prerequisites

This guide will help you get started with Alecs smart camera, a smart embedded camera from Allied Vision. It walks you through setting up the environment and deploying a first application using eVision and Vimba X libraries.

Alecs comes with the following software preinstalled:

- eVision 25.06
- Vimba X SDK 2025-1
- GCC compiler 11.4.0
- Python 3.11 0rc1

For all samples elaborated in this document, we use Alecs G1-1242 NX16 with Sony IMX545 sensor. It integrates Alvium CSI-2 technology with NVIDIA® Jetson Orin™ NX16 System on Modules (SoM).

Requirements:

- Skills
  - Basic C++ or Python programming
  - Understanding of image types (Mono8, RGB), camera exposure/gain/pixel format
  - Basic build tooling (CMake for C++; virtual environments for Python)
  - Familiarity with eVision concepts and using Open eVision Studio to design pipelines and export code
- Hardware
  - **Target:** Alecs smart camera device with power and network connectivity.
  - **Host:** A development PC (Windows or Linux)
- Software
  - CMake (release 3.22 ...3.31)
  - Python 3.13 recommended
- Environment variables
  - VIMBAX_SDK_DIR (e.g., /opt/VimbaX_2025-1)
  - OPENEVISION_DIR (e.g., /opt/euresys/Open_eVision_25_06)
- Tools

// An IDE for code prototyping with remote development capabilities (e.g., VS Code, CLion)
// CMake and a C++17 compiler (refer to compile and run section below)

To use eVision API, you will also need a license. A 30-days trial license is installed by default. Refer to the third step for more information about license management.

Notes:

You may need to install additional dependencies to run Neo License Manager and Open eVision Studio in graphical interface mode via SSH.

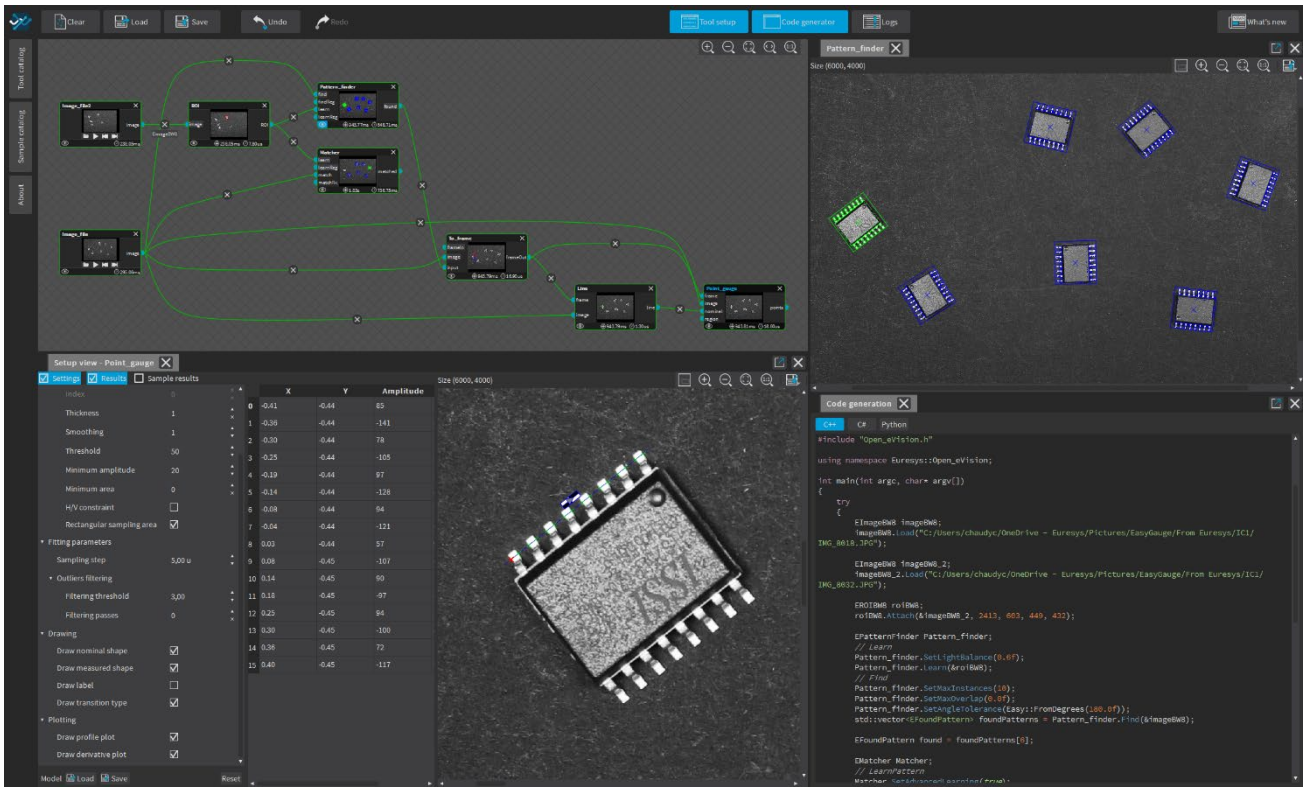
```
# Install the Qt 5 base development package
sudo apt-get install qtbase5-dev


# Install shared library dependencies
sudo apt install libdbus-1-3 libdrm2 libegl1 libfontconfig1 libfreetype6
libglx0 libinput10  libopengl0 libwayland-client0 libwayland-cursor0
libwayland-egl1 libwayland-server0  libx11-6 libx11-xcb1 libxcb-cursor0 libxcb-
glx0 libxcb-icccm4 libxcb-image0 libxcb-keysyms1 libxcb-randr0 libxcb-render-
util0 libxcb-render0 libxcb-shape0 libxcb-shm0 libxcb-sync1 libxcb-xfixes0
libxcb-xkb1 libxcb1 libxkbcommon-x11-0 libxkbcommon0 libxcb-xinerama0
```

For host PCs running on Windows systems, it is recommended to use terminal MobaXterm that supports X11 forwarding.

First step: Design the processing pipeline with Open eVision Studio

What is Open eVision Studio?

Open eVision Studio (short: Studio) is a visual tool to prototype inspection pipelines with eVision libraries. You can load images, configure tools (e.g., filtering, matching, measurement, OCR), try parameters, and export code (C++ and Python snippets) corresponding to your configured tools.



Where to run Studio?

The recommended way is to run Open eVision Studio on your host PC for convenience and speed. You can then copy the exported code into your Alecs project. You can use a remote desktop to connect to Alecs. Then you can run Open eVision Studio and do on-device prototyping directly on Alecs (performance may be lower).

Create and validate a simple pipeline

1. Start Open eVision Studio on the host PC.
2. Load sample images representative of your camera output (Mono8 is the simplest start). If you don't have images yet, capture them from Alecs as described below.
3. Add one or two tools (for example):
   - Preprocessing: smoothing or cropping
   - Code reading: BarCode reader, MatrixCode reader or QRCode reader
4. Validate the pipeline on a small batch of images. Iterate until you get robust results.
5. Use Studio's "Code Generator" to generate C++ and/or Python code snippets for the configured tools. You will paste this code into your application later.

Capturing sample images from Alecs

Use *VimbaXViewer application (path: /opt/VimbaX_2025-1/bin/VimbaXViewer)* for quick access to the Alecs camera to run image acquisition and to access all camera features. Adjust the features (exposure, gain, pixel format) to match your lighting and motion.

Typical first settings for a monochrome workflow: '*PixelFormat* = *Mono8*', '*ExposureAuto* = *Off*', '*ExposureTime*' adjusted for correct brightness, '*Gain*' low to minimize noise.

Refer to Alecs User Guide, chapter "Acquiring a first image"

Second step: Develop the application for Alecs

Below, we outline the core parts of the application. We show a minimal flow in both C++ and Python:

- Initialize camera (Vimba X)
- Grab a frame (Mono8 or RGB8)
- Wrap into eVision image (EImageBW8 or EImageC24)
- Import Studio-exported processing code
- Build/run

1. Image acquisition

// C++

The snippet below initializes Vimba X, opens the camera, and makes it ready for image acquisition.

```
VmbCPP::VmbSystem& sys = VmbCPP::VmbSystem::GetInstance();
VmbErrorType err = sys.Startup();
if (err != VmbErrorSuccess)
{
    std::cerr << "Vimba X Startup failed: " << err << std::endl;
    return false;
}
VmbCPP::CameraPtrVector cameras;
err = sys.GetCameras(cameras);
if (err != VmbErrorSuccess || cameras.empty())
{
    std::cerr << "No camera found." << std::endl;
    return false;
}
m_cam = cameras[0];
err = m_cam->Open(VmbAccessModeFull);
if (err != VmbErrorSuccess)
{
    std::cerr << "Failed to open camera: " << err << std::endl;
    return false;
```

```
}
std::cout << "Camera opened successfully" << std::endl;
SetEnumFeature("PixelFormat", "Mono8");

m_frameObserver = VmbCPP::IFrameObserverPtr(new FrameObserver(m_cam));

err = m_cam->StartContinuousImageAcquisition(5, m_frameObserver);
if (err != VmbErrorSuccess)
{
    std::cerr << "Failed to start continuous acquisition: " << err <<
std::endl;
    return false;
}
```

Frames are acquired through the FrameObserver class. This class will be detailed in the next section.

Setting camera features:

```
VmbErrorType SetEnumFeature(const char* name, const char* value)
{
    VmbCPP::FeaturePtr f;
    VmbErrorType e = m_cam->GetFeatureByName(name, f);
    if (e != VmbErrorSuccess) return e;
    return f->SetValue(value);
}
```

// Python

```
from vmbpy import VmbSystem, PixelFormat
import open_evision as oev
import numpy as np

with VmbSystem.get_instance() as vmb:
    cams = vmb.get_all_cameras()
    if not cams:
        raise RuntimeError('No cameras found')

    with cams[0] as cam:
        cam.PixelFormat.set(PixelFormat.Mono8)
        cam.ExposureAuto.set('Off')
        cam.ExposureTime.set(10000.0)  # microseconds

        frame = cam.get_frame()  # single frame grab helper
        w, h = frame.get_width(), frame.get_height()
        frame = frame.as_numpy_ndarray()  # shape (h, w) for Mono8
        # Copy into eVision image
        eimg = oev.EImageBW8(int(w), int(h))
  np_view = np.asarray(eimg.PixelView)
```

```
    np.copyTo(np_view, frame)

        # TODO: Call Studio processing here
```

2. Image processing

Principle: Design and test tools in Studio, then export code snippets. The exported code typically initializes tool objects, sets parameters, runs the process on input images, and provides result objects.

The code snippet below uses *IFrameObserver* to grab images from the camera, then reads QR Code from an image using eVision *EQRCodeReader* tool.

C++

```
class FrameObserver : public VmbCPP::IFrameObserver
{
public:
    FrameObserver(VmbCPP::CameraPtr pCamera) : IFrameObserver(pCamera), m_frameCount(0)
    {
    }

    void FrameReceived(const VmbCPP::FramePtr pFrame) override
    {
        if (pFrame)
        {
            VmbFrameStatusType status;
            VmbErrorType err = pFrame->GetReceiveStatus(status);
            if (err == VmbErrorSuccess && status == VmbFrameStatusComplete)
            {
                m_frameCount++;
                Euresys::Open_eVision::VimbaXBridge::EVimbaXImageBW8 imageBW8(pFrame);

                // Studio export process
                try
                {
                    Euresys::Open_eVision::EQRCodeReader reader;
                    reader.SetMaxNumCodes(1);
                    auto codes = reader.Read(imageBW8);
                    std::string decoded;
                    if (!codes.empty())
                    {
                        decoded =
codes[0].GetDecodedString(Euresys::Open_eVision::EByteInterpretationMode_Auto);
                        std::cout << "Decoded: " << decoded << std::endl;
                    }
                }
                catch (const Euresys::Open_eVision::EException& ex)
                {
                }
            }
```

```
        }
        m_pCamera->QueueFrame(pFrame);
    }

    uint64_t GetFrameCount() const { return m_frameCount; }

private:
    uint64_t m_frameCount;
};
```

// Python

```
try:
      # after you have eimg
      QRCode_reader = oev.EQRCodeReader()
      QRCode_reader.MaxNumCodes = 1
      QRCode_reader.TimeOut = 5000000
      qrCodes = QRCode_reader.Read(eimg)
      qrCode = qrCodes[0]
      qrCodeDecodedString = qrCode.GetDecodedString(oev.EByteInterpretationMode.Auto)
except oev.EException as exception:
      # Handle exceptions here
      pass
```

3. Use the result

As an illustrative example, we will show how to control Alecs LED panels by attributing green color if QR Code is detected and red color if not.

There are two LED panels on the lateral side of the camera housing. Each one is composed of 5 individually configurable RGB LEDs. Keeping the lens in front of you, the right LED panel is identified by LED Selectors LED0 to 4, and the left LED panel is identified by LED Selector LED5 to 9.

```
bool setLedColor(const Color& color)
{
    if (!m_tlayer) return false;
    VmbErrorType err;
    try
    {
       for (auto i = 0; i < 10; i++)
```

```
        {
            VmbCPP::FeaturePtr f;
            err = m_tlayer->GetFeatureByName("LEDSelector", f);

            if (err == VmbErrorSuccess)
            {
                std::string led_selector = "LED " + std::to_string(i);
                f->SetValue(led_selector.c_str());
            }
            err = m_tlayer->GetFeatureByName("LEDValueBlue", f);
            if (err == VmbErrorSuccess) f->SetValue(color.b);

            err = m_tlayer->GetFeatureByName("LEDValueGreen", f);
            if (err == VmbErrorSuccess) f->SetValue(color.g);

            err = m_tlayer->GetFeatureByName("LEDValueRed", f);
            if (err == VmbErrorSuccess) f->SetValue(color.r);

            err = m_tlayer->GetFeatureByName("LEDValuesSave", f);
            if (err == VmbErrorSuccess) err = f->RunCommand();
        }
    }
    catch (const std::exception& ex)
    {
        std::cerr << ex.what() << std::endl;
        return false;
    }
    return err == VmbErrorSuccess;
}
```

Optionally, you can use Qt GUI to view the live video and results from Alecs smart camera (refer to sample ...). You can also simply save the image on the hard disk with the relevant result overlays (more info on eVision documentation).

Compile and run

// Install system dependencies (Alecs)

// C++

Install required development dependencies on Alecs smart camera for native build and run.

```
sudo apt update
sudo apt install -y cmake build-essential ninja-build
```

Vimba X SDK is installed by default on Alecs smart camera in /opt/VimbaX_2025-1.

// Python

Install Python 3.13:

```
sudo add-apt-repository ppa:deadsnakes/ppa
sudo apt update
sudo apt install python3.13 python3.13-venv
```

Set up a new venv on Alecs, then install dependencies.

```
python3 -m venv .venv && source .venv/bin/activate
pip install vmbpy
Pip install /opt/euresys/Open_eVision_25_06/Python/open_evision-25.6.0+36009-py3-none-linux_aarch64.whl
```

Update the eVision Python binding name according to your release number.

// Set environment variables

On Alecs smart camera, set environment variables to define eVision and Vimba X installation paths to simplify the maintainability of scripts later.

```
export VIMBAX_SDK_DIR=/opt/VimbaX_2025-1
export OPENEVISION_DIR=/opt/euresys/Open_eVision_25_06
```

To make those environment variables permanent, you can append them to .bashrc file of your current shell.

```
$> echo "VIMBAX_SDK_DIR=/opt/VimbaX_2025-1" >> ~/.bashrc
$> echo "OPENEVISION_DIR=/opt/euresys/Open_eVision_25_06" >> ~/.bashrc
$> source ~/.bashrc
```

// Build and run

// C++

- Project skeleton with CMake

```
cmake_minimum_required(VERSION 3.22)
project(AlecsOevDemo CXX)
set(CMAKE_CXX_STANDARD 17)
set(VIMBAX_SDK_DIR $ENV{VIMBAX_SDK_DIR})
set(OPENEVISION_DIR $ENV{OPENEVISION_DIR})

list(APPEND CMAKE_PREFIX_PATH "${VIMBAX_SDK_DIR}/bin" "${VIMBAX_SDK_DIR}/api")
include_directories("${OPENEVISION_DIR}/include")
```

```
add_executable(${PROJECT_NAME} main.cpp)

find_package(Vmb REQUIRED COMPONENTS CPP NAMES Vmb VmbC VmbCPP
VmbImageTransform)
if(Vmb_FOUND)
    target_include_directories(${PROJECT_NAME} PRIVATE
"${VIMBAX_SDK_DIR}/api/include/")
    target_link_directories(${PROJECT_NAME} PRIVATE
"${VIMBAX_SDK_DIR}/api/lib")
    target_link_libraries(${PROJECT_NAME} PRIVATE VmbCPP VmbC
VmbImageTransform)
else()
    message(FATAL_ERROR "VimbaX not found")
endif()

if(DEFINED OPENEVISION_DIR AND EXISTS ${OPENEVISION_DIR})
    target_include_directories(${PROJECT_NAME} PRIVATE
            "${OPENEVISION_DIR}/include"
    )
    target_link_directories(${PROJECT_NAME} PRIVATE
            "${OPENEVISION_DIR}/lib"
            "${OPENEVISION_DIR}/bin"
    )
else()
    message(FATAL_ERROR "OPENEVISION_DIR not set")
endif()
```

- Build on Alecs

```
cd <your_app_dir>
# Configure
# -S : set the source directory
# -B : set the build folder
cmake -S . -B build -DCMAKE_BUILD_TYPE=Release
# Build
cmake --build build --config Release -j 8
# Run
./build/AlecsOevDemo
```

Ensure that eVision and Vimba X runtime libraries are in the system library path.

// Python

Run your script on Alecs

```
python3 your_script.py
```

If you see import/runtime errors, verify that eVision's Python wheel and native libs are installed for your architecture and that the license is recognized.

You will find the full sample code in the annex section.

eVision provides samples on how to use Vimba X SDK with eVision library. You find the samples in:

```
/opt/euresys/Open_eVision_25_06/Sample Programs/…
```

For more examples on how to use Vimba X SDK, you can find a bench of samples in `/opt/VimbaX_2025-1/api/examples/` on Alecs.

Third step: Activate some licenses

eVision's processing capabilities come in thematic sets called libraries. You have, for instance, a library for image pre-processing (EasyImage), a library to read Bar Codes (EasyBarCode)…etc… Each of these libraries requires a specific license to operate.

Alecs comes with a full set of eVision evaluation licenses. These licenses are valid for 30 days starting from your first use of eVision on the camera (for instance through the web demonstrator or a sample program).

***Note**: This evaluation license is only installed at factory time. If you restore or update the firmware of your camera, it won't be available anymore.*

If you ever need to extend your evaluation period, or purchase permanent licenses for some libraries, please contact our Technical Support.

Following your request, you should receive from Euresys one or more ticket numbers that will look like this: `TT5YY-HYAGN-Q2PBU-U7K7J-DKJJQ`

After receiving your ticket, you will need to go on your Alecs and navigate to the following folder: `/opt/euresys/neo_license_manager_25_06/`. There, you will find eVision's Neo License Manager.

We will now show you how to use the command line version of the License Manager to activate your license. Please note that this is the standard use of the License Manager and requires an internet connection. An offline version of this process also exists. For more information, see here.

First, you should create a software container for your license:

```
> ./NeoLicenseManagerCL.exe CreateSoftwareContainer
Software container 130-3245488904 has been created.
```

This command creates the container for all the licenses you will add to your Alecs camera. It will only need to be done once (unless you reset the camera firmware).

You can now activate the license contained in your ticket in the newly created container:

```
> ./NeoLicenseManagerCL.exe Activate -t TT5YY-HYAGN-Q2PBU-U7K7J-DKJJQ -sn
130-3245488904
Activation of ticket TT5YY-HYAGN-Q2PBU-U7K7J-DKJJQ has been done
succesfully.
```

And that's all there is to it. You now have new eVision licenses on your Alecs camera.

You can check that the licenses have been correctly applied by listing them:

```
> ./NeoLicenseManagerCL.exe ListLicenses
Licenses on container 130-3613949292 (software):
eVision EasyOCR
```

***Note**: A GUI version of the license manager also exists if you can access your Alecs camera with a graphical interface.*

Application with Deep Learning inference

This application demonstrates the use of EasyClassify from eVision to classify images using a deep learning model. It displays classification results including predicted labels and Out-of-Distribution (OOD) detection.

To use Deep Learning tools, additional dependencies are required on Alecs:

- open_evision-deep-learning-redist-linux-arm64-25.10.1.37879.deb.tar.gz

They are available for download in Open eVision Deep Learning Studio download area.

Let's start by designing the process using Open eVision Studio.

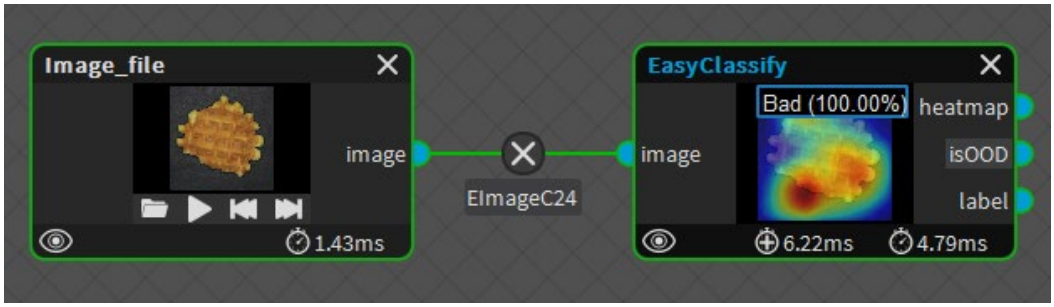


Figure 3: EasyClassify sample from Open eVision Studio.

How it works

1. The application connects to the camera
2. Loads the EasyClassify model from the specified path
3. Configures the classifier:
   - Engine: "EasyDeepLearningEngine_OpenVINO"
   - Device: CPU
4. Starts continuous image acquisition
5. For each acquired image:
   - Converts Vimba X image to eVision EImageC24 format
   - Applies classification using the loaded model
   - Displays classification results:
     - Predicted label (best match)
     - Out-of-Distribution (OOD) status if detected
     - OOD score and threshold

Example output:

```
EasyClassify initialized successfully with model:
/opt/euresys/Open_eVision_25_06/Sample Images/Deep
Learning/EasyClassify/MiniWaffles/Tool 1/MiniWaffles.edltool
Started continuous acquisition ...
Frame 1 - Classification: good
Frame 2 - Classification: bad
Frame 3 - Classification: good (OOD Score: 0.85, Threshold: 0.80)
Acquisition for 10 seconds...
Camera closed
Application stopped.
```

Model file

For this sample we use a pre-trained model (.edltool format) located in:
`/opt/euresys/Open_eVision_25_06/Sample Images/Deep Learning/EasyClassify/MiniWaffles/Tool 1/MiniWaffles.edltool`

You can train the model with your dataset by using eVision Deep Learning Studio. For more details on how to do, refer to Training Deep Learning Tool documentation.

Enable GPU acceleration: To use the embedded GPU instead of CPU, configure the engine in your code:

```
m_classifier.SetEngine("EasyDeepLearningEngine_OpenVINO")
m_classifier.SetActiveDeviceByName("GPU");  // Instead of "CPU"
```

Classification results

The application provides the following information for each classified image:

- Predicted Label: The class with the highest confidence score
- Out-of-Distribution (OOD): Boolean indicating if the image is outside the training distribution
- OOD Score: Confidence score for OOD detection
- OOD Threshold: Threshold value used for OOD determination

For complete source code of the C++ samples, please refer to the sample programs in the annex section.

Summary

In this document, we explained how to develop applications on Alecs using the eVision libraries and tools.

Some topics are not considered, like configuration with web-based panels or results transfer through industrial protocols. These are more specific to a particular industry and are usually custom-made.

The combination of Alecs and eVision libraries offers a comprehensive solution that addresses various challenges of developing Machine Vision solutions, providing a powerful and flexible platform for developing customized and powerful Machine Vision applications.

Resources

1. New Open eVision Studio - Allied Vision
2. eVision Libraries - Allied Vision
3. Open eVision online documentation
4. Alecs - Allied Vision
5. Vimba X SDK - Allied Vision
6. Alecs Downloads - Allied Vision
7. EasyDeepOCR - Allied Vision

Annex

QR code reader sample

```
#include <iostream>
#include <string>
#include <vector>
#include <chrono>
#include <thread>
#include <VmbCPP/VmbCPP.h>
#include "Open_eVision.h"

struct Color
{
  uint8_t r = 0;
  uint8_t g = 0;
  uint8_t b = 0;

  friend std::ostream& operator<<(std::ostream& os, const Color& c)
  {
    os << "[R: " << static_cast<int>(c.r) << " G: " <<
static_cast<int>(c.g) << " B: " << static_cast<int>(c.b) <<
      "]";
    return os;
  }
};

struct CameraInfo
{
  std::string id;
  std::string name;
  std::string model;
  std::string serial;
  std::string interface;

  void print() const
  {
    std::cout << "\tID: " << id
      << "\n\tName: " << name
      << "\n\tModel: " << model
      << "\n\tSerial: " << serial
      << "\n\tInterface: " << interface
      << std::endl;
  }
};

class FrameObserver : public VmbCPP::IFrameObserver
{
public:
  FrameObserver(VmbCPP::CameraPtr pCamera) : IFrameObserver(pCamera),
m_frameCount(0)
  {
```

```
  }

  void FrameReceived(const VmbCPP::FramePtr pFrame) override
  {
    if (pFrame)
    {
      VmbFrameStatusType status;
      VmbErrorType err = pFrame->GetReceiveStatus(status);
      if (err == VmbErrorSuccess && status == VmbFrameStatusComplete)
      {
        m_frameCount++;
        Euresys::Open_eVision::VimbaXBridge::EVimbaXImageBW8
imageBW8(pFrame);

        try
        {
          Euresys::Open_eVision::EQRCodeReader reader;
          reader.SetMaxNumCodes(1);
          auto codes = reader.Read(imageBW8);
          std::string decoded;
          if (!codes.empty())
          {
            decoded =
codes[0].GetDecodedString(Euresys::Open_eVision::EByteInterpretationMode_Au
to);
            std::cout << "Decoded: " << decoded << std::endl;
          }
        }
        catch (const Euresys::Open_eVision::EException& ex)
        {
        }
      }
    }
    m_pCamera->QueueFrame(pFrame);
  }

  uint64_t GetFrameCount() const { return m_frameCount; }

private:
  uint64_t m_frameCount;
};

class Camera
{
public:
  bool open()
  {
    // Vimba X initialization and camera open.
    VmbCPP::VmbSystem& sys = VmbCPP::VmbSystem::GetInstance();
    VmbErrorType err = sys.Startup();
    if (err != VmbErrorSuccess)
    {
      std::cerr << "Vimba X Startup failed: " << err << std::endl;
```

```
      return false;
    }
    VmbCPP::CameraPtrVector cameras;
    err = sys.GetCameras(cameras);
    if (err != VmbErrorSuccess || cameras.empty())
    {
      std::cerr << "No camera found." << std::endl;
      return false;
    }
    m_cam = cameras[0];
    err = m_cam->Open(VmbAccessModeFull);
    if (err != VmbErrorSuccess)
    {
      std::cerr << "Failed to open camera: " << err << std::endl;
      return false;
    }
    std::cout << "Camera opened successfully" << std::endl;

    // print camera basic information
    CameraInfo camera_info;
    m_cam->GetID(camera_info.id);
    m_cam->GetName(camera_info.name);
    m_cam->GetModel(camera_info.model);
    m_cam->GetSerialNumber(camera_info.serial);
    m_cam->GetInterfaceID(camera_info.interface);
    camera_info.print();

    // Set TransportLayer
    m_cam->GetTransportLayer(m_tlayer);

    // Configure stream and pixel format as needed (e.g., Mono8)
    SetEnumFeature("PixelFormat", "Mono8");

    // Create frame observer
    m_frameObserver = VmbCPP::IFrameObserverPtr(new FrameObserver(m_cam));

    // Start continuous acquisition with 5 frame buffers
    err = m_cam->StartContinuousImageAcquisition(5, m_frameObserver);
    if (err != VmbErrorSuccess)
    {
      std::cerr << "Failed to start continuous acquisition: " << err <<
std::endl;
      return false;
    }

    std::cout << "Started continuous video streaming..." << std::endl;
    return true;
  }

  void startAcquisition(int durationSeconds)
  {
    std::cout << "Acquisition for " << durationSeconds << " seconds..." <<
std::endl;
```

```
  std::cout << "Press Ctrl+C to stop early" << std::endl;
  std::this_thread::sleep_for(std::chrono::seconds(durationSeconds));
}

void close()
{
  if (m_cam)
  {
    m_cam->StopContinuousImageAcquisition();
    m_cam->Close();
    m_cam.reset();
  }
  VmbCPP::VmbSystem::GetInstance().Shutdown();
  std::cout << "Camera closed" << std::endl;
}


bool setLedColor(const Color& color)
{
  if (!m_tlayer) return false;

  VmbErrorType err;
  try
  {
    for (auto i = 0; i < 10; i++)
    {
      VmbCPP::FeaturePtr f;
      err = m_tlayer->GetFeatureByName("LEDSelector", f);

      if (err == VmbErrorSuccess)
      {
        std::string led_selector = "LED" + std::to_string(i);
        f->SetValue(led_selector.c_str());
      }
      err = m_tlayer->GetFeatureByName("LEDValueBlue", f);
      if (err == VmbErrorSuccess) f->SetValue(color.b);

      err = m_tlayer->GetFeatureByName("LEDValueGreen", f);
      if (err == VmbErrorSuccess) f->SetValue(color.g);

      err = m_tlayer->GetFeatureByName("LEDValueRed", f);
      if (err == VmbErrorSuccess) f->SetValue(color.r);

      err = m_tlayer->GetFeatureByName("LEDValuesSave", f);
      if (err == VmbErrorSuccess) err = f->RunCommand();
    }
  }
  catch (const std::exception& ex)
  {
    std::cerr << ex.what() << std::endl;
    return false;
  }
  return err == VmbErrorSuccess;
```

```
  }

private:
  VmbCPP::CameraPtr m_cam;
  VmbCPP::FeaturePtr m_featureTmp;
  VmbCPP::IFrameObserverPtr m_frameObserver;
  VmbCPP::TransportLayerPtr m_tlayer;

  // Helpers to set features
  VmbErrorType SetEnumFeature(const char* name, const char* value)
  {
    VmbCPP::FeaturePtr f;
    VmbErrorType e = m_cam->GetFeatureByName(name, f);
    if (e != VmbErrorSuccess) return e;
    return f->SetValue(value);
  }

  VmbErrorType SetBoolFeature(const char* name, bool value)
  {
    VmbCPP::FeaturePtr f;
    VmbErrorType e = m_cam->GetFeatureByName(name, f);
    if (e != VmbErrorSuccess) return e;
    return f->SetValue(value);
  }

  VmbErrorType SetIntFeature(const char* name, int64_t value)
  {
    VmbCPP::FeaturePtr f;
    VmbErrorType e = m_cam->GetFeatureByName(name, f);
    if (e != VmbErrorSuccess) return e;
    return f->SetValue(value);
  }
};


int main()
{
  std::cout << "Alecs QR code reader demo (Vimba X)" << std::endl;

  Camera cam;
  if (!cam.open())
  {
    std::cerr << "Failed to open camera." << std::endl;
    return 2;
  }

  Color color = {255, 0, 0}; //Red
  cam.setLedColor(color);
  cam.startAcquisition(10); //seconds
  color = {0, 255, 0}; //Green
  cam.setLedColor(color);
  cam.close();
  std::cout << "Application end." << std::endl;
```

```
  return 0;
}
```

EasyClassify sample

```
#include <iostream>
#include <string>
#include <vector>
#include <chrono>
#include <thread>
#include <cstring>
#include <VmbCPP/VmbCPP.h>
#include "Open_eVision.h"

using namespace Euresys::Open_eVision::EasyDeepLearning;
using namespace Euresys::Open_eVision;

struct Color
{
    uint8_t r = 0;
    uint8_t g = 0;
    uint8_t b = 0;
    friend std::ostream& operator<<(std::ostream& os, const Color& c)
    {
        os << "[R=" << static_cast<int>(c.r) << " G=" <<
static_cast<int>(c.g) << " B=" << static_cast<int>(c.b) << "]";
        return os;
    }
};

struct CameraInfo
{
    std::string id;
    std::string name;
    std::string model;
    std::string serial;
    std::string interface;

    void print() const
    {
        std::cout << "\tID: " << id
            << "\n\tName: " << name
            << "\n\tModel: " << model
            << "\n\tSerial: " << serial
            << "\n\tInterface: " << interface
            << std::endl;
    }
};

class FrameObserver : public VmbCPP::IFrameObserver
{
public:
    FrameObserver(VmbCPP::CameraPtr pCamera, const std::string& modelPath)
        : IFrameObserver(pCamera), m_frameCount(0),
m_classifierInitialized(false), m_modelPath(modelPath)
    {
```

```
        // Initialize Classifier once
        try
        {
            m_classifier.Load(m_modelPath);
            m_classifier.SetEngine("EasyDeepLearningEngine_OpenVINO");
            m_classifier.SetActiveDeviceByName("GPU");

            m_classifierInitialized = true;
            std::cout << "EasyClassify initialized successfully with model:
" << m_modelPath << std::endl;
        }
        catch (const EException& ex)
        {
            std::cerr << "Failed to initialize EasyClassify: " << ex.what()
<< std::endl;
            m_classifierInitialized = false;
        }
    }

    void FrameReceived(const VmbCPP::FramePtr pFrame) override
    {
        if (pFrame)
        {
            VmbFrameStatusType status;
            VmbErrorType err = pFrame->GetReceiveStatus(status);
            if (err == VmbErrorSuccess && status == VmbFrameStatusComplete)
            {
                try
                {
                    m_frameCount++;

                    if (!m_classifierInitialized) return;

                    VimbaXBridge::EVimbaXImageC24 imageC24(pFrame);

                    // Classify the image
                    EClassificationResult result =
m_classifier.Classify(imageC24);

                    // Get classification results
                    std::string predictedLabel = result.GetBestLabel();
                    bool isOOD = result.IsOOD();
                    float oodScore = result.GetOODScore();
                    float oodThreshold = result.GetOODThreshold();

                    // Display results
                    std::cout << "Frame " << m_frameCount << " -
Classification: " << predictedLabel;
                    if (isOOD)
                    {
                        std::cout << " (OOD Score: " << oodScore
                                  << ", Threshold: " << oodThreshold <<
")";
```

```
                    }
                    std::cout << std::endl;
                }
                catch (const EException& ex)
                {
                    std::cerr << "Failed to classify image: " << ex.what()
<< std::endl;
                }
            }
        }
        m_pCamera->QueueFrame(pFrame);
    }

    uint64_t GetFrameCount() const { return m_frameCount; }

private:
    uint64_t m_frameCount;
    EClassifier m_classifier;
    bool m_classifierInitialized;
    std::string m_modelPath;
};

class Camera
{
public:
    bool open(const std::string& modelPath)
    {
        // Vimba X initialization and camera open.
        VmbCPP::VmbSystem& sys = VmbCPP::VmbSystem::GetInstance();
        VmbErrorType err = sys.Startup();
        if (err != VmbErrorSuccess)
        {
            std::cerr << "Vimba X Startup failed: " << err << std::endl;
            return false;
        }
        VmbCPP::CameraPtrVector cameras;
        err = sys.GetCameras(cameras);
        if (err != VmbErrorSuccess || cameras.empty())
        {
            std::cerr << "No camera found." << std::endl;
            return false;
        }
        m_cam = cameras[0];
        err = m_cam->Open(VmbAccessModeFull);
        if (err != VmbErrorSuccess)
        {
            std::cerr << "Failed to open camera: " << err << std::endl;
            return false;
        }
        std::cout << "Camera opened successfully" << std::endl;

        // print camera basic information
        CameraInfo camera_info;
```

```
        m_cam->GetID(camera_info.id);
        m_cam->GetName(camera_info.name);
        m_cam->GetModel(camera_info.model);
        m_cam->GetSerialNumber(camera_info.serial);
        m_cam->GetInterfaceID(camera_info.interface);
        camera_info.print();

        // Set TransportLayer
        m_cam->GetTransportLayer(m_tlayer);

        if (SetEnumFeature("PixelFormat", "BGR8") != VmbErrorSuccess)
        {
            if (SetEnumFeature("PixelFormat", "RGB8") != VmbErrorSuccess)
            {
                SetEnumFeature("PixelFormat", "Mono8");
            }
        }

        m_frameObserver = VmbCPP::IFrameObserverPtr(new
FrameObserver(m_cam, modelPath));

        err = m_cam->StartContinuousImageAcquisition(5, m_frameObserver);
        if (err != VmbErrorSuccess)
        {
            std::cerr << "Failed to start continuous acquisition: " << err
<< std::endl;
            return false;
        }

        std::cout << "Started continuous acquisition ..." << std::endl;
        return true;
    }

    void streamVideo(int durationSeconds)
    {
        std::cout << "Acquisition for " << durationSeconds << " seconds..."
<< std::endl;
        std::cout << "Press Ctrl+C to stop early" << std::endl;
        std::this_thread::sleep_for(std::chrono::seconds(durationSeconds));
    }

    void close()
    {
        if (m_cam)
        {
            m_cam->StopContinuousImageAcquisition();
            m_cam->Close();
            m_cam.reset();
        }
        VmbCPP::VmbSystem::GetInstance().Shutdown();
        std::cout << "Camera closed" << std::endl;
    }
```

```
bool setLedColor(const Color& color)
{
    if (!m_tlayer) return false;

    VmbErrorType err = VmbErrorSuccess;
    try
    {
        for (auto i = 5; i < 10; i++)
        {
            VmbCPP::FeaturePtr f;

            // Select the LED first
            err = m_tlayer->GetFeatureByName("LEDSelector", f);
            if (err != VmbErrorSuccess) continue;

            std::string led_selector = "LED" + std::to_string(i);
            err = f->SetValue(led_selector.c_str());
            if (err != VmbErrorSuccess) continue;

            // Set Blue value
            err = m_tlayer->GetFeatureByName("LEDValueBlue", f);
            if (err == VmbErrorSuccess)
            {
                err = f->SetValue(static_cast<int64_t>(color.b));
                if (err != VmbErrorSuccess) continue;
            }

            // Set Green value
            err = m_tlayer->GetFeatureByName("LEDValueGreen", f);
            if (err == VmbErrorSuccess)
            {
                err = f->SetValue(static_cast<int64_t>(color.g));
                if (err != VmbErrorSuccess) continue;
            }

            // Set Red value
            err = m_tlayer->GetFeatureByName("LEDValueRed", f);
            if (err == VmbErrorSuccess)
            {
                err = f->SetValue(static_cast<int64_t>(color.r));
                if (err != VmbErrorSuccess) continue;
            }

            // Save the values for this LED
            err = m_tlayer->GetFeatureByName("LEDValuesSave", f);
            if (err == VmbErrorSuccess)
            {
                err = f->RunCommand();
            }
        }
    }
    catch (const std::exception& ex)
    {
```

```
            std::cerr << ex.what() << std::endl;
            return false;
        }
        return err == VmbErrorSuccess;
    }

private:
    VmbCPP::CameraPtr m_cam;
    VmbCPP::FeaturePtr m_featureTmp;
    VmbCPP::IFrameObserverPtr m_frameObserver;
    VmbCPP::TransportLayerPtr m_tlayer;

    // Helpers to set features
    VmbErrorType SetEnumFeature(const char* name, const char* value)
    {
        VmbCPP::FeaturePtr f;
        VmbErrorType e = m_cam->GetFeatureByName(name, f);
        if (e != VmbErrorSuccess) return e;
        return f->SetValue(value);
    }

    VmbErrorType SetBoolFeature(const char* name, bool value)
    {
        VmbCPP::FeaturePtr f;
        VmbErrorType e = m_cam->GetFeatureByName(name, f);
        if (e != VmbErrorSuccess) return e;
        return f->SetValue(value);
    }

    VmbErrorType SetIntFeature(const char* name, int64_t value)
    {
        VmbCPP::FeaturePtr f;
        VmbErrorType e = m_cam->GetFeatureByName(name, f);
        if (e != VmbErrorSuccess) return e;
        return f->SetValue(value);
    }
};


int main(int argc, char* argv[])
{
    std::cout << "Alecs EasyClassify demo (Vimba X)" << std::endl;

    // Default model path - can be overridden via command line argument
    std::string modelPath = "/opt/euresys/Open_eVision_25_06/Sample
Images/Deep Learning/EasyClassify/MiniWaffles/Tool 1/MiniWaffles.edltool";

    if (argc >= 2)
    {
        modelPath = argv[1];
    }
    else
    {
```

```
        std::cout << "Usage: " << argv[0] << " [model_path.edltool]" <<
std::endl;
        std::cout << "Using default model path: " << modelPath <<
std::endl;
        std::cout << "Note: Update the model path in the code or pass it as
argument" << std::endl;
    }

    Camera cam;
    if (!cam.open(modelPath))
    {
        std::cerr << "Failed to open camera." << std::endl;
        return 2;
    }

    cam.streamVideo(10);
    cam.close();
    std::cout << "Application stopped." << std::endl;
    return 0;
}
```