EVD1 Study guide

Microcontroller optimized image processing

OV7670

DVP with SmartDMA

FRDM-MCXN947

MCXN947VDF

*Image processing
QQVGA: 160x120
YUV, RGB888, uint8,
int16, int32, float*

USB video class device

USB MCU-Link: SWD and VCOM

Laptop / PC

made by ESE
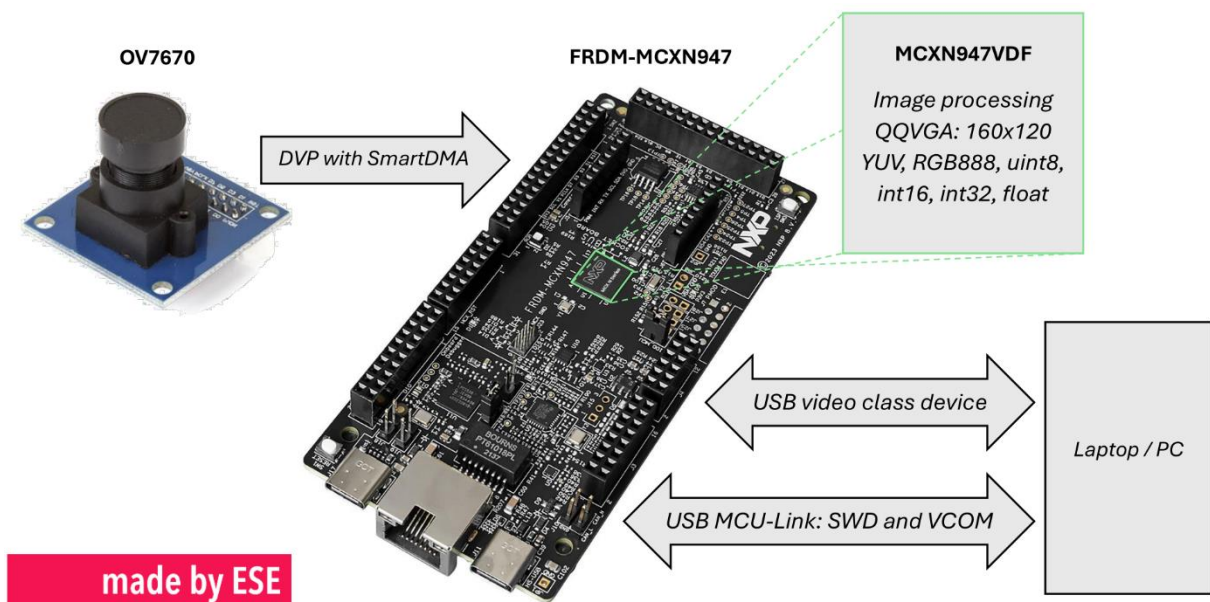
By Hugo Arends
Embedded Vision and Machine Learning
HAN Embedded Systems Engineering

Contents

Revisions

Version	When	Who	What
1.0	Oct. 2025	Hugo Arends	First version of the document for the 2025/2026 academic year.

Introduction

This document is the study guide for the second period of the EVD1 course. The EVD1 course is part of the Embedded Vision and Machine Learning module taught as part of the HAN Embedded Systems Engineering program. The goal of this module is to learn how to implement and use image processing operators on a microcontroller. The course will therefor discuss how to implement image processing operators in de C programming language with special attention for performance considerations.

This document comes with the following other resources:

- Example images
- Source files with template functions for image processing operators
- Powerpoint presentations
- VS Code workspace with one or more hardware target projects
- VS Code workspace with one or more PC app projects

These resources are made available online.

The assignments for each week are described in this document. The theory is discussed in class by means of powerpoint presentations. The first week starts with the installation of the software development environment, configuring the hardware, testing the installation and getting familiar with writing and testing code. In the second week performance enhancement of code execution is discussed. The third week discusses image enhancement by means of filtering. The fourth week discusses algorithms for automatic thresholding for separating objects from the background. The fifth week reduces binary objects as much as possible to their basic shapes. And finally, the sixth week discusses techniques to extract object features. This document starts, however, with a description of the final assignment.

This course is organized as a workshop. The planning and content of the classes is as follows:

Week	Full time	Part time
1	Class 1: Theory + Lab Class 2: Theory + Lab	Class: Theory + Lab
2	Class 1: Theory + Lab Class 2: Theory + Lab	Class: Theory + Lab
3	Class 1: Theory + Lab Class 2: Lab	Class: Theory + Lab
4	Class 1: Theory + Lab Class 2: Lab	Class: Theory + Lab
5	Class 1: Theory + Lab Class 2: Lab	Class: Theory + Lab
6	Class 1: Theory + Lab Class 2: Lab	Class: Theory + Lab
7	Class 1: Lab Class 2: Lab	Class: Lab

As can be seen in this overview, there will be plenty of opportunity to work on the exercises as described in this document and ask questions during lab classes.

Have fun!
Hugo

Final assignment

In the second period, various image processing operators are implemented including a unique operator by each student. During an individual oral assessment, the student's competence in designing, implementing and testing image processing operators for embedded devices is assessed.

During the assessment, the embedded device correctly classifies the following three shapes. All other shapes are classified as 'unknown'.
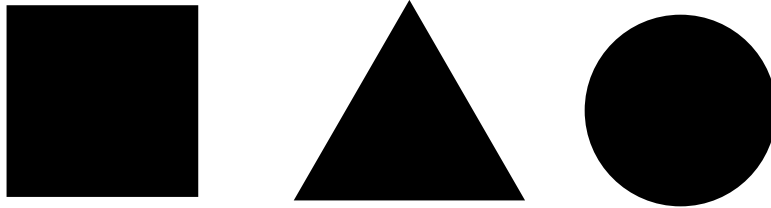
The visualization of the classification is up to the student. Examples are colour overlays in the camera image, coloured bounding boxes, text on the image, LEDs on the development kit, or a combination of these examples.

To accomplish this goal, the student implements a set of mandatory image processing operators. These operators, and other operators which will be made available, can be used to fulfil the task.

The solution has an expected framerate of 30 fps. A faster framerate is considered better.

Although there are many possible solutions, the object recognition task is the same for all students. To increase the authenticity of the examination, all students also design, implement and test a 'unique' image processing operator.

1   Professional products

The assessment consists of the following professional products:

- source code
- demonstration of the object recognition
- powerpoint presentation of a unique operator
- demonstration of a unique operator
- logbook or comment in source code

The mandatory operators to be implemented are discussed during the lessons. *All these operators must function correctly in order to participate in the assessment.* In other words, they must all give a pass in the unit tests. The unique operator must also function correctly. For this, students devise their own (unit) test(s) and demonstrate the correct operation as part of the assessment.

During the classes in the first weeks the lecturer decides which students is assigned which unique operator.

2 Handin

Prior to the oral assessment, the following files must be handed in via https://handin.han.nl/ in a single ZIP archive with the assessor(s):

- All files in the folder **/evdk5/evdk_operators/*.***
- The file **evdk5/evdk_workspace_targets/frdmmcxn947_evdk5_n/source/main.c**
- The file **evdk5/evdk_workspace_targets/frdmmcxn947_evdk5_n/source/ clearUint8Image_cm33.s**
- A (powerpoint) presentation of the unique operator

3 Oral assessment

By participating in the assessment, you declare that all the work submitted was designed, realised and tested by yourself. This means that you are allowed to *collaborate*, but not *copy verbatim*. If code checks by the examiner reveal that large parts of code have been copied verbatim, the assessor will report this to the board of examiners.

A planning of the oral assessments is drawn up during class in the final weeks of the period.

The oral assessment takes 20 minutes and is planned as follows:

Time (min.)	Activity
~ 5	**Demonstration of the assignment** *The EVDK board is used to demonstrate that the three shapes – circle, square, and triangle. The next page of this document includes the shapes that will be used during the assessment. The source code, the used operators, the classification method, and overall performance are explained.*
~ 5	**Questions on a random operator** *The student is asked questions about the functional operation and technical implementation of a random operator based on the source code. The student must be able to provide an explanation and answer questions adequately. The handwritten logbook created by the student may be consulted.*
~ 10	**Presentation & demonstration unique operator** *The functional operation (to-the-point description and example) and the technical implementation (e.g. flowchart, mathematical equations, code snippets), of the unique operator is explained in a short presentation (approximately 5 sheets). The final slide must __cite all used sources__. Next, the operator is demonstrated. The student may use the EVDK board and/or the PC development environment for this demonstration.*

As time is limited, students must ensure the hardware and software is ready for use when entering the exam room. If this is not the case, the examiner will have insufficient time to assess the knowledge and skills, which will result in a fail.

A printed copy of the sheet on the next page is used during the assessment. This page can be printed for testing.

EVD1 Assessment
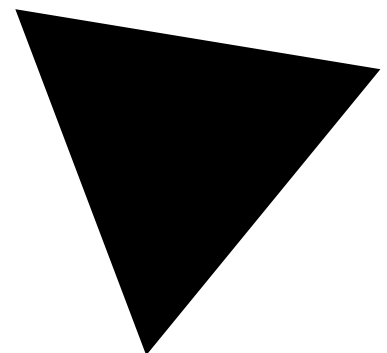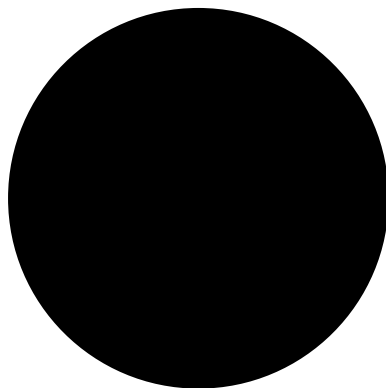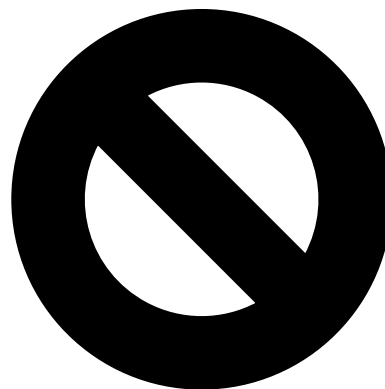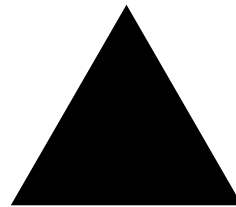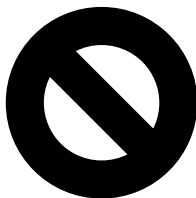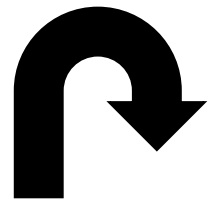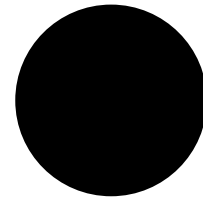
Week 1

The goal for this week is to install the software development environment, configure the hardware, test the installation and get familiar with writing and testing code.

1   SDE setup

This section describes how to setup the Software Development Environment (SDE). The tools described in the first paragraph (1.1) have been installed during EVD1 period 1 so this can be skipped if you have attended these classes.

1.1   VS Code, MinGW and OpenCV

Note. Skip this paragraph if VS Code, the MinGW toolchain and OpenCV have already been installed.

1. Install software
    - Install VS Code
    - Install the MinGW toolchain as described here.
      Note. You can stop at the section "Create a Hello World app".
    - Install the CMake extension for VS Code
    - Download OpenCV – x.y.z sources, for example:
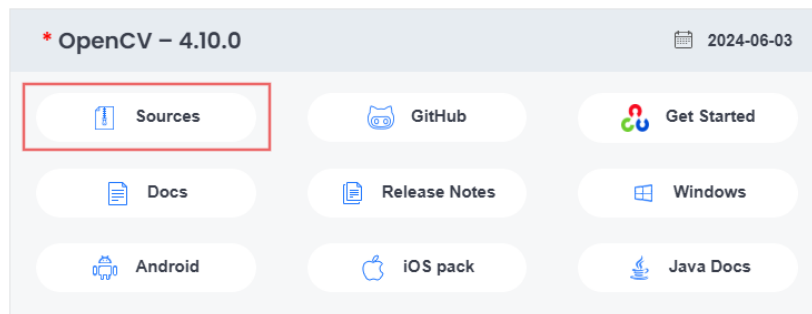
    - Unzip the downloaded sources to folder c:\opencv-x.y.z

2. Update PATH variable
    - Add the following paths to the system environment variable PATH. The system environment variables can be accessed as follows:
        o Press Windows key
        o Type "environment"
        o Select "Edit the system environment variables"
        o Click the *Environment Variables* button
        o Select Path
        o Click *Edit...*
    - Add the following two paths:
        o *C:\opencv-x.y.z\build\bin*
        o *C:\opencv-x.y.z\build\lib*
    - The result should look similar to the adjacent image.
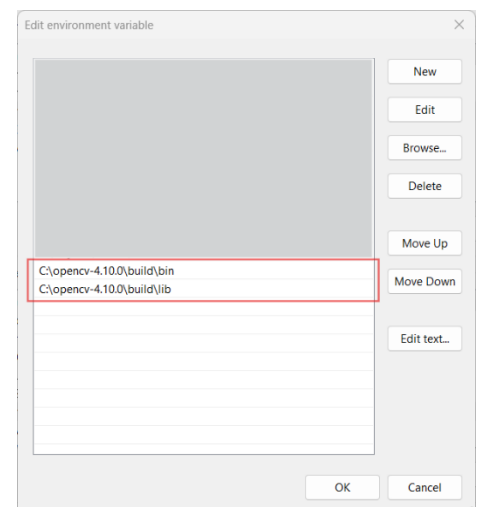
3. Configure and build OpenCV in VS Code
    - Start VS Code

- File > Open folder… > c:\opencv-x.y.z
- Open the CMake extension (see adjacent image)
- Select the kit *GCC a.b.c x86_64-w64-mingw32 (ucrt64)*
  Note. Configuration will take a while
- Select Release.
- Click ⚙*Build*
  Note. Building will take even longer.
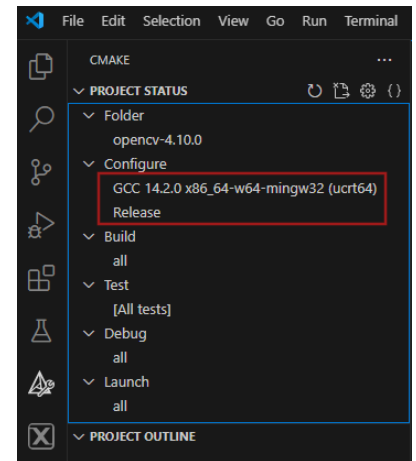
1.2 MCUXpresso for VS Code extension

Download and install the MCUXpresso for VS Code extension as follows:

1. Start VS Code
2. Install the MCUXpresso for VS Code extension: [MCUXpresso for VS Code](#)

1.3 MCUXpresso for VS Code components

After installing the MCUXpresso for VS Code extension, you should be prompted to start the *MCUXpresso Installer*. If not, click *Open MCUXpresso Installer* from the Quickstart panel in the MCUXpresso for VS Code extension.

The *MCUXpresso Installer* looks like the adjacent image:

1. In this installer, select (at least) the following options:
   - MCUXpresso SDK Developer
   - ARM GNU Toolchain
   - Standalone Toolchain Add-ons
   - LinkServer
   - SEGGER J-Link
2. Click Install.
3. Close the installer when the installation is finished.
   This will take a while.
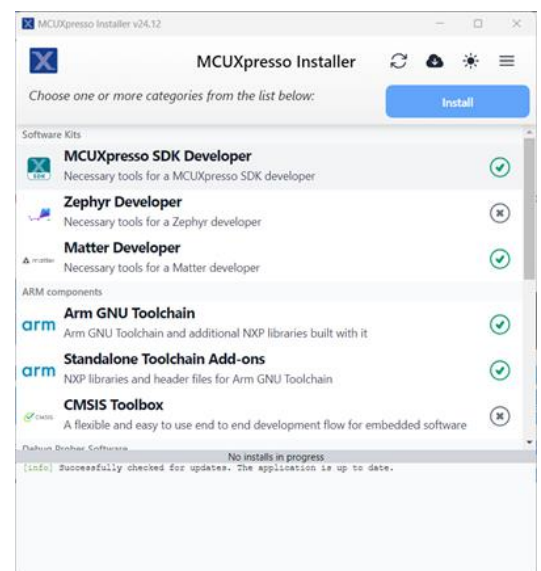
1.4   MCUXpresso for VS Code SDK

Import the Software Development Kit (SDK) for the FRDM-MCXN947 development board.

1. Start VS Code
2. Select the MCUXpresso for VS Code extension
3. In the Quickstart panel, click
   + Import Repository
4. Select *Remote archive*
5. Select package *FRDM-MCXN947*
6. Select version 25.09.00. (Newer versions might also work, but have not yet been tested.)
7. Set the default name
8. Set a location, such as c:\sdks
   Important. *Use a location with short path names and without special characters, such as hyphens ('-')*
9. Deselect *Create a Git repository*
10. Select *I agree*
11. Click Import
12. Wait for the download and extraction to complete (see VS Code notifications in the right bottom).
13. Close VS Code.

1.5   Git client

Download and install a git client (such as [Git for Windows](#)). You can check to see if a git client is already installed by opening a command prompt and type:

```
> git version
```
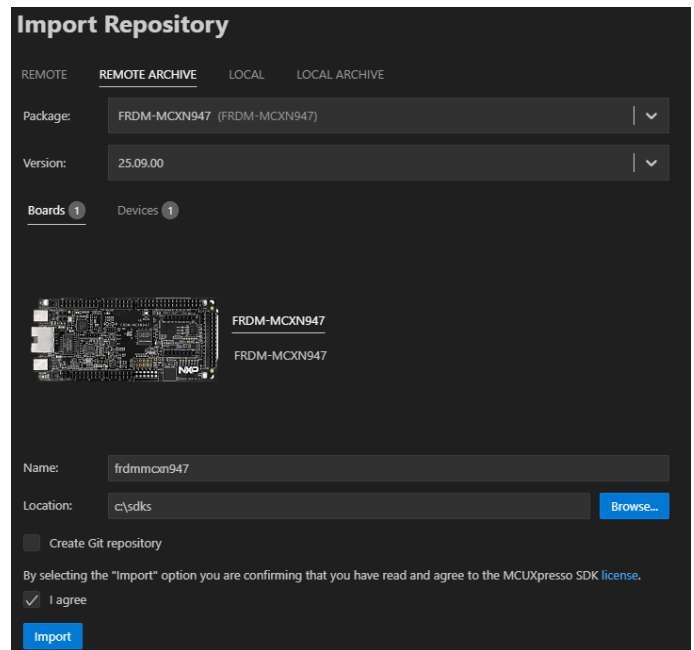
If a git version is printed, a git client is already installed and installation can be omitted.

TIP: For an overview of common git commands, see:

   https://docs.gitlab.com/ee/gitlab-basics/start-using-git.html

1.6   Course files

Clone the course files from the git repo with the following command. **Important**. *Clone it to a location with a short path name and without special characters, such as hyphens ('-'), ampersand ('&'), etc. This will prevent toolchain problems later during compilation*.

Use a command prompt and browse to the folder of your choice, for example c:\evml. Run the following command:

```
> git clone https://gitlab.com/hugoarends/evdk5-2526.git
```

2 FRDM-MCXN947 changes

The FRDM-MCXN947 board does not work out-of-the-box with the provided project. Several jumpers need to be resoldered and it is recommended to change the MCU-Link firmware. This section describes how this is done.

2.1 Resoldering jumpers

Three jumpers need to be resoldered, otherwise the OV7670 camera module cannot be connected to the J9 connector. The following images highlight these three changes:

Jumper position out-of-the-box   *Jumper position after resoldering*
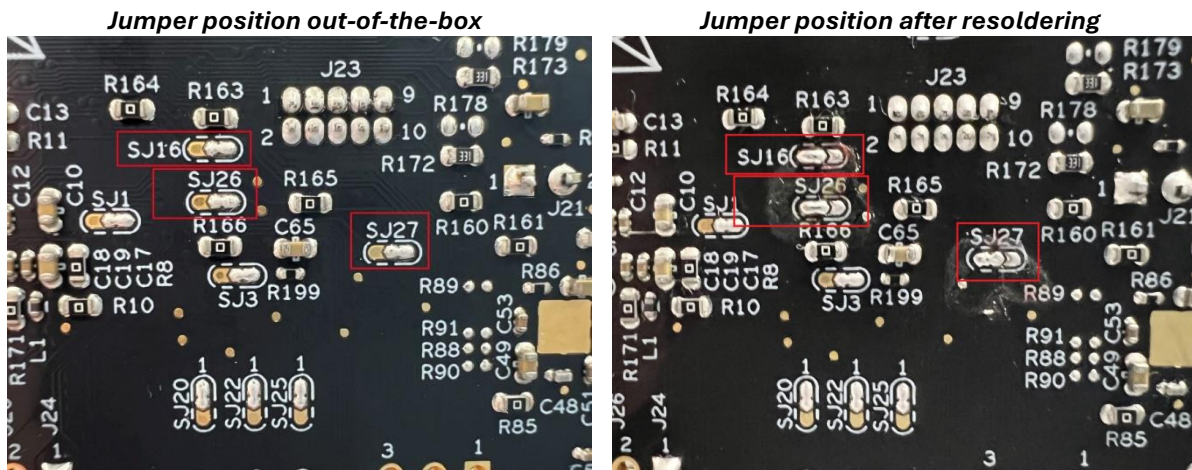
FYI. The following images show the location of the jumpers at the bottom of the FRDM-MCXN947 board and how the changes are reflected in the schematic diagram.
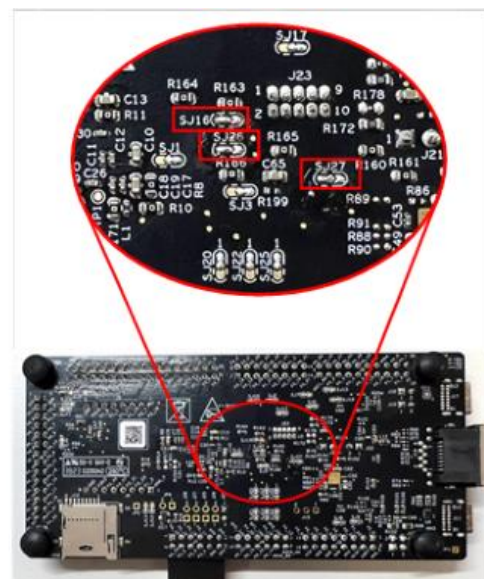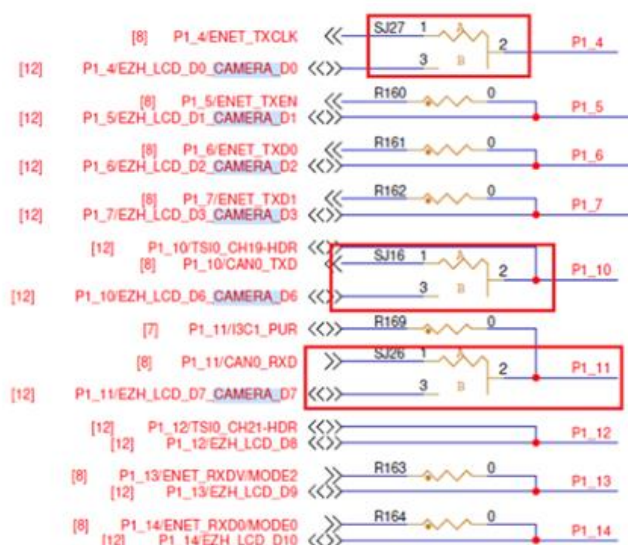
These hardware changes are also described online.

2.2 MCU-Link firmware update

The FRDM-MCXN947 board comes with an MCU-Link debug probe. Out of the box, the CMSIS-DAP firmware is installed. However, J-Link firmware is preferred. Changing the MCU-Link to J-Link firmware is described here. As this is a one-time action, you can also ask your lecturer.

3 Hello Webcam!

In this section the SDE setup and hardware changes will be verified by building the source files in VS Code, programming the FRDM-MCXN947 board and see if your laptop displays images in the default camera app.

3.1 Building the source files

Build the source files:

1. Open the project by double clicking the VS Code workspace file:
   .\evdk5\evdk_workspace_targets**evdk5_targets.code-workspace**
2. Open the *MCUXpresso for VS Code* extension
3. Right click the project **frdmmcxn947_evdk5_0**
4. Select Configure -> Associate Repository
   - Select the installed SDK
   - If prompted for a board, select the FRDM-MCXN947
   - If prompted for a core, select cm33_core0
5. Right click the project **frdmmcxn947_evdk5_0**
6. Select Configure -> Associate Toolchain
   - Select the installed ARM GNU toolchain
7. Build the project:
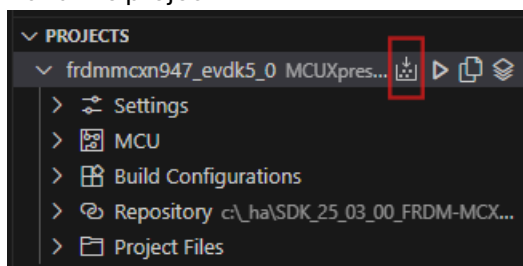
   TIP. While the project is building, the terminal will display several warnings. These warnings were added to the project intentionally and will help to identify what image processing operators need to be implemented. When the source code is handed in, no more warnings are allowed!

   NOTE. If you run into build issues, chances are that project paths are too long and/or paths contains characters that are not supported by the toolchain (such as hyphens). If so, move the project and the SDK to a location with a shorter path and/or remove unsupported characters from the path.

3.2 Programming the FRDM-MCXN947

Program the MCXN947 microcontroller as follows:

1. Connect the FRDM-MCXN947 development board to your laptop using connector *J17 MCU-Link USB*.
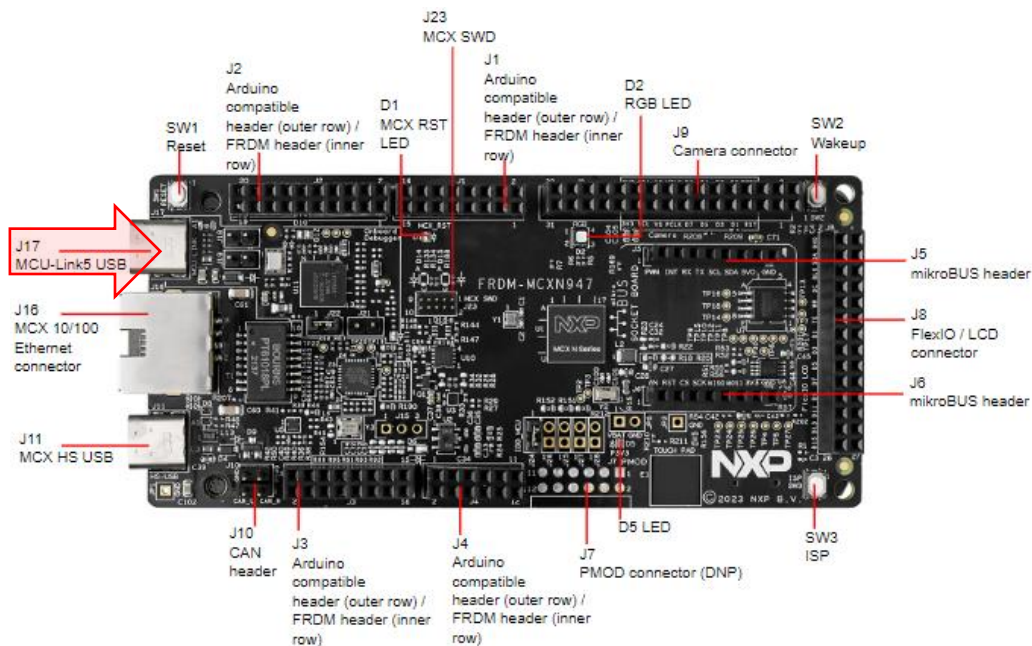
2. Click the Debug button
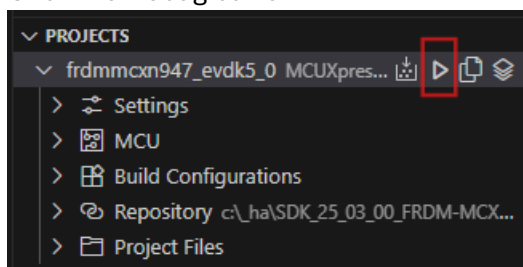
   TIP. The SEGGER tooling will automatically detect if the J-Link firmware is not up-to-date and prompts to upgrade. Always click yes. Furthermore, J-Link can be used for free, however you will get a daily reminder of the fact that you are using the free version.

3. Click Continue (F5):

   TIP: Instead of starting the debugger, the target can be flashed directly with:
   - *Right clicking the project*
   - *Click Flash the Selected Target*

3.3 Display images in a camera app

Verify all of the steps above by displaying live images from the FRDM-MCXN947 board.

1. Connect the FRDM-MCXN947 development board to your laptop using connector *J11 MCX HS USB*.
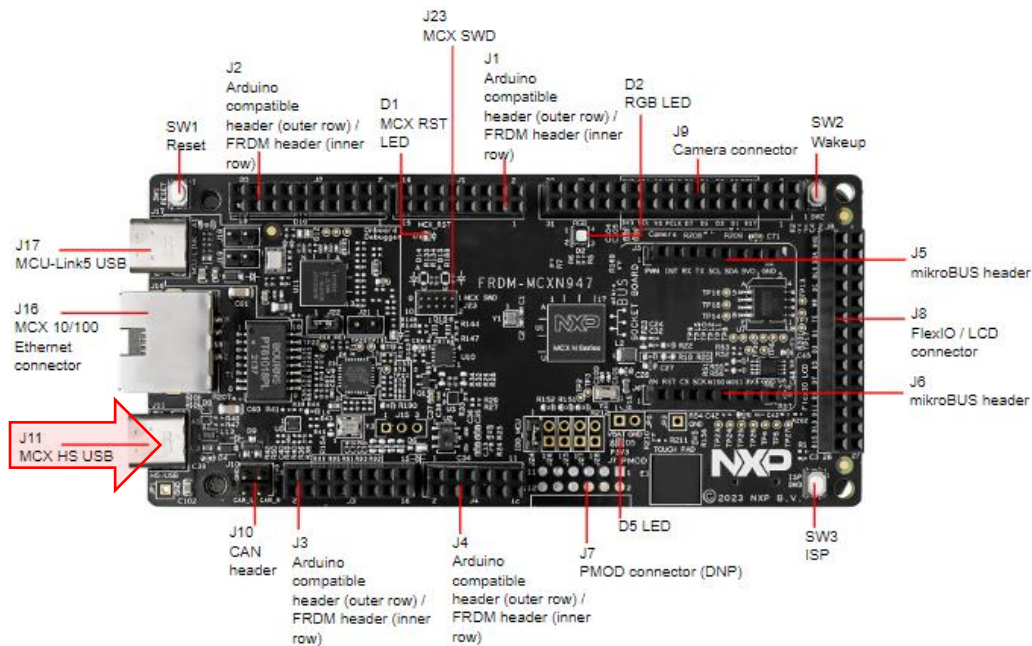
2. Open a camera app. On Windows: open Start menu and type *Camera*.
3. If your laptop's internal webcam is selected, change to the EVDK5 by clicking

4   Change and run the example project

The default example selected in main() is exampleWebcamBgr888().

- Find the main() function in the file main.c.
- Comment the function exampleWebcamBgr888().
- Uncomment the function exampleWebcamBgr888TestPattern().
- Build and run the application.

When opening the camera app on your laptop, the image doesn't make sense. It is either completely black, or it might show (coloured) noise.

- Find the function exampleWebcamBgr888TestPattern() in the file main.c.
- In this function, locate the following todo:
  `// \todo Copy-and-paste week 1 code here`

- Change the image processing pipeline by copy-and-pasting the following code:

```c
// \todo Copy-and-paste week 1 code here

// Set all pixels to color
bgr888_pixel_t *bgr888_pixel = (bgr888_pixel_t *)bgr888->data;
int32_t len = bgr888->rows * bgr888->cols;

while(len > 0)
{
    *bgr888_pixel = color;

    bgr888_pixel++;
    len--;
}
```

Rebuild and run the application and verify using the camera app on your laptop that eight different colours are displayed, changing colour every second.

The example also prints a messages using the PRINTF() function to the serial interface. Open a terminal application (115200,8,n,1) and verify that the BGR values are printed in hexadecimal format.

TIP. A serial monitor can be integrated in VS Code by installing the [Serial Monitor](#) extension.

5   Run OpenCV webcam app

Connect the FRDM-MCXN947 board to your laptop and make sure that exampleWebcamBgr888TestPattern() is running (with the added code as mentioned in the previous paragraph). Also make sure that the (Windows) camera app is not running.

1. Open the project by double clicking the VS Code workspace file:
   *.\evdk5\evdk_workspace_apps***evdk5_apps.code-workspace**
2. Open the CMake extension
3. In project status, select the following options (see adjacent image):
   - Folder: evdk5_webcam
   - Configure:
     GCC a.b.c x86_64-w64-mingw32 (ucrt64)
     Release
4. Launch the selected target, as depicted in the adjacent image. This will build and run the application.

When the app runs successfully, it shows an image similar to what could be seen in the camera app. Now, however, it is an OpenCV image viewer, demonstrating that we can use the image in any OpenCV project as well.

5. With the image view selected, press 's' on your keyboard. This will save an image to your disk, which might be convenient for future testing. The saved image is located
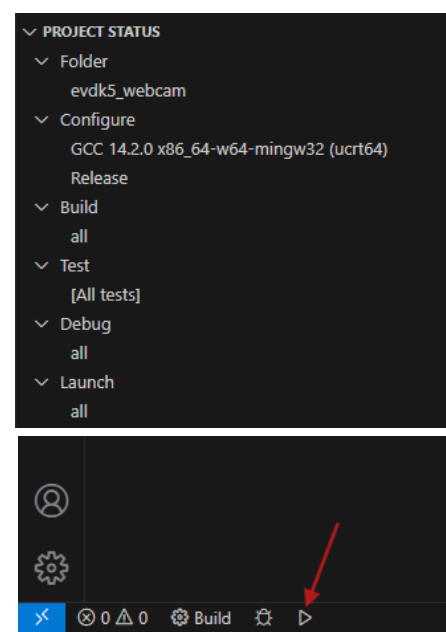
at
./evdk-2526/evdk_workspace_apps/evdk5_webcam/build/*image.bmp*

6   Unit testing

The Unity test framework is used for unit testing the image processing operators. The unit test project contains static images and several test cases per image processing operator. There is no need to connect the FRDM-MCXN947 hardware. The unit tests are executed as follows.

1. Open the project by double clicking the VS Code workspace file:
    .*evdk5\evdk_workspace_apps***evdk5_apps.code-workspace**
2. Open the CMake extension
3. In project status, select the following options:
    a. Folder: evdk5_unit_test
    b. Configure:
        GCC a.b.c x86_64-w64-mingw32 (ucrt64)
        Debug
4. Launch the selected target. This will build and run the application.

The unit test output in the terminal window will show failed tests, because these operators have not yet been implemented. The implementation is part of your assignments and these unit tests can be used to test your implementation.

TIP. The unit tests can also be used as a TODO list, because part of the EVD1 assignments is to implement the missing image processing operators.

7   Image fundamentals – convertUyvyToUint8()

Implement and test the function *convertUyvyToUint8()*.

1. Open the unit test project by double clicking the VS Code workspace file:
    .*evdk5\evdk_workspace_apps***evdk5_apps.code-workspace**
2. Open the file evdk_operators/image_fundamentals.c
3. Find the function *convertUyvyToUint8()*.
4. Remove the warning and start the implementation.
5. Make sure the unit test gives a <mark>PASS</mark> instead of a <mark>FAIL</mark>.

TIP: In the file test_image_fundamentals.c, you will find the test function test_convertUyvyToUint8(). In this function, the source image data, destination image data and expected image data can be printed for inspection! All it takes is changing the following 0 to 1:

```
#if 0 // Change 0 to 1 to enable printing

    // Print testcase info
    printf("\n-----------------------------------\n");
    printf("%s\n", name);

    // Print image data
    prettyprint(&src, "src");
    prettyprint(&exp, "exp");
    prettyprint(&dst, "dst");

#endif
```

Also test the implementation on the target hardware.

1. Open the project by double clicking the VS Code workspace file:
   *.\evdk5\evdk_workspace_targets***evdk5_targets.code-workspace**
2. Open the file main.c.
3. In main(), uncomment *exampleWebcamUint8()* and make sure all other examples are commented.
4. Build the project and upload to the FRDM-MCXN947.
5. Test the result with a camera app (Windows Camera app or OpenCV project). A grayscale image with increasing brightness in horizontal direction should be visible.

8   Histogram operations – contrast()

Implement and test the function *contrast()*.

1. The function is located in the file evdk_operators/histogram_operations.c

2. Make sure the unit test gives a PASS instead of a FAIL.

If you would like to inspect the histogram of an image, an example project is available.

1. Use a webcam or the EVDK5 with the *exampleWebcamBgr888()* running.
2. Open the project *./evdk5/evdk_workspace_apps/evdk5_histogram_webcam*
3. Run the application. Five windows will popup:
   a. The original bgr888_pixel_t camera image
   b. The same image, but converted to uint8_pixel_t
   c. The histogram for this second image
   d. The image after an operation, such as scaling
   e. The histogram for this third image
4. Open the file evdk5_historgram_webcam/main.cpp.
5. Find the following code snippet:

```
// Examples, select one!
scale(src, dst);
// brightness(src, dst, 100);
// contrast(src, dst, 2.0f);
```

6. Comment the call to the *scale()* function and uncomment the call to the *contrast()* function.
7. Test your implementation with the following contrast correction arguments:
   a. 1.0
   b. 2.0
   c. 0.5
   d. 10.0

Week 2

The goal for this week is to enhance the performance of code execution. You will learn how to improve execution time, especially for 32-bit Cortex-M33 microcontrollers. To get the best performance, you will also learn how to implement a function in assembly programming language.

1   Image fundamentals – scaleFast()

Implement and test the function *scaleFast()*. It always stretches from 0 up and until 255.

1. Open the file evdk_operators/image_funcdamentals.c
2. Find the function *scaleFast()*.
3. Remove the warning and start the implementation.
4. Combine several of the performance optimization techniques to make the operator execute **within 2.5ms**. This must be tested using MCUXpresso for VS Code by calling the function *scaleFast()* in the function *exampleTemplate()*. For example:

```
// ---------------------------------------------------------------
// Image processing pipeline
// ---------------------------------------------------------------
// Convert uyvy_pixel_t camera image to uint8_pixel_t image
convertToUint8(cam, src);

// Copy timestamp
ms1 = ms;

scaleFast(src, dst);

// Copy timestamp
ms2 = ms;

// Convert uint8_pixel_t image to bgr888_pixel_t image for USB
convertToBgr888(dst, usb);
```

5. Note (in your log book, or in comment in your code) the performance gained for each technique.

> 🏆 *The record is ~1770 us, with the FRDM-MCXN947 board connected to the computer and streaming images to a camera app.*

6. A new record only holds if the *scaleFast()* unit test gives a `PASS` instead of a `FAIL`.

2   Image fundamentals – clearUint8Image_cm33()

Implement and test the function *clearUint8Image_cm33()*. The function clears the image data by setting all pixels to the value 0. This is the same functionality as the function *clearUint8Image()*.

---

1. Open the project by double clicking the VS Code workspace file:
    *.\evdk5\evdk_workspace_targets***evdk5_targets.code-workspace**
2. Make sure the *Release* build configuration is selected.
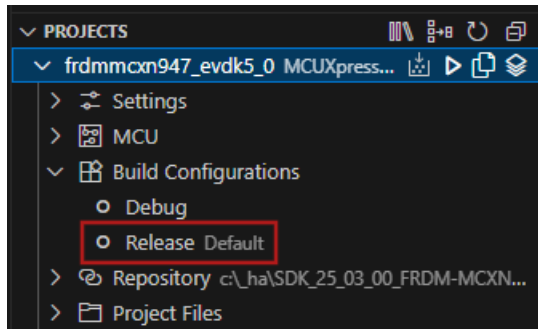
3. Use the following code in the function *exampleTemplate()* to measure how long the execution time of the function *clearUint8Image() is*.

```
// Copy timestamp
ms1 = ms;

clearUint8Image(dst); // Measured execution time ... us

// Copy timestamp
ms2 = ms;
```

4. Replace this function call as follows:

```
// Copy timestamp
ms1 = ms;

//clearUint8Image(dst); // Measured execution time ... us
clearUint8Image_cm33(dst); // Measured execution time ... us

// Copy timestamp
ms2 = ms;
```

5. Open the file *source/clearuint8image_cm33.s*
6. Implement the function.

TIP. The ARM Cortex-M33 instruction set can be found online:

https://developer.arm.com/documentation/100235/0100/The-Cortex-M33-Instruction-Set/Cortex-M33-instructions?lang=en

The pseudo code for this function is:

Push all used core registers to the stack.
Load four words from image pointer into R0 to R3.
Calculate the image size and store it in R1.
Move the image data pointer in R0.
Clear as many core registers as possible (search the instruction set for a suitable instruction).
Do
Store all these core registers to the image data pointer (R0) and make sure that the pointer is incremented after each access.
Use the SUBS instruction to subtract the number of written pixels from the image size (R1).

	Use the BNE instruction to branch to the beginning of the loop.
Pop all used registers from the stack.	
Return from the function.	

As this function is specifically written for an ARM Cortex-M33 microcontroller, there is no unit test. Verify its performance. It must be **< 100 μs**.

Important! If you would like to step through the assembly instructions, make sure the **Debug** build configuration is selected, and not Release! Furthermore, the contents of the core registers can be visualized in the debugger by selecting the *Registers* view. For example:
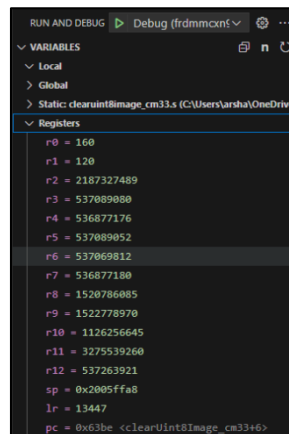
3   Image fundamentals – convertUyvyToUint8_cm33() EXTRA

This is an EXTRA assignment for those students who like a challenge. The function *convertUyvyToUint8()* is called every loop, so it makes sense to enhance its performance.

The function *convertUyvyToUint8()* takes approximately 970 μs to execute (Release build configuration selected and webcam app running on PC). The function can be improved to make it run in approximately 430 μs.

Implement and test the function *convertUyvyToUint8_cm33()*.

TIP. In the loop, consider using the LSR and BFI instructions.

4   Graphics algorithms – affineTransformation()

Examine the following images. These images are also provided in the evdk repo in the folder ./evdk5/evdk_images.
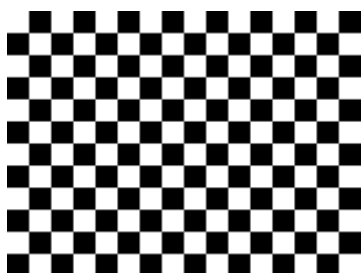
affine-src.png
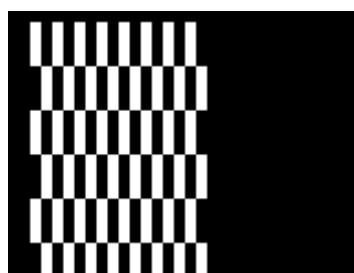
affine-dst1.png

affine-dst2.png

1. Open these images for a detailed, pixel level analysis. Use an image viewer such as the [Luna Paint – Image Editor](#) extension in VS Code.
2. Answer the following questions for both destination images:
   a. Has backward (inverse) or forward transformation been applied?
   b. Has scaling been applied? If yes, how much?
   c. Has rotation been applied? If yes, how much?
   d. Has translation been applied? If yes, how much?
   e. Has shearing been applied? If yes, how much?
3. Determine for both destination images the values for the affine transformation coefficients *a..f*.
4. Use these coefficients to produce the same images from the given *affine-src.png*. Make sure the dst image is cleared (*clearUint8Image()*) before calling the *affineTransformation()* function.

For affine_dst2.png: $\begin{vmatrix} 2 & 1 & 0 \\ 1 & 4 & 0 \\ 0 & 0 & 1 \end{vmatrix}$ and forward transformation

For affine_dst1.png: $\begin{vmatrix} 0.5 & 0 & 10 \\ 0 & 2 & 5 \\ 0 & 0 & 1 \end{vmatrix}$ and backward transformation

Week 3

The theme for this week is image enhancement by means of filtering. Most of the filters are already implemented and can be used as a reference implementation when enhancing their performance.

1 Unique operator selection

During this week each student is assigned a unique image processing operator. This is an operator that is not part of the EVDK framework. This operator is unique in that sense, that it is implemented by a single student. All other students are assigned a different operator.

The lecturer maintains a list of operators to choose from. This list is on a first-come-first serve basis. During this week, each student must have chosen an operator. Otherwise, the lecturer will assign one.

During this week, students have to come up with a function prototype and a description of the arguments and return value. This function prototype is discussed with the lecturer during class and is a go/no-go moment.

Use the following template for your function. Make sure to give the function a suitable name, change the arguments and/or the return value as needed.

```c
/*!
 * \brief Description of the function in one sentence
 *
 * A detailed description of the function that takes
 * several lines, but each line doesn't contain more than 80 characters.
 *
 * \param[in]  src A pointer to the source image, must be of type ...
 * \param[out] dst A pointer to the destination image, must be of type ...
 * \param[in]  val Value that is ...
 *
 * \return \li -1 Failure
 *         \li  0 Memory error
 *         \li  1 Success
 */
int32_t myUniqueOperator(image_t *src, image_t *dst, int32_t val)
{

}
```

Depending on the type of function, it should be located in one of the existing files.

2 Image fundamentals – convolveFast()

Implement and test the function *convolveFast()*.

1. The function is located in the file evdk_operators/image_fundamentals.c
2. Remove the warning and start the implementation.
3. Make sure the unit test gives a PASS instead of a FAIL.

4. Combine several of the performance optimization techniques to make the operator execute **within 10 ms**. Use the FRDM-MCXN947 board to measure the performance in ms precision. Here is an example of how to convert to- and from int16_pixel_t images.

```c
void exampleTemplate(void)
{
    PRINTF("%s\r\n", __func__);

    // Update SysTick to have better precision
    SysTick_Config(SystemCoreClock/100000);

    // ----------------------------------------------------------------
    // Local image memory allocation
    // ----------------------------------------------------------------
    // Create additional int16_pixel_t images
    image_t *src_int16 = newInt16Image(EVDK5_WIDTH, EVDK5_HEIGHT);
    image_t *dst_int16 = newInt16Image(EVDK5_WIDTH, EVDK5_HEIGHT);
    image_t *msk_int16 = newEmptyInt16Image(3, 3);

    // Prepare images
    clearInt16Image(src_int16);
    clearInt16Image(dst_int16);

    // Set convolution mask
    int16_pixel_t msk_data[3 * 3] =
    {
        // Identity
        0,0,0,
        0,1,0,
        0,0,0,

        // Edge enhancement
//      -1,-1,-1,
//      -1, 8,-1,
//      -1,-1,-1,

        // Sharpen
//       0,-1, 0,
//      -1, 5,-1,
//       0,-1, 0,
    };

    msk_int16->data = (uint8_t *)msk_data;

    image_t *src = newUint8Image(EVDK5_WIDTH, EVDK5_HEIGHT);
    image_t *dst = newUint8Image(EVDK5_WIDTH, EVDK5_HEIGHT);

    if(dst == NULL)
    {
        PRINTF("Could not allocate image memory\r\n");
        while(1)
        {}
    }

    while(1U)
    {
        // ----------------------------------------------------------------
        // Wait for camera image complete
        // ----------------------------------------------------------------
```

```
        while(smartdma_camera_image_complete == 0)
        {}

        smartdma_camera_image_complete = 0;

        // ----------------------------------------------------------------
        // Image processing pipeline
        // ----------------------------------------------------------------
        // Convert uyvy_pixel_t camera image to int16_pixel_t image
        convertToInt16(cam, src_int16);

        // Copy timestamp
        ms1 = ms;

//       convolve(src_int16, dst_int16, msk_int16);
        convolveFast(src_int16, dst_int16, msk_int16);

        // Copy timestamp
        ms2 = ms;

        // Scale for visualisation
        scaleInt16ToUint8(dst_int16, dst);

        // Convert uint8_pixel_t image to bgr888_pixel_t image for USB
        convertToBgr888(dst, usb);

        // ----------------------------------------------------------------
        // Set flag for USB interface that a new frame is available
        // ----------------------------------------------------------------
        image_available_for_usb = 1;

        // Print debug info
        PRINTF("%d | delta: %04d us\r\n", ms1, (ms2-ms1)*10);
    }
}
```

3   Nonlinear filters – meanFast()

Although all nonlinear filters have the same basic implementation structure, for this assignment you will improve the performance of one of them. Implement and test the function *meanFast()*. Remember that the implementation of a nonlinear filter is very similar to a convolution.

1. The function is located in the file evdk_operators/nonlinear_filters.c
2. Remove the warning and start the implementation.
3. Make sure the unit test gives a PASS instead of a FAIL.
4. Combine several of the performance optimization techniques to make the operator execute **within 10 ms**. Use the FRDM-MCXN947 board to measure the performance in ms precision.

4   Nonlinear filters – EXTRA

This is an EXTRA assignment that addresses the use of nonlinear filters. Answer the following questions.

1. Which nonlinear filter performs best on the adjacent image **filters.bmp** (see folder evdk_images)? Why?
2. Which of the nonlinear filters can be used for binary erosion? Provide an example. *Use the function threshold() to create a binary image.*
3. Which of the nonlinear filters can be used for binary dilation? Provide an example.
4. Which of the nonlinear filters can be used for binary detecting edges? Provide an example.
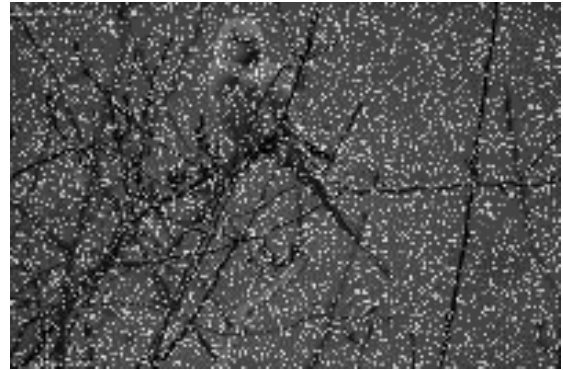
filters.bmp

1. *The median filter, because the image mainly contains salt and pepper noise.*
2. *The minimum filter can be used for binary erosion.*
3. *The maximum filter can be used for binary erosion.*
4. *The range filter can be used for binary edge detection.*

5   Spatial filters – sobelFast()

Implement and test the function *sobelFast()*. Use, amongst others, the convolveFast() function to improve performance.

1. The function is located in the file evdk_operators/spatial_filters.c
2. Remove the warning and start the implementation.
3. Make sure the unit test gives a PASS instead of a FAIL.
4. Combine several of the performance optimization techniques to make the operator execute **within 20 ms**. Use the FRDM-MCXN947 board to measure the performance in ms precision.

Week 4

This week discusses algorithms for automatic thresholding for separating objects from the background. These algorithms are based on the graylevel histogram.

1   Segmentation – threshold2Means()

Implement and test the function *threshold2Means()*.

1. The function is located in the file evdk_operators/segmentation.c
2. Remove the warning and start the implementation.
3. Make sure the unit test gives a `PASS` instead of a `FAIL`.

2   Segmentation – thresholdOtsu()

Implement and test the function *thresholdOtsu()*.

1. The function is located in the file evdk_operators/segmentation.c
2. Remove the warning and start the implementation.
3. Make sure the unit test gives a `PASS` instead of a `FAIL`.

Week 5

The goal for this week is to reduce binary objects as much as possible to their basic shapes. Morphological filters are used for that purpose.

1 Morphological filters – removeBorderBlobsTwoPass()

Implement and test the function *removeBorderBlobsTwoPass()*.

1. The function is located in the file evdk_operators/morphological_filters.c
2. Remove the warning and start the implementation.

Make sure the unit test gives a PASS instead of a FAIL.

TIPS

- The argument *lutSize* is used to dynamically allocate memory in the removeBorderBlobsTwoPass() function for the equivalence lookup table.
    - Use the function malloc() for dynamic memory allocation.
    - Check if the allocation succeeded by verifying if the returned pointer is not equal to NULL.
    - Use the function memset() to set the entire lookup table to zero.
    - Do not forget to use free() when the function finishes.
- The removeBorderBlobsTwoPass() function returns 1 on successful execution and returns 0 in case of the following failures:
    - Memory allocation for the lookup table failed
    - The lookup table is too small. In other words, the image requires more unique labels than can be stored in the lookup table.

2 Morphological filters – fillHolesTwoPass()

Implement and test the function *fillHolesTwoPass()*.

1. The function is located in the file evdk_operators/morphological_filters.c
2. Remove the warning and start the implementation.

Make sure the unit test gives a PASS instead of a FAIL.

TIPS. You should be able to reuse the majority of the code from the removeBorderBlobsTwoPass() function.

Week 6

This last week discusses techniques to extract object features. Some of these features are translation, rotation and scaling invariant. These features can be used in the final assignment for object classification.

1 Mensuration – labelTwoPass()

Implement and test the function *labelTwoPass()* according the two-pass labelling algorithm. To ensure a fast implementation, it is allowed to skip the border pixels.

1. The function is located in the file evdk_operators/mensuration.c
2. Remove the warning and start the implementation.
3. Make sure the unit test gives a PASS instead of a FAIL.
4. Use the FRDM-MCXN947 board to measure the performance in ms precision.

EXTRA. Skipping the border pixels has two major disadvantages. The first is the decrease in resolution, which especially is a problem when working with an image pyramid and very small images. The second is that an object's connectivity might be broken. This is visualized in the following images. The left image handles the border pixels correctly and shows a single 8-connected BLOB. The right image, however, skips the borders and shows two BLOBs.

			1	1			
	1	1			1	1	
	1	1			1	1	
	1	1			1	1	

	1	1			2	2	
	1	1			2	2	
	1	1			2	2	

In this EXTRA assignment, change the implementation so that border pixels are not skipped anymore. Handle these border pixels separately from the inner pixels, to prevent an out-of-bounds check for every pixel.

2 Mensuration – perimeter()

Implement and test the function *perimeter()* according the alternative d algorithm (see week 6 powerpoint presentation).

1. The function is located in the file evdk_operators/mensuration.c
2. Remove the warning and start the implementation.
3. Make sure the unit test gives a PASS instead of a FAIL.
4. Use the FRDM-MCXN947 board to measure the performance in ms precision.

3 Mensuration – circularity()

The *circularity()* function is already implemented. It uses the perimeter() function and as the perimeter() function is finished in the previous assignment, the circularity() function can be used. Call the circularity function in the FRDM-MCXN947 project and print the circularity of the

objects as used in the [assessment sheet](). From these printed values, deduce the expected minimum and maximum values for each shape (square, triangle, circle and unknown).

4   Mensuration – huMoments()

The *huMoments()* function is already implemented. Use it to print the first four hu invariant moments of the objects as used in the [assessment sheet](). From these printed values, deduce the expected minimum and maximum values for each shape (square, triangle, circle and unknown).

5   Final assignment

The final assignment is described in detail in section [Final assignment](). The solution can be realized by implementing the following steps:

- Acquisition: convert the image from uyvy_pixel_t to uint8_pixel_t. Use an image pyramid, so image processing operations are performed on images as small as possible.
- Enhancement: pre-process the image (if needed) to enhance the objects.
- Segmentation: segment objects from background. Are you going to use an operation for automatic thresholding? Or does this take too long?
- Feature extraction: calculate object features, such as circularity or Hu invariant moments.
- Classification: classify the object based on the features.

The camera produces images at 30fps. This makes for a timing budget of $\frac{1}{30} = 33\text{ms}$. Try to make the solution execute on the FRDM-MCXN947 board within this timing budget. Provide an answer to the following questions:

- How long does each operation take?
- How long does the total of all operations take?
- If the total of all operations takes longer than timing budget, what can be done to improve performance?