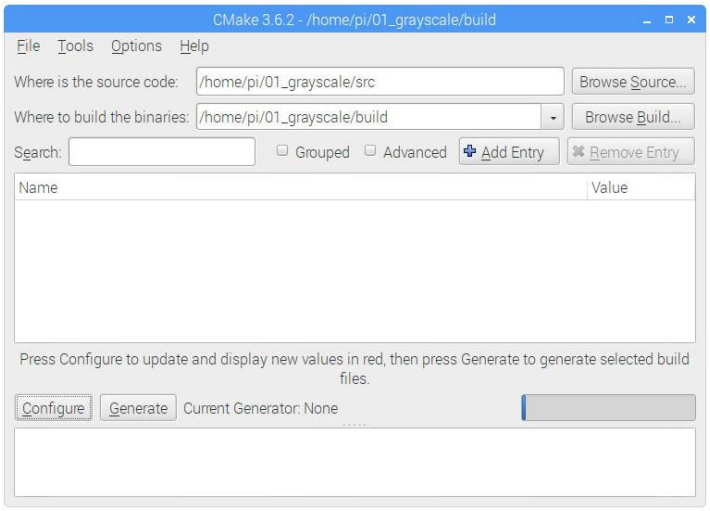
DSP2 SS2018 – Exercise 1: CMake, OpenCV and Grayscale

Part 1: Get the Files

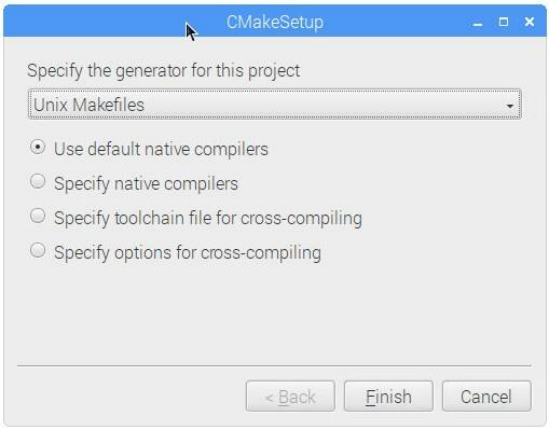
1. Download this exercise (`01_grayscale.zip`) from OPAL[1] (http://opal.sachsen.de/TUC) to your home directory (`/home/pi`) on the Raspberry Pi
2. Start a Terminal, go to your home directory (command `cd` without any parameters) and unzip it with `unzip 01_grayscale.zip`
3. Afterwards you will find a new directory `01_grayscale` with the following content:
   a. build: an empty directory for the program you are going to compile
   b. data: a folder with the image (lena.tiff) which we are going to use as input
   c. src: this directory contains the source files

Part 2: Create the project with CMake

1. Start CMake
   a) RaspPi-Menu -> Programming -> CMake
   or
   b) `cmake-gui` on the terminal
2. Set the path to source and build

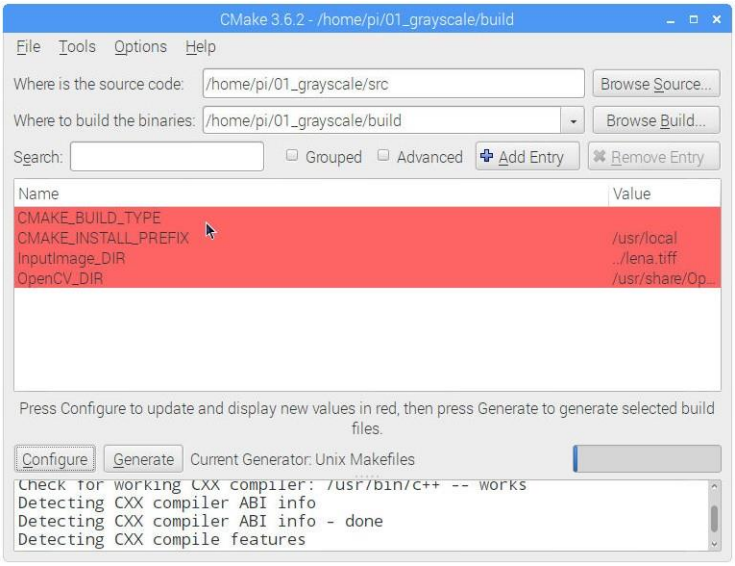


3. Configure (use default settings in the following window)

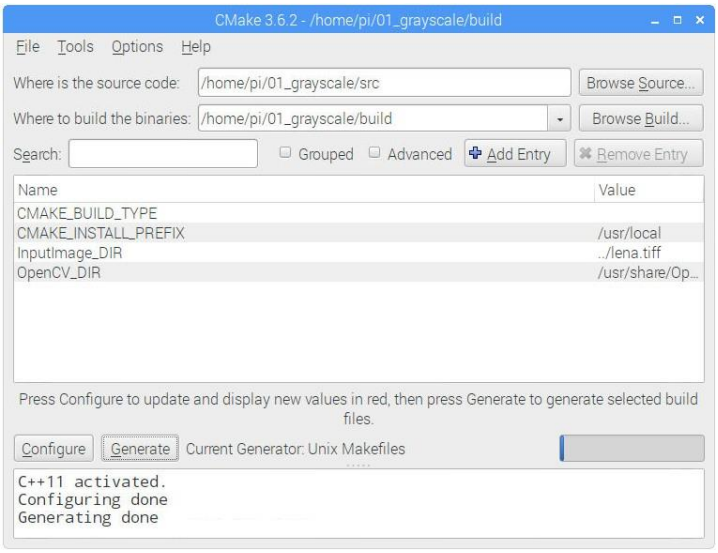


4. Configure again (the red markings will disappear)

[1] OPAL → Technische Universität → Chemnitz Fakultät für Elektrotechnik und Informationstechnik → Professur Digital- und Schaltungstechnik | Chair of Digital Signal Processing and Circuit Technology → DST_RDS



5. Generate



Note: Once the project is created, you will not have to use CMake (and these 5 steps) for this project again, even when you change the source code.

Part 3. Compile and start the program

1. Use the terminal and go to the build directory (e.g. `cd ~/01_grayscale/build`).
2. Compile the program using the command `make`
3. Start the program (in the build directory) with `./grayscale`

Note: After changing the source code you have to use `make` again.

Part 4: The source code (folder src)

Files:

1. Timer.h
   - Provides an easy way to measure the time (you do not need to modify this file at all).
   - `INIT_TIMER` (without ";") initializes the timer
   - `START_TIMER` starts the time counting
   - `STOP_TIMER(<text>)` displays the time elapsed between START_TIMER and STOP_TIMER with the additional text <text>. Use <text> for a short but meaningful description.
1. main.cpp
   - Function `main(…)`: You do not have to change anything.
   - Function `neon_convert(…)`: Converts a BGR colored image into grayscale using NEON code. NEON uses hardware acceleration of the Raspberry Pi and increases performance. NEON code is only used to demonstrate the performance of different methods. You do not have to understand or write any NEON code in this seminar. For a more detailed description of this NEON code, see http://hilbert-space.de/?p=22.
   - Function `reference_convert(…)`
     This is the place where you have to insert your code.

Description of main.cpp

- `cv::Mat img = cv::imread("../data/lena.tiff");`
  creates the matrix img and imports the image (24-bit BGR color) from file
- `img.cols` and `img.rows` : the number of columns and rows of the matrix
- `img.isContinuous()` : true, if the matrix is continuous (without gaps in memory)
- `cv::cvtColor(img, imgGray, CV_BGR2GRAY);`
  converts the BGR color image (img) to a grayscale image (imgGray) with OpenCV
- `reference_convert(…)`
  This function should convert the BGR color image to a grayscale image with standard C++ code. It is your task to write this code.
- `neon_convert(…);`
  converts the BGR color image to a grayscale image with NEON code
- `cv::imshow("description", img);` : displays the image (matrix)
- `cv::waitKey();`
  without it, the program (and the displayed images) will be terminated immediately

Note:

In OpenCV, the channel weights for RGB-to-grayscale conversion are: R*0.299, G*0.587 and B*0.114 (77/256*R, 150/256*G and 29/256*B).

Further information:

OpenCV API Reference: http://docs.opencv.org/2.4/modules/refman.html

Part 5: Exercise 1

Write the function `reference_convert (uint8_t * dest, uint8_t * src, int n)` to convert the BGR colored image `src` into a grayscale image `dest`. `dest` is a continuous matrix with `n` pixels where each pixel represents a grayscale value (0…255). `src` is a continuous matrix where each pixel has 3 consecutive values (0…255) for the BGR colors (i.e. pixel ordering is (b_0, g_0, r_0, b_1, g_1, r_1 … b_{n-1}, g_{n-1}, r_{n-1})). When you are finished, the program will show the original image, 3 identical grayscale images and the computation time for each of the grayscale conversions.

Good luck and feel free to play with the code.