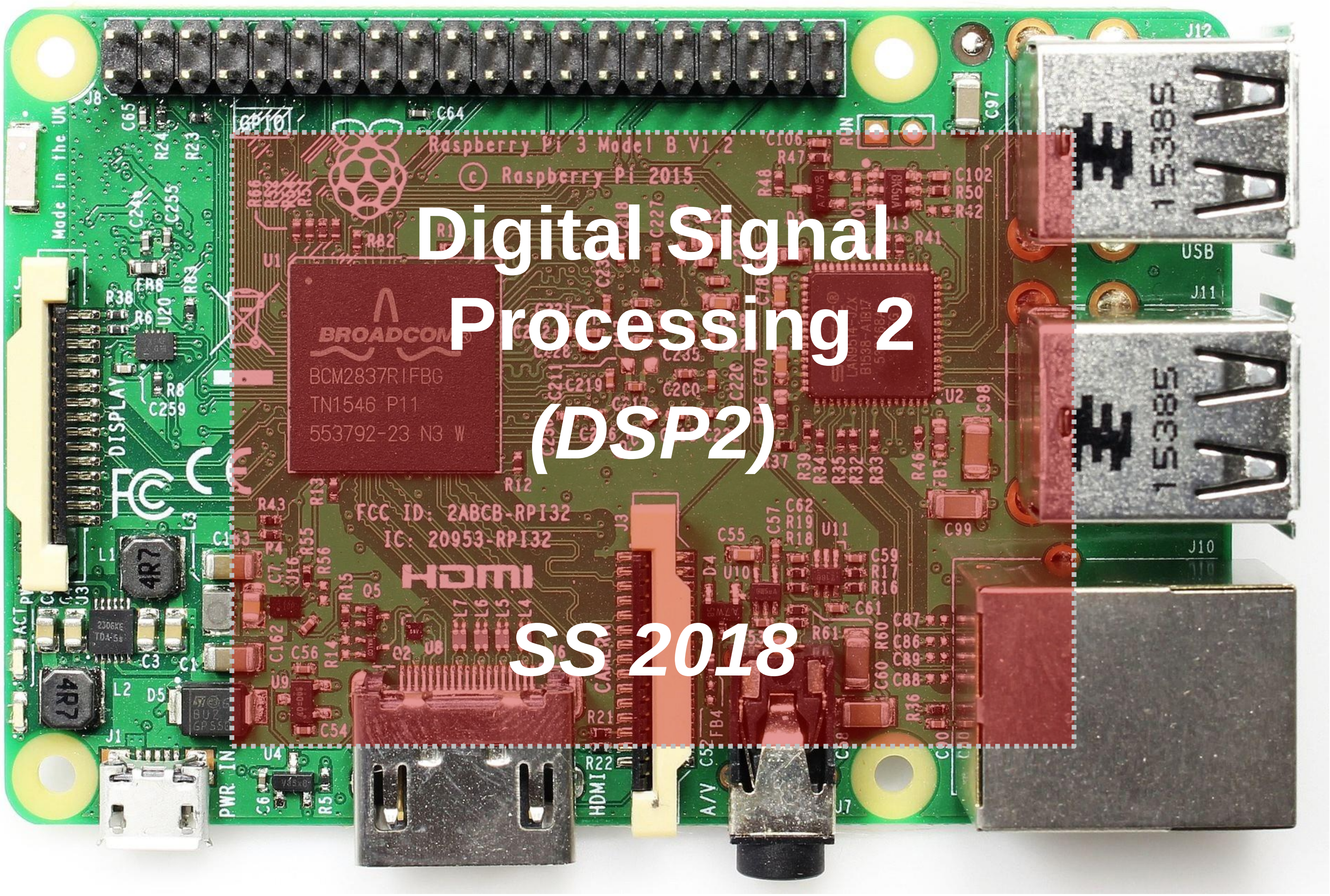


Digital Signal Processing 2

(DSP2)

SS 2018

Supervisors:

Dipl.-Inf. Norbert Nestler
Room: C25.421
Mail: norbert.nestler@etit.tu-chemnitz.de

M.Sc. Danny Heinz
Room: C25.220
Mail: danny.heinz@etit.tu-chemnitz.de

Download:

- All exercises and documents will be available on OPAL:

https://bildungsportal.sachsen.de/opal/auth/RepositoryEntry/9637920769

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

- The solution to an exercise will be given at the beginning of the next seminar

Requirements

Requirements for the course:

- Good knowledge in Digital Systems (especially image processing: image filter, convolution, point operations, segmentation, …)
- Good knowledge in C++ programming

Goal:

- Realize image processing algorithms on an Embedded System

Tools:

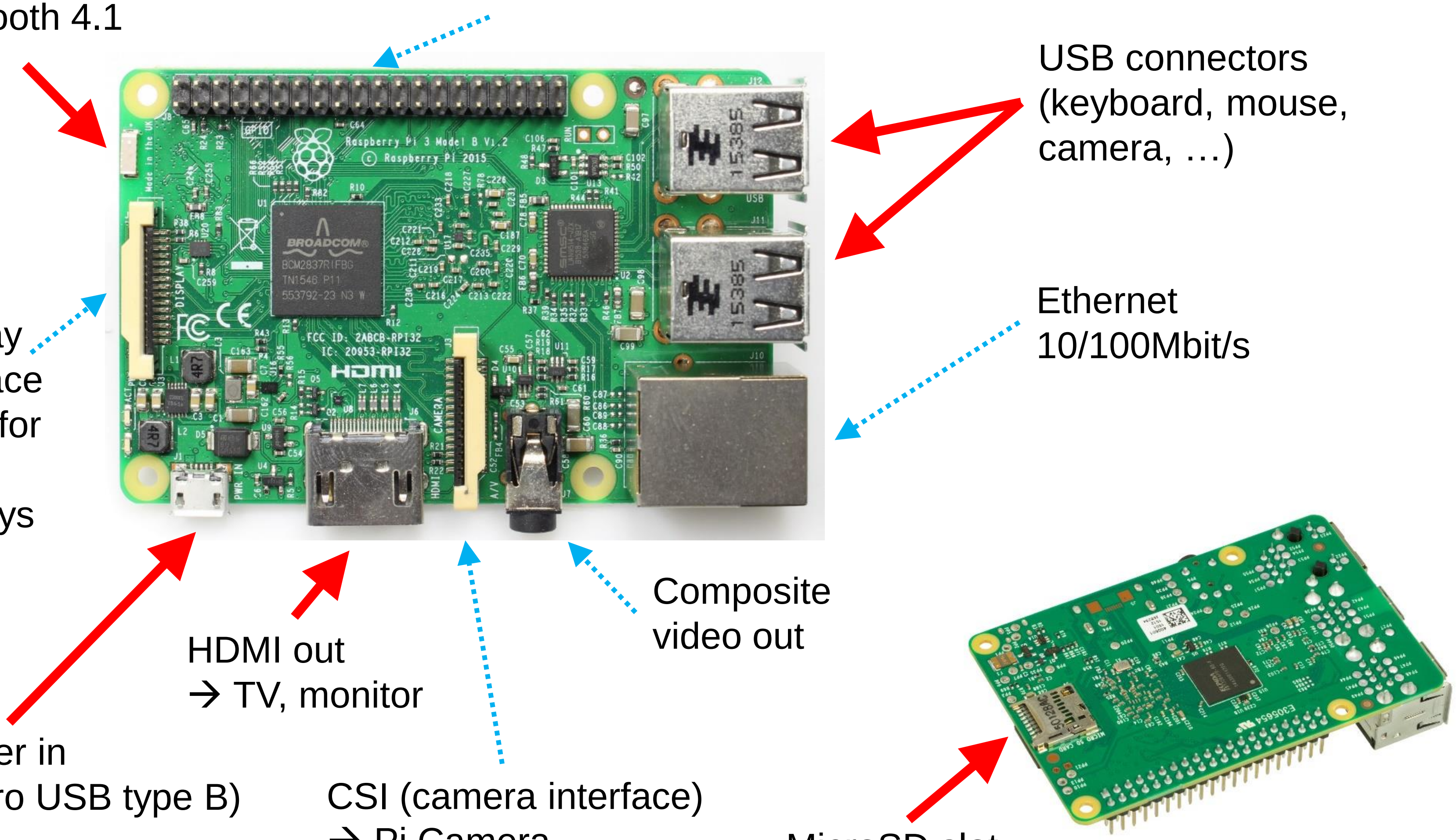
- CMake:
  - Free open-source cross-platform tool for managing the build process of software
- OpenCV:
  - Free open-source computer vision framework
  - Available on a huge variety of platforms (including our Embedded System)

Raspberry Pi

Hardware (Raspberry Pi 3 Model B):

← used in the seminar

← not used in the seminar



On Board Bluetooth 4.1 Wi-fi

GPIOs (general-purpose input/output)

USB connectors (keyboard, mouse, camera, …)

Ethernet 10/100Mbit/s

Display interface (DSI) for LCD displays

Composite video out

HDMI out → TV, monitor

Power in (Micro USB type B)

CSI (camera interface) → Pi Camera

MicroSD slot

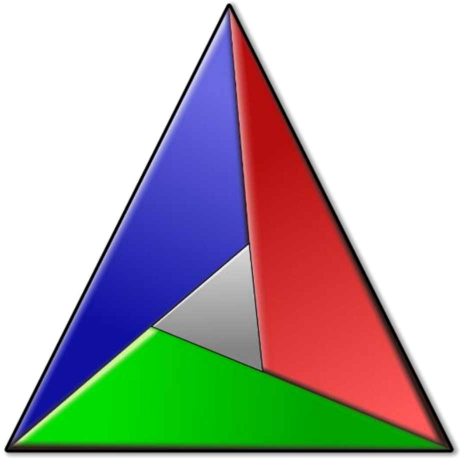
Raspberry Pi:

- A small single-board computer developed in the United Kingdom by the Raspberry Pi Foundation
- Originally designed to promote the teaching of basic computer science in schools and in developing countries
- Became very popular
- Version 3: 1.2 GHz 64-bit quad-core ARM Cortex-A53 processor, 1 GB RAM (shared with GPU)
- Different operating systems available, e.g. Raspbian (a Debian linux based OS)

- More information: https://www.raspberrypi.org (Raspberry Pi Foundation)

CMake:

- Free open-source cross-platform tool to manage the build process of software
- Automatically creates the build environment for you
- Collects all libraries for your project
- Generates the project for your IDE (e.g. Makefile, Visual Studio, Qt Creator, Xcode, KDevelop), so you can use CMake on every OS to create a valid project file
- Needs a project description file “CMakeLists.txt”



- For the complete documentation please look at:

  http://www.cmake.org/documentation/

CMake example

Example for CMakeLists.txt

```
cmake_minimum_required(VERSION 2.8.10)

# set project name
project(Introduction)

# set compile flags
set(CMAKE_CXX_FLAGS "-std=c++11")
set(CMAKE_BUILD_TYPE "Release")

# find libraries
find_package(OpenCV REQUIRED)

# set include directories
include_directories(${OpenCV_INCLUDE_DIR})

# make executable
add_executable(${PROJECT_NAME} main.cpp)

# link against libraries
target_link_libraries(${PROJECT_NAME} ${OpenCV_LIBS})
```

Note:

The CMakeLists.txt files will be given to you for all exercises. You do not need to create or change them.

OpenCV

OpenCV

- Free open-source computer vision framework
- Is the standard in computer vision

Download:

http://opencv.org/downloads.html

Documentation:

http://opencv.org/documentation.html

Reference Manual:

http://docs.opencv.org/2.4.10/modules/refman.html

OpenCV: Basic functions

Read an image from file:

- Mat **imread**(const string& **filename**, int **flags=1**)
- **filename**: Name of file to be loaded.
- **flags**: Flags specifying the color type of the loaded image:
  - CV_LOAD_IMAGE_ANYDEPTH - If set, return 16-bit/32-bit image when the input has the corresponding depth, otherwise convert it to 8-bit.
  - CV_LOAD_IMAGE_COLOR - If set, always convert image to color
  - CV_LOAD_IMAGE_GRAYSCALE - If set, always convert image to grayscale
- http://docs.opencv.org/2.4/modules/highgui/doc/reading_and_writing_images_and_video.html#imread

Show an image:

- void **imshow**(const string& **winname**, InputArray **mat**)
- **winname**: Name of the window.
- **mat**: matrix (image) to be shown.
- http://docs.opencv.org/2.4/modules/highgui/doc/user_interface.html#imshow

OpenCV: Basic functions

cv::Mat class:

- stores the data of an image

- List of important members:
  - int cols: number of columns
  - int rows: number of rows
  - uchar* data: pointer to the image data

- List of important methods:
  - void Mat::**create**(int **rows**, int **cols**, int **type**):
  - **rows**: New number of rows.
  - **cols**: New number of columns.
  - **type**: New matrix type (CV_8U = grayscale image, CV_8UC3 = 24 bit color image)
  - List of image types http://docs.opencv.org/2.4/modules/core/doc/intro.html#fixed-pixel-types-limited-use-of-templates

- template<typename T> T& Mat::at(int **i**, int **j**)
  - Access the data of a image
  - **i**: Index along the dimension 0 (rows)
  - **j**: Index along the dimension 1 (columns)

cv::Mat class methods:

- bool Mat::**isContinuous**()
  - The method returns true, if the matrix elements are stored continuously without gaps at the end of each row. Otherwise, it returns false.
  - If you extract a part of the matrix (e.g. subpart of an image), the matrix is not continuous.

- template<typename _Tp> _Tp* Mat::**ptr**(int **i0**=0)
  - Access image data
  - **i0**: A 0-based row index.

- A full reference of cv::Mat is available here:

  http://docs.opencv.org/2.4/modules/core/doc/basic_structures.html#mat

Color conversion:

- void **cvtColor**(InputArray **src**, OutputArray **dst**, int **code**, int **dstCn=0**)
  - Converts an image from one color space to another.
  - **src**: input image: 8-bit unsigned, 16-bit unsigned (CV_16UC...), or single-precision floating-point
  - **dst**: output image of the same size and depth as **src**
  - **code**: color space conversion code (CV_BGR2GRAY, CV_RGB2GRAY, CV_GRAY2BGR, CV_GRAY2RGB)
  - **dstCn**: number of channels in the destination image; if the parameter is 0, the number of the channels is derived automatically from **src** and **code**

OpenCV: Basic functions

Programming techniques: Access to the image data from a cv::Mat:

- There are 3 ways to access a pixel:
  1. template<typename T> T& Mat::**at**(int i, int j) const method
  2. Pointer with index
  3. Pointer without index
- The result is always the same, but the speed of the data access is different!

1. template<typename T> T& Mat::at(int i, int j) const

```
cv::Mat img = cv::imread(lena.tiff)

for (int r = 0; r < rows; ++r) {
    for (int c = 0; c < cols; ++c) {
        std::cout << img.at<uchar>(r, c) << std::endl;
    }
}
```

→ easy, but slow

2. Pointer with index:

```
cv::Mat img = cv::imread(lena.tiff)

// check for continuous data in memory
if (img.isContinuous()) {
    cols = rows*cols
    rows = 1;
}

for (int r = 0; r < rows; ++r) {
    // pointer to the data
    const uchar *pInput = input.ptr<uchar>(r);

    for (int c = 0; c < cols; ++c) {
        // access image element
        std::cout << pInput[c] << std::endl;
    }
}
```

} → checking if image is conti./not

→ same to '\n'

→ faster, but not the fastest solution

OpenCV: Basic functions

3. Pointer without index:

```
cv::Mat img = cv::imread(lena.tiff)

// check for continuous data in memory
if (img.isContinuous()) {
    cols = rows*cols
    rows = 1;
}

for (int r = 0; r < rows; ++r) {
    // pointer to the data
    const uchar *pInput = input.ptr<uchar>(r);

    for (int c = 0; c < cols; ++c) {
        // access image element
        std::cout << *pInput  << std::endl;

       // increment data address
       ++pInput;
    }
}
```

→ the fastest solution

Wi-Fi configuration

1. Connect to the Wi-Fi network “tu-chemnitz.de”. If this is no available try “web-psk” with the pre-shared key “web-mit-psk”.
2. Open any website in the browser but make sure it uses http and not http**s**. You will be redirected to a web page where you can login.

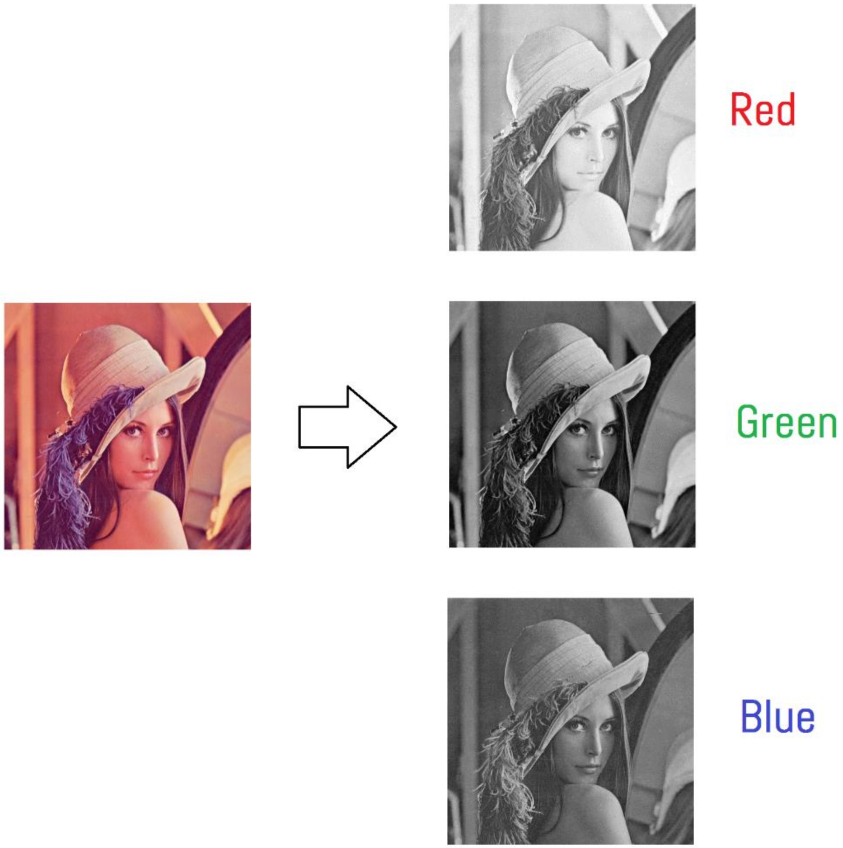
Save your work

Your code will be erased after every exercise so that the next person using “your” Raspberry Pi has a clean working environment. Therefore, we recommend you to save your work.

Exercise

First Exercise

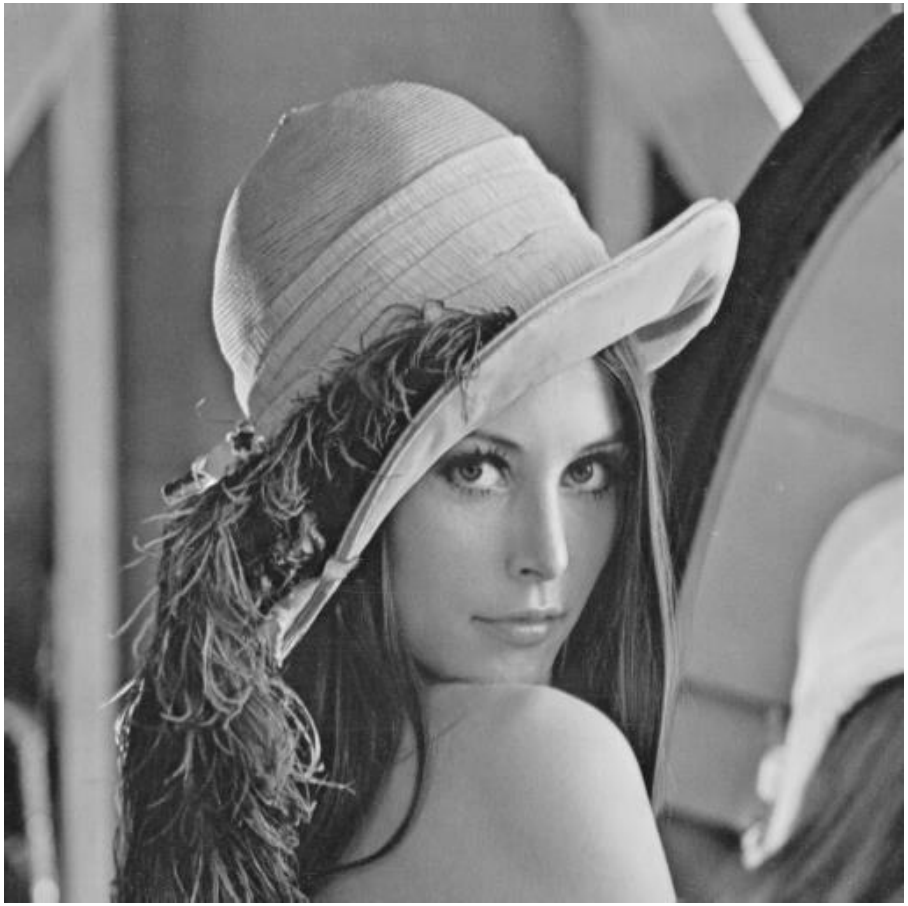
- Compute the Grayscale Image of an RGB-Image
- In OpenCV, the channel weights for RGB-to-grayscale conversion are:
  - R*0.299 (R * 77/256)
  - G*0.587 (G * 150/256)
  - B*0.114 (B * 29/256)
- Weighting is done to account for human color perception
  → most sensitive to green, then red, then blue



Expected Output

Original RGB-Image

3 Grayscale Images



Goodbye

That is all for today.