Introduction

1 General Notes

This exercise is intended to practice the material discussed in the computer vision course. The lecture follows the book by Szeliski: "Computer Vision: Algorithms and Applications", which is freely available online[1]. It is highly advised that you read the book, especially the chapters which are covered in this course. Apart from this, we also recommend the book "Multiple View Geometry in Computer Vision" by Richard Hartley and Andrew Zisserman, a sample chapter of which is available online[2]. Also, the university library offers a pdf version of this book, so you can get access to it from there.

The exercises are structured as follows:

- There are 6 exercise sheets (including this one). Exercises will not be corrected or graded. There will be questions on all exercises in the exam.
- The exercise sheets are meant to be solved during the lecture period. The lecture schedule indicates when an exercise should be solved.
- You may work on the exercises in groups.
- We will provide support in the two exercise slots (in person) and online via FAU's Matrix chat.
- You can find the Matrix chat link on StudOn.

[1] http://szeliski.org/Book/

[2] https://www.cambridge.org/core/books/multiple-view-geometry-in-computer-vision/0B6F289C78B2B23F596CAA76D3D43F7A

2 Compile and Run

The exercises are written in Python and use the open-source library OpenCV. You may work on your own computer, on any OS and with any IDE. We recommend using the provided running scripts and test-cases to make sure your solution is correct.

- Are you new to **Python**? If you are looking for a coding environment, try either Visual Studio Code and the Python extension, the PyCharm IDE or the Spyder IDE. Of course, you can also stick to the console. Choose whatever coding environment suits you best. Here is a quick tutorial for Python.
- Are you new to **NumPy**? NumPy is a famous and reliable math extension for Python and is well suited for working with images, which are multidimensional arrays of numbers. Read the Quickstart guide for a quick introduction.
- Are you new to **OpenCV**? OpenCV implements many powerful computer vision algorithms, such that you can use them as convenient tools in the labs. From the OpenCV-Python tutorials, make sure to read the core guide.

2.1 Using a CIP Pool Computer

If you work on CIP Pool computers, we recommend the editor Visual Studio Code (VS Code). Some issues may arise if you load modules in the "wrong" order (error message about missing font). Stick to the following order of loading VS Code first, then Python.

- Open a terminal and load VS Code via the Environment Module System with the command `module load <software>`, i.e.: `module load vscode`
  You can check for additional modules with the following command: `module available`
- Load the CV-specific Python environment with: `module load python3/CV`
  This module contains all Python packages that we will be needing this semester.
- Once everything is loaded, you can start your editor/IDE as usual, e.g.: `vscode`
- Open the template file(s) for this exercise in VS Code. Before you can run it, you first need to configure the correct Python interpreter in VS Code. Open the command palette (`Ctrl+P`) and type: `>Python: Select Interpreter`, then choose the option `/local/python3.11-Anaconda3-2024.03-CV/bin/python`. If, for some reason, the path to the python executable does not show up, you can manually find the correct path in a terminal with `which python`. Once you have selected the interpreter, click the arrow next to the play button on the top right and change the run setting to "Run Python File", which should also run your script.

If you want to use the IDE PyCharm instead of VS Code in the CIP Pool, you will probably find that your CIP quota is not enough to handle PyCharm's data caching into your home folder. Students have more quota in `/proj/ciptmp`, as data stored there does not get backuped. With a symlink like `/.PyCharmVersion/ -> /proj/ciptmp/$USER/.PyCharm`, using PyCharm should not be a problem.

2.2 Using your own Computer

Note: Students who want to install `anaconda` should have complete knowledge about it and only then set up conda environments in their system, otherwise we highly recommend to use system python and the steps suggested below.

Recommended version numbers:

- python: 3.6+
- OpenCV: 4.1+

2.2.1 Install Python

- Linux

```
sudo apt install python3
```

- Windows
  - Tutorial
  - Documentation

2.2.2 Setup Virtual Environment

- Linux/Mac users

```
mkdir computer_vision      # Make the directory
cd computer_vision         # navigate into it
python3 -m venv venv       # create the virtual environment
source venv/bin/activate   # activate the environment
pip install opencv-python numpy pytest  # install packages
```

- Windows users

```
cd computer_vision
python -m venv venv
.\venv\Scripts\Activate.ps1
pip install opencv-python numpy pytest
```

2.3 Directory Structure

The project directory structure will be persistent across exercises. Each exercise directory will have the following structure:

```
introduction
├─ exercise-sheet
│  └─ sheet.pdf
├─ resources
│  ├─ img.png
│  └─ ...
├─ src
│  └─ main.py
├─ utils
│  ├─ __init__.py
│  ├─ functions.py
│  └─ ...
└─ test
   └─ test.py
```

- The tasks of each exercise sheet can be found in `exercise-sheet/sheet.pdf`.
- Each exercise will consist of writing a python module. You find the launching script in `src` as `main.py`. This script is usually already complete and demonstrates how the module to be implemented is used. The actual implementation work has to be done in the files in the `utils` folder. The resources to your program, e.g., input images, are located in the `resources` folder.
- Each exercise will have its own pytest test-case found in `test`. Passing test-cases are usually a good indication of a correct solution, but always check the results of your solution to make sure.

2.4 Run Code

```
cd introduction
# Activate virtual env. (if you set up one) according to OS
source venv/bin/activate # [Linux/MacOS]
.\venv\Scripts\Activate.ps1 # [Windows]
# Execute
python3 src/main.py
# Test
python3 -m pytest test/test.py
```

3 OpenCV Image Processing

To familiarize with the most basic OpenCV functions, implement the following tasks in `utils/functions.py`. The launching script `main.py` already loads the image `img.png` from the `resources` folder using the OpenCV function `imread`.

3.1 Image Loading and Saving

Implement `show_images` to display the given images on the screen. Use the OpenCV function `imshow`. Once your program exits, all created windows will close automatically. Use the function `waitKey` to stall the program until a key has been pressed.

Implement `save_images` to save the given images to disk. Similar to loading, you can save the image by calling the function `imwrite`. Save the files to the `resources` folder with the given filenames.

3.2 Resizing

Implement `scale_down` to resize the given image by a factor of 0.5 in both directions with the OpenCV function `resize`.

3.3 Color Channels

Implement `separate_channels` to create three images from the single given image: one for each channel (red, green, blue). Make sure these images have the same size and type as the input image. Iterate over every pixel of the input image and store each channel individually in one of the 3 images. A single row of an image is accessed in the following way:

```
// set the blue channel of the fifth image column to 0
// Note: OpenCV stores images in BGR format.
img[i, 4, 0] = 0
```

Hint: An efficient solution has no loops in python/numpy.

The resulting images should look like this:

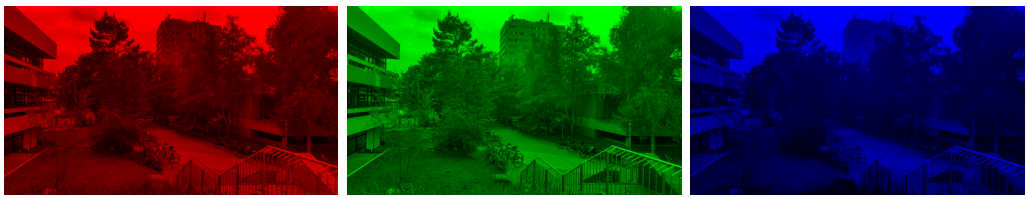

Figure 1: The red, green, and blue channel of the input image.