FH AACHEN
UNIVERSITY OF APPLIED SCIENCES

Prof. Dipl.-Inf. Ingrid Scholl

Kathrin Petters, B.Sc.

3D-Bildverarbeitung
Praktikum

Information System Engineering

WS 2017/18

Inhaltsverzeichnis

1 Working with OpenCV **2**

1.1 First steps 3

1.2 Smoothing Images 3

1.3 Camera calibration 3

1.4 Feature Detection and Matching 4

1 Working with OpenCV

First there are some tasks to learn how to calculate 3D points from 2D images.

In this part we use OpenCV 3.2 in Ubuntu. If you want to use your own computer the Documentation of OpenCV gives a installation guide where you can find all necessary information. The information about installing OpenCV in Linux can be found under `http://docs.opencv.org/3.2.0/d7/d9f/tutorial_linux_install.html`

Follow the steps to install OpenCV, but check for the right version. We use 3.2.

To compile your code you have to use cmake. Follow these steps to compile your program.

- Write your Code into the „.cpp“ file ans save it in your working directory.
- Create a „CMakeLists.txt“ file and write the following code in it and replace„yourProgram“ with the name of your file:

```
cmake_minimum_required(VERSION 2.8)
project( DisplayImage )
find_package( OpenCV REQUIRED )
include_directories( ${OpenCV_INCLUDE_DIRS} )
add_executable( yourProgram yourProgram.cpp )
target_link_libraries( yourProgram ${OpenCV_LIBS} )

```

- Open a terminal (Ctrl + Alt + T) an navigate (cd directory) to your working directory and type following commands:

```
cmake .
make

```

- Now you can run your program with typing

```
./yourProgram parameter

```

  Use the parameter only if needed for example to hand over the image.

1.1 First steps

Here is the first small exercise with OpenCV. Just choose an image you want and follow the instructions. `http://docs.opencv.org/3.2.0/db/d64/tutorial_load_save_image.html`

Be sure you understand everything.

1.2 Smoothing Images

Smoothing an image is essential in image processing. After reading and writing an image now calculate different filter on it. Tutorial: `https://docs.opencv.org/3.2.0/dc/dd3/tutorial_gausian_median_blur_bilateral_filter.html` Use different kernel sizes to see what happens.

Questions:

1. Explain the different filter.
2. How does the kernel size affect?

1.3 Camera calibration

This Task would be done in two parts. First we need to do a mono calibration and after this we calibrate stereo.

Mono:

There is an example at `http://docs.opencv.org/3.2.0/d4/d94/tutorial_camera_calibration.html`. Use this for your project.

There is a configuration file used to define usage (for example to use video or webcam). Read it and then enter the right configuration for the chessboard and image source.

Questions:

1. What do you use image and object points for?
2. What are the „rvecs“ und „tvecs“?
3. How are the flags set and what does it mean?

Stereo:

You find a good explanation at `https://docs.opencv.org/3.0-last-rst/modules/calib3d/doc/camera_calibration_and_3d_reconstruction.html` with some notes for examples.

Basically you need to expand the mono calibration. Following steps must be done:

- Calibrate both cameras with mono calibration to calculate relevant parameter.

- Save the object points, image points, camera matrix and distortion coefficients. For more information what is needed see `http://docs.opencv.org/3.2.0/d9/d0c/group__calib3d.html#ga246253dcc6de2e0376c599e7d692303a`
- Do a rectification by using $stereoRectify(...)$. And print your result by using $imshow()$.

Questions:

1. Why is mono calibration needed for stereo?
2. Which Rotation and Translation is calculated in $stereoCalibrate()$?
3. Why do you need to do $stereoRectify()$?

1.4 Feature Detection and Matching

In this part we use the SURF-Feature-Detector to find same keypoints in two images. Use the tutorial at `https://docs.opencv.org/3.2.0/dd/dd4/tutorial_detection_of_planar_objects.html`

This part is not a copy-paste tutorial, you need to write many parts you own. You can take a few things from the old program like the includes.

Questions:

1. what is the $minHessian$ Parameter?
2. How does the matcher work?
3. what calculates $findHomography$?