VU Stereo Vision – Exercise 1

1 General Task

The overall goal of the practical part of this course is to implement a local stereo matching algorithm based on the ideas of Hosni et al. [HBR+11]. The practical part is based on one preliminary exercise (i.e., exercise 1) to familiarize yourself with OpenCV and on two actual exercises (i.e., exercises 2 and 3) which are describe here in more detail. In exercise 2, you will implement a simple block-based stereo algorithm. In exercise 3, you will add an adaptive support weight approach to your stereo algorithm and you will apply additional disparity refinement steps. In particular, the following four functions have to be implemented in the exercises 2 and 3 (see Figure 1):

1) computeCostVolume \\ Exercise 2 – Task 2a
   A cost volume is a three-dimensional structure and is derived by computing the color dissimilarity for each pixel at each allowed disparity value (see Figure 1b).

2) aggregateCostVolume \\ Exercise 3 – Task 3a
   Smoothing is applied on two-dimensional (x, y) slices of the cost volume at a fixed disparity value (see Figure 1c).

3) selectDisparity \\ Exercise 2 – Task 2b
   The best disparity value for each pixel is selected from the cost volume, i.e., the disparity with the lowest cost (see Figure 1d).

4) refineDisparity \\ Exercise 3 – Task 3b
   Inconsistent pixels are detected in the disparity maps and these pixels are filled with the disparity of the closest consistent pixel (see Figure 1e).

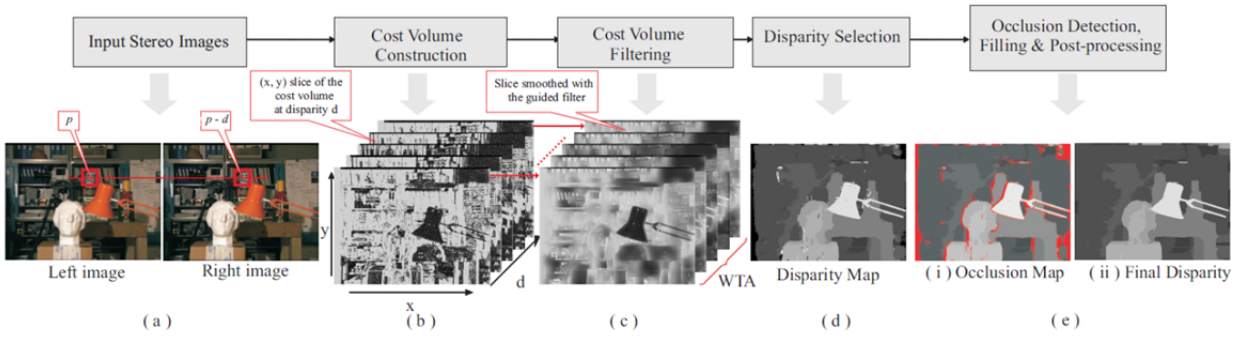


Figure 1: Outline of the algorithm to be implemented. (a) Input stereo images. (b) Cost volume construction. Each *(x, y)* slice holds the dissimilarity costs at a given disparity value *d*. (c) Cost volume filtering. Filtering is applied on each of the *(x, y)* slices. (d) Disparity selection. For each pixel, the disparity of lowest costs in the cost volume is selected. (e) Disparity refinement. Inconsistent pixels which are marked red in the occlusion map are filled with the disparity of the closest consistent pixel in the final disparity map (Figure taken from [HBR+11]).

2 General Instructions

The exercises have to be implemented in C++ using the OpenCV library. In order to set up a C++ project with OpenCV in Visual Studio, a HOWTO is provided in the instructions for exercise 1. Moreover, useful links concerning OpenCV can be found at the section “Useful Links” in this document.

Note that the exercises are built upon each other, e.g., if you do not accomplish exercise 1, you cannot accomplish neither exercise 2 nor exercise 3.

2.1 Recommended Setup

It is recommended to use the following setup:

- Development environment: Visual Studio 2010
- OpenCV: 2.4.10
- Operating system: Windows

Note that for different setups (especially regarding the operation system) either no or only limited support can be provided.

2.2 Submission

The deadline for exercise 1 is **26.04.2015**.

Your submission must include:

- The source code (i.e., the main.cpp file).
- A self-alone executable .exe file which computes and shows the left disparity map. Make sure that (i) you compile the .exe file with the parameters that give the best results and (ii) all necessary files in order to run the .exe file are enclosed.

All files are to be sent by e-mail in a single .zip attachment to “nezveda@ims.tuwien.ac.at”. The subject of the e-mail is to be “Stereo Vision VU – Group ##: Exercise 1 Submission”. One submission per group is sufficient.

The structure of the submission should look like this:

.zip

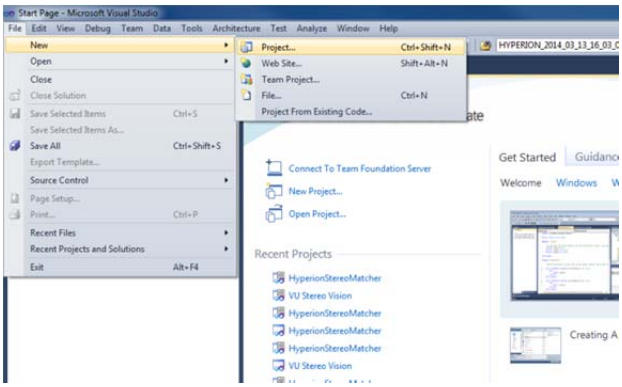
- /source
  - main.cpp
- /exe
  - .exe
  - all necessary files in order to run the .exe file

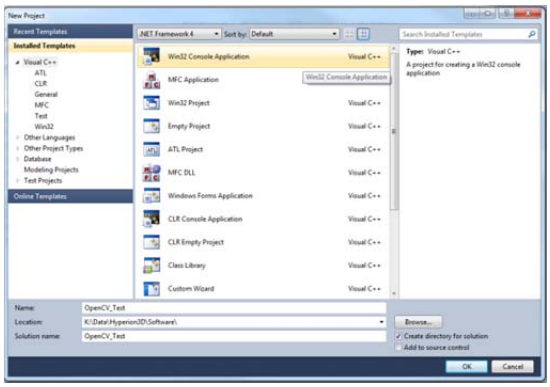
3 **Task 1a:** Set up C++ project with OpenCV library

1) Dowload OpenCV version 2.4.10 from http://opencv.org/downloads.html.

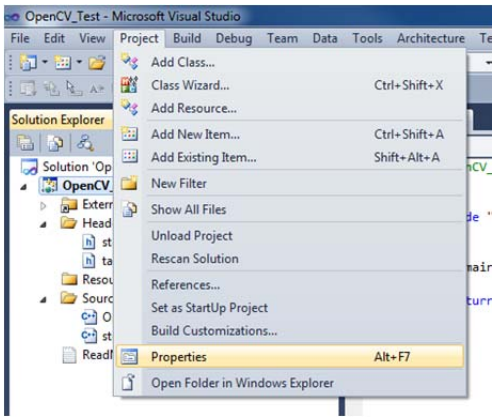
2) Extract the files, e.g., “K:\opencv-2.4.10”

3) Launch Visual Studio 2010 and select “File -> New -> Project”. Select “Win32 Console Application”, enter a name (e.g., OpenCV_Test) and select a location where to place the project. When done, press “OK”.

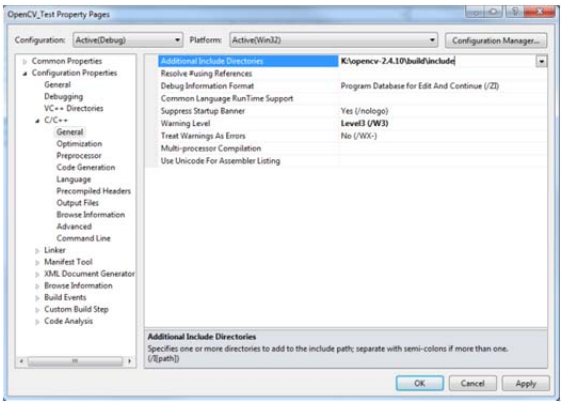




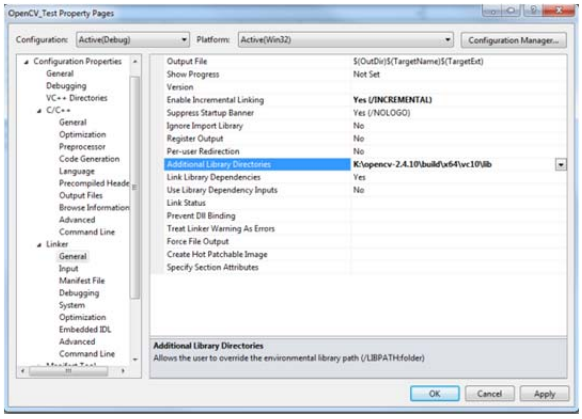
4) Click “Project -> Properties” to access the properties dialog.



5) In the left box choose “Configuration Properties -> C/C+ -> General”, edit “Additional Include Directories” and add “K:\opencv-2.4.10\build\include”.
IMPORTANT: Change the path accordingly, if you have put OpenCV files to a different place.



6) Under “Common Properties -> Linker -> General”, edit “Additional Library Directories” and add “K:\opencv-2.4.10\x86\vc10\lib”.
IMPORTANT: Change the path accordingly, if you have put OpenCV files to a different place.

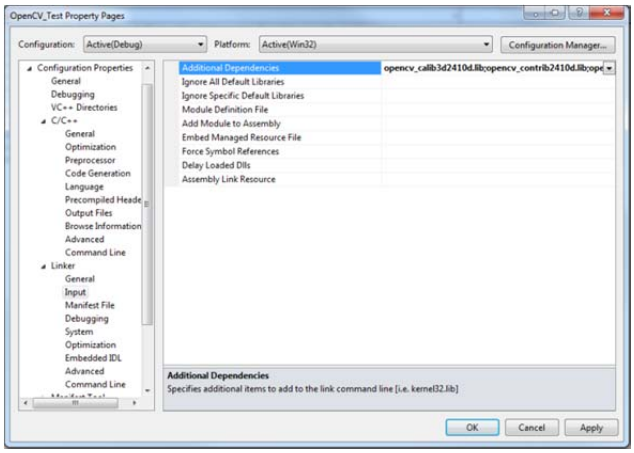


7) Under “Common Properties -> Linker -> Input”, edit “Additional Dependencies” and add the following lib files:

```
opencv_calib3d2410d.lib
opencv_contrib2410d.lib
opencv_core2410d.lib
opencv_features2d2410d.lib
opencv_flann2410d.lib
opencv_gpu2410d.lib
opencv_highgui2410d.lib
opencv_imgproc2410d.lib
opencv_legacy2410d.lib
opencv_ml2410d.lib
opencv_objdetect2410d.lib
```

```
opencv_ts2410d.lib
opencv_video2410d.lib
```

Actually, you will probably not need all the libraries. Linking the only necessary ones may make the linking operation a bit faster but for our case that speed up is not worth the effort. Note the letter "d" in the filenames just before the dot. That "d" stands for "Debug". The steps mentioned above are the same for the "Release" configuration except for the linked library filenames. You just need to get rid of the "d" letters before the dot. Debug libraries include additional instructions for debugging purposes such as variable watching, and they are not optimized. This makes debug libraries run slower. When you are dealing with lots of data or you just need more processing power you should switch to release configuration which is free of extra instructions and optimizes your code, hence runs much faster.



8) Copy the dll files from "K:\opencv-2.4.10\build\x86\vc10\bin" to the project folder where the .exe file is located.

9) Modify "main.cpp" so it looks as follows and run the code. If everything is ok, then a green image should appear.

```
#include "stdafx.h"
#include <stdio.h>
#include <opencv2\opencv.hpp>

int main()
{
    cv::Mat img(500, 500, CV_8UC3, cv::Scalar(0, 255, 0));
    cv::imshow("test", img);
    cv::waitKey(0);

    return 0;
}
```

4 **Task 1b:** Convert color image to grayscale

Implement a function that takes as input a color image “img” and outputs the color image as grayscale image “imgGray”.

Use the following formula in order to obtain a luminance value *L* from the red color channel *R*, the green color channel *G* and the blue color channel *B*:

$$L = 0.21R + 0.72G + 0.07B.$$

Your function should look like this:

```
void convertToGrayscale(const cv::Mat &img, cv::Mat &imgGray)
```

Hint 1: A stereo image pair (i.e., tsukuba_left.png and tsukuba_right.png) is provided on TUWEL. Use one of these two images as input.

Hint 2: You are not allowed to use any in-built OpenCV color conversion functions. Instead, you have to implement the functionality of such functions on your own.

Hint 3: Note that the default color format in OpenCV is often referred to as RGB but it is actually BGR (the bytes are reversed). So the first byte in a standard (24-bit) color image will be an 8-bit blue component, the second byte will be green, and the third byte will be red.

5 Useful Links

Note that all these links refer to OpenCV 2.4.10:

- http://docs.opencv.org/2.4.10/modules/refman.html
- http://docs.opencv.org/2.4.10/doc/user_guide/user_guide.html
- http://docs.opencv.org/2.4.10/doc/tutorials/tutorials.html

References

[HBR+11] A. Hosni, M. Bleyer, C. Rhemann, M. Gelautz and C. Rother: Real-time local stereo matching using guided image filtering. In *IEEE International Conference on Multimedia and Expo*, pp. 1-6, 2011