ECE 510 EmbV I
Fall 2016
Lab04 notes

Feature matching

Command Line Arguments

Table 1. Project command line arguments.

-trn	image_file	The training image.
-tst	image_file	The test image.
-orb		Choose the ORB detector.
-surf		Choose the SURF detector.
-bf		Choose the brute-force matcher.
-flann		Choose the Flann based matcher.
-m	K	Perform a median filter KxK before matching.
-g	K S	Perform a Gaussian filter KxK with sigma S before matching.
-s	K S	Sharpen using a Gaussian filter KxK with sigma S before matching.
-n	N	Number of matches to keep.
-noOut		Disable showing the images (faster processing in batch mode).

Application design

Based on the requirement to perform multiple experiments with various detectors and matchers I decided to implement the application to perform just one experiment at a time based on the provided command line arguments. The user can specify the detector type, the matcher type, a training image and a test image. For additional experiments the user can specify a filter to perform before the key points detection, feature extraction and feature matching.

For this project I created the class called Lab04 which:

- Is initialized with data from the command line arguments.
- Performs the feature extraction using the ExtractFeatures() method.
- Performs the feature matching using the Match() method.
- Performs one experiment using the Experiment() method.

The output of each experiment consists of three files which are dumped in 'c:\OpenCV\Solution\images\lab04\experiments':

1. The training image displaying circles around the identified key point features with the size of the circle indicating the scale of the feature
2. The test image displaying circles around the identified key point features with the size of the circle indicating the scale of the feature
3. The training and test images side by side displaying the matched key points.

Each file name is unique based on the command line arguments:

1. The training image file name:
   Template: test_file_name.detector.matcher.filter.keyPoints.trn.jpg
   Example: bear_test1.ORB.BF.Gaussian.3x3.3.25.keyPoints.trn.jpg

2. The test image file name:
   Template: test_file_name.detector.matcher.filter.keyPoints.tst.jpg
   Example: bear_test1.ORB.BF.Gaussian.3x3.3.25.keyPoints.tst.jpg
3. The matched side by side images file name:
   Template: test_file_name.detector.matcher.filter.matched.jpg
   Example: bear_test1.ORB.BF.Gaussian.3x3.3.25.matched.jpg

After each execution the application will display short statistics on the results of the processing. I collected the statistics for multiple experiments in 'c:\OpenCV\Solution\images\lab04\results.161108.2129.txt'. The results file shows the following columns:

- Nr. key points found on the training image
- Nr. key points found on the testing image
- Nr. of matches between the training and the test image
- The detector type used
- The matcher type used
- The filter type used in additional experiments (filter_name.sizeXsize.sigma)

Logging the results is only partially useful for a few reasons:

1. The ORB detector always reports the same number of key points - 500
2. It is difficult to evaluate the ground truth because there's no automate it – the counting must be done by hand.
3. In the end we're keeping 25 matches at most.

Experiments and statistics

The application is designed to run just one configuration at a time. In order to facilitate the execution of all 16 configurations I created the batch file 'lab04.bat' which must be placed in the same folder as the executable. The batch file can be found in 'c:\OpenCV\Solution\images\lab04\lab04.bat' and should be placed in 'c:\OpenCV\Solution_outdir\Debug_x86\lab04.bat'. In the batch file I am preventing the application from displaying the images using the '-noOut' command line argument in order to reduce processing time.

The first observation is that, in the case of the 'bear' images, the SURF detector seems to do a much better job at matching features than the ORB detector regardless of the matcher used. In fact there seems to be little difference between the brute-force and the Flann-based matchers, especially in the case of SURF. (Figure 1)

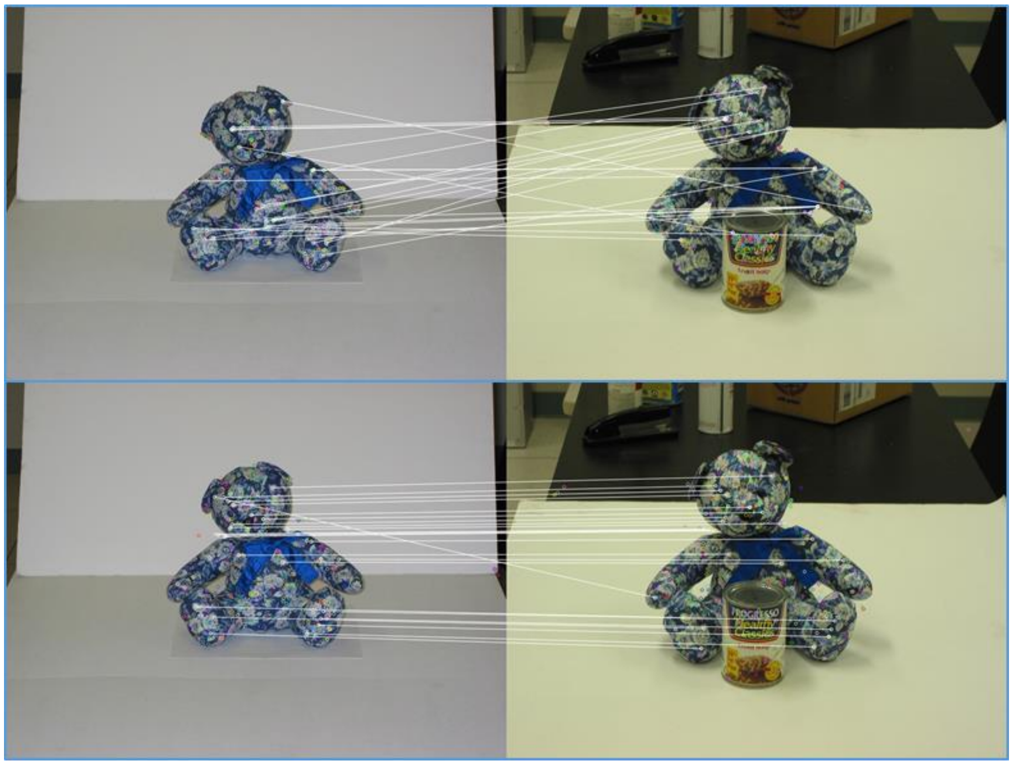


Figure 1. ORB-BF (top) vs. SURF-BF (bottom).
SURF always provides a better match between the 'bear' images.

The situation is reversed in the case of the 'shoe' images. Somehow the ORB detector is able to provide a better matching than SURF regardless of scale. (Figure 3)

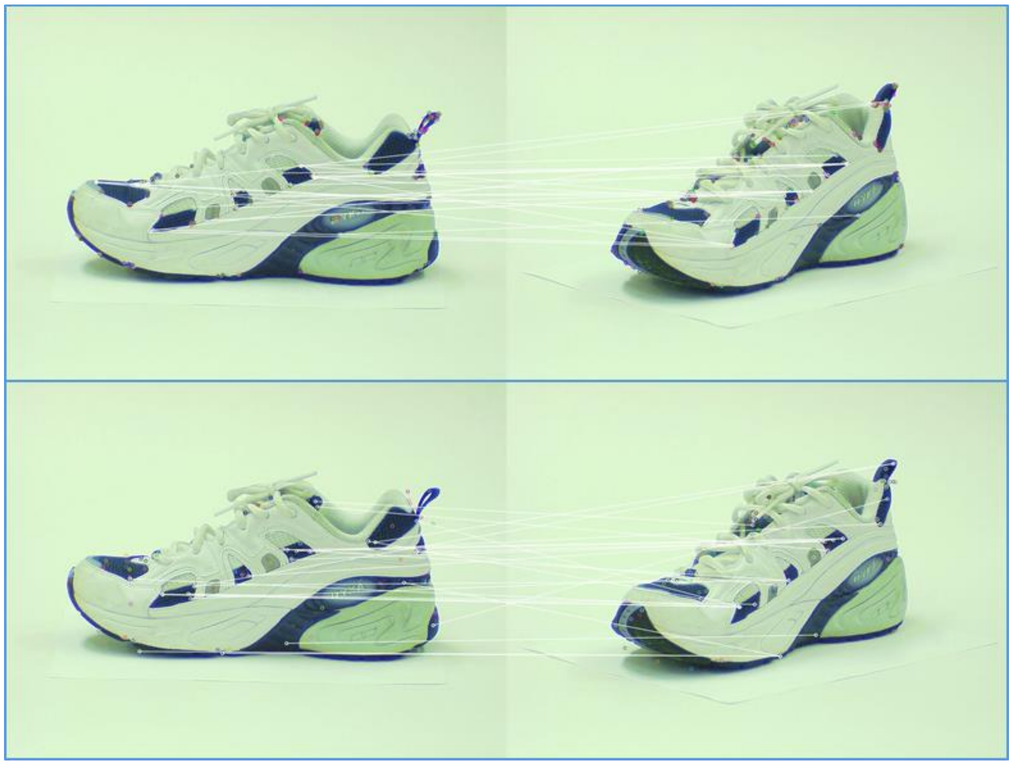


Figure 2. ORB-BF (top) vs. SURF-BF (bottom).
ORB always provides a better match between the 'shoe' images.

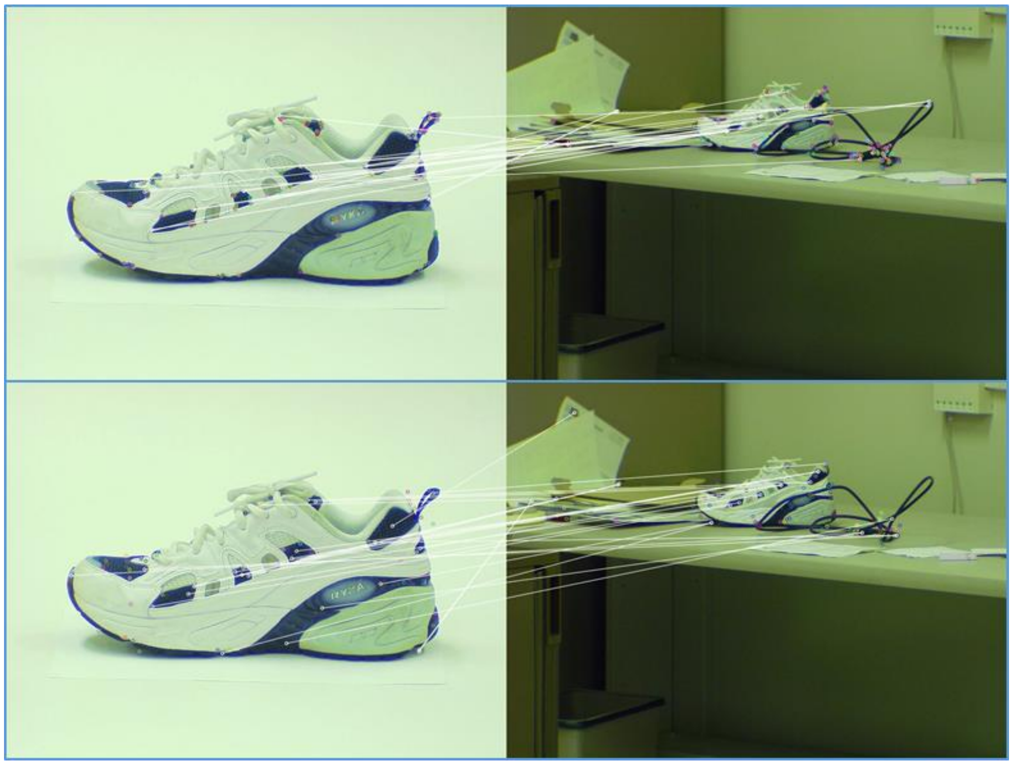


Figure 3. ORB-BF (top) vs. SURF-BF (bottom).
ORB always provides a better match between the 'shoe' images regardless of scale.

Other experiments

I decided to extend the list of experiments by introducing an additional filtering step before key points detection and feature matching. In order to facilitate the execution of all the configurations I created the batch file 'lab04.extended.bat' which must be placed in the same folder as the executable. The batch file can be found in 'c:\OpenCV\Solution\images\lab04\lab04.extended.bat' and should be placed in 'c:\OpenCV\Solution_outdir\Debug_x86\lab04.extended.bat'.

Median & Gaussian Blur

The idea here was that by blurring I might be able to reduce some of the noise in the images. Even though this blurring might cause smaller features to fade out it might allow larger more distinct features to be selected and thus improve the true positive rate.

Sharpen

In order to sharpen the images I subtracted a Gaussian blurred image from the original. I did this using the addWeighted OpenCV function:

addWeighted(Mat src1, double alpha, Mat src2, double beta, double gamma, Mat dst)

This method allows combining two images according to the formula below:

dst = alpha * src1 + beta * src2 + gamma

I used:

- alpha = 1.5
- beta = -0.5
- gamma = 0

in order to preserve the original brightness as much as possible while removing some of the blur around the images.

My ideas was that by sharpening maybe I'll be able to enable the detection of more distinct key points/features and thus increase the true positive rate.

Conclusion

Figure 4 shows the SURF detector with a brute-force matcher in four different cases: without any filtering, with a median filter, with a Gaussian filter and in the end with sharpening before the key point detection, feature extraction and feature matching. The results in this case show that any of the filters improve over the no filter case with sharpening performing slightly better than the Gaussian which in turn performed better than the Median. In several other cases captured by my extended experiments a lot of the false positives were eliminated by the filtering step. However that is not always the case and there are a few situations where the filters led to more false positives.

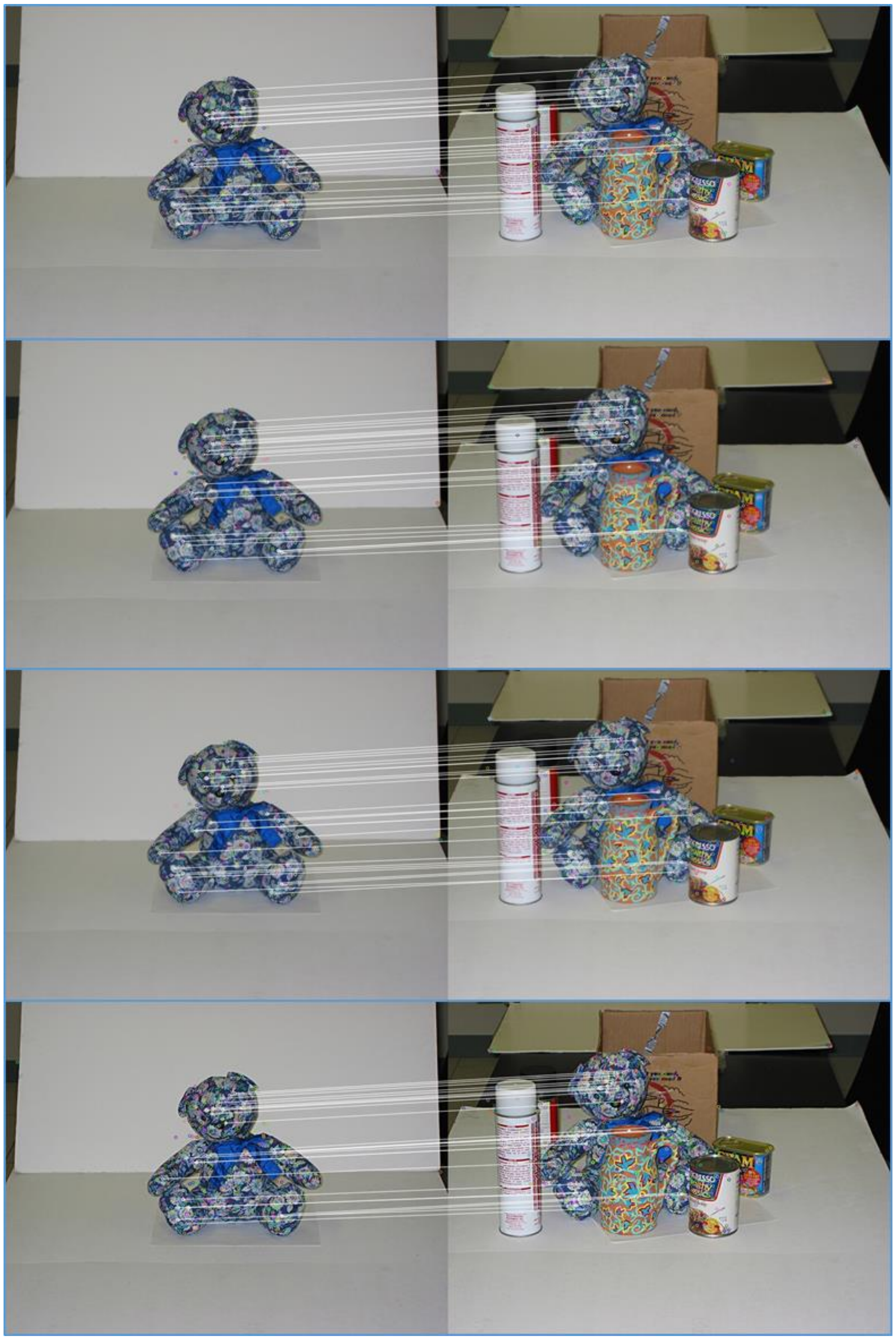


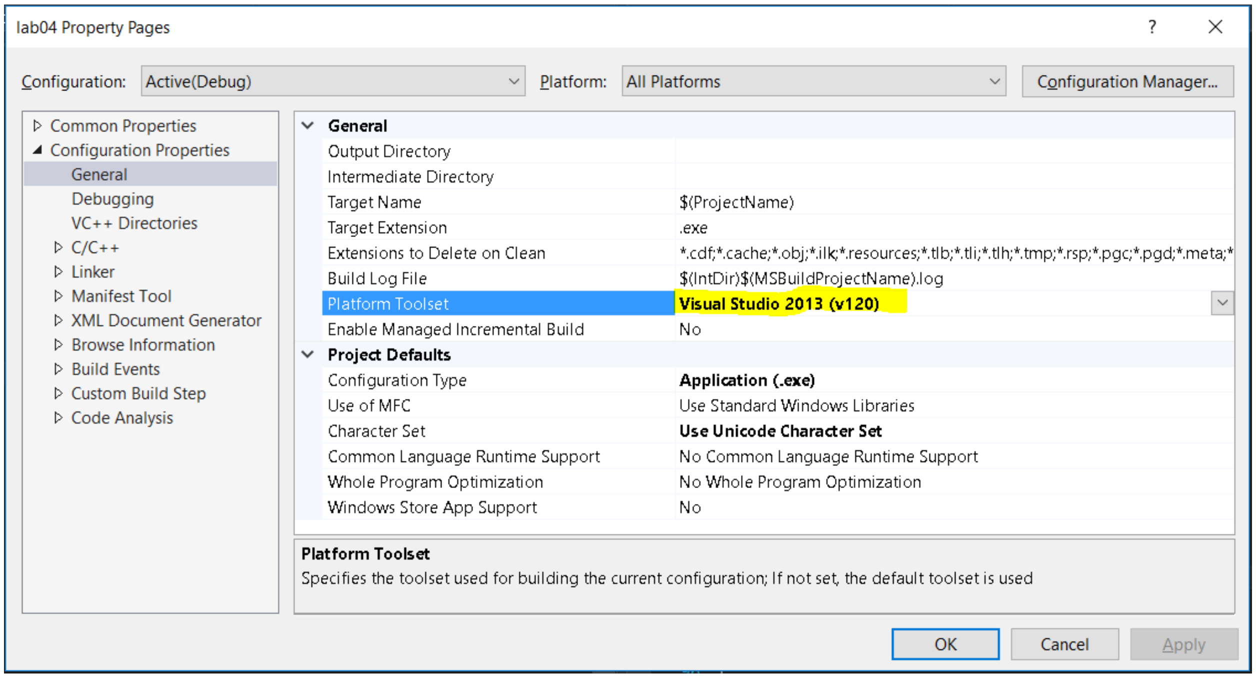
Figure 4.SURF-BF. From top to bottom:
No filter, Median, Gaussian, Sharpen.

Fix application crashing

In one of the previous labs I mentioned the application was crashing on exit. This was due to a conflict between the OpenCV libraries (built for VS2013) and VS2015 I used to build the project with.

I decided to move the whole solution to VS2013. I did this in two steps:

1. Open the solution in VS2013.
   Even though the solution was created with VS2015 I had no problems loading it in VS2013.
2. Update all projects to use VS2013 tools.
   This is done in
   - project Properties
     - Configuration Properties
       - General
         - Platform Toolset: set to **Visual Studio 2013 (v120)** (Figure 5)



lab04 Property Pages

Configuration: Active(Debug) Platform: All Platforms Configuration Manager...

Common Properties
Configuration Properties
General
Debugging
VC++ Directories
C/C++
Linker
Manifest Tool
XML Document Generator
Browse Information
Build Events
Custom Build Step
Code Analysis

General	
Output Directory	
Intermediate Directory	
Target Name	$(ProjectName)
Target Extension	.exe
Extensions to Delete on Clean	*.cdf;*.cache;*.obj;*.ilk;*.resources;*.tlb;*.tli;*.tlh;*.tmp;*.rsp;*.pgc;*.pgd;*.meta;*
Build Log File	$(IntDir)$(MSBuildProjectName).log
Platform Toolset	**Visual Studio 2013 (v120)**
Enable Managed Incremental Build	No
Project Defaults	
Configuration Type	**Application (.exe)**
Use of MFC	Use Standard Windows Libraries
Character Set	**Use Unicode Character Set**
Common Language Runtime Support	No Common Language Runtime Support
Whole Program Optimization	No Whole Program Optimization
Windows Store App Support	No

Platform Toolset
Specifies the toolset used for building the current configuration; If not set, the default toolset is used

Figure 5. Set Platform Properties to Visual Studio 2013 (v120).