Feature Detectors and Applications

I. Objective
The objective of this project is to introduce students to feature detectors and descriptors and their computer vision applications. Feature detectors are also referred to as interest point detectors or salient point detectors. After completing this assignment the student should become familiar with:

- Well-known feature detectors and descriptors
- How to apply feature detectors and descriptors in various computer vision applications

Furthermore, students should be able to investigate more state-of-art feature detectors and descriptors by themselves.

II. Introduction of the Feature Descriptors in OpenCV
Features can refer to specific locations in images, such as mountain peaks, building corners, or other object characteristics depending on the considered application. These kinds of localized features are often called key-point features or interest points (or even corners) and are often described by the appearance of patches of pixels surrounding the point location. So finding these image features is called feature detection. Once a key point is detected, the region surrounding the detected keypoint is used to describe this keypoint’s local characteristics by forming a keypoint descriptor, also called feature descriptor. This descriptor is typically formed by extracting features describing the local characteristics of the neighborhood of the detected keypoint. Descriptors that are invariant to illumination, rotation, translation and/or scaling are typically desirable. Once you have the features (keypoints) and their corresponding descriptors, you can find the same features in other images and match them, stitch them, track them, to name a few applications. In this project, OpenCV will be used to implement feature detectors and descriptors and applications. Some popular feature detectors and descriptors are described briefly below.

- **SIFT (Scale-Invariant Feature Transform)**
  
  The state-of-the-art SIFT feature detector and descriptor was introduced by D. Lowe in his seminal paper: D. Lowe, “Distinctive Image Features from Scale-Invariant Keypoints,” *International Journal of Computer Vision*, vol. 60, issue 2, pages 91-110, Nov. 2004. SIFT extracts keypoints and compute their descriptors. *Note: Please download the paper and read it.*
  
  So now let us examine the SIFT implementation in OpenCV. Let’s start by detecting keypoints and drawing them. First we have to construct a SIFT object. We can set different parameters corresponding to this object. Consider the C++ code below.
#include "opencv2/features2d/features2d.hpp"
#include "opencv2/xfeatures2d.hpp"
#include "opencv2/xfeatures2d/nonfree.hpp"
using namespace cv;
using namespace std;
using namespace cv::xfeatures2d;

void main()
{
    img = imread("lena.jpg");
    Mat gray;
    cvtColor(img, gray, COLOR_BGR2GRAY);
    Ptr<Feature2D> f2d = xfeatures2d::SIFT::create();
    std::vector<KeyPoint> keypoints;
    f2d->detect(gray, keypoints);
    Mat descriptors;
    f2d->compute(gray, keypoints, descriptors);
    Mat img_keypoints;
    drawKeypoints(gray, keypoints, img_keypoints, Scalar::all(-1),
                 cv::DrawMatchesFlags::DRAW_RICH_KEYPOINTS);
}

In the code above, the \texttt{f2d->detect()} function finds the keypoint in the considered image. You can pass a mask if you want to search only a part of an image. Each keypoint is a special structure which has many attributes like its (x,y) coordinates, size of the meaningful neighborhood, angle which specifies its orientation, response that specifies strength of keypoints etc.

OpenCV also provides a \texttt{drawKeyPoints()} function which draws the small circles on the keypoints. If you pass a flag, \texttt{cv::DrawMatchesFlags::DRAW_RICH_KEYPOINTS} to it, it will draw a circle with the size of the keypoint’s neighborhood and it will even show its orientation. See below example.

\begin{verbatim}
drawKeypoints(gray, keypoints, img_keypoints, Scalar::all(-1),
             cv::DrawMatchesFlags::DRAW_RICH_KEYPOINTS);
\end{verbatim}

Now to calculate the descriptor, OpenCV provides two methods. The first method is as follows:
Since you already found keypoints, you can call compute() which computes the descriptors from the keypoints. Example: \texttt{f2d->compute(gray, keypoints, descriptors)};

i. If you didn’t find keypoints, you can directly find keypoints and descriptors in a single step with the function, \texttt{detectAndCompute()} as shown in the sample code below.

\begin{verbatim}
Ptr<Feature2D> f2d = xfeatures2d::SIFT::create();
\end{verbatim}
After computing the keypoints and descriptors, students should be able to detect and characterize, by means of a descriptor, keypoints for images in OpenCV.

- **SURF (Speed-Up Robust Features)**

  Since the SIFT algorithm is comparatively slow, a faster keypoint detector and descriptor (SURF) was presented by Bay et al. in their paper; H. Bay, A. Ess, T. Tuytelaars, and L. Van Gool, “SURF: Speeded Up Robust Features,” *Elsevier Computer Vision and Image Understanding*, vol. 110, no. 3, pp. 346-359, June 2008. **Note: Please download the paper and read it.**

  OpenCV provides an implementation of the SURF feature detector. You can initialize a SURF object with selected parameters such 64/128-dimension descriptors, Upright/Normal SURF etc. Then as we did in SIFT, we can use `SURF.detect()`, `SURF.compute()` (and other functions similar to SIFT) for finding keypoints and descriptors.

- **BRISK (Binary Robust Invariant Scalable Keypoint)**

  In 2011, the BRISK feature detector was presented by Leutenegger et al. in their paper: S. Leutenegger, M. Chli and R. Y. Siegwart, “BRISK: Binary Robust Invariant Scalable Keypoint,” *IEEE International Conference on Computer Vision (ICCV)*, pp. 2548-2555, Nov. 2011. **Note: Please download the paper and read it.** BRISK utilizes a novel scale-space FAST-based detector in combination with the assembly of a bit-string descriptor from intensity comparisons retrieved by dedicated sampling of each keypoint neighborhood.

  OpenCV provides an implementation of BRISK. The functions provided are similar to the ones discussed in the SIFT part. Then as we did in SIFT, we can use `BRISK.detect()`, `BRISK.compute()` etc (and other functions similar to the ones discussed in the SIFT part) for finding keypoints and descriptors. The discussion below shows the parameters of BRISK function in OpenCV.
**BRISK**\( (\text{int \ thresh}=30, \ \text{int \ octaves}=3, \ \text{float \ patternScale}=1.0f) \)

<table>
<thead>
<tr>
<th>Parameters:</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. <strong>thresh</strong> – FAST/AGAST detection threshold score.</td>
</tr>
<tr>
<td>2. <strong>octaves</strong> – detection octaves. Use 0 to do single scale.</td>
</tr>
<tr>
<td>3. <strong>patternScale</strong> – apply this scale to the pattern used for sampling the neighborhood of a keypoint.</td>
</tr>
</tbody>
</table>

Note 1: Useful links to use SIFT, SURF, BRISK descriptors in C++


Note 2: Instructions to add the “xfeatures2d” OpenCV library for SIFT and SURF

In OpenCV 3.1.0, the “xfeatures2d” library is not included in the standard OpenCV library (the one downloaded from the OpenCV website). The source code of SIFT and SURF is included in the xfeatures2d library (opencv_contrib_modules), so we need to add the library to the pre-compiled OpenCV library. Please refer to Appendix 1 for the details of re-compiling the OpenCV library.
III. Computer Vision Applications

- Stereo correspondence algorithms

Stereo matching is one of the most active research areas in computer vision. Over the years, a large number of algorithms for stereo correspondence have been developed. The Middlebury Stereo Vision Dataset, which is available at http://vision.middlebury.edu/stereo/, provides a taxonomy of existing stereo algorithms that allows the dissection and comparison of individual algorithm components design decision, and also a testbed for the quantitative evaluation of stereo algorithms. Please review and read the papers related to the top 5 algorithms in the Middlebury Stereo Vision Dataset. Then students should have a basic understanding about the two-frame stereo correspondence algorithms and also the algorithm evaluation method.

The task for this course project is to find the disparity of keypoints between two-frames. The specific requirements are as follows:

a) The keypoints should be generated by using the aforementioned feature detectors and descriptors in OpenCV.

b) Select a good matching algorithm to match the keypoints of two-frames.

c) Calculate the disparity between the obtained matched keypoints of two-frames.

d) Use the Middlebury Stereo Dataset for evaluation since it provides the ground-truth disparity map.

e) Evaluate the errors in terms of root mean square error (RMSE) and the percentage of incorrectly matched pixels. For more details about the error and accuracy, please read the paper: D. Scharstein and R. Szeliski, “A Taxonomy and Evaluation of Dense Two-Frame Stereo Correspondence Algorithms,” International Journal of Computer Vision, 2002.

f) Compare results obtained by using different feature descriptors.

- Motion Estimation

Motion estimation is the process of determining motion vectors that describe the transformation from one 2D image to another; usually from adjacent frames in a video sequence. Usually, optical flow is used for this application. Optical flow is the pattern of apparent motion of image objects between two consecutive frames caused by the movement of object or camera. It is a 2D vector field where each vector is a displacement vector showing the movement of points from the first reference frame to a second frame. The Lucas-Kanade (LK) and the Kanade-Lucas-Tomasi (KLT) algorithms are popular optical flow computation methods.

The task for this course project is to track objects in a video by using the Lucas-Kanade or KLT optical flow method. The specific requirements are as follows:
a) Read papers related to the Lucas-Kanade and Kanade-Lucas-Tomasi optical flow methods:


b) Use the optical flow method to track keypoints in a video. First, use the aforementioned feature detectors to detect keypoints in the first frame. Then iteratively track those points in subsequent frames using the Lucas-Kanade or KLT optical flow method. A pyramid implementation of the KLT algorithm is available in OpenCV (The function is cvCalcOpticalFlowPyrLK()). Plot the resulting optical flow as shown in the image below:

![Optical Flow Image](image)

Note: Useful links for this application

http://docs.opencv.org/modules/video/doc/motion_analysis_and_object_tracking.html
c) Use the feature descriptors to track keypoints in a video. Use the aforementioned feature detectors to detect keypoints in the video frames and use their descriptors to match these and to generate the optical flow. Highlight matched keypoints for each frame and use different color for each set of matched keypoints. Plot the resulting optical flow as shown in the image above and compare with the optical flow method in (b) in terms of tracking accuracy and timing.

Note: Useful links for this application

http://docs.opencv.org/modules/video/doc/motion_analysis_and_object_tracking.html
- **Image Stitching**

In this project, you will implement a system to combine a series of photographs into a panorama (see panorama below). Your software will detect discriminating features in the images, find the best matching features in the other images, automatically align the photographs (determine their overlap and relative positions) and then blend the resulting photos into a single seamless panorama. You will use the aforementioned feature detectors to detect discriminating features in the images, and associated feature descriptors to find the best matching features in the other images. The main focus here is to automatically align the photographs (determine their overlap and relative positions) and then blend the resulting photos into a single seamless panorama as shown in the figure below.

Input images:

![Input images](http://www.ces.clemson.edu/~stb/klt/)

Resulting stitched panorama:

![Resulting stitched panorama](http://robots.stanford.edu/cs223b04/algo_tracking.pdf)
The task for this course project is to combine a series of photographs into a 360º panorama. The specific requirements are as follows:

a) Compute the alignment of the images in pairs.

b) Select a good stitching algorithm to stitch and crop the resulting aligned images – OpenCV has built in tools for stitching – refer to online OpenCV sample tutorial on stitching and Stitcher class.

c) Save the obtained panorama in JPEG format.

Some groups will be also tasked to compare the results of using feature detectors with those obtained using template matching. A useful reference for stitching using template matching and image blending is the paper by Burt & Adelson, “A Multiresolution Spline With Application to Image Mosaics,” ACM Transactions on Graphics, Vol. 2. No. 4, October 1983, Pages 217-236.

Note: Useful link for this application:

http://docs.opencv.org/modules/stitching/doc/stitching.html

- Object Recognition

One of the most exciting and difficult open problems of machine vision is enabling a machine to recognize objects and object categories in images. Over the years, a lot of object recognition algorithms have been proposed; and most of these state-of-art object recognition methods utilize the aforementioned feature descriptors. For this course, students should refer to the Caltech 101 computer vision database and its documentation at http://www.vision.caltech.edu/Image_Datasets/Caltech101/.

Students need also to search and read object recognition papers using feature detectors and template matching including but not limited to: S. Lazebnik, C. Schmid, and J. Ponce, “Beyond Bags of Features: Spatial Pyramid Matching for Recognizing Natural Scene Categories”, IEEE CVPR, 2006. Then students should have a basic understanding about object recognition.

The task for this course project is to implement the above paper “Beyond Bags of Features: Spatial Pyramid Matching for Recognizing Natural Scene Categories” by using the aforementioned feature detectors or descriptors. The specific requirements are as follows:
a) Read the paper.

b) Implement the paper in OpenCV.

   Note: Useful link for this application:
   http://docs.opencv.org/3.0-beta/doc/py_tutorials/py_feature2d/py_feature_homography/py_feature_homography.html

c) Use another aforementioned feature detector to run the code.

d) Evaluate and compare the accuracy of the object recognition system with different feature descriptors.

IV. Requirements

For this project, students in the class are divided into groups. The tasks for each group are different from other groups in terms of the feature detectors/descriptors and computer vision application. The Table below shows the specific tasks for each group.

V. Submission Instructions

Submit by the due date via Blackboard under Projects a zipped folder named LastNames_Project2.zip, where LastNames are the last names of the members in your group. The submitted zipped folder should contain: 1) Code folder containing Readme file on how to compile and run the code and citing any existing code or part of code used, source code with comments, executable code, input data on which to run code, and corresponding output data that is generated by the code; 2) Report: Report should include title, authors’ names, introduction, description of adopted approaches, results and discussion, conclusion, and references; report should be formatted as single-column, double-spaced, 12-point Times New Roman font.

The following guidelines are suggested in preparing the final report:

   (a) Introduction: clearly identify the problem of keypoint detection and keypoint descriptors, the importance of it, and the areas of application, and present the report organization. (0.75 to 1 page).

   (b) Description of Adopted Feature Detectors and Descriptors: describe the feature detectors/descriptors assigned to your group and compare these. Each of the chosen feature detectors/descriptors should be described briefly and compared by stating what are the main contributions of this detector(descriptor as compared to other ones, what are the main ideas proposed and/or tools used (without details since the reader can refer to the referenced original paper for details), its advantages and disadvantages as compared to other detector/descriptor. The description should be concise (3 to 5 pages).
(c) Results: Show and discuss the results. The requirements are specified in each computer vision application as described previously in this document (2 to 5 pages).

(d) Conclusion: discuss unsolved problems (things that still need to be solved/improved), impediments to further progress, and future directions for possible improvements (0.75 to 1 page).

<table>
<thead>
<tr>
<th>Group</th>
<th>Feature Detector/Descriptors</th>
<th>Computer Vision Applications</th>
</tr>
</thead>
<tbody>
<tr>
<td>Group 1</td>
<td>SIFT &amp; SURF</td>
<td>Stereo correspondence algorithms</td>
</tr>
<tr>
<td>Group 2</td>
<td>SURF &amp; BRISK</td>
<td>Stereo correspondence algorithms</td>
</tr>
<tr>
<td>Group 3</td>
<td>SIFT &amp; BRISK</td>
<td>Stereo correspondence algorithms</td>
</tr>
<tr>
<td>Group 4</td>
<td>SIFT &amp; SURF</td>
<td>Motion Estimation</td>
</tr>
<tr>
<td>Group 5</td>
<td>SURF &amp; BRISK</td>
<td>Motion Estimation</td>
</tr>
<tr>
<td>Group 6</td>
<td>SIFT &amp; BRISK</td>
<td>Motion Estimation</td>
</tr>
<tr>
<td>Group 7</td>
<td>SIFT &amp; SURF</td>
<td>Image Stitching</td>
</tr>
<tr>
<td>Group 8</td>
<td>SURF &amp; BRISK</td>
<td>Image Stitching</td>
</tr>
<tr>
<td>Group 9</td>
<td>SIFT &amp; BRISK</td>
<td>Image Stitching</td>
</tr>
<tr>
<td>Group 10</td>
<td>SIFT &amp; SURF</td>
<td>Object Recognition</td>
</tr>
<tr>
<td>Group 11</td>
<td>SIFT &amp; BRISK</td>
<td>Object Recognition</td>
</tr>
<tr>
<td>Group 12</td>
<td>SURF &amp; BRISK</td>
<td>Object Recognition</td>
</tr>
<tr>
<td>Group 13</td>
<td>SIFT &amp; Template Matching</td>
<td>Image Stitching</td>
</tr>
</tbody>
</table>
Appendix 1. Instructions for Re-compiling the OpenCV library with additional libraries (xfeatures2d)

For the purpose of this project, we need to add the xfeatures2d library module to the existing OpenCV library. There are two approaches as follows:

1. Use the provided re-compiled library files directly.

   If you are using C++ under Windows 7.0 (or up) 64-bit system and you are using Visual Studio 2013 (compiler version vc12), use the updated OpenCV library files provided on Blackboard under Project2 in the zipped folder: OpenCV_UpdateFiles_Win64_vc12.zip.

   Unzip the folder, and copy the included files to the OpenCV path as described below.
   Suppose the OpenCV path on your computer is “C:\opencv”.
   • Copy the files in “OpenCV_UpdateFiles_Win64_vc12\build\include\opencv2”, and paste the files to “C:\opencv\build\include\opencv2”.
   • Copy the files in “OpenCV_UpdateFiles_Win64_vc12\build\x64\vc12\lib”, and paste the files to “C:\opencv\build\x64\vc12\lib”. Replace existing files with the same file names.
   • Copy the files in “OpenCV_UpdateFiles_Win64_vc12\build\x64\vc12\bin”, and paste the files to “C:\opencv\build\x64\vc12\bin”. Replace existing files with the same file names.
   • Copy the folders in “OpenCV_UpdateFiles_Win64_vc12\source\modules”, and paste the folders to “C:\opencv\source\modules”.

   Now you should be able to run the SIFT and SURF in OpenCV.

2. Re-compile the OpenCV library using CMake.

   If the first approach does not apply to your system or programming language, you need to re-compile the libraries and add the opencv-contrib-modules as follows. The following illustration uses the CMake GUI for Windows System.
   • Download the opencv & opencv_contrib repositories https://github.com/Itseez/opencv_contrib
   • Unzip the file, go to “opencv_contrib-master\modules\”, copy the folder “contrib_world” and “xfeatures2d”, and paste the folders to “C:\opencv\sources\modules”.
   • Download and Install CMake from https://cmake.org/download/
   • Open Cmake-GUI, and choose source as “C:\opencv\source”. Create a new folder named “new_build” in “C:\opencv\”. Choose the build path as “C:\opencv\new_build”. This will store the newly built OpenCV files.
   • In Cmake-GUI, press the button “Configure”. A new window will pop up to select the compiler. Select the proper compiler, such as “Visual Studio 12 2013 Win64”.
Click on “Finish”, and wait for the analysis to be done. A list of build options are shown as follows. Check the options: “BUILD_opencv_contrib_world”, “BUILD_opencv_world” and “BUILD_opencv_xfeatures2d”.

Press the button “Generate”, and wait for the process to finish as shown in the message window below:
In Visual Studio, open the solution “OpenCV.sln” at “C:\opencv\new_build\OpenCV.sln”. Wait for the project to load.

In “Debug” configuration and “x64” platform, build the project “ALL_BUILD” and then the project “INSTALL”. Both build processes should finish with no errors.

In “Release” configuration and “x64” platform, build the project “ALL_BUILD” and then the project “INSTALL”. Both build processes should finish with no errors. This will take a while.

Now we have re-compiled the OpenCV library by adding the “xfeatures2d” library. Next we will copy the updated files to the OpenCV folder:

- In “C:\opencv\new_build\lib\Debug”, copy the files “opencv_ts310d.lib” and “opencv_world310d.lib”, and paste the files to “C:\opencv\build\x64\vc12\lib”.
- In “C:\opencv\new_build\lib\Release”, copy the files “opencv_ts310.lib” and “opencv_world310.lib”, and paste the files to “C:\opencv\build\x64\vc12\lib”.

In Visual Studio, open the solution “OpenCV.sln” at “C:\opencv\new_build\OpenCV.sln”. Wait for the project to load.
In “C:\opencv\new_build\bin\Debug”, copy the files “opencv_contrib_world310d.dll” and “opencv_world310d.dll”, and paste the files to “C:\opencv\build\x64\vc12\bin”.

In “C:\opencv\new_build\bin\Release”, copy the files “opencv_contrib_world310.dll” and “opencv_world310.dll”, and paste the files to “C:\opencv\build\x64\vc12\bin”.

Download and unzip the updated OpenCV library files provided on Blackboard under Project 2 in the zipped folder: OpenCV_UpdateFiles_Win64_vc12.zip.

Copy the files in “OpenCV_UpdateFiles_Win64_vc12\build\include\opencv2”, and paste these files to “C:\opencv\build\include\opencv2”.

Now the xfeature2d library and re-compiled OpenCV library have been updated in the OpenCV installation path. You can create a new C++ solution in Visual Studio to work on the project. Set the property settings to link the OpenCV library paths (same procedure as in the OpenCV installation guide).

In your project solution property setting, under Linker -> Input -> Additional Dependencies, add the “opencv_world310d.lib” and “opencv_ts310d.lib” for Debug mode. Add the “opencv_world310.lib” and “opencv_ts310.lib” for Release mode. As shown in the figure below:

Note: Useful Links for re-compiling the OpenCV libraries in Windows

http://audhootchavancv.blogspot.in/2015/08/how-to-install-opencv-30-and.html

https://nishantnath.com/2015/10/19/open-cv-3-x-adding-contrib-support-to-default-installation/

https://putuyuwono.wordpress.com/2015/04/23/building-and-installing-opencv-3-0-on-windows-7-64-bit/