ELEC 474 – Machine Vision
Lab #4 – Feature Matching-based Object Detection and Pose Estimation
Revised Oct. 2019

Feature Matching-based Object Detection and Pose Estimation

1.0 :: Introduction

The purpose of this lab is to introduce you to the concepts of object detection and pose estimation for a given pair of images. The lab has four main parts:

1. Feature Extraction: Extract ORB features from a scene S and a model image M to create a list of Scene features $\{S_i\}_1^n$ and Model features $\{M_j\}_1^m$.
2. Feature Matching: Create a set of possible feature correspondences between S and M as $C = \{S_i, M_j\}$.
3. Transformation Estimation: Using C, estimate the similarity transformation mapping M onto S using a RANSAC framework.
4. Visualization: The results can be visualized by overlaying M on S with the recovered transformation.

2.0 :: Pre-Lab [1 Mark]

***Required Files*:** box.png and box_in_scene.png

In the lecture, we have learned about SIFT (Scale-Invariant Feature Transform) features and have shown its robustness against scale, rotation, and translation variations. Another feature descriptor will be used for this lab i.e., the Oriented Fast, Rotated BRIEF (ORB) feature extractor. ORB is a fusion of the FAST keypoint detector and the BRIEF descriptor with some modifications. BRIEF, being a binary descriptor, is very fast to match since it only requires XOR operations for comparison, whereas SIFT and SURF descriptors require Euclidean distance-based matching.

To detect and extract ORB features from an image using OpenCV, we first create the ORB feature detector and its descriptor. To retain a good number of features and maintain a certain degree of robustness, you should change the preset default parameter values according to the properties of the image you are testing your algorithm on.

```
// Feature Detector
    Ptr<FeatureDetector> detector = ORB::create(int nfeatures=500, float
scaleFactor=1.2f, int nlevels=8, int edgeThreshold=31, int firstLevel=0, int
WTA_K=2, int scoreType=ORB::HARRIS_SCORE, int patchSize=31);
```

```
// Feature Descriptor
    Ptr<DescriptorExtractor> descriptor = ORB::create(int nfeatures=500, float
scaleFactor=1.2f, int nlevels=8, int edgeThreshold=31, int firstLevel=0, int
WTA_K=2, int scoreType=ORB::HARRIS_SCORE, int patchSize=31);
```

Details for such settings can be found at https://docs.opencv.org/3.4/db/d95/classcv_1_1ORB.html

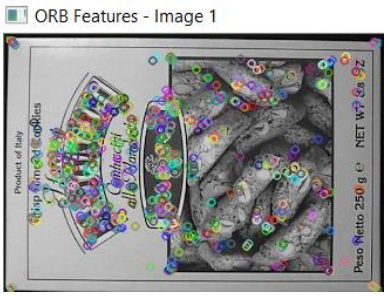

Figure 1: Image showing the detected features from image 1

Figure 2: Image showing the detected features from image 2

Feature Matching

After computing the keypoints their corresponding descriptors from the two images, we can match features based on the binary descriptors using a brute-force comparison:

```
Ptr<DescriptorMatcher> matcher = DescriptorMatcher::create ( "BruteForce-Hamming" );
// Create a vector of matches
vector<DMatch> matches;
// Match the descriptors
matcher->match (descriptors_1, descriptors_2, matches);
```

You can then visualize the matches using the `cv::drawMatches` OpenCV function. The output of such step should look something like the following:

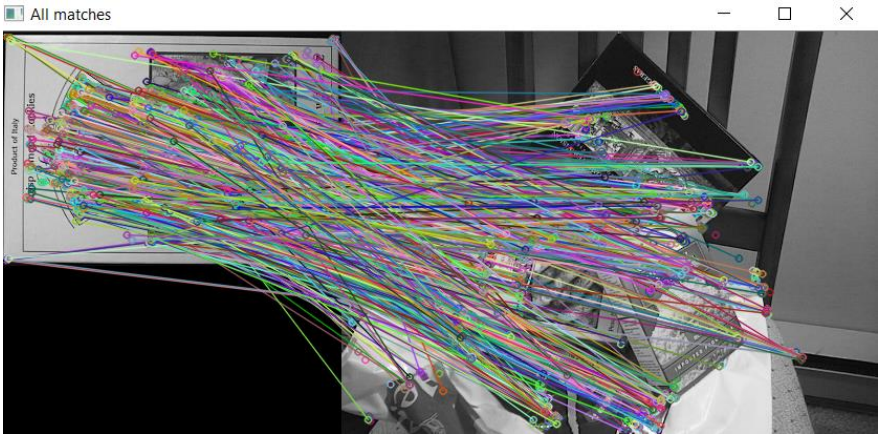

Figure 3: Image showing the correspondence lines in color

3.0 :: Lab [6 Marks]

***Required Files*:**

Original Image (Model)	Target Image (Scene)
box.png	box_in_scene.png
stapleRemover.jpg	clutteredDesk.jpg
66_6_PS_Dataset_Mod.png	66_5_PS_Dataset.png

Transformation Estimation

Using the set of matches, compute the similarity transform using the provided code snippet in "getSimilarityMatrix.cpp".

Recap: Let us assume that the point correspondences are (x',y') <-> (x,y). The similarity transform that maps (x',y') onto (x,y) is:

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \overbrace{\begin{bmatrix} s\cos\theta & -s\sin\theta & t_x \\ s\sin\theta & s\cos\theta & t_y \\ 0 & 0 & 1 \end{bmatrix}}^{\text{Similarity Transformation}} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}, \qquad where\ s = scaling$$

The minimum number of correspondences required to compute such a matrix using SVD is 2.

Optimal Transformation Estimation:

The easiest way to achieve a good transformation estimation is by making use of a RANSAC framework. The following pseudo-code describes a possible way to estimate the "best" 2D Projective Transformation from a set of 2D point correspondences.

Algorithm: Suggested RANSAC framework for estimating the best 2D Projective Transformation

```
k – the number of iterations required
m – the minimum number of required points to compute the needed transformation
t – the distance threshold used to identify a correspondence as an inlier
p – the probability of inliers with respect to the whole set
p_th – the minimum probability of inliers to state that a transformation is good.
Until k iterations have occurred
    Randomly select m matches from the list of matches
        Compute the transformation using SVD
            Transform all the points using the obtained transformation
                Compute the distances between the points in the target image
                and the projected image points.
                    If distance < t (pixels)
                       Then there is a good match (inlier ++)
                    end
                Compute p
                if p>p_th then it is a good transformation
                else continue
                end
end
The best transformation is the one with the highest p.
```

A good way to test your algorithm is to compare your estimated transformation with the one that you can get directly using either the `cv::findHomography` or the `cv::estimateRigidTransform` function. The latter estimates the optimal affine transformation between two 2D point sets, while the former estimates the full 3D Homography. Both have the option to make use of the RANSAC-based transformation estimation method.

```
Homography - 8DOF (OpenCV)=[0.4589614228946763, -0.103000439776077, 116.3581185117107;
 -0.003054628219988535, 0.5041989606294696, 158.8620479595043;
 -0.0002864201797323826, 1.837556401539272e-005, 1]
Affine Transformation - 6DOF (OpenCV)=[0.5178624619845528, -0.1106058823145759, 115.6280465056763;
 0.05897931627119413, 0.5145956424409714, 157.3912459155213;
 0, 0, 1]
Similarity Transformation - 4DOF = [0.51884139, -0.094492272, 113.75034;
 0.094492272, 0.51884139, 153.4742;
 0, 0, 1]
```

Figure 4: cout of the estimated Transformations

You should be getting close values; otherwise you are doing something wrong. A good way for debugging would be to visualize the matches that are selected as inliers. If the corresponding points do not belong to the model, then it may be that you are not projecting the points in the right way.

Hints for the task:

- See the following link for the function definitions:
  - findHomography:
    https://docs.opencv.org/3.4/d9/d0c/group__calib3d.html#ga4abc2ece9fab9398f2e560d53c8c9780
  - estimateRigidTransform
    https://docs.opencv.org/3.4/dc/d6b/group__video__track.html#ga762cbe5efd52cf078950196f3c616d48

Visualization

In order to actually see the effect of the transformation on the model image, you should visualize a transformed version of the original model on top of the scene image. To do that, it is preferable to overlay the model image on the scene image. There exist multiple ways to achieve this. One way to do it is to create an empty CV_8UC3 matrix and draw a colored rectangle the size of the model, using different colors for each of the solutions (OpenCV transformation / Computed similarity transformation). This image can then be transformed by applying the obtained transformation using the `cv::warpPerspective` function. It is worth noting that the output image should have the same size as the target image.

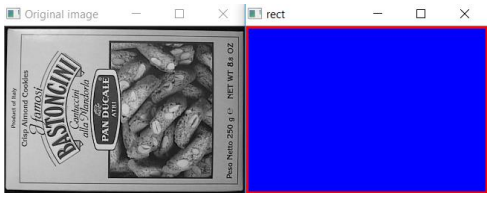

Figure 5: box image and its model to be transformed

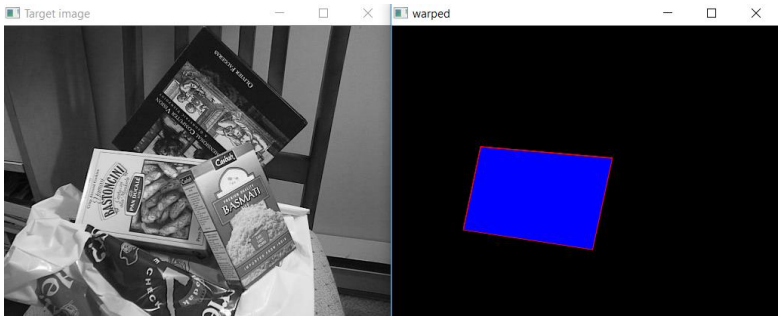

Figure 6: box_in_scene image and warped box image

All that is left now is to overlay the obtained images on the target image. To do this, you can use the `cv::addWeighted` function to calculate the weighted sum of two arrays (src1 and src2) as follows:

$$I = src1 * \alpha + src2 * \beta + \gamma$$

where,

I is a multi-dimensional index of array elements. In the case of multi-channel arrays, each channel is processed independently.

α is the weight of the first array.

β is the weight of the second array.

γ is a scalar added to each sum.

Hints for the task:

See the following link for the function definitions:

- warpPerspective
  https://docs.opencv.org/master/da/d54/group__imgproc__transform.html#gaf73673a7e8e18ec6963e3774e6a94b87
- addWeighted
  https://docs.opencv.org/2.4/doc/tutorials/core/adding_images/adding_images.html

You should scale the coefficients accordingly in order to obtain something like this:

Figure 7: box_in_scene with overlaid solutions

Figure 8: clutteredDesk with overlaid solutions

Figure 9: 66_5_PS_Dataset image with overlaid solutions

Each model image has a specific border color to differentiate between the results. Compare the output of the obtained similarity transformed image with the homography transformed image.

Question 1: What do you notice from the obtained results?

Question 2: What modifications could be made to the algorithm in order to enhance the accuracy and/or the computation time of the RANSAC-based pose estimation framework?

4.0 :: Mark Distribution

	Detailed Distribution			
Pre-Lab	Complete [1 Mark]		Incomplete [0 Marks]	**/1**
Task 1	Estimate the best Similarity Transformation in a RANSAC loop [3 Mark]	Estimate Similarity Transformation from very good matches [2 Mark]	Did not estimate Similarity Transformation [0 Marks]	**/3**
Task 2	Close transformation values compared to obtained OpenCV homography/affine transformation [1 Mark]	Partially close transformation values [0.5 Marks]	Transformation values not close at all [0 Marks]	**/1**
Task 3	Model warping is done correctly using obtained similarity transformation and OpenCV's transformation [1 Mark]	Warped original image using OpenCV-only homography/affine transformation [0.5 Marks]	Did not warp the original image using the obtained transformation [0 Marks]	**/1**
Task 4	Correctly displaying and comparing all overlaid solutions [1 Mark]	Displaying only overlaid OpenCV homography/affine transformation [0.5 Marks]	Did not display any solution [0 Marks]	**/1**
			Total:	**/7**