Computer Vision 2020
HW#2
Due on February 25, 2020

1. Scale-space blob detection 50%

In this question you will implement a scale-space blob detection algorithm.
Algorithm outline:

1. Generate a Laplacian of Gaussian filter (design the mask).
2. Build a Laplacian scale space, starting with some initial scale and going for n scales:
    1. Filter image with scale-normalized Laplacian at current scale.
    2. Save square of Laplacian response for current level of scale space.
    3. Increase scale by a factor k.
3. Perform nonmaximum suppression in scale space.
4. Display resulting circles at their characteristic scales.

The file input_blob.zip contains four images to test your code, and output_blob.zip contains sample output images for your reference. Keep in mind, though, that your output may look different depending on your threshold, range of scales, and other implementation details. In addition to the images provided, also **run your code on at least four images of your own choosing**.

Here are some hints to implement the blob detector:

Convert images to grayscale (rgb2gray command) and double (im2double) in Matlab.

You have to choose the initial scale, the factor k by which the scale is multiplied each time, and the number of levels in the scale space. Set the initial scale to 2 and use 10 to 15 levels in the scale pyramid. The multiplication factor should depend on the largest scale at which you want regions to be detected.

- You may want to use a three-dimensional array to represent your scale space.
- To perform nonmaximum suppression in scale space, you should first do nonmaximum suppression in each 2D slice separately.
- You also have to set a threshold on the squared Laplacian response above which to report region detections. You should play around with different values and choose one you like best.
- You should display the highest scoring 100 detected blobs for each image. If your code returns fewer than 100 blobs then you may have to lower your threshold. Hint: Don't forget that there is a multiplication factor that relates the scale at which a region is detected to the radius of the circle that most closely "approximates" the region (review the lecture slides).

2. Stitching pairs of images 50%

For this question you will be stitching the following pair of images.

1. Download the zip file hw2q2.zip
2. Load both images, convert to double and to grayscale.
3. Detect feature points in both images. We provided Harris detector code you can use.
4. Extract local neighborhoods around every keypoint in both images, and form descriptors simply by "flattening" the pixel values in each neighborhood to one-dimensional vectors. Experiment with different neighborhood sizes to see which one works the best.
   For a 25% extra mark experiment with SIFT descriptors.
   - **MATLAB:** There is a basic code for computing SIFT descriptors of circular regions in the zip file.
   - **Python:** You can use the OpenCV library to extract keypoints through the function cv2.xfeatures2D.SIFT_create().detect and compute descriptors through the function cv2.xfeatures2D.SIFT_create().compute.
5. Compute distances between every descriptor in one image and every descriptor in the other image. There is a MATLAB code for fast computation of Euclidean distance is the zip file. In Python, you can use scipy.spatial.distance.cdist(X,Y,'sqeuclidean') for fast computation of Euclidean distance. If you are not using SIFT descriptors, you should experiment with computing normalized correlation, or Euclidean distance after normalizing all descriptors to have zero mean and unit standard deviation.
6. Select putative matches based on the matrix of pairwise descriptor distances obtained above. You can select all pairs whose descriptor distances are below a specified threshold or select the top few hundred descriptor pairs with the smallest pairwise distances.
7. Write a RANSAC code. Run RANSAC to estimate a homography (projective) mapping one image onto the other. Report the number of inliers and the average residual for the inliers (squared distance between the point coordinates in one image and the transformed coordinates of the matching point in the other image). Also, display the locations of inlier matches in both images.
8. Warp one image onto the other using the estimated transformation. To do this in MATLAB, you will need to learn about maketform and imtransform functions. In Python, use skimage.transform.ProjectiveTransform and skimage.transform.warp.
9. Create a new image big enough to hold the panorama and composite the two images into it. You can composite by simply averaging the pixel values where the two images overlap. Your result should look something like this.
10. Submit your RANSAC code, the number of inliers and outliers you are getting, average residue for the inliers and the stitched image.

Tips and Details

- For RANSAC, a very simple implementation is sufficient. Use four matches to initialize the projective transform in each iteration. You should output a single transformation that gets the most inliers in the course of all the iterations. For the various RANSAC parameters (number of iterations, inlier threshold), play around with a few "reasonable" values and pick the ones that work best.
- Details of projective transform:

$$
\begin{aligned}
\mathbf{p}_1 &= \mathbf{M}\mathbf{p}_0 \\
\begin{bmatrix} x_1' \\ y_1' \\ w_1' \end{bmatrix} &= \begin{bmatrix} m_0 & m_1 & m_2 \\ m_3 & m_4 & m_5 \\ m_6 & m_7 & 1 \end{bmatrix} \begin{bmatrix} x_0 \\ y_0 \\ 1 \end{bmatrix} \\
\begin{bmatrix} x_1 \\ y_1 \end{bmatrix} &= \begin{bmatrix} x_1'/w_1' \\ y_1'/w_1' \end{bmatrix} \\
x_1 &= \frac{m_0 x_0 + m_1 y_0 + m_2}{m_6 x_0 + m_7 y_0 + 1} \\
x_2 &= \frac{m_3 x_0 + m_4 y_0 + m_5}{m_6 x_0 + m_7 y_0 + 1}
\end{aligned}
$$

$$
\begin{bmatrix} x_1 \\ y_1 \end{bmatrix} = \begin{bmatrix} x_0 & y_0 & 1 & 0 & 0 & 0 & -x_0 x_1 & -y_0 x_1 \\ 0 & 0 & 0 & x_0 & y_0 & 1 & -x_0 y_1 & -y_0 y_1 \end{bmatrix} \begin{bmatrix} m_0 \\ m_1 \\ m_2 \\ m_3 \\ m_4 \\ m_5 \\ m_6 \\ m_7 \end{bmatrix}
$$

$$
\begin{bmatrix} x_{1a} \\ y_{1a} \\ x_{1b} \\ y_{1b} \\ x_{1c} \\ y_{1c} \\ \vdots \end{bmatrix} = \begin{bmatrix} x_{0a} & y_{0a} & 1 & 0 & 0 & 0 & -x_{0a}x_{1a} & -y_{0a}x_{1a} \\ 0 & 0 & 0 & x_{0a} & y_{0a} & 1 & -x_{0a}y_{1a} & -y_{0a}y_{1a} \\ x_{0b} & y_{0b} & 1 & 0 & 0 & 0 & -x_{0b}x_{1b} & -y_{0b}x_{1b} \\ 0 & 0 & 0 & x_{0b} & y_{0b} & 1 & -x_{0b}y_{1b} & -y_{0b}y_{1b} \\ x_{0c} & y_{0c} & 1 & 0 & 0 & 0 & -x_{0c}x_{1c} & -y_{0c}x_{1c} \\ 0 & 0 & 0 & x_{0c} & y_{0c} & 1 & -x_{0c}y_{1c} & -y_{0c}y_{1c} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \end{bmatrix} \begin{bmatrix} m_0 \\ m_1 \\ m_2 \\ m_3 \\ m_4 \\ m_5 \\ m_6 \\ m_7 \end{bmatrix}
$$

$$
= \mathbf{C}\,[m_i]
$$

- Projective transform fitting calls for finding the M matrix in the above formula. Use pseudo inverse of C to find M.