Submission details

- This assignment should be done in pairs (contact the TA if this is a problem).
- The topic of this assignment is Image Features.
- The submission date is **3/27/2016**. Please pay attention to the late submission policy.
- Coding should be done in Matlab or Python. We prefer Python 2, but Python 3 is possible.
- You are not required to use any specific function or library, unless stated otherwise. If in doubt, please contact the TA via email or the Piazza website.
- For submission, package up your code as a zip file. Include your written answers as a pdf file named writeup.pdf. Include graphs and images in your writeup, and please label them so we know which figures go with which sub-problem.
- Send the final zip file to the TA. Add the course number to the subject of the email.
- If you have any questions about this assignment, please contact the TA stinger@tx.technion.ac.il.

Special instructions

- You must **choose ONE task** out of task 1 and task 2. Task 1 is probably safer, while task 2 is probably more interesting. You don't need to declare which task you choose until the actual submission, so you may switch between them as much as you like while implementing.
- The shorter task 3 is **mandatory**. You may choose to do both task 1 and task 2, but it is **definitely not required**. If you do choose to do both task 1 and task 2, you may skip task 3.
- Time permitting, it is highly recommended that you implement RANSAC on your own.

Task 1: Image Stabilization using Harris

In this task we will explore an important use of image features which is stabilizing image sets (and, basically, videos). We will use the Harris corner detector to perform the stabilization, and you may use any library function, apart from functions that actually perform image or video stabilization.

You are given a set of 7 images of the same scene, with minor movements of the camera. You need to stabilize the images so that the camera movements will not be noticed. You will implement two ways for matching feature points: Manual matching in item 3, and automatic matching in item 6. Since the automatic matching may sometime fail, you will use an algorithm in item 7 to take only matched feature points with a high probability to be correct.

The original Harris paper (the method was improved later; see Wikipedia under "Corner Detection"): C. Harris and M. Stephens, "A combined corner and edge detector", *Proc. of the 4th Alvey Vision Conference*, pp. 147–151, 1988.

1. **Viewing.** Build a small tool that will allow you to watch the given images as a movie, where you can control the time between every two frames (a single parameter). Watch the original set.
2. **Detection.** Use a Harris corner detector (If you choose to implement on your own, use algorithm 4.1 in the book or the paper). Use the detector on all the given images. Set a single threshold value for all of them, such that at least 100 feature points per image will be detected. **Note:** Sometimes the detector finds adjacent points, all passing the threshold. In this case you must select and return the point being the local maximum out of them and discard the rest. This is called Non-Maximum Suppression. Use an efficient way to keep only the local maximum

in every local area of several feature points. Use your own judgment to decide the size of the local area. Explain your solution and your considerations for area size selection. Show images with their feature points.

3. **Manual Matching.** Set the first image as the reference image and select at least 6 detected points. For each such point, find a feature point appearing in every one of the other 6 images matching it. Show all the images with their matched points, each appearing differently (e.g. use a red star for the first, a blue circle for the second, …).

4. **Transformation.** Calculate and show the affine transformation between the reference image and each of the other images, using the matching pairs of feature points.

$$\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}_n \begin{bmatrix} x_R \\ y_R \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \end{bmatrix}_n$$

   Where (x_R, y_R) are the coordinates of a feature point in the reference image, and n is the image number. Use a Least Squares method to compute the transformation parameters.

   <u>**Note:**</u> Each image has a unique transformation, and you are expected to show all the parameters of the 6 transformations.

5. **Stabilization.** For each image, calculate its stabilized version by applying the relevant transformation in the opposite direction:

$$\begin{bmatrix} x \\ y \end{bmatrix} = A_n \begin{bmatrix} x_S \\ y_S \end{bmatrix} + b_n$$

   Where (x_S, y_S) are the coordinates of the stabilized image (the destination image). Start by initializing an all zero image in the size of the reference image. Then, for each coordinate in the stabilized image, calculate the value of its matching coordinate in the non-stabilized image (the source image). For non-integer coordinate values, use bilinear interpolation that operates on the pixel values in the non-stabilized image. Show the reference image followed by the 6 stabilized images as a movie.

   After completing the manual method of matching and then stabilization, you will now implement an automatic method for matching, and then the same stabilization. You will attempt to find a matching feature point in the destination image for each feature point in the reference source image. To do that you will use image content similarity in the local feature point neighborhood.

6. **Automatic Matching.** Implement the following simplistic algorithm for automatic matching of feature points:
   a. Define a $L \times L$ search window around each feature point in the reference window.
   b. Apply the search window on the non-stabilized image and check which of the feature points in that image are included in that window. If no feature points are found, cancel the original feature point in the reference image and return to (a).
   c. For each feature point inside the search window, measure its compatibility to the original point. Define a $W \times W$ window around it and calculate the SSD measure between the two windows (compare to a window around the original point).

d. The feature point with the smallest SSD compared to the original point will be its matching point in this image.

The output vectors of the points must have the same number of rows, and matching feature points should be in matching rows.

What will be a good value for L and W, and why? Explain.

Show 10 automatically matched feature points that were found in all the images. Each feature point should appear differently (as in item 3).

7. **RANSAC.** The process you preformed in the previous question does not guarantee that all matching pairs can be used to calculate the transformation, since it cannot guarantee to eliminate false matching of feature points. To eliminate the incorrectly matched pairs, use and apply the RANSAC algorithm on the feature point pairs. If you choose to implement:

   a. Randomly select 3 points of matching feature point pairs.
   b. Calculate the transformation between the images using these 3 pairs.
   c. Using this transformation, Calculate for all the pairs the mapping of the coordinates from the reference image to the non-stabilized image, by applying the transformation formula:

$$\begin{bmatrix} x \\ y \end{bmatrix} = A_n \begin{bmatrix} x_I \\ y_I \end{bmatrix} + b_n$$

   Where (x_I, y_I) are the coordinates of the stabilized image. Calculate the Euclidean distance between every calculated coordinate (x, y) and its real value from the non-stabilized image.
   d. Define the inlier group as the group of all feature point pairs that the distance D calculated for them is small (meaning that the transformation is accurate).
   e. Repeat steps (a)-(d) for 20-100 times. Save the inlier group each time.
   f. Recalculate the transformation using all the pairs in the biggest inlier group.

   Why are we using only 3 points in the process above?

8. **Stabilization II.** Repeat item 4 with the transformation from item 6 and compare to the results of item 4. Which is better visually? Explain. **Bonus:** Find additional sets to stabilize and show your algorithm's performance on them.

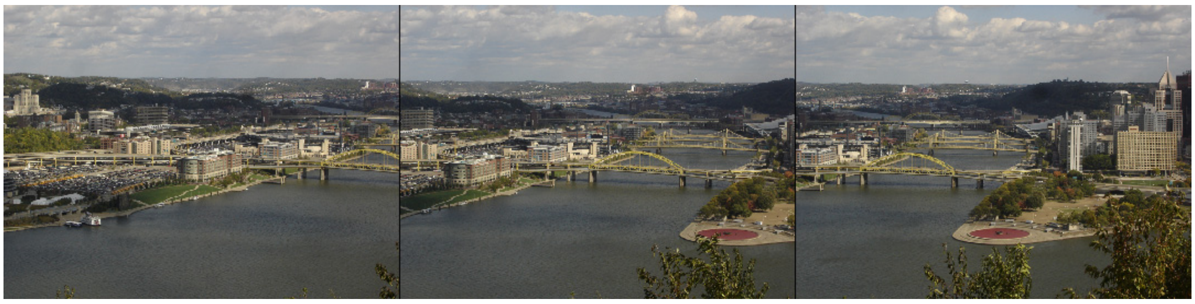
Task 2: Creating a Panorama Image using SIFT

In this task we will see how to create the ever-so-popular Panorama image from a set of images using the SIFT image features. You may use any library function, apart from functions that create Panorama images. Do **NOT** implement SIFT on your own! Possible sources for SIFT implementation:

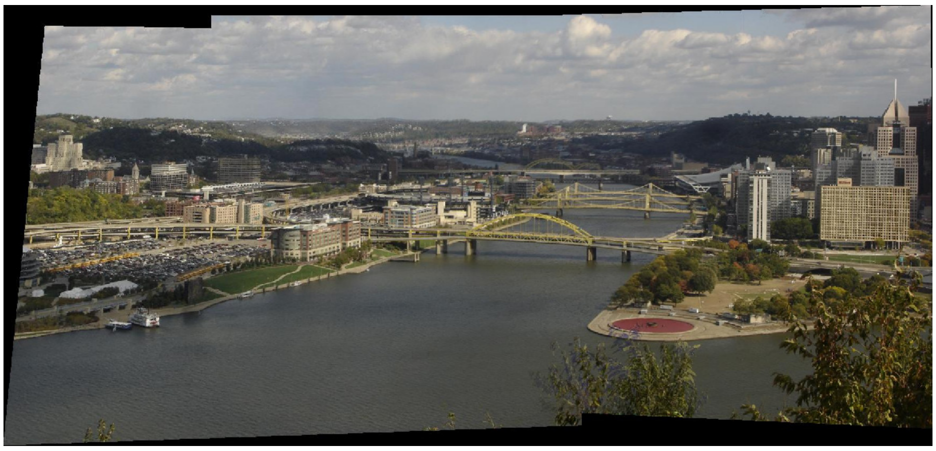
1. http://opencv-python-tutroals.readthedocs.org/en/latest/py_tutorials/py_feature2d/py_sift_intro/py_sift_intro.html
2. http://www.janeriksolem.net/2009/02/sift-python-implementation.html
3. Matlab: http://www.vlfeat.org/~vedaldi/code/sift.html

The original SIFT paper:
Lowe, D. G., "Object recognition from local scale-invariant features", *Proc. of ICCV,* pp. 1150–1157, 1999.

You are given a series of 3 images with a part of a single scene in each with some overlapping, while the camera moved from one part to the next. You will attempt to merge the parts into one big Panorama image using SIFT. For example, these three images

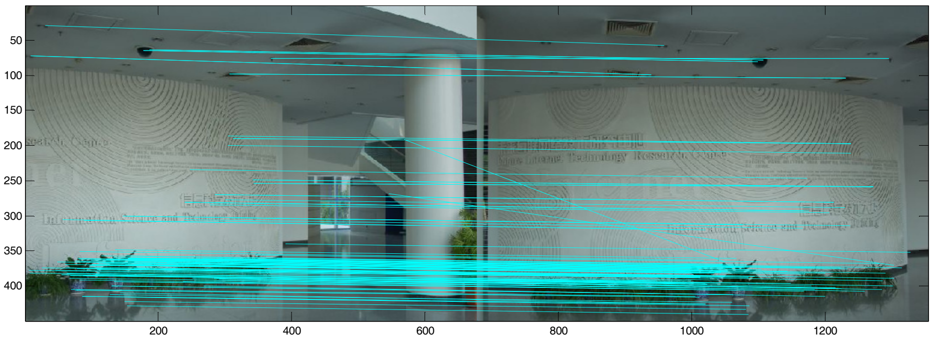


will give us this Panorama:



We will be working on the “Working set” until the very late stages of this task. To save on text, I will be referring to items from task 1 in certain places.

1. **Viewing.** Display the images and set them in the correct order of appearance.
2. **Detection.** Detect features on the first two images of the set using SIFT, similarly to item 2 in task 1. Note that here we do not need 100 points, we will use whatever SIFT gives us. SIFT returns a 128 element vector as the feature vector for each interest point.
3. **Matching.** Find matching feature points on both images by comparing feature vectors. Calculate the Euclidean distance between the vector of one point from the first image and the vectors of points in the second image. Set the match for the smallest distance, but pay attention that you do not allocate the same point twice. Show 50 of your matches by drawing lines between them like this (It is ok to get a few errors here):



4. **Homography.** Find the Homography matrix between the two images by using RANSAC (see task 1 item 7). In every RANSAC iteration the Homography matrix is calculated from scratch using a minimal number of interest point pairs. What is that minimal number?
   When RANSAC is done, calculate the final Homography from the biggest inliers group of interest point pairs. Use the Direct Linear Transformation method to do that (See below).

5. **Mapping.** Map the pixels from the projection plane of the first image to the projection plane of the second image using the Homography matrix you calculated. You need to create a new image that will hold this projection, and this image may be bigger and may contain areas of black (zero values) pixels.

6. **Merging I.** Now we are going to place both images on the same canvas. Note that we chose to project Image A on the projection plane of Image B, therefore you have to place image B and the projection of image A on that canvas. you need to create a canvas (a big black matrix) that can hold the two merged images. To calculate the size, you can project the coordinates of the corners of image A to the plane of image B. For example, the upper left corner:

$$\begin{bmatrix} \tilde{x}_i \\ \tilde{y}_i \\ \tilde{w}_i \end{bmatrix} = H \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$$

   Important notes:
   a. Remember you have to normalize the resulting coordinates.
   b. It is possible to get negative coordinates (since they are relative). Take this into account when calculating the locations on the canvas. The mapping of the corners of image A should be reasonable and not too far away from image B. Check the quality here.
   c. Note that in the overlapping regions in the Panorama, one image may "run over" the other image, but that should not create a visual problem, apart from noticeable "stitching"edges. That is, if you placed the images correctly.

   Show the Panorama image that you received.

7. **Merging II.** Now let's deal with those "stitching" lines. Upgrade your merging using a Pyramid Blending algorithm, similar to the one from HW2, task 1. Try two or three levels only in the pyramid. Repeat item 6, but instead of just placing the images on the canvas, do that with Pyramid Blending. Calculate the appropriate mask for each image as if it is placed on the canvas alone, while paying attention that that certain areas from the overlapping regions between the images will be taken from one of the images only (The "stitching" line between a pair of images will define the border of the mask of each image). Show what you got.
   Can you improve this result? Hint: it's in the mask. Show the improvement.

8. **Generalizing I.** Until now we built a Panorama only from two images. Now let's add the third image by repeating the above process (with the appropriate changes). Explain the needed changes and show the Panorama image composed of the three images.

9. **Generalizing II.** Go wild and try the other image sets. Some of them are composed of four images and even more. Consider changing the order of the image projection and placing on the canvas – Maybe better results can be achieved by starting from the middle image and adding to both sides? Show your results and discuss them. You may reduce image resolutions.

Direct Linear Transformation:

We want to calculate the Homography matrix H from the projection planes of image A to the projection plane of image B, by using pairs of matching points:

$$\begin{bmatrix} \tilde{x}_i \\ \tilde{y}_i \\ 1 \end{bmatrix} = \underbrace{\begin{bmatrix} h_{00} & h_{01} & h_{02} \\ h_{10} & h_{11} & h_{12} \\ h_{20} & h_{21} & h_{22} \end{bmatrix}}_{H} \begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix}$$

Where (x_i, y_i) are image A coordinates and $(\tilde{x}_i, \tilde{y}_i)$ are the matching coordinates of interest point i in image B. Given n interest point pairs, it is possible to write the following equation:

$$\underbrace{\begin{bmatrix} x_1 & y_1 & 1 & 0 & 0 & 0 & -\tilde{x}_1 x_1 & -\tilde{x}_1 y_1 & -\tilde{x}_1 \\ 0 & 0 & 0 & x_1 & y_1 & 1 & -\tilde{y}_1 x_1 & -\tilde{y}_1 y_1 & -\tilde{y}_1 \\ & & & & \vdots & & & & \\ x_n & y_n & 1 & 0 & 0 & 0 & -\tilde{x}_n x_n & -\tilde{x}_n y_n & -\tilde{x}_n \\ 0 & 0 & 0 & x_n & y_n & 1 & -\tilde{y}_n x_n & -\tilde{y}_n y_n & -\tilde{y}_n \end{bmatrix}}_{A} \underbrace{\begin{bmatrix} h_{00} \\ h_{01} \\ h_{02} \\ h_{10} \\ h_{11} \\ h_{12} \\ h_{20} \\ h_{21} \\ h_{22} \end{bmatrix}}_{\mathbf{h}} \cong \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

It is possible to approximate **h** by solving the following Least Squares problem:

$$\hat{\mathbf{h}} = \arg\min_{\mathbf{h}} \|A\mathbf{h}\|_2^2 \ \ \textit{subject to} \ \ \|\mathbf{h}\|_2 = 1$$

Which is solved by: $\hat{\mathbf{h}} = \text{eigenvector of } A^T A \text{ corresponding to the smallest eigenvalue of } A^T A$.

<u>Task 3: Computing Textons</u>

In this task we will get to know a type of features called textons. They provide us a great example of how to use gradients as features. You may use any library function in this task, apart from functions that compute textons.

1. Load the given images and show them. Describe the type of content you see in the images (edges, textures, smooth surfaces, …)
2. Apply Sobel operators (in X and Y directions separately) to the image. Do **<u>NOT</u>** threshold the results. We will mark the outputs M_x and M_y. Show them.
3. Compute the orientation based on M_x and M_y like this: $\theta = \tan^{-1}\left(\frac{M_y}{M_x}\right)$ for each pixel, and also the magnitude $M = \sqrt{M_x^2 + M_y^2}$ for each pixel. Show the results and explain.
4. Discretize the orientation image uniformly to 8 different angles using round. Pay attention to the fact that the data is cyclic.

5. For each pixel, compute a descriptor that will be the histogram of discretized orientations in a neighborhood of 11×11 pixels. Orientations should only contribute to the histogram if the magnitude M at the corresponding pixel is above some threshold (you have to select this threshold empirically).
6. Using K-Means, cluster the descriptors into 5, 10 and 50 clusters which are the textons. For each setting, assign textons to each pixel (allocate each pixel to a texton using some distance function), and color each texton differently.
7. Plot the results and discuss them. What do you see? Does it make sense? Explain.

Note: Possible distance functions for this task include, apart from the regular known metrics, also the χ^2 **(**Chi-squared) distance between histograms. See scipy.stats.chisquare(f_obs, f_exp).

Appendix A – List of useful Python library functions for these tasks

TBA