Exercise 2

Guidelines:

- Implement the code in each question using either Matlab or Python, including their libraries for basic operations like reading/writing images, overlaying in a figure, and off-the-shelf functions that are permitted in each question. Recommended are the Matlab Computer-Vision and Image-Processing toolboxes, the vl_feat library as well as Python Pillow, Opencv and scikit-image libraries.
- For each question, in addition to the required outputs and explanations to submit, you should submit all the code files that you have written (not including other libraries you have used).
- The input images for each question are organized in the respective folders Q1, Q2, Q3.
- In addition to the code files per question, you should submit one document, preferably in pdf format, with all the text and output figures required in each question.
- Submit the exercise to the email: cv.checker.21@gmail.com with the title "תרגיל בית 2". You may send updated versions (before the deadline) as well as grade appeals using the same thread.
- The submission deadline is Thursday June 3 at 23:59. Every late day will incur a reduction of 10 points off the grade.
- Make sure to work on your own and to implement yourselves the required parts of the code. Special attention will be given to this on the grading side.

Questions:

1. Epipolar Geometry (35 pts)

The goal is to practice the application and quality assessment of an estimated fundamental matrix. We will visualize the epipolar lines that result from a set of hand marked correspondences and compute the Epipolar and Algebraic distance measures.

Stages:

a. For each pair of images, manually mark 8 (or more) matching image points. Make sure that they are spread across the image and do not reside on a single scene plane.
b. Compute the fundamental matrix using two estimators out of the following possibilities (you may use off-the-shelf implementations):
    i. 7 point algorithm
    ii. 8 point algorithm
    iii. normalized 8 point algorithm

c. For each of the two estimators, visualize the correspondences and the epipolar lines on the opposite images that correspond to each of the correspondence points. Use your own code for applying the fundamental matrices (using matrix multiplication). You may use an off-the-shelf function to draw the lines (If you do it yourself you might need to compute the endpoints of the line, which are its intersections with the image boundary). See output example provided.
d. For each of the two estimators, compute the following errors (with your own code), averaged over the 8 (or more) input correspondences (x, x'):
   i. Algebraic Distance: $x'^T Fx$
   ii. (Symmetric) Epipolar Distance: $d(x', Fx)^2 + d(x, F^T x')^2$, where if we denote by $l = (a, b, c)^T$ the line Fx, then $d(x', Fx) = x'^T Fx / \sqrt{a^2 + b^2}$ (Make sure that you understand why this is the Euclidean distance between the point and the line, which is supposed to be zero in perfect conditions).

Submit:
1. Resulting visualizations from part c. for each of the two estimators and for each of the image pairs.
2. A table with the different computed types of the distances for each of the two estimators (averaged over the image pairs).

2. Photometric Stereo (30 pts)

The goal is to compute the disparity map for a left view of a rectified stereo pair using a simple scan-line approach.

Stages:
a. For each pair of corresponding image rows compute the optimal disparities row, separately for each of the sum-square-differences (SSD) and the normalized-cross-correlation (NCC) errors over $k \times k$ patches (use $k = 3, 9, 15$).
b. Compare your disparity maps with the given ground-truth maps in terms of four different measures (note: disparity values should be divided by 3 before use):
   i. AvgErr - mean of absolute differences in pixels
   ii. MedErr - median of absolute differences in pixels
   iii. Bad05 - percentage of disparities whose error is above 0.5
   iv. Bad4 - percentage of disparities whose error is above 4

Submit:
1. For each of the 6 combinations of {k=3,9,15} x {SSD,NCC} display the resulting disparities images for each image pair, with the 4 error measures in the figure title
2. Discuss shortly which are the preferable settings in terms of quality and runtime for the given examples

3. New View Synthesis (35 pts)

The goal is to generate new views of a scene, given a single image and a respective depth map. This can be done by changing the pose of the camera and synthesizing the image that would have been captured from the new view-point.

Stages:

a. Reproject all the image pixel locations into 3D space. This can be done using the (inverse of the) intrinsics matrix K and the depth map D. The 3D coordinates will be in the camera coordinate system, which means that we are using the camera matrix $P = K[R|T]$, where $R = I$ and $T = 0$. (note: We've provided useful matlab and python code for reading the camera matrix (.cam) and depth-map (.dpt)).
b. Project the 3D points back to the original view using the camera matrix. Now, synthesize the image by copying the intensity values (RGB) from the original image. The result should be identical to the original image - this is just a sanity check of the 2D-3D-2D process.
c. Generate a sequence of novel view images by changing the pose of the camera. This can be done by manipulating R and T in the extrinsics matrix. See the attached example for a reference to follow (If you view them at speed with a photo viewer, like irfanview, it will look like a video). More specifically, if we denote by $M_0 = [R\ |T]$ the original camera position, synthesize a sequence of jpeg images (you may save in ½ or ¼ resolution), by:
    i. Rotating the camera back and forth around its x-axis (ending at M_0)
    ii. Rotating the camera back and forth around its y-axis (ending at M_0)
    iii. Rotating the camera back and forth around its z-axis (ending at M_0)
    iv. Translating the camera back and forth along its x-axis (ending at M_0)
    v. Translating the camera back and forth along its y-axis (ending at M_0)
    vi. Translating the camera back and forth along its z-axis (ending at M_0)

Submit:

1. The synthesized jpg frames for each of the two scenes (choose 2 out of the 3). Make sure to have the sequences in separate sub-folders per scene.
2. Notice the difference between the rotation and the translation parts of the sequence.
    a. Which of the two gives the viewer a sense of the scene geometry?
    b. Explain why this is the case.
3. Notice the "holes" or black pixels in the results. There are 2 kinds of such holes: (i) Thin lines (as can be seen for example in frames 32/52 of the example) ; (ii) Larger blobs (as in frames 144/228).
    a. What are the sources of each of these holes?

b. Which one of them is inherent and which can be solved? Explain your answers (as for a solution - suggest a solution, without implementing it).