Image Processing: Assignment #2

Submission date: 11\02\2024, 07:59.

Please submit the assignment via Moodle.

For questions regarding the assignment please contact:

Alon Papini (imageprocessinghaifau@gmail.com).

Assignment Instructions:

- Submissions in ***pairs***.
- The code must be written in Python 3.11 or higher.
- The code must be reasonably ***documented***.
- Please submit one single ***.zip*** whose name should be using this format: ID1_ID2_HW2.zip. It should contain ***only***:
  - 'q1' folder, and inside it will be:
    - Your python script for question 1.
    - The 3 original images you received.
    - (Optional) The fixed images you were told to attach to the PDF file.
  - 'q2' folder, and inside it will be:
    - Your python script for question 2.
    - 'puzzles' folder, and inside it will be:
      - 'puzzle_affine_1' folder
      - 'puzzle_affine_2' folder
      - 'puzzle_homography_1' folder

      Note: The contents of each puzzle folder will be laid out at the end of question 2's instructions.
  - The PDF file where your written answers will be.

Assignment Topics:

- Point operations.
- Geometric operations.

Good Luck 😊

Problem 1 – Point operations (30 points):

Inside the q1 directory, you are given three broken images and a skeleton of a python script file called imageFix.py.

For each of those images:

a) Choose which correction is the best one for the image.

- Brightness and contrast stretching.
- Gamma correction.
- Histogram equalization.

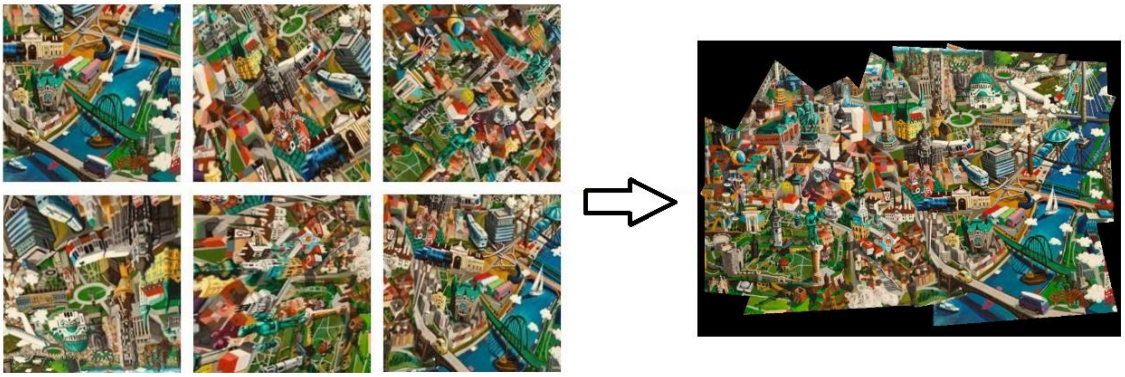
Explain your choices and what you expect to happen using the other corrections.

b) Apply the chosen correction for the image, attach the fixed image along with the parameters used (it's up to you to have figured out the parameters that best fit that image) in your PDF file.

Remember to also include in your submission zip the python file you used for this question!

Problem 2 – Puzzle solving with geometric operations (70 points):

We're given pieces of a puzzle, and we want to solve the puzzle by "stitching" the pieces together. For example:



However, it's not that simple since some mischievous entity applied affine or projective transformations to our puzzle pieces. Therefore, we need to find the inverse-transformations for those pieces so we can put them back together.

There are 3 different puzzles, each in its own folder inside the 'puzzles' folder, and for each of them you are given the following:

- A folder ('pieces') containing all the transformed puzzle pieces.
- A matches.txt file containing pixel coordinate matches between pairs of images. Each pair consists of 3 or 4 couples of matches, depending on whether the transform is affine or projective. The matches are between the coordinates of a point in a puzzle piece and its appropriate place in the entire puzzle canvas.

Assume that we always start from the first image and aim to stitch the other image to the first one. Therefore, the matches are between the pairs (1^{st},2^{nd}), (1^{st},3^{rd}), etc. There is a reason why the matches are always between the first puzzle piece and a different puzzle piece, and it will be explained in the following section.

Before we dive into the puzzle solving steps, let's take a look at an example solution of a flower puzzle, in the demo folder. You'll find an example of matches.txt with appropriate pixel coordinates filled out in the format we'll use for this question, and a solution.jpg file showing the fully solved puzzle.

Additionally, there are two sub-folders:

- pieces: There are 8 puzzle pieces files in the folder.
- abs_pieces: The same 8 pieces after they've been inverse-transformed to how we'd like them to be in the final solution canvas (so basically the solution image in the pieces folder is all the images in the abs_pieces folder joined together).

You should take note of the following, as it's true for the puzzles you'll be solving in this question yourselves:

- The first piece is **already located** in the absolute correct position. Moreover, the image size of the first piece is the size of the entire solution image. In other words, the first piece is properly placed in the puzzle solution canvas, and you'll want to stitch the *other* pieces to the canvas that starts with the first piece.
- The structure of the matches.txt file. You MUST use the same format for your own solutions for the 3 puzzles you'll do yourselves.

After understanding everything you need from the demo, it's time to start puzzle solving. Besides the 'puzzles' folder, you're given a skeleton python script file in the form of puzzleSolver.py, so let's start working with it.

Apply the following steps to each of the 3 puzzles you need to solve:

a) Go through all the pieces, use a program like paint to measure exact pixel coordinates and pick pairs of pixel coordinates that you think are best for calculating the transformations that will help you solve the puzzle (Note: Some points you may choose might not work properly for you). Add those coordinates appropriately to matches.txt. Remember: we are doing the matches between the (1^{st},2^{nd}), (1^{st},3^{rd}),…, so we can later transfer those pieces onto the final canvas that is already represented by the first piece.

b) Call the function ‘prepare_puzzle’. It returns the matches between pixel coordinates, whether the current puzzle is affine or not and the number of images.

   The shape of the matches returned by this function is (n-1, 3|4, 2, 2): We have n-1 image pairs and for each pair we need either 3 or 4 pixel coordinate pairs (depending on whether the current puzzle is limited to affine transformations or not). Each pair looks like $(x_1,y_1),(x_2,y_2)$, which is a match between pixel coordinates in the puzzle canvas (the first image) and pixel coordinates in the destination (which is the transformed puzzle piece we’ll want to inverse-transform to the canvas).

c) Implement the function 'get_transform', that takes the matches of specific image pair (1^{st},x^{th}) and returns the transformation from 1^{st} to x^{th} image (affine or projective). This will be used later for finding the inverse-transform, the thing you’ll need to “piece the puzzle” with. You may use existing cv2 implementations for calculating the transformation.

d) Implement the function ‘inverse_transform_target_image’, which receives a target image (x^{th}), the transformation from 1^{st} to x^{th} you got from ‘get_transform’ and 1^{st} image size (in other words, the output solution’s size).

   The function will then perform the inverse-transform needed to bring x^{th} to its proper place in 1^{st}’s canvas. You may use existing np and cv2 functions to achieve this (for example, with cv2.warpPerspective you can also specify the interpolation method for the inverse transformation you’ll use).

   By the way, the 1^{st} image size parameter is particularly important, as it removes the concern of out-of-bound pixels – we know the canvas size, so we’ll know how to inverse-map the puzzle piece, and where to (without accidentally landing on out-of-bounds pixels).

   Note: Before we move on to piecing the puzzle, you should remember that there will be some overlap between the pieces. Don’t expect the pieces to just fit side-by-side.

e) Implement the 'stitch' function. At this stage you have a pair of images (1st,xth) both placed properly on the final canvas, coordinate-wise. However, you still need to stitch them together into one actual image. Think of a solution that takes both of those images (like in the demo/flowers/abs_pieces folder) and stitch them into one image (it's up to you if you want to create a partial stitch per pair you've solved, or if you want to stitch all pieces after you're done with all pairs individually). It may be simpler than it seems – the implementation could even be as simple as the return value of one cv2 function!

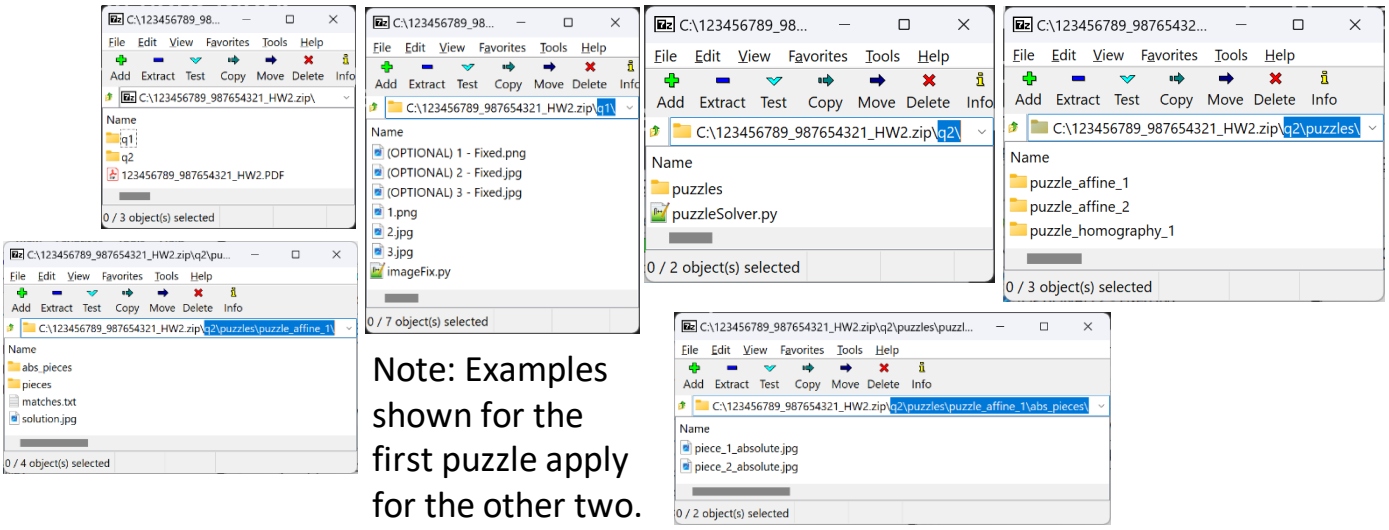
f) After doing everything you needed with each (piece1,pieceX) pair, output the final image (after adding all the pieces to the puzzle) as 'solution.jpg'

After you're done with all 3 puzzles, here's what you'll need for the submission. You received a 'puzzles' folder, and you'll also include it in your submission, except now it also has your own additions to it. In each of the 3 puzzle folders you received, include the following:

1) solution.jpg, the completed puzzle image.
2) An 'abs_pieces' folder containing the absolute correct image of each piece over the black canvas (just like in demo/flowers/abs_pieces).
3) matches.txt, properly filled.

Besides the 'puzzles' folder, include your python script file for this question. Additionally, please write a few words and maybe include a few screenshots of your work on this question in the PDF file you'll include because of Question 1. No specific format requirement for this, but it will be useful for checking your work.

The following screenshots should clarify how the zip needs to look like:



Note: Examples shown for the first puzzle apply for the other two.