ECE661: Homework 3

Fall 2020
Due Date: Sept 21,2020

Turn in typed solutions via BrightSpace. Additional instructions can be found at BrightSpace.

1 Introduction

The goal of this homework is to remove projective and affine distortions in the given images. This is referred as *metric rectification* in the text [1] (Second Edition). For your programming tasks, you will implement the approaches given the textbook for metric rectification. Note that you also have one new extra credit task for this homework and that would require additional reading. Make sure to read the description for the given tasks carefully. This homework is based on the concepts covered in Lectures 4 and 5.

You will show results after the distortions removal on the given input images as well as using your own images for the following approaches

1. **Point-to-point correspondences:** This approach is a trivial extension of what you implemented in your Homework 2. You will use the given measurements to get points in the original scene as your domain and the corresponding pixel coordinates in the image as your range which has projective and affine distortions. The inverse homography will eliminate these distortions.

2. **Two-step method** in which you first remove the projective distortion using the Vanishing Line (VL) method discussed in Lecture 4. Subsequently, you remove the affine distortion by using the $\cos\theta$ expression with θ equal to 90°. Note that you must first remove the projective distortion before you can remove the affine distortion with the $\cos\theta$ based method.

3. **One-step method** removes both the projective and affine distortions in one step.

For the two-step approach you estimate a VL in the image plane by picking pixel coordinates of at least two pairs of parallel lines in the original scene. Taking the cross-product of two such pixels on any line in the image

will give you the HC representation of that line. Taking the cross-product of 3-vectors for two different lines (which are parallel in the original scene) will give you the HC representation for the Vanishing Point for those two lines. Then taking the cross-product of two such vanishing points for two different pairs of parallel lines will give you the VL you need for getting rid of the projective distortion. For identifying pixel coordinates of parallel lines, you can use the same set of tools (GIMP, IrfanView, etc) as you used in Homework 2. As you learned in Lecture 5, you can obtain the expression for $\cos\theta$ in the form of Dual Degenerate Conic $\mathbf{C}_\infty^*$ as follows

$$\cos\theta = \frac{\mathbf{l}^T\mathbf{C}_\infty^*\mathbf{m}}{\sqrt{(\mathbf{l}^T\mathbf{C}_\infty^*\mathbf{l})(\mathbf{m}^T\mathbf{C}_\infty^*\mathbf{m})}} \tag{1}$$

where $\mathbf{C}_\infty^* = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$.

For affine distortion removal, you need to identify orthogonal line pairs, i.e., angle-to-angle correspondence between orthogonal lines in the original scenes and in the image planes. Then, estimate the homography matrix for affine distortion correction as explained in Lecture 5 which essentially is based on Eqn. 1. Note that in the text, the two-step approach is termed as *stratified*.

For the one-step approach, let $\mathbf{C}_\infty^{*'}$ be a projection of the $\mathbf{C}_\infty^*$ and it's represented as

$$\mathbf{C}_\infty^{*'} = \begin{bmatrix} a & b/2 & d/2 \\ b/2 & c & e/2 \\ d/2 & e/2 & f \end{bmatrix}$$

Identify lines $\mathbf{l}'$ and $\mathbf{m}'$ in an image such that their corresponding lines $\mathbf{l}$ and $\mathbf{m}$ in world coordinates are orthogonal. You need to identify at least five such pairs. Then you can estimate the unknowns a, b, c, d, and e by solving

$$\mathbf{l}'^T\mathbf{C}_\infty^{*'}\mathbf{m}' = 0$$

Note that since you're working with HC where only ratios matter, you can fix $f = 1$.

After estimating $\mathbf{C}_\infty^{*'}$, you can estimate the homography that corrects both projective and affine distortion via SVD of $\mathbf{C}_\infty^{*'}$. That is,

$$\mathbf{C}_\infty^{*'} = \mathbf{U}\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}\mathbf{U}^T$$

(a) Image 1 (b) Image 2 (c) Image 3

Figure 1: Input images for Task1

where the homography $\mathbf{H} = \mathbf{U}$ can be used to rectify both projective and affine distortions in one-step. For further information on the one-step approach, see pages 42,55, and 56 of the text [1].

2 Programming Tasks

2.1 Task 1

Download the input images, shown in Fig. 1 and their world coordinates (height and width measurements of some planar object in the scenes), separately.

1. Show results after distortion corrections using **point-to-point correspondences**, this is a trivial extension of Homework 2. With the given height and width, your points in the undistorted image should be (0,0), (0, width), (height, 0), and (height, width). Their corresponding points in the distorted input image can be manually measured using a software such as GIMP. After you have found the correspondences you simply need to apply the homography to the input image to remove distortion.

2. Show results after distortion correction using the **two-step** and **one-step** approaches.

3. Outline your observations on the results obtained using the above three methods.

2.2 Task 2

Repeat the steps, outlined in Task 1, on at least two sets of your own images. Usually, images with repetitive patterns on planar surfaces such as facades, walls with portraits, etc have many parallel or orthogonal line features and are easier to work with. Make sure to use images with significant projective and affine distortions. You can use approximate estimations of world coordinates, however state your assumptions clearly.

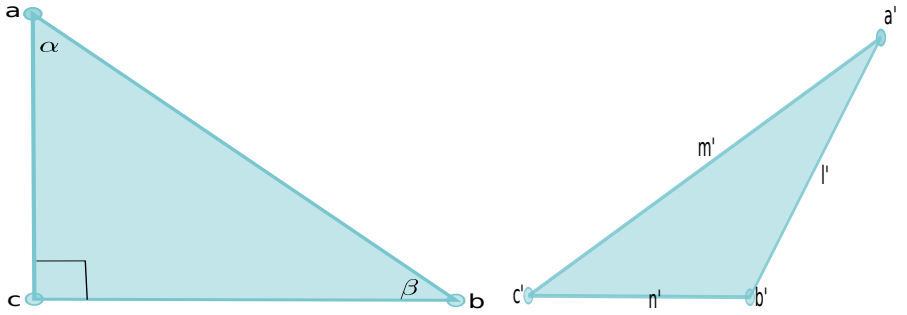
2.3 Notes

- Numpy vectorized operations are usually faster than "for" loop for numpy arrays. If you want to write an efficient python program, follow some online tutorials on "Numpy Vectorized Operations" or "pythonic for loops".
- You can avoid storing very large images for output using the following steps. First estimate the $x_{min}, y_{min}, x_{max}$, and y_{max} using the homography matrix $\mathbf{H}$
  1. Compute the aspect ratio from the estimated bounds as w_o/h_o, where w_o and h_o are the estimated output width and height.
  2. Fix either width w_i or height h_i of your input image and estimate the other using the aspect ratio computed in the previous step.
  3. Compute the scale factor as $s = h_o/h_i$, assuming we fixed h_i in the previous step. Alternatively, you can choose the $s = \max(s_w, s_h)$, i.e., maximum possible scaling either along width or height.
  4. Now your row iteration should look like the following $i = y_{min} + i/s$, where $i \in \{0, 1, \cdots, h_i - 1\}$. Apply the same logic for column iteration during the image warping iterations or vectorized numpy operations.

  In essence, you will be updating image scaling at the time of image warping using the above steps. For programming examples, refer to HW3 solutions from Fall2016 and HW2 solutions (C/C++) from Fall2012 and Fall2010 offerings.

3 Extra Credit Task

If you look at the material on pages 54-55 in the textbook, you will find one more important property of the $\mathbf{C}^*_\infty$. That is, you can compute the length ratios in the original scene using $\mathbf{C}^*_\infty$.



(a) Known geometry in World Coordinates

(b) The corresponding distorted view.

Figure 2: A view with known geometry in world coordinates and the corresponding view with distortion.

For example, as shown in Fig. 2, using standard trigonometric sine rule you can estimate the length ratios in world coordinates as $d(\mathbf{b}, \mathbf{c}) : d(\mathbf{a}, \mathbf{c}) = \sin\alpha : \sin\beta$. To obtain the same length ratios using the $\mathbf{l}'$, $\mathbf{m}'$, and $\mathbf{n}'$, you can estimate the angles α and β using the Eqn. 1, and then estimate the length ratios using the sine rule.

For completing this task, you will identify known α and β from the given world coordinates. Then estimate the new values using $\mathbf{C}_\infty^*$ and verify if they're equal to the original values in the world coordinates.

The key challenges to complete this task include

1. Determine at what stage of your algorithm you can measure the length ratios using $\mathbf{C}_\infty^*$.
2. Show the initial edge length ratios using the provided world coordinates and then using $\mathbf{C}_\infty^*$. As mentioned in the previous item, you need to first identify where you can verify this property in your implementation.
3. Your observations on all the three approaches. If this property cannot be verified for a specific approach then provide justifications why it cannot be done.
4. Considering human annotation errors and numerical issues, the errors within $\pm 5^\circ$ margin are acceptable for the angles α, and β. Since the length ratios are computed from α and β, it's redundant to provide error margins for length ratio estimations.

4 Submission Instructions

Include a typed report explaining how did you solve the given programming tasks.

1. Turn in a zipped file, it should include (a) a typed pdf report with source code files and results, (b) source code files (.py, .cpp,.c), (c) Separate input and output images are required for this task to see the effects of scaling technique. If you have followed all the notes correctly, the zip file should be of reasonable size. Rename your .zip file as hw3_<First Name><Last Name>.zip and follow the same file naming convention for your pdf report too.
2. Your pdf must include a description of
   - The logic that you used to solve the given tasks.
   - The steps that you used for each of the tasks with relevant equations.
   - The input and output images for each task. Clearly show the plotted parallel / orthogonal lines that you chose in the input images.
   - At least some use vectorized numpy operations, if not fully optimized, is expected in your Python code.
   - Your observations on the output quality and performance of each approach.
   - Your source code. Make sure that your source code files are adequately commented and cleaned up.
3. In order to avoid large file size of your submission, include JPEG images in your report for showing your results and your input images for Task2. Additionally, make sure to apply the scaling technique recommended in Sec 2.3.
4. The sample solutions from previous years are for reference only, it's important not to get too biased by those solutions. **Your code and final report must be your own work.**

References

[1] Richard Hartley and Andrew Zisserman. *Multiple view geometry in computer vision*. Cambridge university press, 2003.