Cornell University
CS 5670: Introduction to Computer Vision
Fall 2025. Instructor: Andrew Owens

Problem Set 5: Panorama Stitching

Posted: Wednesday, November 5, 2025 **Due:** Wednesday, November 19, 2025

We have created two Gradescope assignments for PS5. This is because problems 5.1 and 5.2 are theory questions, and problems 5.3 and 5.4 are programming questions. For the theory questions, please prepare a PDF with your solutions and upload it to the corresponding assignment on Gradescope. For the programming questions, you should finish the Colab Notebook, generate a PDF (using the conversion script at the end of the notebook) and then upload it to the corresponding assignment on Gradescope.

Colab Notebook: https://drive.google.com/file/d/1LmE4bffsR6XOJghgOYW4dTPagFrQxPC5

Problem 5.1 *Linear transformations*

(a) **Effects of linear transformations (4 points).** For each of the following matrices, show the corresponding property. All variables are real numbers.

(i) **Transformation matrix**: $\begin{bmatrix} \cos\theta & -\sin\theta & a \\ \sin\theta & \cos\theta & b \\ 0 & 0 & 1 \end{bmatrix}$

Property: Lengths of line segments are always preserved.

(ii) **Transformation matrix**: $\begin{bmatrix} a & 0 & 0 \\ 0 & a & 0 \\ 0 & 0 & 1 \end{bmatrix}$

Property: Angles between lines are always preserved.

(iii) **Transformation matrix**: $\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ a & b & c \end{bmatrix}$

Property: Parallel lines are *not* necessarily parallel.

Problem 5.2 *Calculating homography from camera parameters*

We will explore two situations where the image content in two images can be related by a homography. In each case, we will derive the corresponding homography.

(a) **Homography induced by camera rotation (5 points).** Suppose we have two images, I_1 and I_2, captured by cameras C_1 and C_2 respectively. Their pose differs by a rotation (there they have the same camera center). Let the intrinsic matrices for the cameras be K_1 and K_2 respectively. Finally, let R be a 3×3 rotation matrix that defines the relative pose between the cameras.

(i) Given a 3D point X_1 in the C_1 coordinate system, please provide an expression that you can use to compute the location of the pixel in the image I_1.

(ii) Similarly, given the same 3D point X_1 in the C_1 coordinate system, please provide an expression that you can use to compute the location of the corresponding pixel in the image I_2.

(iii) Given a pixel x_1 in the image I_1, provide an expression that can represents the set of possible 3D points in the C_1 coordinate system that project to this pixel. (*Hint*: It would be useful to think about the properties of homogeneous coordinates)

(iv) Finally, given a pixel x_1 in the image I_1, please provide an expression that computes a corresponding pixel x_2 in the image I_2.

(v) Show that the expression derived above is a homography.

(b) **Planar homography (4 points).** Consider the plane $z = 0$ in a world coordinate system. Suppose that there is a camera C_1 that views the plane, and has the 3×4 extrinsic matrix $E = [R \mid t]$. Here, R is a 3×3 rotation matrix and t is a 3×1 translation vector. The camera captures an image I_1, and has an associated intrinsics matrix K. Show that the projection of any point on the $z = 0$ plane to the image I_1 can be represented by a homography. (*Hint*: use a step-by-step derivation similar to your solution to 5.2(a)).

Problem 5.3 *Homography estimation*

In this problem, we will implement a function to compute a homography matrix H from a set of point correspondences. You can implement this function using nonlinear least squares (or, alternatively, the direct linear transform as described in class).

Hint: For nonlinear least squares, we recommend using `scipy` library's built-in nonlinear least squares. To use that function, you need to:

- Define a cost function, $f(\texttt{h}; \texttt{pts1}, \texttt{pts2})$, that calculates the projection error (a vector of length $2N$) between `pts1` and projected `pts2` using homography H. The vector $\texttt{r} \in \mathbb{R}^2$ of residuals for point i is:

$$\texttt{r} = \texttt{pts1}[\texttt{i}] - \texttt{cart}(H * \texttt{homog}(\texttt{pts2}[\texttt{i}])). \tag{1}$$

Here, `homog(x)` converts `x` into homogeneous coordinates, `cart(x)` converts `x` to Cartesian coordinates, and h is a flattened version of the homography H, which is what we will estimate.[1]

- Provide an initial guess for the homography `h`. A length 9 vector filled with '1's should be good enough.

(a) **(4 points)** Implement a function `fit_homography(pts1, pts2)` that computes the homography matrix from a set of point correspondences.

(b) **(4 points)** Complete the function `apply_homography` and apply the computed homography H to the original points. Plot these original points, the desired destination points, and the transformed points using matplotlib, as a scatter plot. Make sure the destination and transformed points are very close together. Include the visualization in the Colab notebook.

Problem 5.4 *Panorama stitching*

In this problem we will develop an algorithm for stitching a panorama from overlapping photos (Figure 1), which amounts to estimating a transformation that aligns one image to another. To do this, we will compute ORB features[2] in both images and match them to obtain correspondences. We will then estimate a homography from these correspondences, and we'll use it to stitch the two images together in a common coordinate system.

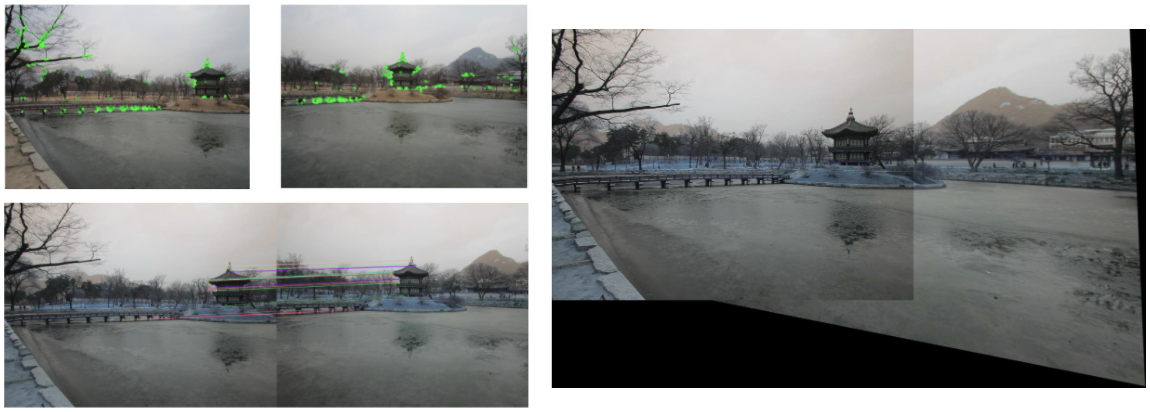

Figure 1: Panorama produced using our implementation. The image pair shown on the left represents the keypoints in the two source images and below them are the predicted feature correspondences. On the right is the stitched panorama.

In order to get an accurate transformation, we will need many accurate feature matches. Unfortunately, feature matching is a noisy process: even if two image patches (and their ORB descriptors) look alike, they may not be an actual match.

[1] Alternatively, you can try minimizing this loss using PyTorch and gradient descent. We didn't do this in our implementation, but it should have a very similar result — nonlinear least squares can sometimes provide a big improvement, but probably not for a simple problem like this one!

[2] ORB is a hand-crafted feature similar to SIFT. Until recently, SIFT was unfortunately patented, and was therefore difficult to use in OpenCV.

To make our algorithm robust to matching errors, we will use RANSAC, a method for estimating a parametric model from noisy observations. We will detect keypoints and represent descriptors using ORB. We will then match features, using heuristics to remove bad matches. We have provided you with two images (Figure 1) that you'll use to create a panorama.

(a) You will start by computing features and image correspondences.

- **(2 points)** Implement `get_orb_features(img)` to compute orb features for both of the given image.
- **(2 points)** Implement `match_keypoints(desc1, desc2)` to compute keypoint correspondences between the two source images using the ratio test. Run the plotting code to visualize the detected features and resulting correspondences.

(b) **(2 points)** Your homography-fitting function from problem 8.1 will only work well if there are no mismatched features. To make it more robust, implement a function `fit_homography_ransac(pts1, pts2)` that fits a homography using RANSAC. You can call `fit_homography(pts1, pts2)` inside the inner loop of RANSAC. You will also be responsible for figuring out the set of parameters to use to produce the best results.

(c) **(2 points)** Now, using the functions implemented so far, complete `compute_homography_between_images(img1, img2)`. This function should return two homography matrices, one representing the forward transformation from `img1` to `img2`, and the other representing the reverse transformation from `img2` to `img1`.

(d) We are now ready to perform panorama stitching! To stitch the images together, you will implement two types of image warping techniques. Your deliverables are:

(i) **(2 points)** Complete the function `forward_warp_image` and visualize the stitched images.

(ii) **(2 points)** Complete the function `backward_warp_image` and visualize the stitched images.

(iii) **(1 points)** Do you notice any differences between your forward warping and backward warping implementations? Please comment briefly in the Colab Notebook (there should be a cell where you can type your answer).

Hints:

- Do note that for both types of warping we are computing the color values for the destination image. We have initialized the destination image with all zeros so that you can fill in the color values.
- You may run into issues where the transformed pixels fall in locations that are outside the image boundaries. One way to handle this issue is to clip such values to the minimum or maximum allowed value depending on the type of the issue.
- The result for backward warping would look similar to the stitched panorama in Figure 1 if implemented correctly.

Acknowledgements. Part of the homework dataset and the starter code were taken from David Fouhey's EECS 442 class in the University of Michigan. The panorama stitching problem is adapted from MIT's 6.869 problem set. The theory problems were created by Bharath Raj.