ELEC 6910A Project 2

Planar Homographies

Professor: TAN, Ping

Due Nov. 1, 2023

1 Augmented Reality With Planar Homographies

1.1 Introduction

In this project, you will implement an AR application step by step using planar homography. Before we step into the implementation, we will walk you through the theory of planar homographies. In the programming section, you will first learn to find point correspondences between two images and use them to estimate their homography. Using this homography, you will warp images and finally implement your AR application.

Every script and function you write in this section should be included in the *matlab/* or *python/* directory. Please include the resulting images in your write-up. **If you have the proper individual Matlab subscription for Matlab, you can still finish this project in Matlab. Otherwise, you build it up from the Python version.**

Post questions to Canvas so everybody can share unless the questions are private. Please look at Canvas first if similar questions have been posted.

1.2 Preliminary

1.2.1 Homographies

A planar homography is a warp operation (which is a mapping from pixel coordinates from one camera frame to another) that makes a fundamental assumption of the points lying on a plane in the real world. Under this particular assumption, pixel coordinates in one view of the points on the plane can be directly mapped to pixel coordinates in another camera view of the same points.

There exists a homography $\mathbf{H}$ that satisfies equation 1 below, given two 3×4 camera projection matrices $\mathbf{P}_1$ and $\mathbf{P}_2$ corresponding to the two cameras and a plane Π.

$$x_1 \equiv \mathbf{H} x_2, \tag{1}$$

where $\equiv$ is identical to or equal up to a scale. The points x_1 and x_2 are in homogeneous coordinates, which means they have an additional dimension. If x_1 is a 3D vector $[u_1, v_1, w_1]^\top$, it represents the 2D point $[\frac{u_1}{w_1}, \frac{v_1}{w_1}]$(called heterogeneous coordinates).

This additional dimension is a mathematical convenience to represent transformations

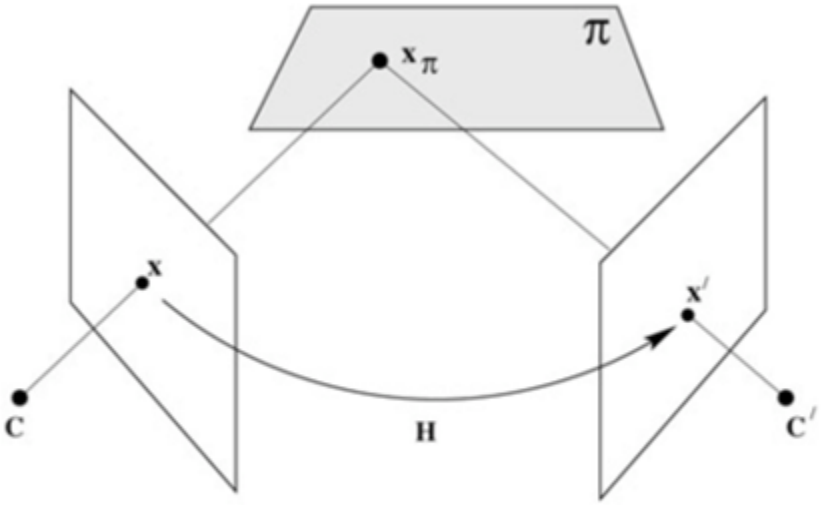

Figure 1: A homography **H** links all points $\mathbf{x}_\pi$ lying in plane π between two camera views $\mathbf{x}$ and $\mathbf{x}'$ in cameras C and C' respectively such that $\mathbf{x}' = \mathbf{H}\mathbf{x}$. *[From Hartley and Zisserman]*

Figure 1: The figure of Homography

(like translation, rotation, scaling, etc) in a concise matrix form. The $\equiv$ means that the equation is correct to a scaling factor.

Note: A degenerate case happens when the plane Π contains both cameras' centers, in which case there are infinite choices of **H** satisfying equation 1.

1.3 Direct Linear Transform(DLT)

A very common problem in projective geometry is often of the form $\mathbf{x} \equiv \mathbf{A}y$, where $\mathbf{x}$ and $\mathbf{y}$ are known vectors, and $\mathbf{A}$ is a matrix which contains unknowns to be solved. Given matching points in two images, our homography relationship is an instance of such a problem. Note that the equality holds only up to scale (which means that the set of equations is of the form $\mathbf{x} = \lambda\mathbf{H}\mathbf{x}^{'}$), which is why we cannot use an ordinary least squares solution such as what you may have used in the past to solve simultaneous equations. A standard approach to solving these problems is called the Direct Linear Transform, where we rewrite the equation as proper homogeneous equations, which are then solved in the standard least squares sense. Since this process involves disentangling the structure of the H matrix, it transforms the

problem into a set of linear equations, thus giving it its name.

Let $\mathbf{X}_1 = \{x_1, x_2, \ldots, x_n\}$ be a set of points in an image and $\mathbf{X}_2 = \{x_1^{'}, x_2^{'}, \ldots, x_n^{'}\}$ be the set of corresponding points in an image taken by another camera. Suppose there exists a homography $\mathbf{H}$ such that:

$$x_1 \equiv \mathbf{H} x_1^{'}, \tag{2}$$

where x_i and $x_i^{'}$ are in homogeneous coordinates. For each point pair, this relation can be rewritten as:

$$\mathbf{A}_i \mathbf{h} = 0, \tag{3}$$

where $\mathbf{h}$ is a column vector reshaped from $\mathbf{H}$, and A_i is a matrix with elements derived from the points x_i and $x_i^{'}$. You can solve for $\mathbf{h}$ by finding the right null space by Singular Value Decomposition or Eigen Decomposition as described below.

1.4 Eigenvalue Decomposition

One way to solve $\mathbf{Ax} = 0$ is to calculate the eigenvector corresponding to the smallest eigenvalue when A is a square matrix. Consider this example:

$$\begin{bmatrix} 3 & 6 & -8 \\ 0 & 0 & 6 \\ 0 & 0 & 2 \end{bmatrix} \tag{4}$$

Using the Matlab function *eig*, we get the following eigenvalues and eigenvectors:

$$\begin{aligned} \mathbf{V} &= \begin{bmatrix} 1.000 & -0.8944 & -0.9535 \\ 0 & 0.4472 & 0.2860 \\ 0 & 0 & 0.0953 \end{bmatrix} \\ \mathbf{D} &= \begin{bmatrix} 3 & 0 & 2 \end{bmatrix} \end{aligned} \tag{5}$$

Here, the columns of $\mathbf{V}$ are the eigenvectors, and each corresponding element in $\mathbf{D}$ is an

eigenvalue. The second eigenvalue is 0, which is the solution to our problem.

$$\mathbf{Ax} = \begin{bmatrix} 3 & 6 & -8 \\ 0 & 0 & 6 \\ 0 & 0 & 2 \end{bmatrix} \begin{bmatrix} -0.8944 \\ 0.4472 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \tag{6}$$

However, $\mathbf{h}$ has a dimension of 9. One-point correspondence provides two constraints. So, if you utilize all the information, you may never encounter this scenario in solving homographies, that is, you never have a square matrix (8×9 or 10×9 matrices, for example).

1.5 Singular Value Decomposition(SVD)

The Singular Value Decomposition (SVD) of a rectangular matrix $\mathbf{A}$ is expressed as:

$$\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^\top. \tag{7}$$

Here, $\mathbf{U}$ is a matrix of column vectors called the "left singular vectors". Similarly, $\mathbf{V}$ is called the "right singular vectors". The matrix $\mathbf{\Sigma}$ is a rectangular matrix with off-diagonal elements 0 (or only diagonal elements are non-zero). Each diagonal element σ_i is called the "singular value," and these are sorted in order of magnitude. In our case, you might see nine values.

If $\sigma_9 = 0$, the system is exactly determined, a homography exists, and all points fit exactly. The corresponding right singular vector in $\mathbf{V}$ is the solution we want.

If $\sigma_9 \geq 0$, the system is over-determined. A homography exists, but not all points fit exactly (they fit in the least-squares error sense). This value represents the goodness of fit. The corresponding right singular vector in V is then the solution we want.

Usually, you will have at least four correspondences. If not, the system is under-determined. We will not deal with those here.

The columns of $\mathbf{U}$ are eigenvectors of $\mathbf{AA}^\top$. The columns of $\mathbf{V}$ are the eigenvectors of $\mathbf{A}^\top\mathbf{A}$. With this fact, the following holds. If $\mathbf{A}$ is not a square matrix, then you can solve $\mathbf{Ah} = 0$ by finding the eigenvector corresponding to the smallest eigenvalue of $\mathbf{A}^\top\mathbf{A}$ (instead of SVD if you want).

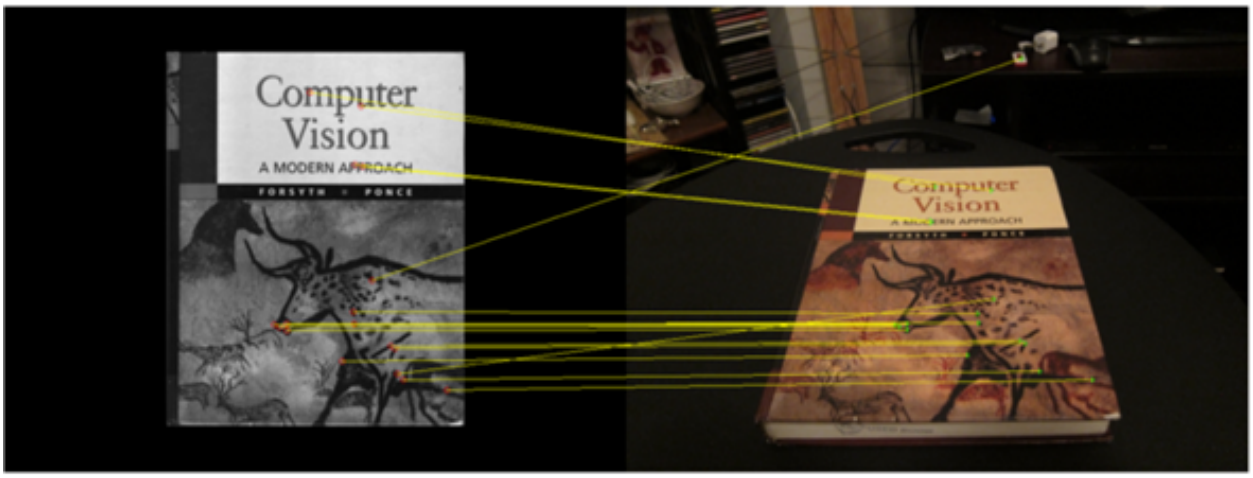

Figure 2: A few matched FAST feature points with the BRIEF descriptor.

Figure 2: The figure of feature and matching.

2 Tasks

2.1 Feature Detection, Description, and Matching

Before finding the homography between an image pair, we need to find corresponding point pairs between two images. But how do we get these points? One way is to select them manually (using cpselect), which is tedious and inefficient. The CV way is to find interest points in the image pair and automatically match them. In the interest of being able to do cool stuff, we will not implement a feature detector or descriptor here but use built-in MATLAB methods. The purpose of an interest point detector (e.g., Harris, SIFT, SURF, etc.) is to find particular salient points in the images around which we extract feature descriptors (e.g., MOPS, etc.). These descriptors try to summarize the content of the image around the feature points in as succinct yet descriptive manner as possible (there is often a trade-off between representational and computational complexity for many computer vision tasks; you can have a very high dimensional feature descriptor that would ensure that you get good matches, but computing it could be prohibitively expensive). Matching, then, is a task of trying to find a descriptor in the list of descriptors obtained after computing them on a new image that best matches the current descriptor. This could be something as simple as the Euclidean distance between the two descriptors or something more complicated, depending on how the descriptor is composed. For this exercise, we shall use the widely used FAST

detector in concert with the BRIEF descriptor.

Implementing the following function:

$$[\mathbf{X}_1, \mathbf{X}_2] = \text{matchPics}(I_1, I_2), \tag{8}$$

where I_1 and I_2 are the images you want to match. $\mathbf{X}_1$ and $\mathbf{X}_2$ are $N \times 2$ matrices containing the x and y coordinates of the matched point pairs. Use the Matlab built-in function *detectFASTFeatures* to compute the features, then build descriptors using the provided *computeBrief* function and finally compare them using the built-in method *matchFeatures*. Use the function *showMatchedFeatures*(I_1, I_2, $\mathbf{X}_1$, $\mathbf{X}_2$, *'montage'*) to visualize your matched points and include the resulting image in your write-up. An example is shown in Fig 2.

For the Python version, you can use the similar functions provided in cv2. You should also detect the FAST feature, extract the brief feature, and use the BF matcher to find correspondence.

There is a threshold parameter on *matchFeatures(...,'MatchThreshold', threshold)* that must be tweaked to see things. The threshold should be 10.0 at default for binary descriptors and 1.0 otherwise. BRIEF is a binary descriptor, but matlab fails to set 10.0 for some reason (use 1.0 instead). Specify the threshold to be 10.0 for BRIEF descriptor. You may also need to increase *MaxRatio* parameter.

We provide you with the function:

$$[\mathbf{D}, \mathbf{X}] = \text{computeBrief}(I, X), \tag{9}$$

which computes the BRIEF descriptor for I. $\mathbf{X}$ in is an $N \times 2$ matrix in which each row represents a feature point's location (x, y). Please note that the number of valid output feature points could be less than the number of input feature points. $\mathbf{D}$ is the corresponding matrix of BRIEF descriptors for the interest points.

2.2 BRIEF and Rotation

Let's investigate how BRIEF works with rotations. Write a script briefRotTest.m that:

- Takes the cv cover.jpg and matches it to itself rotated [Hint: use imrotate] in increments of 10 degrees.

- Stores a histogram of the count of matches for each orientation.
- Plots the histogram using plot

Visualize the feature-matching result at three orientations and include them in your write-up. Explain why you think the BRIEF descriptor behaves this way. Next, use a feature detector *detectSURFFeatures* and *extractFeatures(..., 'Method', 'SURF')* instead and show the results. Does the plot change significantly?

You can use the functions provided in cv2 to extract SURF feature in the python version.

2.3 Homography Computation

Write a function *computeH* that estimates the planar homography from a set of matched point pairs:

$$[\mathbf{H}_{2\to1}] = \text{computeH}(\mathbf{X}_1, \mathbf{X}_2), \tag{10}$$

$\mathbf{X}_1$ and $\mathbf{X}_2$ are $N \times 2$ matrices containing the coordinates (x, y) of point pairs between the two images. H2to1 should be a 3×3 matrix for the best homography from image 2 to image 1 in the least-square sense. You can use eig or svd to get the eigenvectors as described above in this handout. **For at least one pair of images, pick a certain number of points (say randomly 10 points) from the first image and show the corresponding locations in the second image after the homography transformation.**

For the python version, you can use svd function in numpy to compute H matrix.

2.4 Homography Normalization

Normalization improves numerical stability of the solution and you should always normalize your coordinate data. Normalization has two steps:

- Translate the mean of the points to the origin.
- Scale the points so that the average distance to the origin (or you could also try "the largest distance to the origin" to compare) is sqrt(2). This is a linear transformation

and can be written as follows:

$$\begin{aligned} \mathbf{X}_1^{'} &= \mathbf{T}_1\mathbf{X}_1 \\ \mathbf{X}_2^{'} &= \mathbf{T}_2\mathbf{X}_2. \end{aligned} \tag{11}$$

The $\mathbf{X}_1^{'}$ and $\mathbf{X}_2^{'}$ are the normalized homogeneous coordinates of $\mathbf{X}_1$ and $\mathbf{X}_1$. $\mathbf{T}_1$ and $\mathbf{T}_1$ are 3×3 matrices. The homography $\mathbf{H}$ from $\mathbf{X}_2^{'}$ to $\mathbf{X}_1^{'}$ computed by *computeH* satisfies:

$$\mathbf{X}_1^{'} = \mathbf{H}\mathbf{x}_2^{'}. \tag{12}$$

By substituting $\mathbf{X}_1^{'}$ and $\mathbf{X}_2^{'}$ with $\mathbf{T}_1\mathbf{X}_1$ and $\mathbf{T}_2\mathbf{X}_2$, we have:

$$\begin{aligned} \mathbf{T}_1\mathbf{X}_1 &= \mathbf{H}\mathbf{T}_2\mathbf{X}_2 \\ \mathbf{X}_1 &= \mathbf{T}_1^{-1}\mathbf{H}\mathbf{T}_2\mathbf{X}_2. \end{aligned} \tag{13}$$

By following the above procedure, implement the function *computeH_norm*:

$$\mathbf{H}_{2\to1} = \text{computeH\_norm}(\mathbf{X}_1, \mathbf{X}_2). \tag{14}$$

This function should normalize the coordinates in $\mathbf{X}_1$ and $\mathbf{X_2}$ and call *computeH*. **Again, for at least one pair of images, pick a certain number of points (say randomly 10 points) from the first image, and show the corresponding locations in the second image after the homography transformation.**

2.5 RANSAC

The RANSAC algorithm can generally fit any model to noisy data. You will implement it for (planar) homographies between images. Remember that four point-pairs are required at a minimum to compute a homography.

Implementing a function:

$$[\mathbf{H}_{2\to1}^{\text{best}}, \textit{inliers}] = \text{computeH\_ransac}(\mathbf{X}_1, \mathbf{X}_2), \tag{15}$$

where $\mathbf{H}_{2\to1}^{\text{best}}$ should be the homography $\mathbf{H}$ with most inliers found during RANSAC. H will be a homography such that if $\mathbf{x}_2$ is a point in $\mathbf{X}_2$ and $\mathbf{x}_1$ is a corresponding point in $\mathbf{X}_1$, then $\mathbf{x}_1 \equiv \mathbf{H}\mathbf{x}_2$. $\mathbf{X}_1$ and $\mathbf{X}_2$ are $N \times 2$ matrices containing the matched points. Inliers is a vector

Figure 3: Text book

Figure 4: HarryPotterized Text book

Figure 3: The figure of the HarryPotterize.m

of length N with 1 at those matches that are part of the consensus set and 0 elsewhere. Use *computeH_norm* to compute the homography.**For at least one pair of images, visualize the four point-pairs (that produced the most number of inliers) and the inlier matches that were selected by the RANSAC algorithm.**

2.6 HarryPotterizing a Book

You can modify the HarryPotterize_auto.py if needed.

Implementing the scripts *HarryPotterize.m* that:

- Reads cv_cover.jpg, cv_desk.png, and hp_cover.jpg.
- Computes a homography automatically using *MatchPics* and *computeH_ransac*.
- Warps hp_cover.jpg to the dimensions of the cv_desk.png image using the provided *warpH* function.
- At this point, you should notice that although the image is being warped to the correct location, it is not filling up the same space as the book. Implement the function that modifies hp_cover.jpg to fix this issue:

$$I_c = \text{compositeH}(\mathbf{H}_{2\to 1}, \text{template}, I). \tag{16}$$

Figure 5: Rendering video on a moving target
Figure 4: The figure of AR.

2.7 Creating your Augmented Reality application

Now, with the code you have, you can create your own Augmented Reality application. What you're going to do is HarryPotterize the video at source.mov onto the video book.mov. More specifically, you will track the computer vision textbook in each frame of book.mov and overlay each frame of ar source.mov onto the book in book.mov. Please write a script ar.m to implement this AR application and save your result video as *ar.avi* in the *result/* directory. You may use the function *loadVid.m* we provide to load the videos. Your result should be similar to the LifePrint project. You'll be given full credits if you can put the video together correctly, while having strange frames here and it is OK. Also, warped images may fluctuate as keeping the results temporarily consistent is difficult, which is also OK. See Figure 4 for an example frame of what the final video should look like.

Note that the book and the videos we have provided have very different aspect ratios (the ratio of the image width to the image height). You must either use *imresize* or crop each frame to fit onto the book cover. The number of frames may be slightly different, and you do not have to worry about the glitch at the end of the video.

Cropping an image in Matlab is easy. You just need to extract the rows and columns you are interested in. For example, if you want to extract the subimage from point (40, 50) to point (100, 200), your code would look like img cropped = img(50:200, 40:100). In this

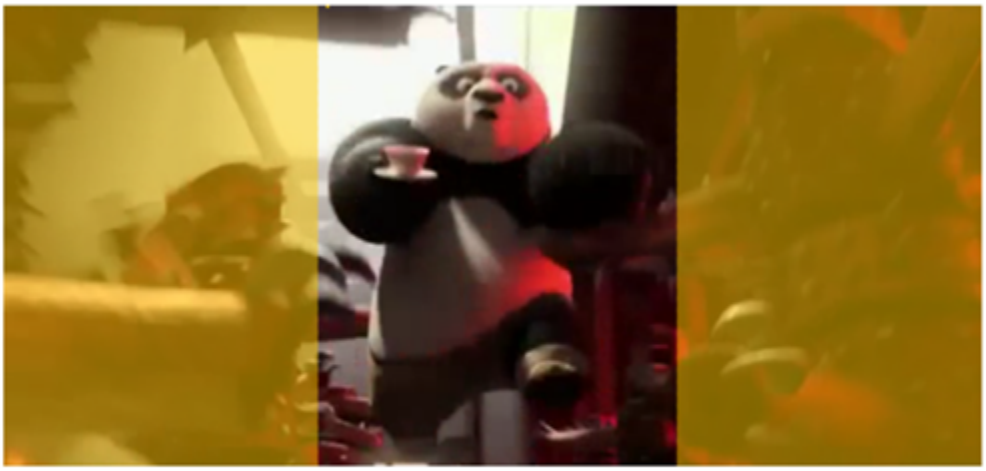

Figure 6: Crop out the yellow regions of each frame to match the aspect ratio of the book

Figure 5: The figure of crop.

project, you must crop that image such that only the central region of the image is used in the final output. See Figure 5 for an example. **You can write your own python to load the image from the video.**

3 Submission

You only need to upload **ONE** zip file containing the code, the README file, and the results. The code should be in Matlab format. Implementing another language will be skipped, and get 0 pts in this project.

Many of the algorithms you will be implementing as part of this project are functions in the Matlab image processing toolbox. You are not allowed to use these functions in this project without permission. However, You may compare your output to the output generated by the image processing toolboxes to ensure you are on the right track. You should write the necessary information in the README file, including the environment, the cmd, the file format, etc. You should make the code in the submission self-contained. The script output should be matched with the file in the results folder.

Cite the paper, GitHub repo, or code url if you use or reference the code online. Please keep academic integrity; plagiarism is not tolerated in this course.

4 Tips

You can access the Matlab with a proper subscription via virtual-barn.

You can use python packages like, cv2, numpy, to finish this project.