Project 2
Augmented Reality with Planar Homographies

Due date: 23:59 Sunday 2/9th (2020)

In this project, you will be implementing an AR application step by step using planar homographies. 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 these to estimate the homography between them. Using this homography you will then warp images and finally implement your own AR application.

1. Instructions

These instructions are also true to all the remaining projects unless indicated. I am repeating them again for project 2 just in case. Please read them carefully.

1. Students are encouraged to discuss projects. However, each student needs to write code and a report all by him/herself. Code should NOT be shared or copied. Do NOT use external code unless permitted.
2. Post questions to Piazza so that everybody can share, unless the questions are private. Please look at Piazza first if similar questions have been posted.
3. Generate a zip or tgz package, and upload to coursys. The package must contain the following in the following layout (they will be different for the other projects but will be similar):
    - {SFUID}
        - {SFUID}.pdf (your write-up, the main document for us to look and grade)
        - matlab
        - Result
4. File paths: Make sure that any file paths that you use are relative and not absolute so that we can easily run code on our end. For instance, you cannot write "imread('/some/absolute/path/data/abc.jpg')". Write "imread('../data/abc.jpg')" instead.
5. If a movie is too large and your file size is bigger than the coursys limit. You can just upload the movie itself as the second component (after zipping the movie as the file name must end with .zip I believe).
6. As indicated below, project 2 will have 17 pts.

2. 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.

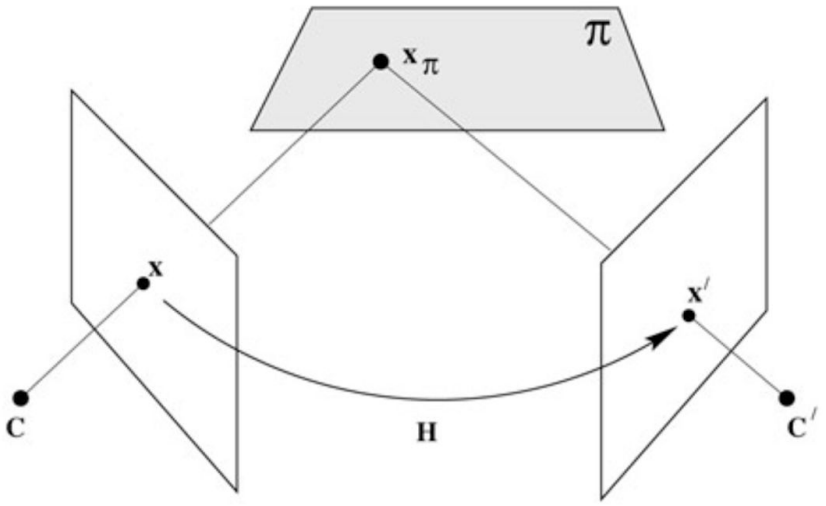

Figure 1: A homography $\mathbf{H}$ links all points $\mathbf{x}_\pi$ lying in plane π between two camera views $\mathbf{x}$ and $\mathbf{x}'$ in cameras C and C' respectively such that $\mathbf{x}' = \mathbf{H}\mathbf{x}$. *[From Hartley and Zisserman]*

There exists a homography H that satisfies equation 1 below, given two 3×4 camera projection matrices P_1 and P_2 corresponding to the two cameras and a plane Π.

$$x_1 \equiv H\, x_2 \qquad (1)$$

The ≡ symbol stands for 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 [xi yi zi]^T , it represents the 2D point [xi/zi yi/zi]^T (called heterogeneous coordinates).

This additional dimension is a mathematical convenience to represent transformations (like translation, rotation, scaling, etc) in a concise matrix form. The ≡ 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.

3. Direct Linear Transform

A very common problem in projective geometry is often of the form x ≡ Ay, where x and y are known vectors, and A is a matrix which contains unknowns to be solved. Given matching points in two images, our homography relationship clearly is an instance of such a problem. Note that the equality holds only up to scale (which means that the set of equations are of the form $x = \lambda Hx'$), 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 solve these kinds of 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's a transform of the problem into a set of linear equation, thus giving it its name.

Let x_1 be a set of points in an image and x_2 be the set of corresponding points in an image taken by another camera. Suppose there exists a homography H such that:

$$x_{i1} \equiv H\, x_{i2} \quad (i \in \{1...N\})$$

where xi1 = [xi1[1] x1i[2] 1]^T are in homogeneous coordinates, xi1 ∈ x1 and H is a 3 × 3 matrix. For each point pair, this relation can be rewritten as

$$A_i h = 0$$

where h is a column vector reshaped from H, and A_i is a matrix with elements derived from the points x_{i1} and x_{i2}. You can solve for h by finding the right null space by Singular Value Decomposition or Eigen Decomposition as described below.

3.1. Eigenvalue Decomposition

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

$$\mathbf{A} = \begin{bmatrix} 3 & 6 & -8 \\ 0 & 0 & 6 \\ 0 & 0 & 2 \end{bmatrix}$$

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

$$V = \begin{bmatrix} 1.0000 & -0.8944 & -0.9535 \\ 0 & 0.4472 & 0.2860 \\ 0 & 0 & 0.0953 \end{bmatrix}$$
$$D = \begin{bmatrix} 3 & 0 & 2 \end{bmatrix}$$

Here, the columns of V are the eigenvectors and each corresponding element in D is an eigenvalue. The second eigenvalue is 0, and hence that is the solution to our problem.

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

However, h has a dimension of 9. One point correspondence provides 2 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 (8x9 or 10x9 matrices for example).

3.2. Singular Value Decomposition

The Singular Value Decomposition (SVD) of a rectangular matrix A is expressed as:

$$A = U\Sigma V^T$$

Here, U is a matrix of column vectors called the “left singular vectors”. Similarly, V is called the “right singular vectors”. The matrix Σ 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 9 values.

- If $\sigma_9 = 0$, the system is exactly-determined, a homography exists and all points fit exactly. The corresponding right singular vector in V is then 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 U are eigenvectors of AA^T . The columns of V are the eigenvectors of A^T A. With this fact, the following holds. If A is not a square matrix, then you can solve Ah=0 by finding the eigenvector corresponding to the smallest eigenvalue of A^T A (instead of SVD if you want).

4. Tasks: Computing Planar Homographies

4.1. Feature Detection, Description, and Matching (3 pts)

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 reimplement 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 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 the purpose of this exercise, we shall use the widely used FAST detector in concert with the BRIEF descriptor.

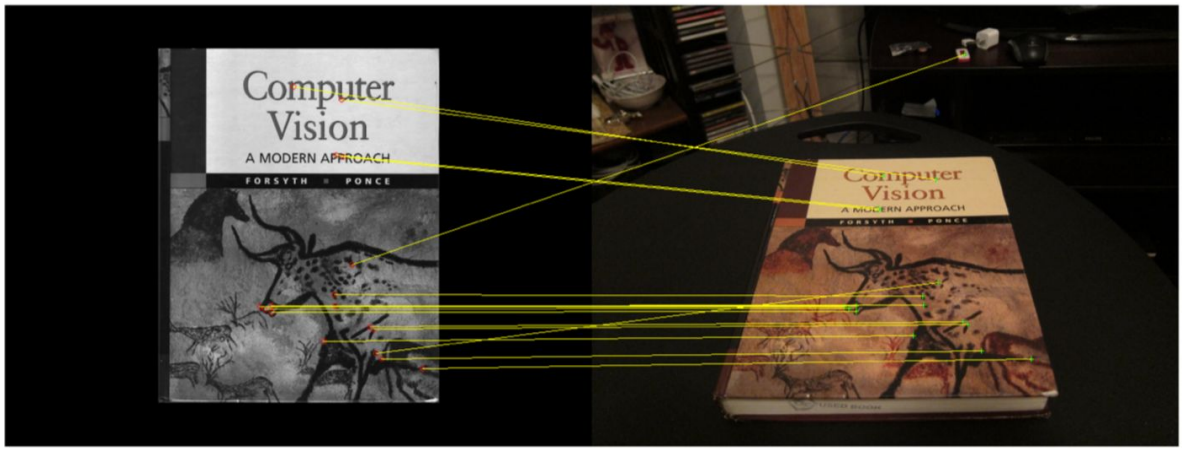

Figure 2: A few matched FAST feature points with the BRIEF descriptor.

Now implement the following function:

[locs1, locs2] = matchPics(I1, I2)

where I1 and I2 are the images you want to match. locs1 and locs2 are N × 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(im1, im2, locs1, locs2, 'montage') to visualize your matched points and include the result image in your write-up. An example is shown in Fig. 2.

There is a threshold parameter on matchFeatures() that must be tweaked to see things:

```
matchFeatures(..., 'MatchThreshold', threshold);
```

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:

[desc, locs] = computeBrief(img, locs in)

which computes the BRIEF descriptor for img. locs in is an N × 2 matrix in which each row represents the location (x, y) of a feature point. Please note that the number of valid output feature points could be less than the number of input feature points. desc is the corresponding matrix of BRIEF descriptors for the interest points.

4.2. BRIEF and Rotations (3 pts)

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 different 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?

4.3. Homography Computation (3 pts)

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

function [H2to1] = computeH(x1, x2)

x1 and x2 are N × 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.

4.4. Homography Normalization (2 pts)

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

1. Translate the mean of the points to the origin.
2. 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:

$$x'_1 = T_1 x_1$$
$$x'_2 = T_2 x_2$$

where x'_1 and x'_2 are the normalized homogeneous coordinates of x_1 and x_2. T_1 and T_2 are 3 × 3 matrices. The homography H from x'_2 to x'_1 computed by computeH satisfies:

$$x'_1 = H x'_2$$

By substituting x'_1 and x'_2 with $T_1 x_1$ and $T_2 x_2$, we have

$$T_1 x_1 = H T_2 x_2$$
$$\mathbf{x}_1 = \mathbf{T}_1^{-1}\mathbf{H}\mathbf{T}_2\mathbf{x}_2$$

By following the above procedure, implement the function computeH_norm:

function [H2to1] = computeH_norm(x1, x2).

This function should normalize the coordinates in x1 and x2 and call computeH(x1, x2). 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.

4.5. RANSAC (2 pts)

The RANSAC algorithm can generally fit any model to noisy data. You will implement it for (planar) homographies between images. Remember that 4 point-pairs are required at a minimum to compute a homography.
Write a function:

function [bestH2to1, inliers] = computeH_ransac(locs1, locs2)

where bestH2to1 should be the homography H with most inliers found during RANSAC. H will be a homography such that if x_2 is a point in locs2 and x_1 is a corresponding point in locs1, then $x_1 \equiv H\, x_2$. locs1 and locs2 are N × 2 matrices containing the matched points. inliers is a vector of length N with a 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 4 point-pairs (that produced the most number of inliers) and the inlier matches that was selected by RANSAC algorithm.

4.6. HarryPotterizing a Book (2 pts)

Write a script HarryPotterize.m that

1. Reads cv_cover.jpg, cv_desk.png, and hp_cover.jpg.
2. Computes a homography automatically using MatchPics and computeH_ransac.
3. Warps hp_cover.jpg to the dimensions of the cv_desk.png image using the provided warpH function.
4. 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:
   function [composite img] = compositeH(H2to1, template, img)

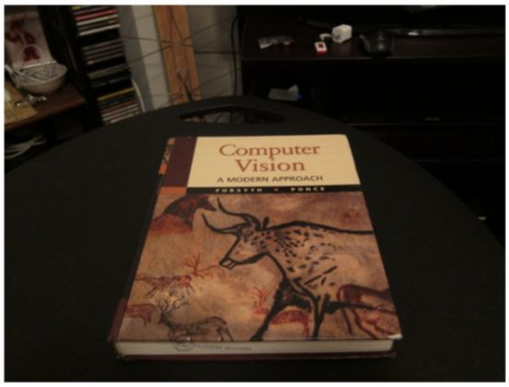

Figure 3: Text book

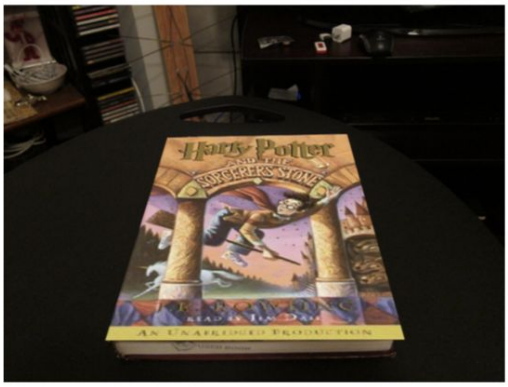

Figure 4: HarryPotterized Text book

5. Creating your Augmented Reality application (2 pts)

Now with the code you have, you're able to create your own Augmented Reality application. What you're going to do is HarryPotterize the video ar source.mov onto the video book.mov. More specifically, you're going to 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 that 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 it is OK to have strange frames here and there. Also warped images may fluctuate as it is difficult to keep the results exactly temporarily consistent, which is also OK. See Figure 5 for an example frame of what the final video should look like.

Figure 5: Rendering video on a moving target

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.

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 project, you must crop that image such that only the central region of the image is used in the final output. See Figure 6 for an example.

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