CSE 491 Introduction to Computer Vision: Project 3
Augmented Reality with Planar Homographies

Instructor: Vishnu Boddeti

Due Date: Tue April 10, 2018 23:59:5$\bar{9}$

In this assignment, you're going to implement an AR application step by step using planar homographies. 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.

Instructions

1. **Integrity and collaboration:** Students are encouraged to work in groups but each student must submit their own work. If you work as a group, include the names of your collaborators in your write up. Code should **NOT** be shared or copied. Please **DO NOT** use external code unless permitted. Plagiarism is strongly prohibited and may lead to failure of this course.

2. If you have any question, please look at Google Classroom first. Other students may have encountered the same problem, and is solved already. If not, post your question there. We will respond as soon as possible.

3. **Write-up:** Items to be included in the write-up are mentioned in each question, and summarized in the Writeup section. Please note that we **DO NOT** accept handwritten scans for your write-up in this assignment. Please type your answers to theory questions and discussions for experiments electronically.

4. **Handout:** The handout zip file contains 3 items. `hw3.pdf` is the assignment handout and `data/` contains image files.

5. **Code:** Stick to the function prototypes mentioned in the handout. This makes verifying code easier for the instructor. If you do want to change a function prototype or add an extra parameter, please talk to the instructor.

6. **Submission:** Your submission for this assignment should be a pdf file `<msu-id.pdf>`, a zip file `<msu-id.zip>`, composed of your write-up, your Python implementations (including helper functions), and your implementations, results for extra credit (optional). Do not hand in the image files we distributed in the handout zip, or any of the generated word map files. However you should hand in the dictionary and vision npz files that you generate.

   Your final upload should have the files arranged in this layout:

- `<msu-id>.pdf`
- `<msu-id>.zip`
  - `python/`
    - `MatchPics.py`
    - `briefRotTest.py`
    - `computeH.py`
    - `computeH_norm.py`
    - `computeH_ransac.py`
    - `HarryPotterize_auto.py`
    - `yourHelperFunctions.py` *(optional)*
  - `result/`
    - `harrypotter.jpg`
  - `ec/` *(optional for extra credit)*
    - `ar.py`
    - `ar.avi`

Please make sure you do follow the submission rule mentioned above before uploading your solution to Google Classroom. Assignments that violate this submission rule will be **penalized 10% of the total score**.

1 Programming

Every script/function you write in this section should be included in the `python/` directory. Please include any result images and outputs in your write-up as well.

1.1 Finding Point Pairs

To find 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, which will take a lot of time. Another way is to find interest points in the image pair and automaically match them. Here, we will use interest point detectors (e.g. FAST detector) to find particular salient points in the images upon which we can extract feature descriptors (e.g. BRIEF). Once we have extracted the interest points, we will use the descriptors to match them automatically.

Q1.1.1 FAST (5 points)

Please search online to learn about the FAST detector and describe briefly how FAST detector finds interest points in your write-up.

Q1.1.2 BRIEF (5 points)

Please search online to learn about the BRIEF descriptor and describe BRIEF descriptor briefly in your write-up.

Q1.1.3 Hamming and Nearest Neighbour (5 points)

Please search online to learn about *Hamming distance* and *Nearest Neighbor*, and describe how Hamming distance and Nearest Neighbor can be used to match interest points with BRIEF descriptors.

Q1.1.4 Feature Matching (10 points)

Please write a function:

```
[locs1, locs2] = MatchPics(I1, I2)
```

`I1` and `I2` are the images you want to match. `locs1` and `locs2` are $N \times 2$ matrices containing the x and y coordinates of the matched point pairs. An example is shown in Figure 1. You may use the Python built-in function `skimage.feature.BRIEF` `skimage.feature.corner_fast` and `skimage.feature.match_descriptors`. Use the function `skimage.feature.plot_matches` to visualize your matched points and include the result image in your write-up. An example is shown in Figure 1.

Q1.1.5 BRIEF and Rotations (15 points)

Now you're going to investigate how BRIEF works with rotations. Write a script `briefRotTest.py` that:

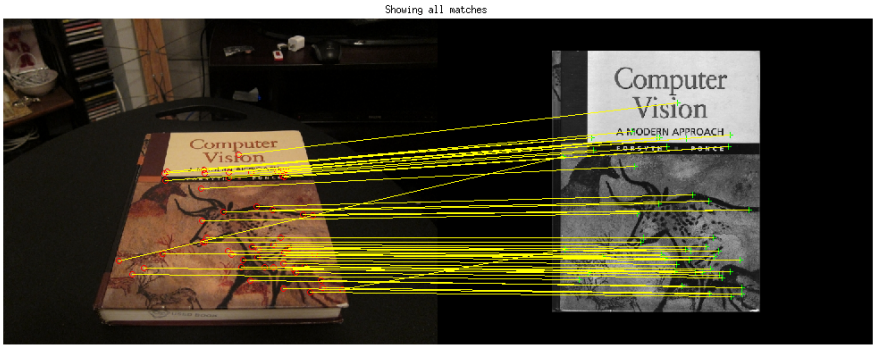

Figure 1: Matched points.

- Take the `cv_cover.jpg` and match it to itself while rotating the second image (hint: use `skimage.transform.rotate`) in increment of 10 degrees.
- Visualize the feature matching result of each pair of images. Please include the result image in your write-up and explain why you think the BRIEF descriptor behaves this way. Can you think of a method to fix this problem? Describe it in your write-up.

1.2 Homography Estimation

Q1.2.1 Computing the Homography (15 points)

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

```
function [H2to1] = computeH(x1, x2)
```

`x1` and `x2` 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 (see Section 3 of this handout for details).

Q1.2.2 Homography with normalization (10 points)

Normalization improves numerical stability of the solution and you should always normalize your coordinate data. In this section, we will do just that.

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)`. Normalization has two steps.

(1) Translate the mean of the points to the origin.

(2) Scale the points so that the largest distance to the origin is $\sqrt{2}$.

Normalization is done separately for `x1` and `x2`. Suppose that the normalization for `x1` and `x2` can be written as linear transforms $\mathbf{T}_1$ and $\mathbf{T}_2$, respectively. Then the final output `H2to1` should be

$$\mathbf{T}_1^{-1}\mathbf{H}\mathbf{T}_2$$

where $\mathbf{H}$ is the homography computed by `computeH`. If you don't understand this, please see the explanation below.

The normalization process consists of translation and scaling the 2D points as described above, which is a linear transformation and can be written as follows:

$$\widetilde{\mathbf{x}}_1 = \mathbf{T}_1\mathbf{x}_1$$
$$\widetilde{\mathbf{x}}_2 = \mathbf{T}_2\mathbf{x}_2$$

where $\widetilde{\mathbf{x}}_1$ and $\widetilde{\mathbf{x}}_2$ are the normalized homogeneous coordinates of $\mathbf{x_1}$ and $\mathbf{x_2}$. $\mathbf{T}_1$ and $\mathbf{T}_2$ are 3×3 matrices.

The homography $\mathbf{H}$ from $\widetilde{\mathbf{x}}_2$ to $\widetilde{\mathbf{x}}_1$ computed by `computeH` satisfies:

$$\widetilde{\mathbf{x}}_1 = \mathbf{H}\widetilde{\mathbf{x}}_2$$

By substituting $\widetilde{\mathbf{x}}_1$ and $\widetilde{\mathbf{x}}_2$ with $\mathbf{T}_1\mathbf{x}_1$ and $\mathbf{T}_2\mathbf{x}_2$, we have:

$$\mathbf{T}_1\mathbf{x}_1 = \mathbf{H}\mathbf{T}_2\mathbf{x}_2$$
$$\mathbf{x}_1 = \mathbf{T}_1^{-1}\mathbf{H}\mathbf{T}_2\mathbf{x}_2$$

Therefore, the homography from $\mathbf{x}_2$ to $\mathbf{x}_1$ computed by `computeH_norm` should be $\mathbf{T}_1^{-1}\mathbf{H}\mathbf{T}_2$ where $\mathbf{H}$ is the homography from $\widetilde{\mathbf{x}}_2$ to $\widetilde{\mathbf{x}}_1$ computed by `computeH`.

Q1.2.3 RANSAC (25 points)

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:

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

where `bestH2to1` should be the homography $\mathbf{H}$ with most inliers found during RANSAC. $\mathbf{H}$ will be a homography such that if $\mathbf{x}_2$ is a point in `locs2` and $\mathbf{x}_1$ is a corresponding point in `locs1`, then $\mathbf{x}_1 \equiv \mathbf{H}\mathbf{x}_2$. `locs1` and `locs2` are $N \times 2$ 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.

Q1.2.4 Putting it together (10 points)

Write a script `HarryPotterize_auto.py` that

1. Reads `cv_cover.jpg`, `cv_desk.jpg`, and `hp_cover.jpg`.
2. Computes a homography automatically using `MatchPics` and `computeH_ransac`.
3. Warps `hp_cover.jpg` onto the cover of the book in `cv_desk.jpg` as in in Figure 2 using `skimage.transform.warp` and `skimage.transform.ProjectiveTransform`.
4. Include your result in the Q1.2.4 section of your write-up and in the `result/` folder. An example is shown in figure 2 and figure 3.

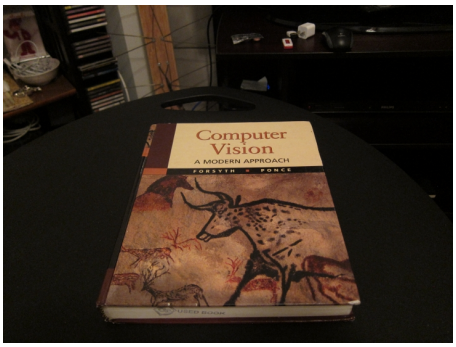

Figure 2: Text book

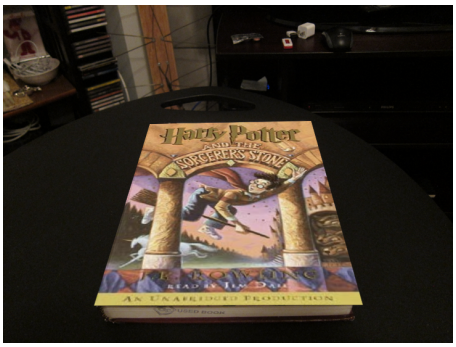

Figure 3: HarryPotterized Text book

2 Extra Credits

2.1 Create your AR

Q1.3.1 Incorporating video (40 points)

Now with the code you have, you're able to create you own Augmented Reality application with `cv_cover.jpg`, `book.mov` and `ar_source.mov`. What you're going to do is to HarryPoterize the video `ar_source.mov` onto the video `book.mov`. More specifically, you're going to track the computer vision text book 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.py` to implement this AR application and save your result video as ar.avi and put it in the `result/` directory. You may use `MoviePy` or `Imageio` to load the videos. Your result should be similar to the LifePrint project: `https://www.indiegogo.com/projects/lifeprint-photos-that-come-to-life-in-your-hands#/`. You'll be given full credits if you can put the video together correctly.

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 resize or crop each frame to fit onto the book cover.

Cropping an image in Numpy 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.

Also, the video `book.mov` only has translation of objects. If you want to account for rotation of objects, scaling, etc, you would have to pick a better feature point representation (like ORB, `skimage.feature.ORB`).

Figure 4: Rendering video on a moving target

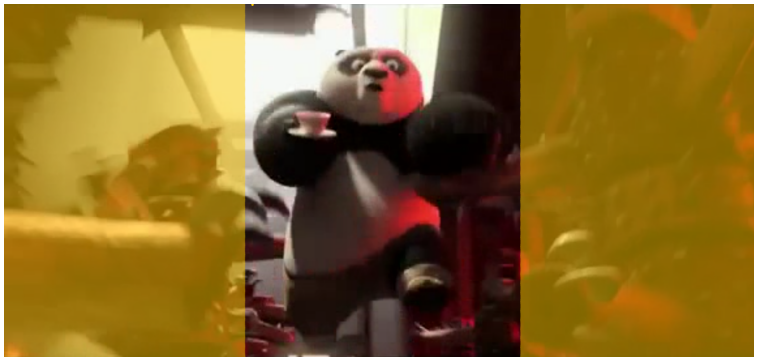

Figure 5: Crop out the yellow regions of each frame to match the aspect ratio of the book

3 Using SVD to calculate the homography

A homography $\mathbf{H}$ transforms one set of points (in homogenous coordinates) to another set of points. In this project, we know the corresponding point coordinates and need to calculate the homography. In this section, we will look at how to solve equations like $\mathbf{Ax} = \mathbf{0}$.

3.1 Eigenvectors as solutions

One way to solve $\mathbf{Ax} = \mathbf{0}$ is to calculate the eigenvalues and eigenvectors of $\mathbf{A}$. The eigenvector corresponding to $\mathbf{0}$ is the answer for this. Consider this example:

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

Using the Numpy function `np.linalg.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 $\mathbf{V}$ are the eigenvectors and each corresponding element in $\mathbf{D}$ it's eigenvalue. We notice that there is an eigenvalue of 0. The eigenvector corresponding to this is the solution for the equation $\mathbf{Ax} = \mathbf{0}$.

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

3.2 Using SVD for least square error

In computer vision, you will very often encounter equations in this form ($\mathbf{Ax} = \mathbf{0}$). Such equations can also be solved using Singular Value Decomposition (SVD). Computing the SVD of $\mathbf{A}$ gives us:

$$\mathbf{A} = U\Sigma V^T$$

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 diagonal matrix. Each diagonal element σ_i is called the "singular value" and these are sorted in order of magnitude. In our case, it is a 9×9 matrix.

- If $\sigma_9 = 0$, the system is *exactly-determined*, a homography exists and all points fit exactly.
- 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.

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

The columns of U are eigenvectors of $\mathbf{A}\mathbf{A}^T$. The columns of V are the eigenvectors of $\mathbf{A}^T\mathbf{A}$. We can use this fact to solve for $\mathbf{h}$ in the equation $\mathbf{Ah} = \mathbf{0}$.

Using this knowledge, let us reformulate our problem of solving $\mathbf{Ax} = \mathbf{0}$. We want to minimize the error in solution in the least-squares sense. The sum-squared error can be written as:

$$\begin{aligned} f(\mathbf{h}) &= \frac{1}{2}(\mathbf{Ah} - \mathbf{0})^T(\mathbf{Ah} - \mathbf{0}) \\ &= \frac{1}{2}(\mathbf{Ah})^T(\mathbf{Ah}) \\ &= \frac{1}{2}\mathbf{h}^T\mathbf{A}^T\mathbf{Ah} \end{aligned}$$

Minimizing this error with respect to $\mathbf{h}$, we get:

$$\begin{aligned} \frac{d}{d\mathbf{h}}f &= 0 \\ \implies \frac{1}{2}(\mathbf{A}^T\mathbf{A} + (\mathbf{A}^T\mathbf{A})^T)\mathbf{h} &= 0 \\ \mathbf{A}^T\mathbf{Ah} &= 0 \end{aligned}$$

This implies that the value of $\mathbf{h}$ equals the eigenvector corresponding to the zero eigenvalue (or closest to zero in case of noise). Thus, we choose the smallest eigenvalue of $\mathbf{A}^T\mathbf{A}$, which is σ_9 in Σ and the least-squares solution to $\mathbf{Ah} = \mathbf{0}$ is the the corresponding eigenvector (in column 9 of the matrix $\mathbf{V}$).