Carnegie Mellon University
16-720: Computer Vision
**Homework 2: Augmented Reality with Planar Homographies**

- **Due date.** Please refer to course schedule for the due date for **HW2**.
- **Gradescope submission.** You will need to submit both (1) a `YourAndrewName.pdf` and (2) a `YourAndrewName.zip` file containing your code, either as standalone python (`.py` or colab notebooks `.ipynb`). Remember you *must* use Gradescope's functionality to mark pages in your PDF that answer individual questions; if not, you will risk loosing points!.
- **Suggestions for creating a PDF.** We suggest you create a PDF by `print`ing your colab notebook from your browser. However, you are responsible for making sure all your code is visible and not cut off. Long lines of code can print poorly; we suggest you add a backslash to break a single long line into multiple lines. You also may wish to look this video for alternate pathways to convert notebooks to PDFs: https://youtu.be/-Ti9Mm21uVc?si=bo4kHfp2BoJvPpZI. In some cases, you may wish to download image or screengrabs and explicitly append them to your PDF using online tools such as https://combinepdf.com/. You may also find it useful to look at this post: https://askubuntu.com/questions/2799/how-to-merge-several-pdf-files.

The starter code can be found at the course gdrive folder (you will need your andrew account to access): https://drive.google.com/drive/folders/1Uqpry1qXKRpVTOv_W9YbKiGvxY1re-hV?usp=sharing

We recommend editing and running your code in Google Colab, although you are welcome to use your local machine instead.

**Please remember to list your collaborators in your report.**

# Overview

In this assignment, you will be implementing a panorama 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 panorama application.

# 1 Preliminaries

## 1.1 The Direct Linear Transform

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

$$\mathbf{x_1^i} \equiv \mathbf{H}\mathbf{x_2^i} \quad (i \in \{1 \dots N\})$$

where $\mathbf{x_1^i} = [x_1^i \;\; y_1^i \;\; 1]$ are in homogenous coordinates, $\mathbf{x_1^i} \in \mathbf{x_1}$ and $\mathbf{H}$ is a $3 \times 3$ matrix. The $\equiv$ symbol stands for identical to. For each point pair, this relation can be rewritten as

$$\mathbf{A_i h} = 0$$

where $\mathbf{h}$ is a column vector reshaped from $\mathbf{H}$, and $\mathbf{A_i}$ is a matrix with elements derived from the points $\mathbf{x_1^i}$ and $\mathbf{x_2^i}$. This can help calculate $\mathbf{H}$ from the given point correspondences.

**Q1.1.1 (3 points):** How many degrees of freedom does $\mathbf{h}$ have?

**Q1.1.2 (2 points):** How many pairs of points are required to solve $\mathbf{h}$?

**Q1.1.3 (5 points):** Derive $\mathbf{A_i}$.

**Q1.1.4 (5 points):** When solving $\mathbf{Ah} = 0$, in essence you're trying to find the $\mathbf{h}$ that exists in the null space of $\mathbf{A}$. What that means is that there would be some non-trivial solution for $\mathbf{h}$ such that the product $\mathbf{Ah}$ turns out to be 0. What will be a trivial solution for $\mathbf{h}$? Is the matrix $\mathbf{A}$ full rank? Why/Why not? What impact will it have on the singular values (i.e. eigenvalues of $\mathbf{A}^\top\mathbf{A}$)?

### 1.2 Homography Theory Questions

**Q1.2.1 (5 points): Homography under rotation** Prove that there exists a homography $\mathbf{H}$ that satisfies $\mathbf{x_1} \equiv \mathbf{Hx_2}$, given two cameras separated by a pure rotation. That is, for camera 1, $\mathbf{x_1} = \mathbf{K_1}[\mathbf{I}\ \ \mathbf{0}]\mathbf{X}$ and for camera 2, $\mathbf{x_2} = \mathbf{K_2}[\mathbf{R}\ \ \mathbf{0}]\mathbf{X}$. Note that $\mathbf{K_1}$ and $\mathbf{K_2}$ are the $3 \times 3$ intrinsic matrices of the two cameras and are different. $\mathbf{I}$ is $3 \times 3$ identity matrix, $\mathbf{0}$ is a $3 \times 1$ zero vector and $\mathbf{X}$ is a point in 3D space. $\mathbf{R}$ is the $3 \times 3$ rotation matrix of the camera.

**Q1.2.2 (5 points): Understanding homographies under rotation** Suppose that a camera is rotating about its center $\mathbf{C}$, keeping the intrinsic parameters $\mathbf{K}$ constant. Let $\mathbf{H}$ be the homography that maps the view from one camera orientation to the view at a second orientation. Let $\theta$ be the angle of rotation between the two. Show that $\mathbf{H}^2$ is the homography corresponding to a rotation of $2\theta$. Please limit your answer within a couple of lines. A lengthy proof indicates that you're doing something too complicated (or wrong).

## 2 Computing Planar Homographies

### 2.1 Feature Detection 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, 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, you will not reimplement a feature detector or descriptor here by yourself. The logic for these can be found in the Initialization block.**.

The purpose of an interest point detector (e.g. Harris) is to find particular salient points in the images around which we extract feature descriptors (e.g. SIFT). These descriptors try to summarize the content of the image around the feature points in as succint 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.

**Q2.1.1 (5 points): FAST Detector** How is the FAST detector different from the Harris corner detector that you've seen in the lectures? Can you comment on its computational performance compared to the Harris corner detector? Reference links: Original Paper [2], OpenCV Tutorial

**Q2.1.2 (5 points): BRIEF Descriptor** How is the BRIEF descriptor different from the filterbanks you've seen in the lectures? Could you use any one of those filter banks as a descriptor?

**Q2.1.3 (5 points): Matching Methods** The BRIEF descriptor belongs to a category called binary descriptors. In such descriptors the image region corresponding to the detected feature point is represented as a binary string of 1s and 0s. A commonly used metric used for such descriptors is called the Hamming distance. Please search online to learn about Hamming distance and Nearest Neighbor, and describe how they can be used to match interest points with BRIEF descriptors. What benefits does the Hamming distance distance have over a more conventional Euclidean distance measure in our setting?

**Q2.1.4 (10 points): Feature Matching**

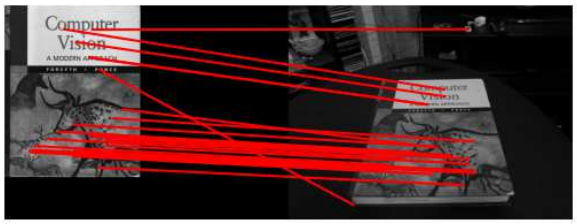


Figure 1: A few matched FAST feature points with the BRIEF descriptor.

Please implement the function `matchPics()`:

```
matches, locs1, locs2 = matchPics(I1, I2, ratio, sigma)
```

where `I1` and `I2` are the images you want to match, `ratio` is the ratio for the BRIEF feature descriptor, and `sigma` is threshold for corner detection using FAST feature detector. `locs1` and `locs2` are $N \times 2$ matrices containing the $x$ and $y$ coordinates of the matched point pairs. `matches` is a $p \times 2$ matrix where the first column is indices into descriptor of features in `I1` and similarly second column contains indices related to `I2`. Use the provided helper function `corner_detection()` to compute the features, then build descriptors using `computeBrief()`, and finally compare them using `briefMatch()`.

```
locs = corner_detection(img, sigma)
desc, locs = computeBrief(img, locs)
matches = briefMatch(desc1, desc2, ratio)
```

`locs` is an $N \times 2$ 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.

Then, implement the function `displayMatched()` using `matchPics()` and the provided function `plotMatches()`. Call `displayMatched()` to visualize your matched points and include the result in your report.

```
plotMatches(img1, img2, matches, locs1, locs2)
```

```
displayMatched(I1, I2, ratio, sigma)
```

The number of matches between the images may vary based on the `ratio` and `sigma` values. You can vary these to get the best results. The example shown in Fig. 1 is with `ratio`= 0.7 and `sigma`= 0.15.

**Q2.1.5 (10 points): Feature Matching and Parameter Tuning**

Conduct a small ablation study by calling `displayMatched` with various `sigma` and `ratio` values. Include the figures displaying the matched features with various parameters in your writeup, and explain the effect of these two parameters respectively.

**Q2.1.6 (10 points): BRIEF and Rotations**

Let's investigate how BRIEF works with rotations. Implement the function `briefRot()` that :

- Takes the `cv_cover.jpg` and matches it to itself rotated from 0 to 360 degrees in increments of 10 degrees. [*Hint*: use `scipy.ndimage.rotate`]
- Stores the degrees of rotation and the number of matches at each degree which will be later used to plot a histogram.
- Visualizes the feature matching results at at least three different orientations.

```
briefRot(min_deg, max_deg, deg_inc, ratio, sigma, filename)
```

Visualize the feature matching results at three different orientations and include them in your write-up. As well, plot the histogram. Explain why you think the BRIEF descriptor behaves this way.

**Debugging Tips:** To save time debugging, try reducing the number of iterations (increase min_deg or decrease max_deg) to make sure your code runs correctly. When it is working, you can then run it for all 360 degrees. Because this function can be time consuming, once complete the number of matches for each degree are saved as a pickle file as brief_rot_test.pkl. It is recommended you save this file and re-upload it (if running on Colab) between sessions to save time from having to run it again if you just want to visualize the histogram. This file is later used by dispBriefRotHist to visualize the count of matches histogram.

**Q2.1.7 (Extra Credit): Improving Performance** The extra credit opportunities described below are optional and provide an avenue to improve the performance of the techniques developed above.

**Q2.1.7.1 (Extra Credit - 5 points):** As we have seen, BRIEF is not rotation invariant. Design a simple fix to solve this problem using the tools you have developed so far (think back to edge detection and/or Harris corner's covariance matrix). Implement `briefRotInvEc`. You are not allowed to use any additional OpenCV or Scikit-Image functions. But you are allowed to define additional helper functions and modify the input parameters. Demonstrate the effectiveness of your algorithm on image pairs related by large rotation. Visualize the feature matching results at three different orientations and plot the histogram. Compare the histogram to that without rotation invariance. Explain your design decisions and how you selected any parameters that you use.

**Debugging Tips:** Similary to Q2.1.6, try reducing the number of iterations to make sure the code runs correctly. Once complete, the number of matches for each degree are aved as ec_brief_rot_inv_test.pkl. It is recommended you save this file and re-upload it (if running on Colab) between sessions to save time from having to run it again if you just want to visualize the histogram. This file is later used by dispBriefRotHist to visualize the count of matches histogram.

**Q2.1.7.2 (Extra Credit - 5 points):** This implementation of BRIEF has some scale invariance, but there are limits. What happens when you match a picture to the same picture at half the size? Look to section 3 of [Lowe2004 [1]], for a technique that will make your detector more robust to changes in scale. Implement a fix to solve this problem. Implement `briefScaleInvEc`. You are not allowed to use any additional OpenCV or Scikit-Image functions (except for when rescaling the test images). But you are allowed to define additional helper functions and modify the input parameters. Demonstrate the effectiveness of your algorithm by evaluating it on several test images. Explain your design decisions and how you selected the parameters that you use.

## 2.2 Homography Computation

### Q2.2.1 (15 points): Computing the Homography

Implement the function `computeH` that estimates the planar homography from a set of matched points.

```
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 $3 \times 3$ matrix for the best homography from image 2 to image 1 in the least-squares sense. The `numpy.linalg` function `eig()` or `svd()` will be useful to get the eigenvectors (see Section 1 of this handout for details).

### Q2.2.2 (10 points): Homography Normalization

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 largest distance to the origin is $\sqrt{2}$

This is a linear transformation and can be written as follows:

$$\tilde{\mathbf{x}}_1 = \mathbf{T_1}\mathbf{x_1}$$
$$\tilde{\mathbf{x}}_2 = \mathbf{T_2}\mathbf{x_2}$$

where $\tilde{\mathbf{x}}_1$ and $\tilde{\mathbf{x}}_2$ are the normalized homogeneous coordinates of $\mathbf{x_1}$ and $\mathbf{x_2}$. $\mathbf{T_1}$ and $\mathbf{T_2}$ are $3 \times 3$ matrices.
The homography $\mathbf{H}$ from $\tilde{\mathbf{x}}_2$ to $\tilde{\mathbf{x}}_1$ computed by `computeH` satisfies

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

By substituting $\tilde{\mathbf{x}}_1$ and $\tilde{\mathbf{x}}_2$ with $\mathbf{T_1}\mathbf{x_1}$ and $\mathbf{T_2}\mathbf{x_2}$, we have:

$$\mathbf{T_1}\mathbf{x_1} = \mathbf{H}\mathbf{T_2}\mathbf{x_2}$$
$$\mathbf{x_1} = \mathbf{T_1}^{-1}\mathbf{H}\mathbf{T_2}\mathbf{x_2}$$

Implement the function `computeH_norm`:

```
H2to1 = computeH_norm(x1, x2)
```

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

### Q2.2.3 (25 points): Implement RANSAC

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, best_inliers = computeH_ransac(locs1, locs2, max_iters, inlier_tol)
```

where `locs1` and `locs2` are $N \times 2$ matrices containing the matched points. `max_iters` is the number of iterations to run RANSAC for, and `inlier_tol` is the tolerance value for considering a point to be an inlier. `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 $\mathbf{x_1} \equiv \mathbf{H}\mathbf{x_2}$ . `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.

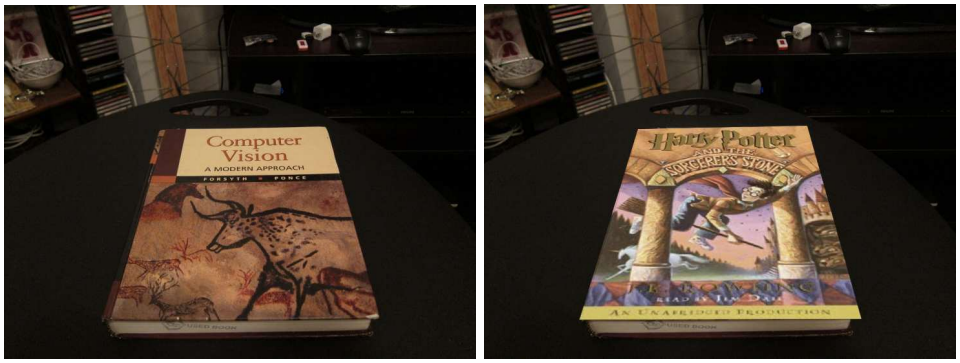


Figure 2: An ordinary textbook (left). Harry-Potterized book (right)

### Q2.2.4 (10 points): Automated Homography Estimation and Warping

Implement the function `compositeImage()` that composes a warped template image on top of another image. You may want to use the OpenCV function `cv2.warpPerspective`. Then, implement `warpImage()` that

- Reads `cv_cover.jpg`, `cv_desk.png`, and `hp_cover.jpg`.
- Computes a homography automatically using `matchPics()` and `computeH_ransac`() (remember to adjust for the size different between `cv_cover.jpg` and `hp_cover.jpg`.
- Uses `compositeImage()` to compose this warped image with the desk image as in in Figure 2

```
composite_img = compositeH(H2to1, template, img)
 warpImage(ratio, sigma, max_iters, inlier_tol)
```

The example shown in Figure 2 is with **ratio**=0.7, **sigma**=0.15, **max_iters**=600, and **inlier_tol**=1.0.

### Q2.2.5 (10 points): RANSAC Parameter Tuning

Just like how we tune parameters for feature matching, there are two tunable parameters in RANSAC as well: max_iters and inlier_tol. Conduct a small ablation study by calling `warpImage()` with various `max_iters` and `inlier_tol` values. Plot the result images and explain the effect of these two parameters respectively.

# 3 Create a Simple Panorama

## 3.1 Create a panorama (10 points)

Implement the function `createPanorama()` which computes the homography between two images separated by a pure rotation and stitches them together. Then, take two pictures with your own camera, separated by a pure rotation as best as possible, call `createPanorama()`, and visualize the results. Be sure that objects in the images are far enough away so that there are no parallax effects. You can use the python module `cpselect` to select matching points on each image or some automatic method. We have provided two images for you to get started (`data/pano_left.jpg` and `data/pano_right.jpg`). Please use your own images when submitting this project. Include original images and the final panorama result image in the write-up. See Figure 3 below for an example.

```
panorama_im = createPanorama(left_im, right_im, ratio, sigma, max_iters, inlier_tol)
```

Figure 3: Original Image 1 (top left). Original Image 2 (top right). Panorama (bottom).

The example in Figure 3 is with **ratio**=0.7, **sigma**=0.15, **max_iters**=600, and **inlier_tol**=1.0.

# 4 FAQs

*Credits: Cherie Ho*

1. In `matchPics`, `locs` stands for pixel locations. `locs1` and `locs2` can have different sizes, since matches gives the mapping between the two for corresponding matches. We use `skimage.feature.match` descriptors (API) to calculate the correspondences.
2. Normalized homography - The function `computeH` norm should return the homography `H` between unnormalized coordinates and not the normalized homography `H` norm. As mentioned in the writeup, you can use the following steps as a reference:

$$\mathbf{H}_{\text{norm}} = \texttt{computeH}(\texttt{x1\_normalized}, \texttt{x2\_normalized})$$
$$\mathbf{H} = \mathbf{T}_1^{-1} @ \mathbf{H}_{\text{norm}} @ \mathbf{T}_2$$

3. The `locs` produced by matchPics are in the form of [`row`, `col`], which is (y,x) in coordinates. Therefore, you should first swap the columns returned by matchPics and then feed into Homography estimation.
4. Note that the third output `np.linalg.svd` is `vh` when computing homographies.
5. When debugging homographies, it is helpful to visualize the matches, and checking homographies with the same image. If there is not enough matches, try tuning the parameters.

# 5 Helpful Concepts

*Credits: Jack Good*

- **Projection vs. Homography:** A projection, usually written as $\mathbf{P}_{3\times4}$, maps homogeneous 3D world coordinates to the view of a camera in homogeneous 2D coordinates. A planar homography, usually written as $\mathbf{H}_{3\times3}$, under certain assumptions, maps the view of one camera in 2D homogeneous coordinates to the view of another camera in 2D homogeneous coordinates.
- **Deriving homography from projections:** When deriving a homography from projections given assumptions about the world points or the camera orientations, make sure to include the intrinsic matrices of the two cameras, $\mathbf{K}_1$ and $\mathbf{K}_2$, which are $3 \times 3$ and invertible, but generally cannot be assumed to be identity or diagonal. Note the rule for inverting matrix products: $(AB)^{-1} = B^{-1}A^{-1}$. When this rule is applied, even when both views are the same camera and $\mathbf{K} = \mathbf{K}_1 = \mathbf{K}_2$, $\mathbf{K}$ is still part of H and does not cancel out.

# References


[1] D. G. Lowe. Distinctive image features from scale-invariant keypoints. *Int. J. Comput. Vision*, 60(2):91–110, Nov. 2004. ISSN 0920-5691. doi: 10.1023/B:VISI.0000029664.99615.94. URL http://dx.doi.org/10.1023/B:VISI.0000029664.99615.94.

[2] E. Rosten, R. Porter, and T. Drummond. Faster and better: A machine learning approach to corner detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 32(1):105–119, 2010. doi: 10.1109/TPAMI.2008.275.