1️⃣ HW1 Planar Homographies

Due Date: April 14, 23:59

In this assignment, 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.

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, including the names of your collaborators in your write-up. Code **MUST 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. **Questions:** If you have any questions, please look at Blackboard first. Other students may have encountered the same problem, and it may be solved already. If not, post your question on the discussion board. The teaching staff will respond as soon as possible.
3. **Write-up:** Your write-up should mainly consist of three parts, your answers to theory questions, some pictures showing your results of each step, and the discussions for experiments. 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. **Compliance with instructions**: Please comply with the function prototypes mentioned in the handout. This makes verifying code easier for the TAs.
5. **Submission:** Create a zip file, *<name-id>.zip*, composed of your write-up, your Python implementations (including helper functions), and your implementations, results for extra credits. Please make sure to remove any other temporary files you've generated. Your final upload should have the files arranged in this layout:

```
•   <name_id>.zip
    –   <name_id>/
        *   <name_id>.pdf
        *   python/
            ·   MatchPics.py
            ·   planarH.py
            ·   loadVid.py
            ·   helper.py
            ·   Q5_featureMatching.py
            ·   Q6_briefRotTest.py
            ·   Q8_computeH.py
            ·   Q9_computeH_norm.py
            ·   Q10_RANSAC.py
            ·   Q11_HarryPotterize.py
            ·   Q12_ar.py
            ·   Q13_Panorama.py
            ·   Q14_Multi_panorama.py
            ·   Q15_Degenerate.py
            ·   yourHelperFunctions.py (optional)
```

```
        *   result/
            ·   ar.avi
            ·   yourpanoImage (the images required for generating the results in Q13)
            ·   panorama_2.png
            ·   panorama_3.png
            ·   panorama_4.png
        *   data/
            ·   ar_source.mov
            ·   book.mov
            ·   cv_cover.jpg
            ·   cv_desk.png
            ·   hp_cover.jpg
            ·   image1.png
            ·   image2.png
            ·   image3.png
            ·   image4.png
```

Please make sure you do follow the submission rules mentioned above before uploading your zip file to Blackboard. Assignments that violate this submission rule will be **penalized by up to 10% of the total score**.

6. **File paths:** Please make sure that any file paths that you use are relative and not absolute.

   Not cv2.imread('/name/Documents/subdirectory/hw2/data/xyz.jpg') but cv2.imread('../data/xyz.jpg').

I. Homographies

1.1 Planar Homographies as a Warp

Recall that 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, through a homography $\mathbf{H}$:

$$\mathbf{x_1} \equiv \mathbf{H}\mathbf{x_2}$$

The $\equiv$ symbol stands for *identical to*. The points $\mathbf{x_1}$ and $\mathbf{x_2}$ are in *homogeneous coordinates*, which means they have an additional dimension. If $\mathbf{x_1}$ is a 3D vector $\begin{bmatrix} x_i & y_i & w_i \end{bmatrix}^\top$, it represents the 2D point $\begin{bmatrix} \frac{x_i}{w_i} & \frac{y_i}{w_i} \end{bmatrix}^\top$ (called *inhomogeneous* or *heterogeneous coordinates*). This additional dimension is a mathematical convenience to represent transformations (like translation, rotation, scaling, etc) in a concise matrix form. The $\equiv$ means that the equation is correct to a scaling factor.

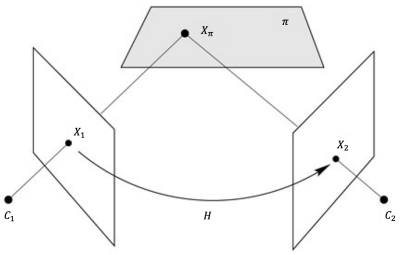

Figure 1: A homography $\mathbf{H}$ links all points $\mathbf{x}_\pi$ lying in plane between two camera views $\mathbf{x}_1$ and in cameras C_1 and C_2 respectively such that $\mathbf{x}_2 = \mathbf{H}\mathbf{x}_1$.
[From Hartley and Zisserman]

1.2 The Direct Linear Transform

A very common problem in projective geometry is often of the form $\mathbf{x} \equiv \mathbf{A}\mathbf{y}$, where $\mathbf{x}$ and $\mathbf{y}$ are known vectors, and $\mathbf{A}$ is a matrix that 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 $\mathbf{x} = \lambda\mathbf{H}\mathbf{x}'$), 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 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
$\mathbf{H}$ matrix, it's a *transform* of the problem into a set of *linear equations*, thus giving it its name.

Q1 Correspondences (15 pts)

Let $\mathbf{x_1}$ be a set of points 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 $\mathbf{x}_1^i = \begin{bmatrix} x_1^i & y_1^i & 1 \end{bmatrix}^\top$ are in homogeneous coordinates, $\mathbf{x}_1^i \in \mathbf{x}_1$ and H is a 3 × 3 matrix. For each point pair, this relation can be rewritten as

$$\mathbf{A}_i\mathbf{h} = 0$$

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.

In your write-up

1. How many degrees of freedom does $\mathbf{h}$ have? (3 pts)
2. How many point pairs are required to solve $\mathbf{h}$? (2 pts)
3. Derive $\mathbf{A}_i$. (5 pts)
4. When solving $\mathbf{A}\mathbf{h} = 0$, 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 product $\mathbf{A}\mathbf{h}$ 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 eigenvalues? What impact will it have on the eigenvectors? (5 pts)

1.3 Using Matrix Decompositions to calculate the homography

A homography $\mathbf{H}$ transforms one set of points (in homogeneous coordinates) to another set of points. In this project, we will obtain the corresponding point coordinates using feature matches and will then need to calculate the homography. You have already derived that $\mathbf{A}\mathbf{x} = 0$ in **Question 1.2.** In this section, we will look at how to solve such equations using two approaches, either of which can be used in the subsequent assignment questions.

1.4 Eigenvalue Decomposition

One way to solve $\mathbf{A}\mathbf{x} = 0$ is to calculate the eigenvalues and eigenvectors of $\mathbf{A}$. The eigenvector corresponding to 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.linalg* function *eig*, we get the following eigenvalues and eigenvectors:

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

Here, the columns of $\mathbf{V}$ are the eigenvectors, and each corresponding element in $\mathbf{D}$ is an eigenvalue. We notice that there is an eigenvalue of 0. The eigenvector corresponding to this is the solution for the equation $\mathbf{Ax} = 0$.

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

1.5 Singular Value Decomposition

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

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

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 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 $\sigma_9 > 0$, 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 *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 $\mathbf{A}^\top\mathbf{A}$. We can use this fact to solve for $\mathbf{h}$ in the equation $\mathbf{Ah} = 0$. Using this knowledge, let us reformulate our problem of solving $\mathbf{Ax} = 0$. We want to minimize the error in solution in the least-squares sense. Ideally, the product $\mathbf{Ah}$ should be 0. Thus the sum-squared error can be written as:

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

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

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

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 $\mathbf{A}^\top\mathbf{A}$, which is σ_9 in Σ and the least-squares solution to $\mathbf{Ah} = 0$ is the corresponding eigenvector (in column 9 of the matrix $\mathbf{V}$).

II. 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 other 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 python modules. 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 an 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 the purpose of this exercise, we shall use the widely used FAST detector in concert with the BRIEF descriptor.

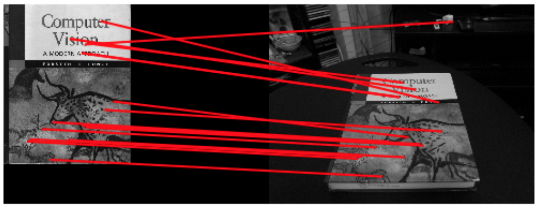

Figure 2: A few matched FAST feature points with the BRIEF descriptor.

Q2 FAST Detector (7 pts)

In your write-up

1. How is the FAST detector different from the Harris corner detector that you've seen in the lectures? You will probably need to look up the FAST detector online. (4 pts)
2. Can you comment on its computational performance compared with the Harris corner detector? (3 pts)

Q3 BRIEF Descriptor (7 pts)

In your write-up

1. How is the BRIEF descriptor different from the filterbanks you've seen in the lectures? (4 pts)
2. Could you use any one of those filterbanks as a descriptor? (3 pts)

Q4 Matching Methods (7 pts)

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*.

In your write-up

1. What benefits does the Hamming distance have over a more conventional Euclidean distance measure in our setting? (4 pts)
2. 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. (3 pts)

Q5 Feature Matching (10 pts)

Please implement a function:

> 💡 *matches, locs1, locs2 = matchPics(I1, I2)*

where *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 feature points. matches is a $p \times 2$ matrix where the first column is indices into features in *I1*, and similarily the second column contains indices related to *I2*. Use the provided helper function *corner detection* to compute the features, then build descriptors using the provided helper function , and finally compare them using the provided helper function *briefMatch*.

Use the provided helper function *plotMatches* to visualize your matched points and include the result image in your write-up. An example is shown in Fig. 2.

The number of matches between the two images varies based on the parameter *sigma* used in *corner detection*, and also on the value *ratio* in *briefMatch*. You should vary these to get the best results. The example shown in Fig. 2 is with *sigma=1.5* and *ratio=0.65*.

We provide you with the following helper functions:

> 💡 *locs = corner detection(img, sigma)*
> *desc, locs = computeBrief(img, locs)*
> *matches = briefMatch(desc1, desc2)*
> *plotMatches(im1, im2, matches, locs1, locs2)*

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 can be less than the number of input feature points. *desc* is the corresponding matrix of BRIEF descriptors for the interest points.

In your write-up

1. Please include an image similar to Figure 3. **You must use different colors from Figure 3**. (10 pts)

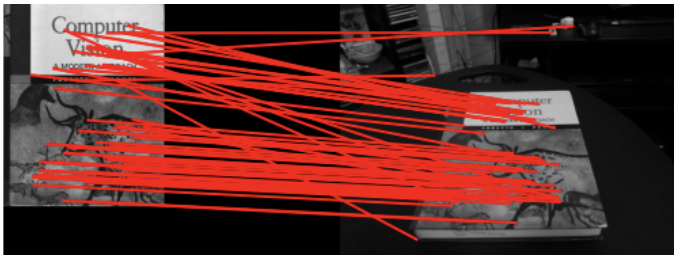

Figure 3: The matching result

Q6 BRIEF and Rotations (10 pts)

Let's investigate how BRIEF works with rotations. Write a script Q6_*briefRotTest.py*

- Takes the *cv cover.jpg* and matches it to itself rotated [*Hint: use scipy.ndimage.rotate*] in increments of 10 degrees.
- Stores a histogram of the count of matches for each orientation.
- Plots the histogram using *matplotlib.pyplot.hist*.
- Visualize the feature matching result at three different orientations that represent the characteristics of the brief descriptor's behaviors.

In your write-up

1. Please include a histogram of BRIEF **similar** to Figure 4 and explain why BRIEF descriptor behaves this way. (**You must use different colors from Figure 4**) (5 pts)

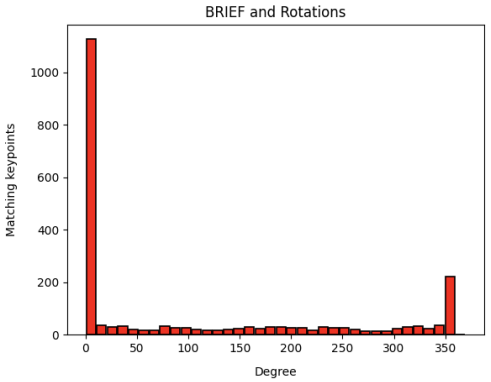

Figure 4: The histogram that rotates 0~360

2. BRIEF matching result at different orientation. See Figure 5-7 as an example (5 pts)

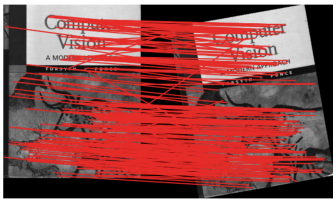

Figure 5: The result of matching that rotates 10 degrees

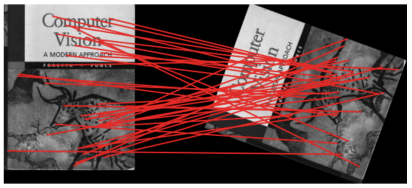

Figure 6: The result of matching that rotates 70 degrees

Figure 7: The result of matching that rotates 190 degrees

Q7 ORB Descriptor(10 pts)

In the next chapter, we will use ORB Descriptor instead FAST and BRIEF. Please write down the benefits of ORB.

In your write-up

1. What differences did ORB have that make benefits over FAST and BRIEF descriptors? (5 pts)
2. Is ORB descriptor invariant to rotation and scale? Why/Why not? (5 pts)

2.2 Homography Computation and Image Warping

Q8 Computing the Homography (30 pts)

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

> 💡 *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. The *numpy.linalg* functions *eig* or *svd* will be useful to get the eigenvectors (see Section 1 of this handout for details).

2. Write a function *warpPerspective* that warps image using pre-computed homography matrix.

   💡 *img_warped, mask = warpPerspective(img, H, size)*

   parameters:

   - *img*: input image
   - *H*: 3×3 transformation matrix
   - *size*: tuple representing the size of the output image, in the format (*HEIGHT*, *WIDTH*).

   return:

   - *img_warped*: transformed image after applying the perspective warp
   - *mask*: mask indicating empty and filled pixels in the warped image

   ※ You can also utilize OpenCV's warpPerspective function for debugging purposes. In this case, it's better to set its flag to *cv2.INTER_NEAREST*.

In your write-up

1. Uses the computeH to get homography and warp cv_*cover.jpg* to the dimensions of the *cv_desk.png* image using the *warpPerspective*. See Figure 8 for an example. (30 pts)

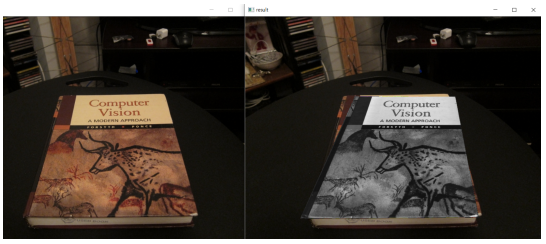

Figure 8 The result of the warping function using unnormalized homography

2.3 Homography Normalization

Normalization improves the 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:

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

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 $\widetilde{\mathbf{x}}_1\widetilde{\mathbf{x}}_\mathbf{1}$ 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:

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

Q9 Homography with normalization (10 pts)

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.

In your write-up

1. Please include your normalization code. (as a capture) (5 pts)
2. Could you provide an example where not applying homography normalization caused an issue? After that, please share the improved example with homography normalization applied. Finally, discuss how applying homography normalization helps prevent problems in your experiments. You can use your own images to illustrate this study. (5 pts)

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.

Q10 Implement RANSAC for computing a homography (25 pts)

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 $\mathbf{x}_1 = \mathbf{H}\mathbf{x}_2$. *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.

In your write-up

1. Visualize the number of inliers for iteration $1 < iter < 240$ in 2D plot (iteration step size: 30). When you get the number of inliers, please repeat 10 times for each iteration and use average of those trials. (15 pts)
2. How can you set the number of iterations to achieve a desired accuracy level? (10 pts)

Automated Homography Estimation and Warping

Q11 Putting it together (10 pts)

Write a script Q11_*HarryPotterize.py* that

1. Reads *cv.jpg*, *cv_desk.png*, and *hp_cover.jpg*.
2. Computes a homography automatically using *MatchPics* and *computeH_ransac*.
3. Uses the computed homography to warp *hp_cover.jpg* to the dimensions of the *cv_desk.png* image using your *warpPerspective*. (If you encounter difficulty in implementing *warpPerspective* in Q8, you can use OpenCV's warpPerspective function as an alternative in this and subsequent questions.)
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. Why do you think this is happening? How would you modify *hp_cover.jpg* to fix this issue?

5. Implement the function:

> 💡 *composite img = compositeH(H2to1, template, img)*

to now compose this warped image with the desk image as in Figure 10.

In your write-up

1. Please include the result of *Q11_HarryPotterize.py*. See Figure 10 as an example. (10 pts)

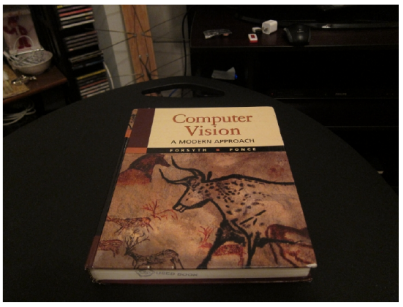

Figure 9: cv_desk.png

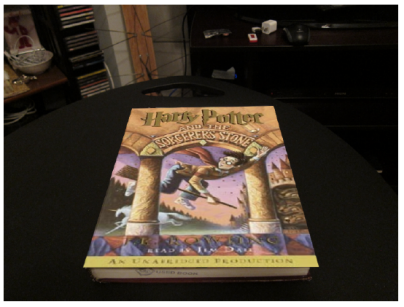

Figure 10: HarryPotterized Text book

III. Applications: Augmented Reality and Panorama

Q12 Incorporating video (15 pts)

Now with the code you have, you're able to create your own Augmented Reality application. What you're going to do is *HarryPoterize* 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 Q12_ar.*py* to implement this AR application and save your result video as *ar.avi* in the *result/* directory. You may use the function loadVid that we provide to load the videos. You'll be given full credits if you can put the video together correctly. See Figure 11 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 crop each frame to fit onto the book cover. You should crop each frame such that only its central region is used in the final output. See Figure 12 for an example.

Also, the video *book.mov* only has a translation of objects. If you want to account for the rotation of objects, scaling, etc.

In your write-up

1. Please include the result video ar.avi. (duration of this video should be same as *ar_source*.mov) (15 pts)

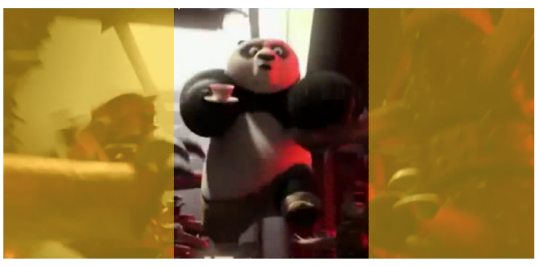

Figure 12: Crop out the yellow regions of each frame to match the aspect ratio of the book

Figure 11: Rendering video on a moving target

Q13 Create a Simple Panorama (10 pts)

Take two pictures with your own camera, separated by a pure rotation as best as possible, and then construct a panorama with Q13_*Panorama.py*. 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. Submit the original images, the final panorama, and the script Q13_*Panorama.py* that loads the images and assembles a panorama. We have provided two images for you to get started (*data/pano left.png* and *data/pano right.png*). Please use your own images when submitting this assignment. In your submission, include your original images and the panorama result image in your write-up.

In your write-up

1. Please include the result of Q13_*Panorama.py*. See Figure 13-15 below for an example. (10 pts)

Figure 13: Original Image 1 (left)

Figure 14: Original Image 2 (right)

Figure 15: Panorama

Q14 Create a Panorama Image from Unordered Multi-images (20 pts)

Until now, we have focused on generating a panorama image for two images. Then, how can we generate a panorama image for multiple images? Intuitively, we can generate a panorama image for two images and then can repeat this process for a new image and the previous panorama images. But, multiple images we have could be unordered in reality.

To handle this issue, please design an alogorithm to automatically select a pair of image to generate a panorama image in an iterative manner. (Please create final panorama image with minimal distortion)

In this problem, we provide four input images (*data/image*.png*) and they are not sequentially ordered. For example, you can follow the below steps:

1. Find the order of the images (*i.e.*, determine which image comes left, right, up, and bottom from the images). [Please note that in step 1, you **should not manually order the images**]
2. Create a panorama image by combining two neighbor images.
3. Create a partial panorama image by combining the panorama image from step 2 with its neighbor image.
4. Generate the final panorama image by combining the partial panorama image from step 3 with the remaining image.

In your write-up

1. Please describe how you can define image sequences in step 1. (10 pts)
2. Show the result of step 2, step 3, and step 4. (which are saved in ***result*** folder) (10 pts)

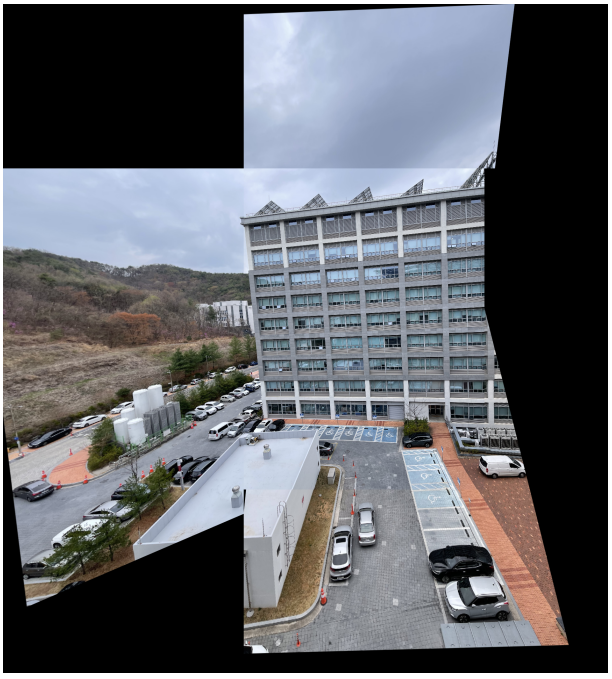

Figure 16: Multi image panorama

Q15 Discussion: Limitations of Homography Warping (degenerated case) (20 pts)

In this PA, we have implemented 2D image transformations (*i.e.*, homography estimation), where we assumed that a given set of images satisfies the condition for applying for 2D transformations. However, this assumption does not hold in real-world situations.

To observe this degenerated case, in this problem, you need to capture two images that our implementation (Q13) does not work and analyize why this problem occur.
If you can handle this problem, improve your code to overcome real-wold problems and describe your method with your result.

In your write-up

1. Please show the result of degenerated case with your own images and analyze why it is hard to apply 2D transformation in real-world situations. (5 pts)
2. Write a code that overcome the real-world problems and show the result with desctiption. (15 pts)