16-720 Computer Vision: Homework 2 (Spring 2021) Augmented Reality with Planar Homographies

Instructor: Deva Ramanan
TAs: Michael Lee, Zhiqiu Lin, Cormac O'Meadhra, Thomas Weng, Gengshan Yang, Jason Zhang
Due: Thursday March 11, 2021 23:59:59 ET

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 applications.

0 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. Do not use external code unless permitted (if in doubt, use Piazza to clarify whether a package can be used). Plagiarism is strongly prohibited and may lead to failure of this course.

2. **Start early!** This is a *much bigger* assignment than homework 1.

3. **Questions:** If you have any questions, please look at Piazza first. Other students may have encountered the same problem, and it may be solved already. If not, post your question on the discussion board. Please keep questions about homework problems in the corresponding threads.

4. The assignment must be completed using Python 3. We recommend setting up a conda environment. See this Piazza post for instructions. You will need the cv2, skimage, scipy modules, and would benefit from using imagio to create a video for Section 3. You can install these packages with `pip install opencv-python scikit-image scipy imageio imageio-ffmpeg`.

5. Please stick to the function prototypes mentioned in the handout. This makes verifying code easier for the TAs.

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')`.

7. **Write-up:** Your write-up should mainly consist of four parts: your answers to theory questions, resulting images of each step, the discussions for experiments, and **as well as snippets of all your code in an Appendix** *(make sure that there aren't any lines that run beyond the page width and that all code is visible, else you may be penalized up to 5% of the total score)*. Please note that we will not accept handwritten scans for your write-up. You may complete your write-up using LaTeX, Microsoft Word, or in-line on a Jupyter Notebook. We will provide a Latex template, but you are not required to use it as long as your submission follows all of the submission guidelines.

8. In your PDF, *add a page break after each question.* **When submitting to Gradescope, make sure that you select each page corresponding to your answer for each question (*if the corresponding write-up and code live on separate pages, select all of them*).** Not doing this makes it difficult for us to find your answer and you will be penalized accordingly.

9. **Submission:** Create a zip file, `<andrew-id>.zip`, composed of your write-up, your Python implementations (including helper functions) and results, and the implementations and results for extra credits (optional). Please make sure to remove the `data/` folder, and any other temporary files you've generated. Your final upload should have the files arranged in this layout:

- `<AndrewId>.zip`
  - `<AndrewId>/`
    - `<AndrewId>.pdf`
    - `python/`
      - `ar.py`
      - `ar_helper.py`
      - `helper.py`
      - `loadVideo.py`
      - `main.py`
      - `opts.py`
      - `planarH.py`
      - `yourHelperFunctions.py` *(optional)*
    - `result/`
      - `ar.avi`
    - `ec/` *(optional for extra credit)*
      - `panorama.py`
      - the images required for generating the results.
      - `yourHelperFunctions.py` *(optional)*

Please make sure you follow the submission rules mentioned above before uploading your zip file to Canvas. Assignments that violate this submission rule will be **penalized by up to 5% of the total score**.

1 Homographies

Planar Homographies as a Warp

Recall that a planar homography is an 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.

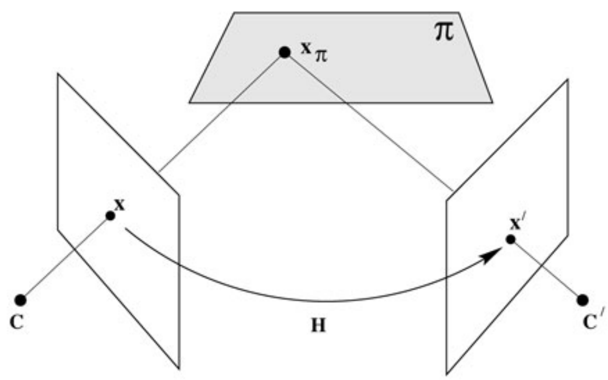

Figure 1: A homography $\mathbf{H}$ links all points $\mathbf{x}_\pi$ lying in plane π between two camera views $\mathbf{x}$ and $\mathbf{x}'$ in cameras C and C' respectively such that $\mathbf{x}' = \mathbf{H}\mathbf{x}$. *[From Hartley and Zisserman]*

Q1.1 Homography (5 points)

Prove that there exists a homography $\mathbf{H}$ that satisfies equation 1 given two 3×4 camera projection matrices $\mathbf{P_1}$ and $\mathbf{P_2}$ corresponding to the two cameras and a plane Π. You do not need to produce an actual algebraic expression for $\mathbf{H}$. All we are asking for is a proof of the existence of H.

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

The $\equiv$ symbol stands for *identical to*. The points $\mathbf{x}_1$ and $\mathbf{x}_2$ are in *homogenous coordinates*, which means they have an additional dimension. If $\mathbf{x}_1$ is a 3D vector $\begin{bmatrix} x_i & y_i & z_i \end{bmatrix}^T$, it represents the 2D point $\begin{bmatrix} \frac{x_i}{z_i} & \frac{y_i}{z_i} \end{bmatrix}^T$ (called *inhomogenous 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.

Note: A degenerate case happens when the plane Π contains both cameras' centers, in which case there are infinite choices of $\mathbf{H}$ satisfying equation 1. You can ignore this special case in your answer.

The Direct Linear Transform

A very common problem in projective geometry is often of the form $\mathbf{x} \equiv \mathbf{Ay}$, 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 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 $\mathbf{x} = \lambda \mathbf{Hx}'$), 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 $\mathbf{H}$ matrix, it's a *transform* of the problem into a set of *linear* equation, thus giving it its name.

Q1.2 Correspondences (15 points)

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:

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

where $\mathbf{x_1^i} = \begin{bmatrix} x_1^i & y_1^i & 1 \end{bmatrix}^T$ are in homogenous coordinates, $\mathbf{x_1^i} \in \mathbf{x_1}$ and $\mathbf{H}$ is a 3×3 matrix. 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.

1. How many degrees of freedom does $\mathbf{h}$ have? (3 points)
2. How many point pairs are required to solve $\mathbf{h}$? (2 points)
3. Derive $\mathbf{A_i}$. (5 points)
4. 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 that 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? (5 points)

Using Matrix Decompositions 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 will obtain the corresponding point coordinates using feature matches and will then need to calculate the homography. You have already derived that $\mathbf{Ax} = \mathbf{0}$ in Question 1. 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.

Eigenvalue Decomposition

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.linalg` 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 $\mathbf{V}$ 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 $\mathbf{Ax} = \mathbf{0}$.

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

Singular Value Decomposition

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

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

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 *under-determined*. 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 $\mathbf{A}\mathbf{h} = \mathbf{0}$. Using this knowledge, let us reformulate our problem of solving $\mathbf{A}\mathbf{x} = \mathbf{0}$. We want to minimize the error in solution in the least-squares sense. Ideally, the product $\mathbf{A}\mathbf{h}$ should be 0. Thus the sum-squared error can be written as:

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

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

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

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 $\mathbf{A}\mathbf{h} = \mathbf{0}$ is the the corresponding eigenvector (in column 9 of the matrix $\mathbf{V}$).

Theory Questions

Q1.3 Homography under rotation (5 points)

Prove that there exists a homography $\mathbf{H}$ that satisfies $\mathbf{x_1} \equiv \mathbf{H}\mathbf{x_2}$, given two cameras separated by a pure rotation. That is, for camera 1, $\mathbf{x_1} = \mathbf{K_1}\begin{bmatrix}\mathbf{I} & \mathbf{0}\end{bmatrix}\mathbf{X}$ and for camera 2, $\mathbf{x_2} = \mathbf{K_2}\begin{bmatrix}\mathbf{R} & \mathbf{0}\end{bmatrix}\mathbf{X}$. Note that $\mathbf{K_1}$ and $\mathbf{K_2}$ are the 3×3 intrinsic matrices of the two cameras and are different. $\mathbf{I}$ is 3×3 identity matrix, $\mathbf{0}$ is a 3×1 zero vector and $\mathbf{X}$ is a point in 3D space. $\mathbf{R}$ is the 3×3 rotation matrix of the camera.

Q1.4 Understanding homographies under rotation (5 points)

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 θ be the angle of rotation between the two. Show that $\mathbf{H^2}$ is the homography corresponding to a rotation of 2θ. Please limit your answer within a couple of lines. A lengthy proof indicates that you're doing something too complicated (or wrong).

Q1.5 Limitations of the planar homography (5 points)

Why is the planar homography not completely sufficient to map any arbitrary scene image to another viewpoint? State your answer concisely in one or two sentences and provide an example of a failure case.

Q1.6 Behavior of lines under perspective projections (5 points)

We stated in class that perspective projection preserves lines (a line in 3D is projected to a line in 2D). Verify algebraically that this is the case, i.e., verify that the projection $\mathbf{P}$ in $\mathbf{x} = \mathbf{PX}$ preserves lines.

2 Computing Planar Homographies

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, we will not reimplement a feature detector or descriptor here by yourself, but use python modules (both of them are provided in helper.py)** 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 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 FAST Detector (5 points)

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.) Can you comment on its computational performance compared to the Harris corner detector?

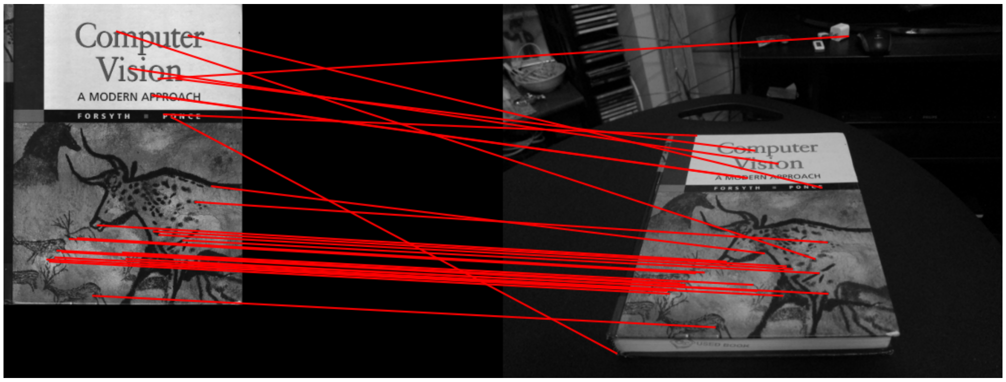

Figure 2: A few matched FAST feature points with the BRIEF descriptor.

Q2.1.2 BRIEF Descriptor (5 points)

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 Matching Methods (5 points)

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 Feature Matching (10 points)

Please implement the function

```
matches, locs1, locs2 = ar_helper.matchPics(img1, img2, opts)
```

where `img1` and `img2` are the images you want to match. `opts` stores two parameters. `sigma` is threshold for corner detection using FAST feature detector, and `ratio` is the ratio for BRIEF feature descriptor. `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 `img1` and similarly second column contains indices related to `I2`. Use the provided helper function `detectCorners()` to compute the features, then build descriptors using `computeBrief()`, and finally compare them using `briefMatch()`. Use `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 2 images varies based on the values of `sigma` in `corner_detection()` and `ratio` in `briefMatch()`. You can vary these to get the best results. The example shown in Fig. 2 is with `sigma`=0.15 and `ratio`=0.7.

We provide you with the following helper functions in `helper.py`:

```
locs = detectCorners(img, sigma)
desc, locs = computeBrief(img, locs)
matches = briefMatch(desc1, desc2, ratio)
plotMatches(img1, img2, 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 from `computeBrief` could be less than the number of input feature points. `desc` is the corresponding matrix of BRIEF descriptors for the interest points.

Display matched features for `cv_cover.png` and `cv_desk.png` and include it in your write-up.

Q2.1.5 Feature Matching Parameter Tuning (10 points)

There are two tunable parameters, both stored in the `opts` variable, and are loaded from `opts.py`. You can change the values by changing their default fields or by command-line arguments. For example, `python main.py --sigma 0.15 --ratio 0.7`.

Conduct a small ablation study by running `matchPics` on `cv_cover.png` and `cv_desk.png` with various `sigma` and `ratio` values. Include the figures displaying the matched features with various parameters in your write-up, and explain the effect of these two parameters respectively.

Q2.1.6 BRIEF and Rotations (10 points)

Let's investigate how BRIEF works with rotations. Implement the function `ar_helper.briefRotTest(img, opts)` that:

- Takes the `cv_cover.jpg` and matches it to itself rotated in increments of 10 degrees from 0 to 360 degrees.
  (*Hint:* use `scipy.ndimage.rotate` to rotate the image.)
- Stores a histogram of the count of matches for each orientation.
- Plots the histogram using `matplotlib.pyplot.bar` (use log scaling for the y-axis).

Visualize the histogram and 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.

Homography Computation

Q2.2.1 Computing the Homography (15 points)

Implement the function `computeH` in `planarH.py` that estimates the planar homography from a set of matched point pairs.

```
H2to1 = planarH.computeH(locs1, locs2)
```

`locs1` and `locs2` 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` function `eig()` or `svd()` will be useful to get the eigenvectors (see Section 1 of this handout for details).

Homography Normalization

Normalization of data points can improve the numerical stability of the solution. You should thus normalize your coordinates by doing the following:

1. Translate the mean of the points to the origin.
2. Scale the points so that the largest distance to the origin is $\sqrt{2}$, i.e the absolute value of the coordinates do not exceed 1.

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 coordinates of $\mathbf{x_1}$ and $\mathbf{x_2}$ (where $\mathbf{x_1}$, $\mathbf{x_2}$, $\widetilde{\mathbf{x}}_1$, and $\widetilde{\mathbf{x}}_2$ are all homogeneous). $\mathbf{T}_1$ and $\mathbf{T}_2$ are 3×3 matrices.

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

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

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:

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

which provides the effective homography matrix which can be used to transform the original unnormalized points (i.e. a way to 'denormalize' $\widetilde{\mathbf{H}}$).

Q2.2.2 Homography with normalization (10 points)

Implement the function `computeH_norm`:

```
H2to1 = computeH_norm(locs1, locs2)
```

This function should normalize the coordinates `locs1` and `locs2`, then call `computeH(locs1, locs2)`, and finally return the effective homography matrix between `locs1` and `locs2` that has been 'denormalized' as described above.

`locs1` and `locs2` are $N \times 2$ matrices containing the coordinates (x, y) of point pairs between the two images. `H2to1` should again be a 3×3 matrix of the best homography from image 2 to image 1 in the least-square sense.

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.

Q2.2.3 Implement RANSAC for computing a homography (25 points)

Implement the function:

```
best_H2to1, inliers = planarH.computeH_ransac(locs1, locs2, opts)
```

where `locs1` and `locs2` are $N \times 2$ matrices containing the matched points. `opts` stores two RANSAC parameters. `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.

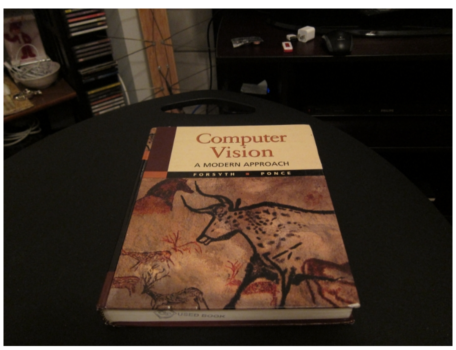

Figure 3: Text book

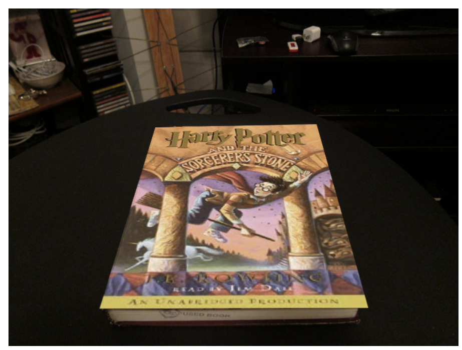

Figure 4: HarryPotterized Text book

Automated Homography Estimation and Warping

Q2.2.4 Putting it together (10 points)

Implement following function to replace the cover of the CV textbook in `cv_desk.png` with the cover of Harry Potter in `hp_cover.jpg`

```
composite_img = ar_helper.composeWarpedImg(img_source, img_target,
                                           img_replacement, opts)
```

1. Compute a homography between `cv_cover.jpg` and `cv_desk.png` automatically using `matchPics` and `computeH_ransac`.

2. Use the computed homography to warp `hp_cover.jpg` to the dimensions of the `cv_desk.png` image using the OpenCV function `cv2.warpPerspective` function.
3. If you notice that `hp_cover.jpg` is being warped to the correct location, but it is not filling up the same space as the CV book in `cv_desk.jpg`, you must account for the appropriate scaling between `hp_cover.jpg` and `cv_cover.jpg`.
4. Include your result in your write-up, which should look similar to Figure 4.

Q2.2.5 RANSAC Parameter Tuning (10 points)

Just like how we tune parameters for feature matching, there are two tunable parameters in RANSAC as well.

Conduct a small ablation study by running `ar_helper.composeWarpedImg` with various `max_iters` and `inlier_tol` values. Include the result images in your write-up, and explain the effect of these two parameters respectively.

3 Creating your Augmented Reality application

Q3.1 Incorporating video (20 points)

Now with the code you have, you're able to create you own Augmented Reality application. What you're going to do is warp 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 fill out the `main` function in `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. Your result should be similar to the LifePrint project. You'll be given full credits if you can put the video together correctly. See Figure 5 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 must also crop that image such that only the central region of the image is used in the final output. See Figure 6 for an example.

Finally, note that this is a very time-intensive job and may take many hours on a single core (parallel processing on 8 cores takes around 30 minutes). Debug before running your full script (e.g. by saving a few early AR frames and verifying that they look correct).

Also include three screenshots of your ar.avi at three distinct timestamps (e.g. when the overlay is near the center, left, and right of the video frame) in your write-up. See Figure 5 as an example of where the overlay is in the center of the video frame.

Figure 5: Rendering video on a moving target

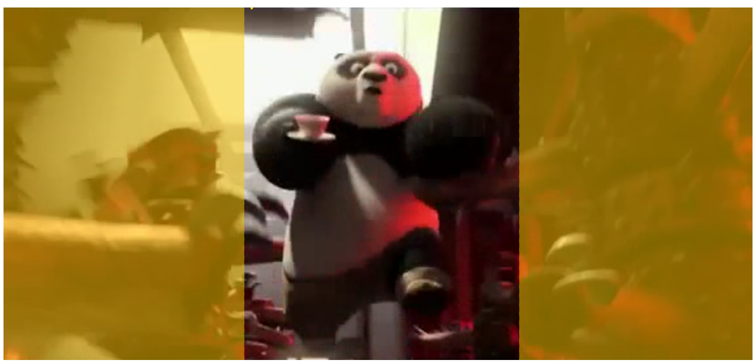

Figure 6: Crop out the yellow regions of each frame to match the aspect ratio of the book

4 Extra Credit

Q4.1x: Create a Simple Panorama (10 points)

Take two pictures with your own camera, separated by a pure rotation as best as possible, and then construct a panorama with `panorama.py`. Be sure that objects in the images are far enough away so that there are no parallax effects. You can use python module `cpselect` to select matching points on each image or some automatic method. Submit the original images, the final panorama, and the script `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 project. **In your submission, include your original images and the panorama result image in your write-up.** See Figure 7-9 below for example.

Figure 7: Original Image 1 (left)

Figure 8: Original Image 2 (right)

Figure 9: Panorama

Submission

A reminder to follow the instructions specified in Section 0 (i.e. instructions 7-9) as you finish the assignment and finalize your submission! Best of luck!