16-720A Computer Vision: Assignment 4
Image matching, stitching and homographies

Instructor: John Galeotti, Kris Kitani
TAs: Leonid Keselman, Tanya Marwah,
Shashank Tripathi, Vibha Nasery, Jiayuan Li
Due Date: Friday, October 12, 2018 11:59pm
Version 1 (Sept 26, 12pm)

Total Points: 100
Extra Credit: 30

Interest point detectors and descriptors are at the heart of most computer vision applications. The goal of this assignment is to gain insights by working on one such application in determining homographies. We first begin with a quick recap and establish our fundamental understanding of the theory behind planar homographies, and then proceed towards showing how to transform between template images and source images, as is commonly used in panoramic photo stitching and augmented reality applications.

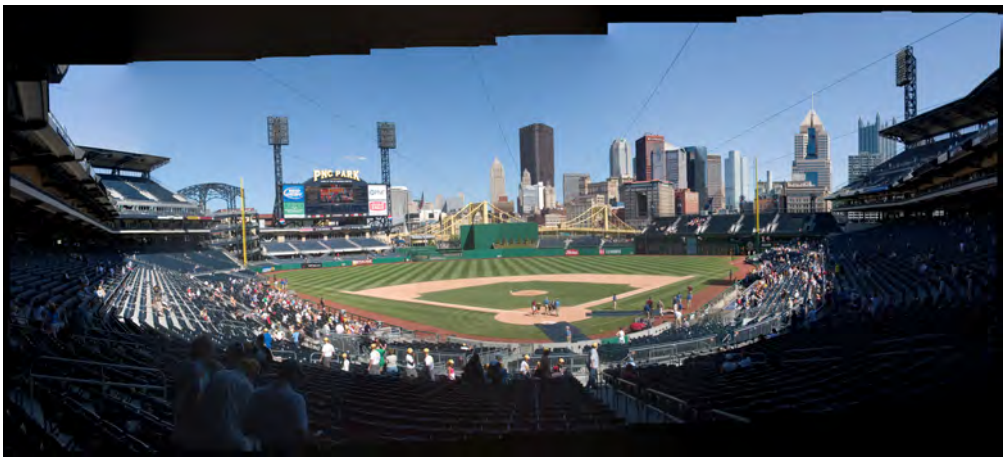

Figure 1: At the end of this assignment, you'll be able to stitch multiple photos into a seamless single photograph, source image taken from M. Kazhdan

Instructions

1. **Integrity and collaboration:** 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 in this course.
2. **Start early!** Especially if you want to play around with the AR app in the end!
3. **Piazza:** If you have any questions, please look at Piazza first. We've created folders for every question, and a sticky thread at the top for each question. Please go through all the previous posts tagged in the appropriate comment before submitting a new question and adding its tag in the sticky.
4. **Write-up:** 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.
5. **Code:** Please stick to the function prototypes mentioned in the handout. This makes verifying code easier for the TAs.
6. **Submission:** Please include all results in your writeup pdf submitted on Gradescope. For code submission, create a zip file, `<andrew-id>.zip`, composed of your Python files. Please make sure to remove any temporary files you've generated. Your final upload should have the files arranged in this layout.

    - `<AndrewId>.zip`
        - `<AndrewId>/`
            - `python/`
                - `q2.py`
                - `q3.py`
                - `q4.py`
                - `ec.py`
                - `run_q2.py`
                - `run_q3.py`
                - `run_q4.py`
                - `run_ec.py`

1 Homographies (30 points)

Suppose we have two cameras $\mathbf{C}_1$ and $\mathbf{C}_2$ looking at a common plane Π in 3D space. Any 3D point $\mathbf{P}$ on Π generates a projected 2D point located at $\mathbf{p} \equiv (u_1, v_1, 1)^T$ on the first camera $\mathbf{C}_1$ and $\mathbf{q} \equiv (u_2, v_2, 1)^T$ on the second camera $\mathbf{C}_2$. Since $\mathbf{P}$ is confined to the plane Π, we expect that there is a relationship between $\mathbf{p}$ and $\mathbf{q}$. In particular, there exists a common 3×3 matrix $\mathbf{H}$, so that for any $\mathbf{p}$ and $\mathbf{q}$, the following conditions holds:

$$\mathbf{p} \equiv \mathbf{Hq} \tag{1}$$

We call this relationship *planar homography*. Recall that both $\mathbf{p}$ and $\mathbf{q}$ are in homogeneous coordinates and the equality $\equiv$ means $\mathbf{p}$ is proportional to $\mathbf{Hq}$ (recall homogeneous coordinates). It turns out this relationship is also true for cameras that are related by pure rotation without the planar constraint.

Q1.1 Homography (5 points)

Prove that there exists an $\mathbf{H}$ that satisfies Equation 1 given two 3×4 camera projection matrices $\mathbf{M_1}$ and $\mathbf{M_2}$ corresponding to cameras $\mathbf{C_1}$, $\mathbf{C_2}$ and a plane Π. Do not produce an actual algebraic expression for $\mathbf{H}$. All we are asking for is a proof of the existence of $\mathbf{H}$.

Note: A degenerate case may happen 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 case in your answer.

Q1.2 Homography under rotation (5 points)

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, $\mathbf{x_1} = \mathbf{K_1}\begin{bmatrix}\mathbf{I} & \mathbf{0}\end{bmatrix}\mathbf{X}$ 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.

Q1.3 Correspondences (10 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:

$$\mathbf{x_1^i} \equiv \mathbf{Hx_2^i} \quad (i \in \{1 \dots N\})$$

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)

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 (2 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.

Q1.6 Behavior of lines under perspective projections (3 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. *Hint: consider how to parameterize a line and show that the same thing holds after perspective projection*

Note: Interest Points

Before finding the homography between an image pair, we need to find corresponding point pairs between two images. But how do we get these points? One way is to select them manually (using `cpselect`), which is tedious and inefficient. The CV way is to find interest points in the image pair and automatically match them. In the interest of being able to do cool stuff, we will not re-implement a feature detector here, but use built-in MATLAB methods. The purpose of an interest point detector (e.g. Harris, FAST, SIFT, SURF, etc.) is to find particular salient points in the images around which we extract feature descriptors (e.g. BRIEF, ORB, etc.). These descriptors try to summarize the content of the image around the feature points in as succinct yet descriptive manner possible (there is often a trade-off between representational and computational complexity for many computer vision tasks; you can have a very high dimensional feature descriptor that would ensure that you get good matches, but computing it could be prohibitively expensive). Matching, then, is a task of trying to find a descriptor in the list of descriptors obtained after computing them on a new image that best matches the current descriptor. This could be something as simple as the Euclidean distance between the two descriptors, or something more complicated, depending on how the descriptor is composed.

For the purpose of this exercise, you can use FAST or Harris detector in concert with the BRIEF descriptor. You can use the Matlab built-in function `detectFASTFeatures` or `detectHarrisFeatures` to compute the interest points. Likewise, in Python you can use functions form skimage.feature such as skimage.feature.corner_fast or skimage.feature.corner_harris.

2 BRIEF Descriptor (25 points)

Now that we have interest points that tell us where to find the most informative points in the image, we can compute descriptors that can be used to match to other views of the same point in different images. The BRIEF descriptor encodes information from a 9×9 patch p centered around the interest point at the *characteristic scale* of the interest point. See the lecture notes for Point Feature Detectors if you need to refresh your memory.

2.1 Creating a Set of BRIEF Tests (5 pts)

The descriptor itself is a vector that is n-bits long, where each bit is the result of the following simple test:

$$\tau(p; x, y) := \begin{cases} 1, & \text{if } p(x) < p(y). \\ 0, & \text{otherwise.} \end{cases} \tag{2}$$

Set n to 256 bits. There is no need to encode the test results as actual bits. It is fine to encode them as a 256 element vector.

There are many choices for the 256 test pairs (x, y) used to compute $\tau(p; x, y)$ (each of the n bits). The authors describe and test some of them in [1]. Read section 3.2

of that paper and implement one of these solutions. You should generate a static set of test pairs and save that data to a file. You will use these pairs for all subsequent computations of the BRIEF descriptor.

Q 2.1: Write the function to create the x and y pairs that we will use for comparison to compute τ:

```
[compareX, compareY] = makeTestPattern(patchWidth, nbits)
```

`patchWidth` is the width of the image patch (usually 9) and `nbits` is the number of tests n in the BRIEF descriptor. `compareX` and `compareY` are linear indices into the `patchWidth` $\times$ `patchWidth` image patch and are each $nbits \times 1$ vectors. Run this routine for the given parameters `patchWidth` $= 9$ and $n = 256$ and save the results in `testPattern.mat`. **Include this file** in your submission.

2.2 Compute the BRIEF Descriptor (5 pts)

It is now time to compute the BRIEF descriptor for the detected keypoints.

Q 2.2: Write the function:

```
[locs,desc] = computeBrief(im, locs, compareX, compareY)
```

where `im` is a grayscale image with values from 0 to 1, `locs` are the keypoint locations returned by the Harris keypoint detector and `compareX` and `compareY` are the test patterns computed in Section 2.1 and were saved into `testPattern.mat`.

The function returns `locs`, an $m \times 2$ vector, where the first two columns are the image coordinates of keypoints, and `desc`, an $m \times nbits$ matrix of stacked BRIEF descriptors. m is the number of valid descriptors in the image and will vary. You may have to be careful about the input detector locations since they may be at the edge of an image where we cannot extract a full patch of width `patchWidth`. Thus, the number of output `locs` may be less than the input.

2.3 Putting it all Together (5 pts)

Q 2.3: Write a function :

```
[locs, desc] = briefLite(im)
```

which accepts a grayscale image `im` with values between zero and one and returns `locs`, an $m \times 2$ vector, where the first two columns are the image coordinates of keypoints, and `desc`, an $m \times n$ *bits* matrix of stacked BRIEF descriptors. m is the number of valid descriptors in the image and will vary. n is the number of bits for the BRIEF descriptor.

This function should perform all the necessary steps to extract the descriptors from the image, including

- Load parameters and test patterns
- Get keypoint locations
- Compute a set of valid BRIEF descriptors

2.4 Check Point: Descriptor Matching (5 pts)

A descriptor's strength is in its ability to match to other descriptors generated by the same world point, despite change of view, lighting, etc. The distance metric used to compute the similarity between two descriptors is critical. For BRIEF, this distance metric is the Hamming distance. The Hamming distance is simply the number of bits in two descriptors that differ. (Note that the position of the bits matters.)

To perform the descriptor matching mentioned above, we have provided the function in `matlab/briefMatch.m`):

```
[matches] = briefMatch(desc1, desc2, ratio)
```

Which accepts an $m_1 \times n$ *bits* stack of BRIEF descriptors from a first image and a $m_2 \times n$ *bits* stack of BRIEF descriptors from a second image and returns a $p \times 2$ matrix of matches, where the first column are indices into `desc1` and the second column are indices into `desc2`. Note that m_1, m_2, and p may be different sizes and $p \leq \min(m_1, m_2)$.

Q 2.4.1: Write a test script `testMatch` to load two of the **chickenbroth** images, compute feature matches. Use the provided `plotMatches` and `briefMatch` functions to visualize the result.

```
plotMatches(im1, im2, matches, locs1, locs2)
```

Where `im1` and `im2` are grayscale images from 0 to 1, `matches` is the list of matches returned by `briefMatch` and `locs1` and `locs2` are the locations of keypoints from `briefLite`.

Save the resulting figure and submit it in your PDF. Briefly discuss any cases that perform worse or better.

Figure 2 is an example result. Suggestion for debugging: A good test of your code is to check that you can match an image to itself.

2.5 BRIEF and rotations (5 pts)

You may have noticed worse performance under rotations. Let's investigate this!

Q 2.5: Take the `model_chickenbroth.jpg` test image and match it to itself while rotating the second image (hint: `imrotate`) in increments of 10 degrees. Count the number of correct matches at each rotation and construct a bar graph showing rotation angle vs the number of correct matches. Include this in your PDF and explain why you think the descriptor behaves this way. **Create a script** `briefRotTest.m` that performs this task.

Figure 2: Example of BRIEF matches for `model_chickenbroth.jpg` and `chickenbroth_01.jpg`.

2.6 Improving Performance - (Extra Credit)

The extra credit opportunities described below are optional and provide an avenue to explore computer vision and improve the performance of the techniques developed above.

1. **(5 pts)** 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). Explain in your PDF your design decisions and how you selected any parameters that you use. Demonstrate the effectiveness of your algorithm on image pairs related by large rotation.
2. **(5 pts)** 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 [2] for a technique that will make your detector more robust to changes in scale. Implement it and demonstrate it in action with several test images. You may simply rescale some of the test images we have given you. Your detector should not assume that it's given the scale difference, it should work with known scales.

3 Homography Computation (35 points)

Q3.1 Computing the Homography (10 points)

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

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

`x1` and `x2` are $N \times 2$ 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. This should follow from your matrix as derived in 1. For the solver, feel free to use whatever you like. Although the SVD notes in at the end of this handout are one of the more straightforward methods.

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

Q3.2 Homography with normalization (5 points)

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)` as described above.

RANSAC

The RANSAC algorithm can generally fit any model to noisy data. You will implement it for (planar) homographies between images.

Q3.3 Implement RANSAC for computing a homography (10 points)

Write a function:

`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.

Automated Homography Estimation/Warping for Augmented Reality

Q3.4 Putting it together (10 points)

Write a script `HarryPotterize.m` that

1. Reads `cv_cover.jpg`, `cv_desk.png`, and `hp_cover.jpg`.
2. Computes a homography automatically using `computeH_ransac`.
3. Warps `hp_cover.jpg` to the dimensions of the `cv_desk.png` image using the provided `warpH` function.
4. At this point you should notice that although the image is being warped to the correct location, it is not filling up the same space as the book. Why do you think this is happening? How would you modify `hp_cover.jpg` to fix this issue?
5. Implement the function:
   `function [ composite_img ] = compositeH( H2to1, template, img )`
   to now compose this warped image with the desk image as in in Figure 4
6. Include your resulting image in your write-up. Please also print the final H matrix in your writeup (normalized so the bottom right value is 1)

Figure 3: Text book

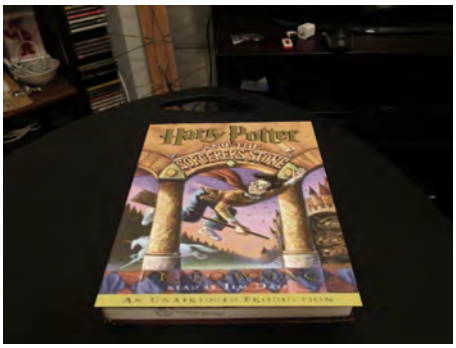

Figure 4: HarryPotterized Text book

Figure 5.a: `img1`

Figure 5.b: `img2`

Figure 5.c: `img2` warped to `img1`'s frame

Figure 5: Example output for **Q 4.1**: Original images `img1` and `img2` (left and center) and `img2` warped to fit `img1` (right). Notice that the warped image clips out of the image. We will fix this in **Q 4.2**

4 Stitching Panoramas (10 points)

We can also use homographies to create a panorama image from multiple views of the same scene. This is possible for example when there is no camera translation between the views (e.g., only rotation about the camera center). First, you will generate panoramas using matched point correspondences between images using the BRIEF matching. **We will assume that there is no error in your matched point correspondences between images (Although there might be some errors, and even small errors can have drastic impacts)**. In the next section you will extend the technique to deal with the noisy keypoint matches.

You will need to use the provided function `warp_im = warpH(im, H, out_size)`, which warps image `im` using the homography transform `H`. The pixels in `warp_im` are sampled at coordinates in the rectangle $(1, 1)$ to $(\texttt{out\_size(2)}, \texttt{out\_size(1)})$. The coordinates of the pixels in the source image are taken to be $(1, 1)$ to $(\texttt{size(im,2)}, \texttt{size(im,1)})$ and transformed according to `H`.

Q 4.1 (5 pts) In this problem you will implement and use the function:

```
[panoImg] = imageStitching(img1, img2, H2to1)
```

on two images from the Dusquesne incline. This function accepts two images and the output from the homography estimation function. This function:

1. Warps `img2` into `img1`'s reference frame using the provided `warpH()` function
2. Blends `img1` and warped `img2` and outputs the panorama image.

For this problem, use the provided images `pnc1` as `img1` and `pnc0` as `img2`. The point correspondences `pts` are generated by your BRIEF descriptor matching.

Apply your `ransacH()` to these correspondences to compute `H2to1`, which is the homography from `pnc0` onto `pnc1`. Then apply this homography to `pnc0` using `warpH()`. **Note:** Since the warped image will be translated to the right, you will need a larger target image.

Visualize the warped image. Please include the image and your H2to1 matrix (with the bottom right index as 1) in your writeup PDF. (along with stating which image pair you used)

Q 4.2 (3 pts) Notice how the output from Q 4.1 is clipped at the edges? We will fix this now. Implement a function

```
[panoImg] = imageStitching_noClip(img1, img2, H2to1)
```

that takes in the same input types and produces the same outputs as in Q 4.1.

To prevent clipping at the edges, we instead need to warp *both* image 1 and image 2 into a common third reference frame in which we can display both images without any clipping. Specifically, we want to find a matrix M that *only* does scaling and translation such that:

```
warp_im1 = warpH(im1, M, out_size);
warp_im2 = warpH(im2, M*H2to1, out_size);
```

This produces warped images in a common reference frame where all points in `im1` and `im2` are visible. To do this, we will only take as input either the width or height of `out_size` and compute the other one based on the given images such that the warped images are not squeezed or elongated in the panorama image. For now, assume we only take as input the width of the image (i.e., `out_size(2)`) and should therefore compute the correct height(i.e., `out_size(1)`).

Hint: The computation will be done in terms of `H2to1` and the extreme points (corners) of the two images. Make sure M includes only scale (find the aspect ratio of the full-sized panorama image) and translation.

Visualize the warped image. Please include the image in your writeup PDF. (along with stating which image pair you used)

Q 4.3 (2 pts): You now have all the tools you need to automatically generate panoramas. Write a function that does the entire pipeline in one place. That is, that accepts two images as input, computes keypoints and descriptors for both the images, finds putative feature correspondences by matching keypoint descriptors, estimates a homography using RANSAC and then warps one of the images with the homography so that they are aligned and then overlays them.

```
im3 = generatePanorama(im1, im2)
```

Run your code on the image pair `data/pnc1.jpg`, `data/pnc0.jpg` or `data/incline_L.jpg`, `data/incline_R.jpg`. However during debugging, try on scaled down versions of the images to keep running time low. **Save the resulting panorama on the full sized images and include the figure and computed homography matrix in your writeup.**

Q 4.4 (3pts) extra credit: Collect a pair of your own images (with your phone) and stitch them together using your code from the previous section. Include the pair of images and their result in the write-up.

Q 4.5 (2pts) extra credit: Collect at least 6 images and stitch them into a single noClip image. You can either collect your own, or use the PNC Park images from Matt Uyttendaele. We used the PNC park images (subsmapled to 1/4 sized) and ORB keypoints and descriptors for our reference solution.

Figure 6: Final panorama view. With homography estimated using RANSAC.

5 Extra Credit (15 points)

For extra credit, we will implement a smarter form of image blending, where seams are minimized. This can be used to handle exposure differences between stitches images, correct for motion blur, and even act as a photoshop healing tool. For examples in photo stitching, see `http://www.cs.jhu.edu/~misha/Code/SMG/` or `http://www.cs.jhu.edu/~misha/Code/DMG/Version4.5/`

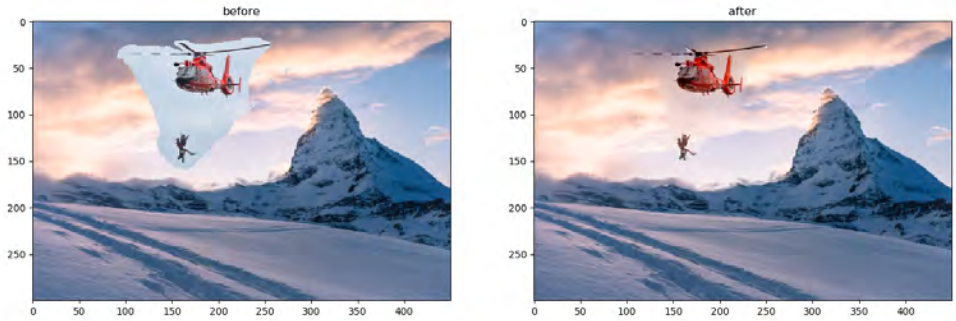

Figure 7: Example of Poisson stitching with included images.

5.1 Q 5.1 (15 points)

In this section we will implement **Poisson Image Stitching** [3] which can be used to create seamless photo montages.

Write a function called `poisson_blend(background,foreground,mask)` which takes 3 equal sized images (background and foreground as RGB, mask as binary) and solves the Poisson equation, using gradients from foreground and boundary conditions from the background. Please include results from both the `fg1,bg1,mask1` and `fg2,bg2,mask2` images in your write-up.

The basic idea behind Poisson Image Stitching is to setup a linear system $Ax = b$ where each row of A is the gradient operator for the image pixel mask (e.g. 4 for each pixel, and -1 +1 for it's 4 neighbors), while the row in b contains known gradients from the foreground. Additional rows in A are pixels on the boundary, which have known values, stored in b. Solving for x then gives you pixel values for the entire masked patch. This is done for each image channel seperately. See the paper, this page or this page if you need more information.

References

[1] Michael Calonder et al. "BRIEF: Binary robust independent elementary features". In: *ECCV*. 2010. URL: https://www.robots.ox.ac.uk/~vgg/rg/papers/CalonderLSF10.pdf.

[2] David G. Lowe. "Distinctive Image Features from Scale-Invariant Keypoints". In: *IJCV* (2004). URL: https://www.cs.ubc.ca/~lowe/papers/ijcv04.pdf.

[3] Patrick Pérez, Michel Gangnet, and Andrew Blake. "Poisson image editing". In: *TOG* (2003). URL: https://www.cs.virginia.edu/~connelly/class/2014/comp_photo/proj2/poisson.pdf.

6 SVD Notes

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 $\mathbf{Ah} = \mathbf{0}$. 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. Ideally, the product $\mathbf{Ah}$ should be 0. Thus the sum-squared error can be written as:

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

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 $\mathbf{V}$).