CS 435 - Computational Photography

Final Project - Panoramic Stitching

YOU MAY WORK WITH A PARTNER IF YOU LIKE!!!
But if you do so, look at the additional information you need to provide in your submission (stated at the end of the document).

Introduction

For our final assignment, we'll attack the problem of creating a panoramic photo. This will require several ideas from this course, including:

- Least Squares Estimate (LSE) for Transformation Matrix Discovery
- Projection
- Blending
- Interest Point Discovery (subsampling, gradients, edges)
- Respresentation (feature extraction)
- Feature Matching (point correspondences).

Grading

Hard Coded Correspondences	10pts
Panoramic using hard-coded correspondences	30pts
Image Pyramids	10pts
Extrema Points	10pts
Keypoint Matching	10pts
Automatic Stitching	10pts
Success on Additional Tests	12pts
Report quality an ease of running code	8pts
TOTAL	100pts

Table 1: Grading Rubric

The Dataset

For the programming component of this assignment, take two pictures, one slightly offset from the other (via rotation and/or translation). Make sure that the two images have significant overlap of content.

1 (10 points) Hard Coding Point Correspondences

Let's start off by hard coding some point correspondences. Look at each image and choose **four** point correspondences.

Display the images side-by-side (as one image) with the point correspondences color coded as dots in the image. An example can be found in Figure 1.

Figure 1: Manual Correspondences

2 (30 points) Compute Transformation Matrix, Project, and Blend!

Next, use the four points you identified in the previous part to compute the transformation matrix that maps one image to the other. You can determine which image you want to be the "base" image.

After determining the transformation matrix, we need to determine the dimensions of the new combined image. The height of this image should be the maximum of the base image's height or the maximum projected y value from the other image. The width will be equal to the maximum of the base image's width or the maximum projected x value from the other image.

Finally we need to populate our new image with pixel(s) from the base and projected images. To do this, go through each location in your new image and grab the corresponding pixels from the base and/or projected image (you'll need to determine where, if anywhere, these come from). If both images map to that location, you'll want to *blend* them (using a technique of your choosing).

An example can be found in Figure 2.

Figure 2: Stitched images using manual correspondences

3 (10 points) Create Scale-Space Image Pyramids

Now on to the tough(er) stuff! We want to automate all this!

The first step is to automatically identify locations of interest. To do this we'll find the stable local maximas in scale-space for each image. And the first step of that is to create image pyramids!

Here are some hyperparameters we'll use to create our image pyramids:

- Find the extremas in *grayscale*.
- Create five scales per octave.
- The initial scale will have a standard deviation of $\sigma_0 = 1.6$.
- Each subsequent scale will have a σ value that is $k = \sqrt{2}$ times larger than the previous.
- Each Gaussian kernel will have a width and height that is three times the filter's σ value, i.e. $w = \lceil 3\sigma \rceil$.
- Create four octaves, each 1/4 of the size of the previous octave, obtained by subsampling ever other row and column of the previous column (no interpolation).

In general, given octave n and scale m, you can compute σ as:

$$\sigma = 2^{n-1} k^{m-1} \sigma_0$$

In your report show all the images for each octave for one of you images. Something similar to Figure 3.

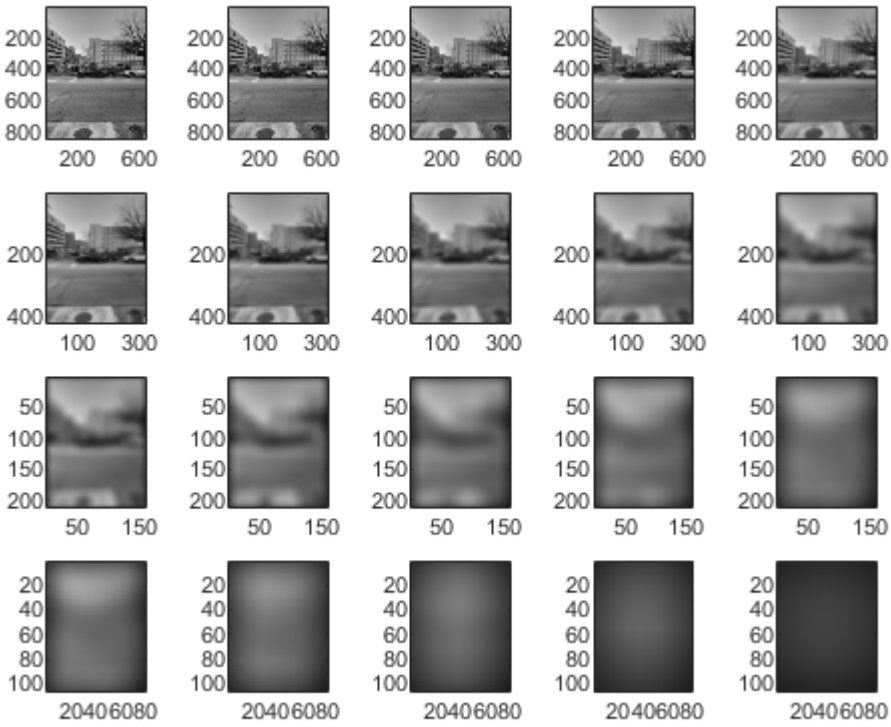

Figure 3: Image Pyramid

4 (10 points) Finding the Local Maximas

Next, for each octave of each image, locate the local maxima, as discussed in class. These locations then need to be in terms of the original image's size (i.e. the first octave), which can be done by multiplying their locations by 2^{n-1}, where again n is the current octave.

After identifying all the extrams, we want to remove the *unstable* ones, i.e. those that are edge pixels and/or in areas of low contrast. To do this:

- Find edge pixels use Matlab's *edge* function. This will return a binary image (where a value of one indicates that the pixel is an edge pixel). Use that (perhaps along with Matlab's *find* and *setdiff* functions) to eliminate extremas that are also edge pixels.
- We will also eliminate extremas that are too close to the border of the image. You can determine what "too close" means, but your choice will likely be related to your descriptor decision in Part 5 (and how large of a region around they keypoints you'll use to form the descriptors).
- Finally, for each remaining extrema, compute the standard deviation of a patch around it. If this standard deviation is less than some threshold, then the patch has low contrast and thus should be eliminated from the extrema list. Once again, you can decide on the size of the patch and the threshold based on experimentation.

For your report, provide two images for each input image. One with **all** the extremas superimposed on it (indicated by red circles), and one after unstable extremas were removed. As an example, see Figures 4-5.

Figure 4: All extrema points

Figure 5: Pruned extrema points

5 (10 points) Keypoint Description and Matching

For each remaining extrema/keypoint in each image, we'll want to extract a descriptor and then match the descriptors from one image to ones in the other. To compare keypoints, you will have to determine what distance or similarity measurement to use. Common distance ones are Eucliden and Manhattan. Common similarity ones are Cosine, Gaussian, and Histogram Intersection.

The following sections discuss strategies for describing keypoint regions (descriptor extraction) and keypoint matching.

5.1 Descriptors

Given the constraints/assumptions of the problem, describing a patch around a keypoint using the RGB values will likely work well (since it encodes both color and positional information). Thus, if we had 9×9 region around a keypoint, we could describe that keypoint with a vector of size $9 \times 9 \times 3 = 243$ values. However, feel free to experiment with other descriptors (SIFTs, Local Histograms, Local GISTs, etc..).

5.2 Keypoint Correspondences

To find keypoint correspondences between images, we'll make a few problem-specific assumptions:

- Correspondences should have roughly the same y value.
- The camera was rotated and/or translated right to obtain the second image.

Our general keypoint matching strategy will be:

1. For each keypoint in the first image, find the best match (using the distance or similarity measurement of your choice) in the second image that satisfies the aforementioned constraints. Call this set C_1.
2. For each keypoint in the second image, find the best match (using the distance or similarity measurement of your choice) in the first image that satisfies the aforementioned constraints. Call this set C_2.
3. Computer the set intersection of these two sets: $C = C_1 \cap C_2$.

4. Remove from C all correspondences that have a distance above some threshold (or if you use similarity, below some threshold).

For visualization (and your report), draw lines between a few matching keypoints, as seen in Figure 6.

Figure 6: Some Point Correspondences

6 (10 points) Find the Transformation Matrix via RANSAC and Stitch

Finally we want to use the keypoint correspondences to compute a transformation matrix that we can then use to auto-stitch our images.

However, as you may have noticed, many of the point correspondences might not be correct :(. So instead we'll use a RANSAC *RANdom SAmpling Consensus* strategy.

To perform RANSAC for our panoramic stitching:

1. For experiments 1 through N (you choose N)
   (a) Select four correspondences *at random*.
   (b) Compute the transformation matrix using these correspondences.
   (c) Using the discovered transformation matrix, count how many point correspondences (among **all** of them) would end up within a few pixels of one another after projection.
2. Keep the transformation matrix the resulting in the largest number of point correspondences (among **all** of them) that ended up within a few pixels of one another after projection.

Now use this transformation matrix to stitch your images!

In your report:

- Draw lines between the keypoint coorespondences used to computer your final transformation matrix. See in Figure 7.
- Your final stitched image.

Figure 7: Point Correspondences for final transformation matrix

7 (12 points) Additional Tests

For the remaining points we'll test your code against three other picture pairs. You will get 0-4 points for each, depending on how well they stitched together.

Submission

NOTE: that 8 points of your grade is based on being able to run your code easily.

IN ADDITION: With your your submission, if you worked with someone else, let me know how evenly the work was split. If each contributed evenly it would be 50/50. I will use this information to adjust grades for pairs where one partner did more of the work.

For your submission, upload to Blackboard a single zip file containing:

1. PDF writeup that includes:
   (a) Visualization for Part 1
   (b) Stitched image for Part 2
   (c) Visualization for Part 3
   (d) Visualization for Part 4
   (e) Visualization for Part 5
   (f) Visualization and stitched image for Part 6
2. A README text file (not Word or PDF) that explains
   - Features of your program
   - Name of your entry-point script
   - Any useful instructions to run your script.
3. Your source files