Homework 3 - Panorama Stitching, Computer Vision Fall 2018

Prof. Dr. Francesco Maurelli, Jacobs University Bremen

Handed out XX.10.2018, due XX.10.2018 23:55:00

Please use the moodle system to upload your homework (`moodle.jacobs-university.de`). The system will shut down at the deadline and homeworks will not be accepted at a later time. No other form of submission is allowed.
Please upload one single zip file, with a main pdf file which can guide the instructors through your work and the various Matlab files and images generated. Be as clear as possible.
Use of LaTeXis recommended, though not compulsory. Add your source code and result images when requested and properly reference them in the pdf file.

1 Introduction

Panoramic stitching is an early success of computer vision. Matthew Brown and David G. Lowe published a famous panoramic image stitching paper in 2007 (included in the HW material). Since then, automatic panorama stitching technology has been widely adopted in many applications such as Google Street View, panorama photos on smartphones, and stitching software such as Photosynth and AutoStitch. In this programming assignment, we will match SIFT keypoints from multiple images to build a single panoramic image. This will involve several tasks:

- Use a Difference-of-Gaussians (DoG) detector to find keypoints, returning their location and scale. (This is done for you.)
- Build the SIFT descriptor for each keypoint in an image.
- Compare two sets of SIFT descriptors coming from two different images and find matching keypoints.
- Given a list of matching keypoints, use least-square method to
  find the affine transformation matrix that maps positions in image 1 to positions in image 2.
- Use RANSAC to give a more robust estimate of affine transformation matrix.
- Given that transformation matrix, use it to transform (shift, scale, or skew) image 1 and overlay it on top of image 2, forming a panorama. (This is done for you.)
- Stitch multiple images together under a simplified case of real-world scenario.

2 Building the SIFT Descriptor

Review the lecture slides on SIFT. Edit the provided `SIFTDescriptor.m` to produce a SIFT keypoint descriptor for each DoG keypoint. Note that the keypoint location and scale are provided for you, as is the Gaussian pyramid. Run the provided `EvaluateSIFTDescriptor.m` to check your implementation.

3 Matching SIFT Descriptors

Edit `SIFTSimpleMatcher.m` to calculate the Euclidean distance between a given SIFT descriptor from image 1 and all SIFT descriptors from image 2. Then use this to determine if there is a good match: if the distance to the closest vector is significantly (by a factor which is given) smaller than the distance to the second-closest, we call it a match. The output of the function is an array where each row holds the indices of one pair of matching descriptors.

Hints: Remember, Euclidean distance between vectors a and b is

$$\sqrt{(a[1]-b[1])^2+(a[2]-b[2])^2+...+(a[n]-b[n])^2}$$

You can calculate this entirely with matrix math if you use the `repmat` command. You may want to also read about the `[Y,I] = sort(...)` version of the `sort` command. Run the provided `EvaluateSIFTMatcher.m` to check your implementation.

4 Fitting the Transformation Matrix

We now have a list of matched keypoints across the two images! We will use this to find a transformation matrix that maps an image 1 point to the corresponding coordinates in image 2. In other words, if the point $\begin{bmatrix} x_1 \\ y_1 \end{bmatrix}$ in image 1 matches with $\begin{bmatrix} x_2 \\ y_2 \end{bmatrix}$ in image 2, we need to find a transformation matrix H such that

$$\begin{bmatrix} x_2 \\ y_2 \\ 1 \end{bmatrix} = H \begin{bmatrix} x_1 \\ y_1 \\ 1 \end{bmatrix}$$

With a sufficient number of points, MATLAB can solve for the best H for us. Review the included handout on Least Squares. Edit `ComputeAffineMatrix.m` to calculate H given the list of matching points. Run the provided `EvaluateAffineMatrix.m` to check your implementation.

Hints:

- You will want to convert the input points to homogeneous coordinates.
- So far in this class we have used a transformation matrix H to transform a single column vector x into a result column vector $b : Hx = b$. But notice that you can also pack multiple column vectors $x_1, x_2, ...$ side-by-side to form a matrix X, and the transformation matrix will work on each column to produce resulting column vectors $b_1, b_2, ...$, packed into a matrix $B : HX = B$. (You may want to go through a sample matrix multiplication to prove this to yourself.)
- The MATLAB "backslash" command, as discussed in the handout, requires an equation in the form $Ax = b$, where the x is the unknown matrix. In our equation above, $Hp_1 = p_2$, the H is the unknown matrix, so it is not in that form. But we can get it in that form with algebra. Take the transpose of both sides:

$$(Hp_1)^T = p_2^T$$

and distribute the transpose:

$$p_1^T H^T = p_2^T$$

Now we can use MATLAB to solve for H^T .

5 RANSAC

Rather than directly feeding all of our SIFT keypoint matches into `ComputeAffineMatrix.m`, we will use RANSAC ("RANdom SAmple Consensus") to select only "inliers" to use to compute the transformation matrix. In this case, inliers are pairs whose relationship is described by the same transformation matrix. We have implemented RANSAC for you, except for the cost function which determines how well two points are related by a given matrix H. Edit the `ComputeError()` function in `RANSACFit.m` to find the Euclidean distance between Hp_1 and p_2:

$$\left\| \begin{bmatrix} x_2 \\ y_2 \\ 1 \end{bmatrix} - H \begin{bmatrix} x_1 \\ y_1 \\ 1 \end{bmatrix} \right\|_2$$

where $\|\|_2$ is the Euclidean distance, as defined in part above. After you finish `RANSACFit.m`, you can test your code by running `TransformationTester.m`. You should get very similar results to Figure 1.

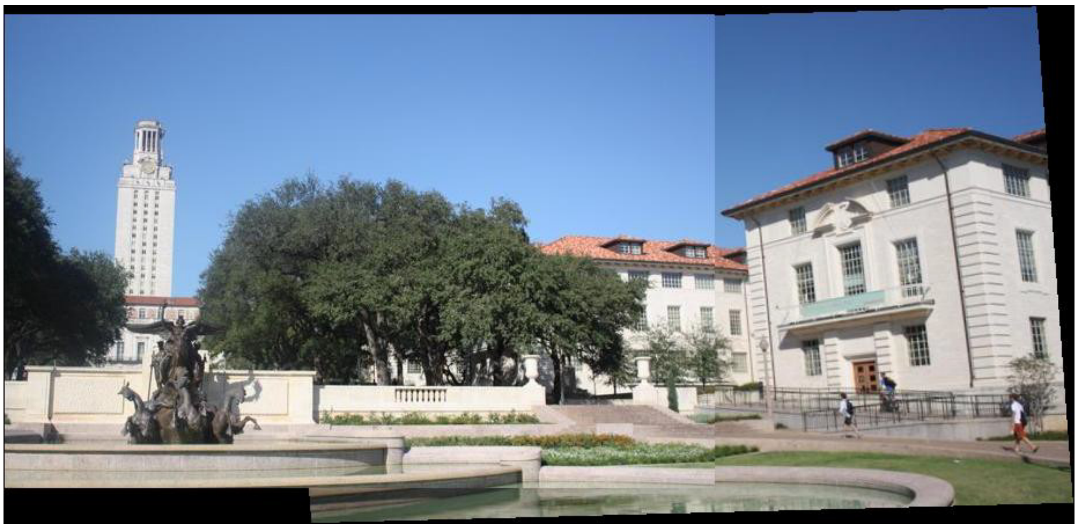


Figure 1: Stitched result from two images

6 Stitching Multiple Images

We have provided a function which uses the code you have written to efficiently stitch an ordered chain of images. (i.e. multiple photos are taken on a line, and each photo captures part of the final panorama. For example, see Figure 2.)

Given a sequence of m images $Img_1, Img_2, ...Img_m$ our code takes every neighboring pair of images and computes the transformation matrix which converts points from the coordinate frame of Img_i to the frame of Img_{i+1}. (It does this by simply calling your code on each pair.)

We then select a reference image Img_{ref}, which is in the middle of the chain. We want our final panorama image to be in the coordinate frame of Img_{ref}. So, for each Img_i that is not the reference image, we need a transformation matrix that will convert points in frame i to frame ref. (MATLAB can then take this transformation matrix and transform the images for us.) Your task is to implement the function `makeTransformToReferenceFrame` in `MultipleStitch.m`. You are given the list of matrices which convert each frame i to frame $i + 1$. You must use these matrices to construct a matrix which will convert the given frame into the given reference frame.

Hint:

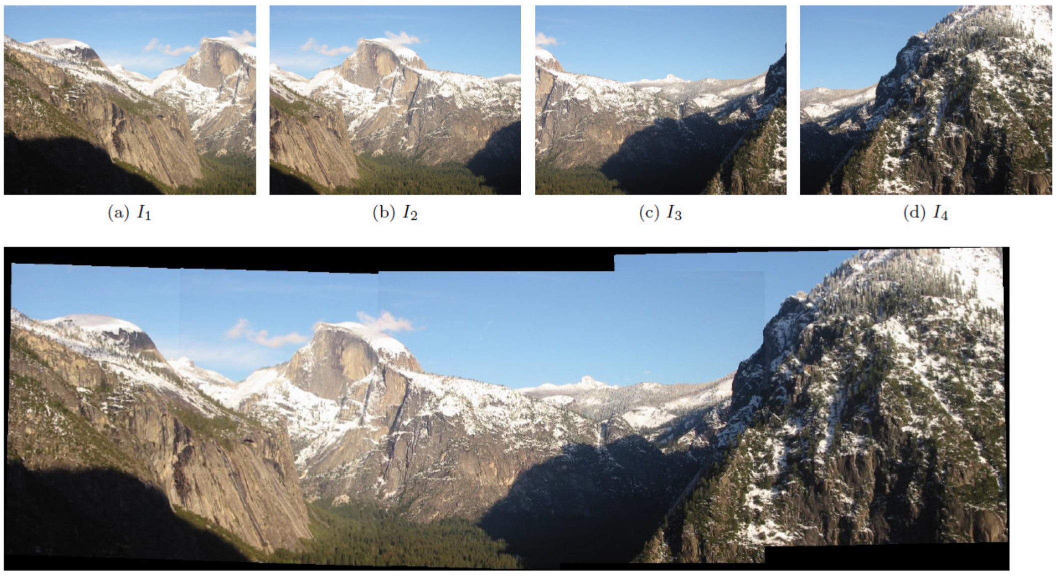
- If you're confused, you may want to review the Linear Algebra review slides on how to combine the effects of multiple transformation matrices.
- The inverse of a transformation matrix has the reverse effect. Please use Matlab's `pinv` function whenever you want to compute matrix inverse. `pinv` is more robust than `inv`.

After finishing this part, you can check your code by running `StitchTester.m`. You should get very similar results to Figure 2.

7 Further Experimentation

Use `StitchTester.m` to produce more panorama images. Edit the input filename at the top of `StitchTester.m` to use these image sets, which are provided under the data folder. Explain any errors in the algorithm or imperfections in the results. (If you get an out-of-memory error, temporarily reduce the "`maxHeight`" variable in `StitchTester.m`, and note in your report that you did so.) Run the algorithm on these image sets:

- Easy examples: `yosemite*.jpg` and `tree*.jpg`.
- Difficult examples: `campus*.jpg` and `pine*.jpg`.
- An image set of your own that you create. Put them in the data folder and name them like the sets we provided: `Name01.jpg`, `Name02.jpg`, ... , where adjacent images overlap.


(a) I_1 (b) I_2 (c) I_3 (d) I_4

(e) Final stitched panorama

Figure 2: Four Yosemite images taken on a line and nal stitched panorama

8 Report

In your report, be sure to highlight all your steps and to provide both your code and the generated images. Additionally:

- Briefly describe what makes SIFT invariant to scale, rotation, and some changes in illumination.
- Experimentally, SIFT can match keypoints which have slightly different shapes in two images. (For example, if part of a 3D object is photographed from a slightly different angle.) Simple pixel-matching algorithms, such as cross-correlation, usually fail in such cases. In addition to the invariances above, what else makes the SIFT descriptor robust to slight changes in an object's shape? (Think about how you made the descriptor, and how it's different than just comparing scaled/rotated pixels.)
- We use a Difference of Gaussian feature detector to choose where we calculate SIFT keypoints. Describe several benefits of this, compared to just choosing points in a uniform grid. (For example, we could calculate a SIFT descriptor every 5 pixels over the whole image, and try to match those with similar grid points in another image. Discuss drawbacks to that approach.
- After we made a list of matching keypoints between two images, what was the benefit of using RANSAC to fit the transformation matrix? (The things you learned in HW2 may help you answer this.)
- Discuss why the image-stitching algorithm had problems with some image sets. (If RANSAC failed completely, look at the images to see why. Think about what RANSAC is trying to find.)