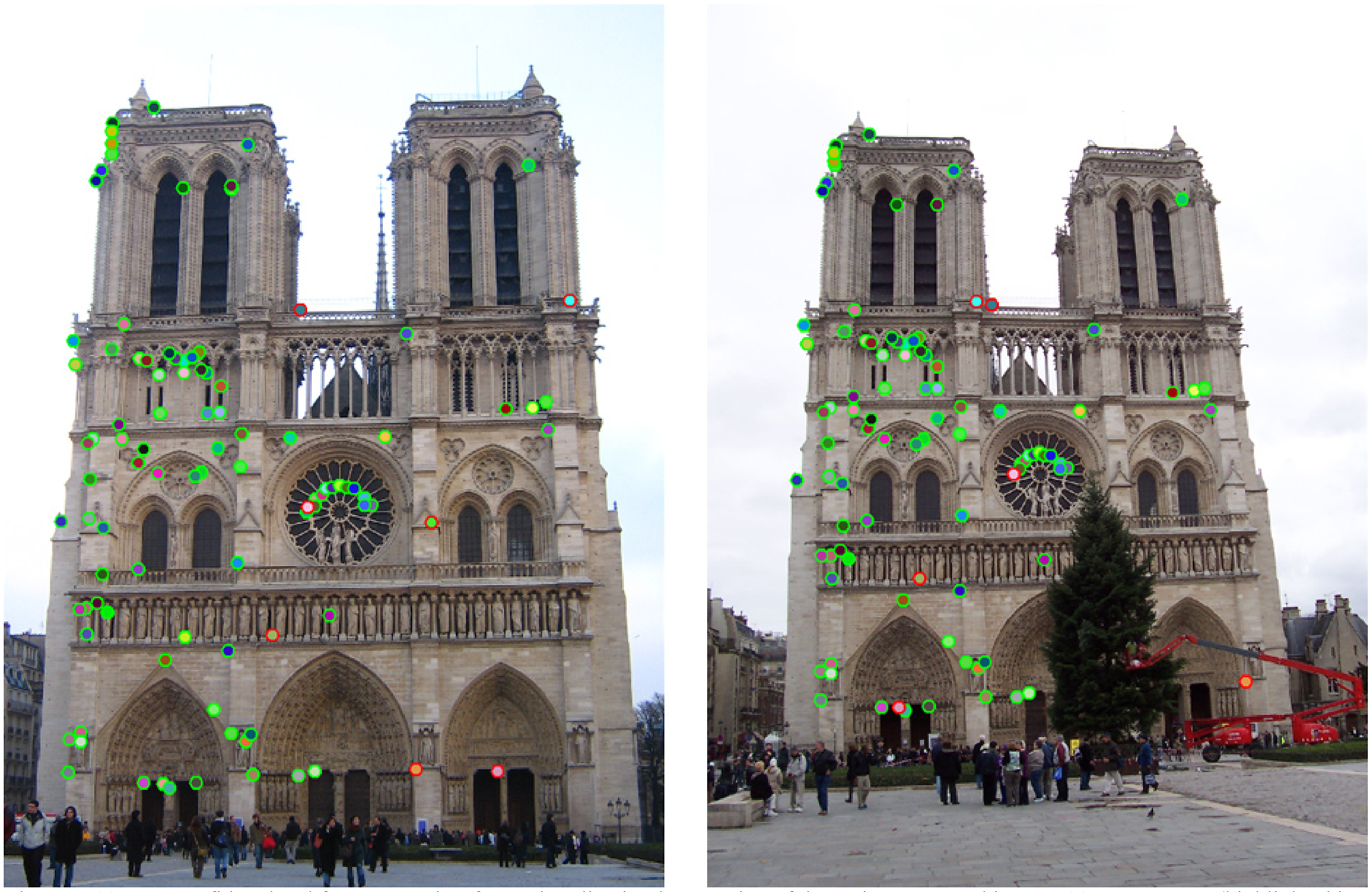

The top 100 most confident local feature matches from a baseline implementation of the assignment. In this case, 93 were correct (highlighted in green) and 7 were incorrect (highlighted in red).

Comp408/508 - HW Assignment 2

Feature Detection and Matching

Due date: 4 November, Friday

Overview

The goal of this assignment is to design a feature detection and matching algorithm using techniques described in lecture slides and Szeliski chapter 4.1. For this purpose you will implement a simplified version of the well-known SIFT pipeline. The matching pipeline is intended to work for *instance-level* matching -- multiple views of the same physical scene. Your implementation will first find corners (keypoints) on a pair of images and then match them. You can find test images and the starter code here.

There is also an additional written part of the assignment on PCA.

Details

You need to implement the three major steps of a feature matching algorithm:

- Interest point (corner) detection in `get_interest_points.m` (see lecture slides and Szeliski 4.1.1)
- Local feature description in `get_features.m` (see see lecture slides and Szeliski 4.1.2)
- Feature Matching in `match_features.m` (see see lecture slides and Szeliski 4.1.3)
- (Bonus) Scale-invariant detection and matching (see lecture slides and Szeliski 4.1)

There are numerous papers in the computer vision literature addressing each stage. For this project, we will suggest specific, relatively simple algorithms for each stage. You are encouraged to experiment with more sophisticated algorithms!

1. Interest point detection (`get_interest_points.m`)

You will implement the Harris corner detector as described in the lecture slides and Szeliski 4.1.1. You may try different alternatives as corner response function (given in the lecture slides). Use color gradient. To compute gradient, you can use Prewitt or Sobel masks available in Matlab. While computing the correlation matrix, use Gaussian or equal weighting over a neighborhood (try 3x3 and 5x5). The starter code gives some additional suggestions. You do not yet need to worry about scale invariance for your baseline Harris corner detector. So you can process all pixels at the given scale of the images.

2. Local feature description (`get_features.m`)

You will implement a SIFT-like local feature as described in the lecture materials and Szeliski 4.1.2. See the placeholder `get_features.m` for more details. For this part, you may try both intensity and color gradient but actually intensity gradient should be enough (you can compute the intensity I for every pixel by averaging the color values, hence I = (R+G+B)/3). Don't worry about the scale and orientation invariance at this stage (so you can consider the first and simpler version of the SIFT algortithm given in lecture slides).
If you want to get your matching pipeline working quickly (and maybe to help debug the other algorithm stages), you might want to start with normalized patches as your local feature, that is, your feature vector for each interest point will simply be composed of image intensities in the neighborhood (in this case intensity normalization can help by subtracting from each pixel's intensity the mean intensity of the neighborhood and then dividing the result by the variance of the intensities in the patch).

3. Feature matching (`match_features.m`)

At this step, you will match the keypoints (corners) detected on a given pair of images, using `get_interest_points.m` and `get_features.m` methods that you already implemented. Simply compute all pairs of distances between all resulting features, and then use the "ratio test" method of matching local features as described in the lecture materials. You may use the following strategy:

For each feature in one (reference) image:

1. Find the best match in the other (test) image by finding the one with the smallest distance in between. This is called the SSD (sum of squared differences) distance.
2. Then compute (score of the best feature match)/(score of the second best feature match). Use a threshold on computed score. This is called the "ratio test". Based on the result of the ratio test, you can discard ambigious matches and identify the most confident ones.

You're free to try and use other thresholding and test schemes for matching if you get better results (you have to explain it clearly in your report).

4. Scale-invariant invariant detection and matching

This is the bonus part. Implement a **scale** and **orientation invariant** version of your feature detection/description methods, as described in the lecture slides. Then match features on pairs of images, using these methods. In this case, you can implement a single matlab function `get_interest_features_scaleinvariant.m`. Here you need to construct Gaussian filtered versions of images at different scales with increasing variance, then find Harris corners on each of these, and keep only those exhibiting also a local maximum across scale considering their normalized LoG responses (see lecture slides, and maybe also Mikolajczy'01). Note also that if you are detecting keypoints at multiple scales, you should build the features at their corresponding best scales.

Using the starter code (`proj2.m`)

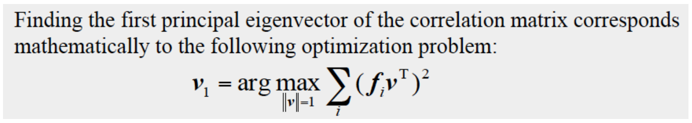
The top-level `proj2.m` script provided in the starter code includes file handling, visualization, and evaluation functions for you as well as calls to placeholder versions of the three functions listed above. Running the starter code without modification will visualize random interest points matched randomly on the particular Notre Dame images shown at the top of this page. For these two images there is a ground truth evaluation in the starter code, as well. `evaluate_correspondence.m` will classify each match as correct or incorrect based on similarity to hand-provided matches (run `show_ground_truth_corr.m` to see the ground truth annotations). The starter code only includes ground truth for this image pair, but you can create additional ground truth matches with `collect_ground_truth_corr.m`, although it is a tedious process.

As you implement your feature matching pipeline, you should see your performance according to `evaluate_correspondence.m` increase. Hopefully you find this useful, but don't overfit to these particular (and relatively easy) images. The baseline algorithm suggested here and in the starter code will give you full credit and work fairly well on these Notre Dame images, but additional image pairs can be more difficult, such as shrek_reference and shrek_test. They might exhibit more viewpoint, scale, and illumination variation. If you do the bonus scale-invariant part, you should be able to match more difficult image pairs.

Potentially useful MATLAB functions: `imfilter(), fspecial(), bwconncomp(), colfilt(), sort()`.
Forbidden functions you can use for testing, but not in your final code: `extractFeatures(), detectSURFFeatures()`.

5. PCA for color gradient (written part):

Consider this lecture slide. Show that



Finding the first principal eigenvector of the correlation matrix corresponds mathematically to the following optimization problem:

$$v_1 = \arg\max_{\|v\|=1} \sum_i (f_i v^T)^2$$

where $\boldsymbol{f}_1$, $\boldsymbol{f}_2$ and $\boldsymbol{f}_3$ correspond to the gradient vectors in R, G, B channels, respectively.
Hint: Use Lagrange multipliers technique. You may choose to refer to any document available (on the web, etc.) for PCA derivation. But actually it's very straight forward: You apply the Lagrange multipliers technique with the constraint that $\boldsymbol{v}$ is unity, and show that it is equivalent to an eigenvalue problem.

Grading

- +25 pts: Implementation of Harris corner detector in `get_interest_points.m`
- +25 pts: Implementation of SIFT-like local feature in `get_features.m`
- +25 pts: Implementation of "Ratio Test" matching in `match_features.m`
- +25 pts: (written part) PCA for color gradient
- +25 pts: Scale and rotation invariant implementation (Bonus)

Report

For this assignment, and all other assignments, you must prepare a report in pdf. In the report you will describe your algorithm and any decisions you made to write your algorithm a particular way. Then you will show and discuss the results of your algorithm. You can include the written assignment (part 5) into this report as well (you can simply write and then scan).

In the case of this assignment, show how well your matching method works not just on the Notre Dame image pair, but on additional test cases. For the Notre Dame images, you can show eval.jpg which the starter code generates. For other image pairs, there is no ground truth evaluation (you can make it!) so you can show vis.jpg instead. A good report will assess how important various design decisions were, like "by using SIFT-like features instead of normalized patches, I went from 70% good matches to 90% good matches". This is especially important if you do some of the bonus part. You should clearly demonstrate how your additions changed the behavior on particular test cases. You can find three test pairs in the homework files provided. Include in your report the results you obtain on these images and discuss your matcher's performance.

Submission

Provide comments where appropriate in the source code. Good programming style will also be taken into account while grading.

You will submit the source files (m files) along with the report. Please try to comply with the announced due date.
You can send the files via e-mail to the course TA (and to me as carbon copy).

Credits

This assignment is based on the project developed by James Hays and Derek Hoiem for a similar computer vision course at Brown University.