Assignment #2
CS510 Spring 2013
"Matching Across Identity and Pose"
Preliminary Report Due Monday, March 4th
Final Report Due Friday, March 8th

Introduction

The first assignment familiarized you with OpenCV and image correlation. This assignment finds out what you have really learned so far: can you match images across variations in object type and pose? You will be given two sets of images, a training set (which is also your gallery set) and a test set. The first task is to write a program that compares a novel test image (from the test set) to the gallery images and returns the filename of the closest gallery image. In the process of developing your program, you may do whatever you like with the training images (some suggestions are below). You may not, however, look at or process the test images in any way.

Your second task is to analyze the performance of your system and write a report. How many test images does it succeed on? How often does it fail? More importantly, when it fails, why does it fail, and when it succeeds, why does it succeed? Speed also matters: how long does your program take to run? (please limit yourself to a single processor: no parallel constructs). If you tried several variations during development, what caused you to select one version over the others? Teach me (or in general the reader) what you learned.

Half of your grade depends on the performance of your program. The other half depends on the quality of your report. To help you along the way, you will write a draft report and submit it to me on Monday, March 4th. I will read it and give you feedback by Wednesday, March 6th in order to help you with your final report. (The drafts are mandatory, but will not be graded.)

Data

The images are taken from the COIL database. These are images of single objects against black backgrounds, so image matching techniques work well. However, there are 12 different objects and the objects are on a black turntable, so they are seen from different viewpoints (but not different scales). In particular, the training/gallery set contains images of every object taken 10° apart. The test images are of the same objects, but taken half way between the test images (i.e. 5° from a test image). Thus there are two correct gallery matches for every test image. Given

twelve objects and gallery images every 10°, the odds of selecting a correct match at random are 1 in 216.

Strategies

There are many ways to tackle this problem. Your task is to find the best one you can. Here some suggestions (but don't limit yourself to these ideas; they are just to get you going).

- Brute force: compute Pearson's correlation between the test image and every gallery image; the gallery image with the best correlation to the test image is your match. This may be slow, but it should perform OK. You wouldn't want to submit a program that was both slower and less accurate than this.
- Nearly brute force: compute linear correlation between the test and gallery images, and then compute Pearson's correlation only for the N best candidates. It won't match better than the brute force approach, but should be faster.
- PCA (version 1): Apply PCA to the training set, discard the eigenvectors with low eigenvalues, and project the training set into PCA space. Project the test image into PCA space, and select the gallery image that is closest to the test image.
- PCA (version 1b): Apply PCA to the training set, look at the eigenvectors, and discard the eigenvectors that don't seem to mean anything. Then project and match as above.
- PCA (version 2): Divide the gallery by object type and compute a PCA space for each object type. Match the test image to the gallery images using the PCA space of the gallery image. Question: this will give you one image for each object type. How do you select the best? Hint: look at the magnitude of the test image projection into each eigenspace.
- Edge matching: Use the Sobel masks to compute edge strength images, and apply any of the techniques above not the test and gallery images, but to the edge images derived from the test and gallery images.
- Log/Polar matching: The objects are pretty much centered; resample the image in log polar space, and use correlation to find the rotation and scale that give the best match. (Matches with large scale changes can be dismissed.)
- Match refinement (version 1): Even the best match is 5° off of the test image. If you have the right match, you should be able to find an affine transformation that improves it. Find 3 corners on the test image, match them to corners on the gallery image, compute the affine transformation, apply it to the test (or gallery) image, and see if the correlation score improves.
- Match refinement (version 2): Search in affine transformation space using gradient descent (instead of matching 3 points). Otherwise, same as version 1 above.

But don't limit yourself to these ideas. Think of other ways to use the techniques we have discussed in class. Even if they don't work, you can get credit in your report if you tell me what you tried *and can explain why it didn't work.*

Submission

Since there are only five students in the class, you will submit your assignment by emailing it to me (draper@cs.colostate.edu). Send me an email with Assignment #2 in the heading and attach a single tar file to it. The tar file should contain all of your source code, a README file, and your report in pdf format. For the draft report, just email me the pdf file (no code required).

The draft report is due Monday, March 4th. The full assignment (including the final report) is due Friday, March 8th. Files not emailed by the 4th/8th with not be accepted (and receive a score of 0). There is always an exception for unforeseen emergencies (e.g. death of the family member, severe illness, etc.), in which case see the instructor. Note that machine/disk crashes do NOT count as a unforeseeable emergency.

Restrictions

To a certain extent, the grading on this assignment will be comparative. Therefore, you would be silly to work with anyone else on this project. You may not submit any work that is not your own (so no copying code from the internet, etc.). See the department's honor code for more details.

You may write your code in C, C++ or Python (but no other languages). You may use the OpenCV library. Students who write in C++ may use the Boost filesystem library to manipulate files and directories. No other non-standard libraries may be used. (If you are unsure whether a library is standard, ask me.)

Hints

OpenCV includes functions for PCA, edge detection, corner detection, correlation, and more. Exploit them. You should not have to re-implement basic techniques, simply learn how to combine them.

Just because you are experimenting, don't forget basic software development principles like unit testing your code. It is amazing how often people give up on good techniques because a bug in their code convinces them the techique doesn't work.

Visualize your results, including as many intermediate stages as possible. (Again, OpenCV makes this easy.)