CS400 — Problem Seminar — Fall 2000
Assignment 2: Face Orientation Detection

Handed out: Sept. 13, 2000
Due: Sept. 27, 2000
TA: Andrea Selinger (`selinger@cs`)

1   Introduction

Face recognition, face detection, and gaze detection all have obvious applications. Think about security, adaptive user interfaces, photo-finishing, realistic animation, monitoring a user's attention or alertness or mood .... The computer vision community has been pursuing face-processing topics for a quarter of a century.

For example, the 1996 *Proceedings of the Conference on Computer Vision and Pattern Recognition* has a section on Face Recognition, as do many other conference proceedings past and present. There are some commercial face-recognition products but there remains a lot to be done and activity is intense. Some of these articles could be of interest to you for this project. Don't be afraid to use the web or library to search for ideas!

In the future, many computers will probably come with TV cameras and microphones attached. We have such setups in the department. One application is video-telephony—communicating with other humans over the Internet. Another application is communicating with the computer. In the TRIPS dialogue system developed here, the user simply speaks to the computer, but video input could help in several ways. One might try to use lip reading to improve the accuracy of speech recognition (as in recent research at IBM's India Research Lab). Or one might try to use images of the user's eyes to figure out what the user is looking at on the screen (as in Dean Pomerleau's gaze detection system at CMU). For this assignment, we will consider a simpler question: Is the user looking at the computer (or the camera) at all? If not, the system should ignore any audio input, because the user is probably having a conversation with someone else![1]

---

[1]At present, the TRIPS system requires the user to hold down a "talk" button while speaking to the computer. This is less natural than just looking at the computer.

2 Assignment

This assignment is designed to get you familiar with the basics of pattern recognition and computer vision. You will need to think about the problem description, applications, and possible approaches, and then design, implement and test a solution. Since none of these is perfectly well-defined, the thinking is as important as the coding.

We will assume grayscale (not color) images. The face orientation detection problem has three levels of complexity:

1. **User-dependent.** You are allowed to use one or more "reference images" of the user. For example, build a system with a "calibrate" key. When the user sits down at the computer in the morning, she presses this key and is photographed as she looks that day when facing the camera. You now have to analyze images of her later that day to determine whether she is again facing the camera.

2. **User-independent.** There is no calibrate key. The system must be able to work on new faces that were not used during system development. In other words, it must solve face orientation in general, not just for particular faces that it has learned.

3. **Efficient.** Solve the above problems fast enough for use in an online system with video input. (Assume you have access to a continuously-digitized image that appears in some buffer in memory. If you get something reliable running that you'd like to test in continuous operation, tell the TA and we'll set you up.)

You must submit a report describing your approach and evaluating its success on each of the three levels. You are expected to develop a program that handles at least one of these levels. If it handles only the user-dependent case, you are also expected to make some progress on the user-independent case; if you can't get the user-dependent method to extend gracefully, explain why in your report. If you are already familiar with computer vision and find the assignment straightforward, you are strongly urged to work with the TA on the live, online version.

Your report should illustrate your methods, including some images of *you*. Try to show visually what your method does to a given photograph. For example, if you have routines that detect eyes, edges, or symmetry, then include a modified version of the photograph where these things are marked. (Obviously, you will also want to create and study such images to help you develop your system in the first place.)

Your report should also honestly describe any limitations of your method, and it should suggest future work.

As always, do not leave the report to the last minute. A well-written report on unsuccessful (but wholehearted) efforts will be preferred over the greatest code in the world poorly described. Mathematical or algorithmic elegance is a plus.

3   Data and Evaluation

First take some photographs to experiment with. Quickly learn how to make images in the vision laboratory, view them, and convert them from one format to another. (This is easy.) You are urged to make a variety of images, and pooling images created by different class members would probably be A Good Idea.

You should probably start by experimenting with some simple low-level techniques on these photographs to get a sense of what is likely to be useful. See the next section for suggestions. Keep a log of the things you try; it will help you when writing your report.

You are also welcome to draw cartoon faces using xfig and experiment with those. (Again, sharing images would be smart.) You may learn something that can be applied to the photo problem. Of course photos are harder: even if you use techniques such as edge detection to make them more cartoonish, there will always be some residual noise.

On Sunday, Sept. 24, three days before the assignment is due, you will be given some data that you should use for a formal evaluation: 3 faces, each photographed in at least 20 orientations. You should test your program on these photographs—simulating the "calibrate" key if appropriate—and include the results in the report. The point is that this is a fair test, since you presumably will not have seen the photos while developing the system. (If you modify the system and test again on the same data, you must say so in your report.)

If you would like a collection of similar data ahead of time for purposes of development—for example, to run an automatic training procedure, or just to validate your techniques—ask me and I'll be glad to provide some. But start by using your own photos because they will be more varied.

4   Approaches

One approach to the face orientation detection problem is to first analyze the content of the image, probably in terms of characteristics and relative positions of interesting "face-related" features in the image. (You are responsible for inventing and finding image features that will differentiate the two classes.) Then, using the extracted features and their relationships, make a decision about the face orientation.

Note that one could design the decision criterion here by hand: use a hand-built decision tree, match for an expected geometrical structure (Ballard and Brown, Chapter 11), compute a simple number (say a symmetry metric), and so on. Or one could use a Pattern Recognition paradigm (Ballard and Brown, Chapter 6.4) and "train" the program on example faces of both types. Then, for a new face, compute the features and see to which cluster of previously-classified feature-vectors the new one is closest. To go even further toward automatic learning, you could take a "neural net" approach in which even the features themselves are learned—the training procedure tries to derive class-differentiating features automatically.

Or do something completely different. The choice of the method is up to you: if you want guidance see your TA, Brown, Ballard, Nelson, or any visionary grad student who has taken CSC449.

Although this is a binary decision problem (*i.e.*, it requires a yes/no answer), you can qualify your answer with certainty factors or whatever other information you can glean from the image. Keep in mind the sort of intended application of this technology when designing your system. Be sure to discuss these issues and applications in your report.

Feel free to make simplifying assumptions about hair, skin color, glasses, background, lighting, whatever you need to make progress, and document these in your report.

Remember, don't leave the report for the last minute.

5   Resources

Computer vision is a huge field with a widely dispersed literature and many techniques that one is not born knowing. You are not expected to know anything in advance about operating on visual data, but you are expected to find out what you can about relevant techniques, to formulate algorithms to solve the problems above, to try them out experimentally, and to report your results and thoughts about the problem.

Images come in many formats. If you want to manipulate images directly, I recommend converting them to the very simple PGM format, which is basically just a 2D array of grayscale levels. `man -k anymap` will tell you about the file format, utilities and C library.

Randal Nelson has a page of image processing resources for his course:

http://www.cs.rochester.edu/u/nelson/courses/vision/resources/resources.html

In particular, this page points to two libraries that can probably do most of your low-level work for you. One is IPP, Nelson's own large and powerful library of vision

routines. The other is the Vista package installed in `/u/cs449`, which provides roughly the same functionality with a more command-line interface.

Nelson's libraries live in `/u/nelson/packages/ipp_package/` and its subdirectories. Documentation is in `Readme` files in these directories, and to some extent in the code itself. Sample routines and code for finding particular features may be found in `/u/nelson/programs/src/ipp/feature/bin/`.

Some bits of Vista seem to be missing, but I was able to get it to do some useful things without much trouble, and you could download the missing pieces from the web if necessary.

In general it will pay to be familiar with the extent and semantics of these libraries to keep from reinventing the wheel. Of course, you needn't use anyone's libraries unless you find them useful—you might come up with a technique that works from scratch.

The text *Computer Vision*, by Ballard and Brown, has some useful information: how to compute useful shape features (pp. 254-261, 376-377), how to encode high-level information about assemblages of shapes, including their individual properties and their geometric relations (Chapter 11). Haralick and Shapiro's *Computer and Robot Vision Volume II* has a useful glossary of computer vision terms, and some chapters on object models and matching.

To show you how to interface with Nelson's libraries, we are providing (thanks to Mac McCauley) a sample program and associated `Makefile` that takes an input binary image, finds connected components (blobs), and uses the bounding boxes computed by the routine to decide if the blob is a "horizontal" or "vertical" one. The directory containing this simple sample application is:

<div align="center">

`/u/jason/teach/cs400/hw2/sample_code/`

</div>

Feel free to copy it and modify it.

Warning: The sample program itself will probably *not* do helpful things on raw grayscale images, especially because it only finds blobs of pixels with *exactly* the same color or gray level. We are providing it only as an example of how to call Nelson's libraries. The `Makefile` in particular may be useful as a description of how to compile and link. If you do want to find blobs for some reason, track down the documentation for the functions used in the sample program; it mentions what to do instead if you have grayscale images.