CS377S
DESIGNING APPLICATIONS THAT SEE
Computer Vision for HCI

ASSIGNMENT #3: APPLYING COMPUTER VISION TECHNIQUES

Due February 26, 2008 (in lecture)

Reflection | Ideation | Exercise | Bonus Challenge

The Right Algorithm for the Right Job *(6 Points)*

At the beginning of the quarter you heard the assertion that "computer vision is not actually that hard; rather, the problems are ill-defined." In other words, there is no single formulation of "the computer vision problem." We are a long way from total image understanding, so instead of a generalized approach we have a broad variety of techniques, each of which works well only for specific tasks. Frequently, the key to solving a particular problem is understanding the constraints of the situation and picking the appropriate technique to meet your application's demands.

Over the past month we have looked at a wide variety of computer vision algorithms. Some of these techniques are listed in the left column of the table below. The right column contains a list of problems that could be solved with computer vision. For each problem description, pick the most appropriate algorithm from the list, and write one or two sentences justifying your decision.

1) Scale Invariant Feature Transforms (SIFT)	a) Detect cuts (sudden scene changes) in a movie or TV show (for example, the beginning of a commercial)
2) Cascade of Boosted Classifiers	b) Navigate a robot along a straight sidewalk by detecting the sidewalk edges and staying within them
3) Edge Finding and Hough Transform	c) During a tennis match, follow the position of the ball using a camera positioned above the court
4) CAMSHIFT Color Histogram Tracking	d) Given the cover images of ten different DVD movie cases, detect when one of the cases is held up in front of the camera
5) Adaptive Background Subtraction	e) Detect when cars pull into and out of a parking spot over the course of the day
6) Frame Differencing	f) Detect the presence of weapon-shaped objects in an airport X-ray scanner

Project Brainstorming *(8 Points)*

The purpose of this exercise is to develop some ideas for our in-class brainstorming session. The in-class brainstorming session will give you an opportunity to share project ideas with the other students in the class and team up with project partners who are interested in pursuing similar ideas.

Based upon your interests, the previous assignments and course readings, and your knowledge of the available computer vision tools, develop four distinct ideas for class projects. These do not have to be fully-formed ideas, just general, high-level concepts. Each project idea should be one that you would be interested in pursuing, and each idea should be different.

For each concept, write down a one-sentence description on 3"x5" index card, along with an explanatory sketch. On the back of the card, write down your name and the following additional information:

- A slightly more detailed summary of the idea
- Which tools you think might be most appropriate for the project
- A sentence or two on how the computer vision component of the project might work. Don't worry too much about this just yet – the brainstorming phase is too early to let technology constraints get in the way of interesting ideas!

Here are some examples of what your cards might look like:

Trashcan opens automatically!

55
ROAD TEDDY
CAM
Teddy bear in car warns you before running red light.
WATCH OUT!
STOP
ROAD TEDDY
CAM

Bring your idea cards to class on Thursday, February 14th. Be prepared to present a quick summary of your favorite ideas to the class. You will later hand in your index cards along with the rest of this assignment.

Programming in OpenCV (*8 Points*)

In this exercise we will try out a few different OpenCV functions to attempt to track an object's position. We will use the same video of a swinging ball that we used in Assignment #2. As before, you should download the video `ball.avi` from http://cs377s.stanford.edu/assignments/.

a) First we will try color histogram tracking using the CAMSHIFT algorithm. Load and compile the example program `camshiftdemo.c` in the `OpenCV/samples/c/` directory. This tracking demo is designed to be run either on a live camera input (if no command line arguments are provided) or on a video sequence (if a video file is specified on the command line). To run it on our sample video, drag the file `ball.avi` onto the icon for `camshiftdemo.exe`, or simply type `camshiftdemo.exe ball.avi` at the command line. You will see the video load and quickly cycle through its frames. The program will not actually try tracking an object in the video until it has been selected with the mouse.

   Find the line of the code that reads

```
c = cvWaitKey(10);
```
and replace it with
```
c = cvWaitKey(0);
```
so that the video will pause on each frame, allowing us to step through the video more slowly. Recompile and run the program, again loading `ball.avi`. Now you can use the return key on your keyboard to step through the frames one by one. Click and drag the mouse across the ball in the video window, and upon advancing the video to the next frame, you will see an ellipse appear over the ball's location. Selecting an image region like this sets the starting position of the search window for the CAMSHIFT algorithm, and builds a color histogram of the selected region to track (this histogram should appear in another window). Continue stepping through the video, and follow the position of the ellipse as the tracking algorithm attempts to rediscover the ball in subsequent frames.

Modify the code so that OpenCV automatically writes out every 100th frame of the video as a JPG image. Include the five resulting output images in your assignment hand-in. How well does CAMSHIFT follow the ball? What causes the algorithm to break down?

b) Next we will try adaptive background subtraction. Load and compile the example program `bgfg_segm.cpp` from the `OpenCV/samples/c/` directory. This program creates a statistical model of the background that updates over time. Note that because the program uses some functionality from OpenCV's experimental CVAUX library, you will need to link it against `cvaux.lib` for a successful build. Run the program using `ball.avi` as input. You should see a marked improvement in tracking performance.

Modify the code to overlay the detected foreground regions on the original input frames using the `cvDrawContours` function:

```
IplImage *fgcopy = cvCloneImage(tmp_frame);
cvDrawContours(fgcopy, bg_model->foreground_regions,
CV_RGB(0,255,0), CV_RGB(255,255,255),1,-1);
cvShowImage("FG", fgcopy);
cvReleaseImage(&fgcopy);
```

Also modify the code to save the `fgcopy` image as a JPG file every 100th frame of the video, and include the five resulting output images in your assignment hand-in.

c) Finally, we will try tracking the ball using optical flow. Load and compile `flow.cpp`, the optical flow example program from the in-class tutorial session. Change the code to load `ball.avi` and take a look at the output.

If we want to track the ball, we will need to isolate the motion of the ball from any other motion going on in the video (for example, motion of the camera, or motion of background objects). In complex cases we might build a model of the background motion and then look for motion that does not fit this model. In this case, however, things are much simpler, because the camera is relatively stable and the only movements we need to discard are the small local displacements of the background due to camera jitter and feature matching errors.

Because the optical flow algorithm is performing matches between every single pair of frames, the ball does not actually move very much from frame to frame. Modify the code so that it skips two frames in between each pair of optical flow frames, to find flow vectors with a larger feature displacement. If necessary, modify the optical flow search window or the number of pyramid levels to make sure that the ball's motion is still discovered by the algorithm.

Now compare the magnitudes of the motion vectors produced by the ball and the motion vectors produced by the background objects. Choose a minimum motion threshold, and discard flow vectors that are shorter than this threshold. Finally, average the positions of the remaining flow vectors to approximate the center of the ball. Use `cvCircle` to draw a green circle at the detected ball location. Save some sample frames as JPG images and include them with your assignment hand-in to demonstrate the performance of your flow-based tracker.

Eyepatch Introduction *(4 Points)*

This exercise will introduce you to Eyepatch, an experimental tool for rapid example-based prototyping of computer vision. Eyepatch lets you design, test, and refine computer vision algorithms without writing any code. Because it is in the early stages of development, you will most likely discover bugs. Please use the online bug-tracking tool to report any bugs you find:

http://code.google.com/p/eyepatch/issues/list

You can also use this tool to submit suggestions and feature requests.

a) Download and install the latest version of Eyepatch from http://eyepatch.stanford.edu/. As of this writing, the most recent version is 0.70. The DivX codec is included in the Eyepatch installation; to ensure that Eyepatch works correctly, you should install DivX if you do not have it already.

b) Start Eyepatch. The Eyepatch window consists of four panes:
   - When you load or record videos, they show up in the *Video* pane. You can drag the slider beneath the video to move back and forth through the video.
   - The pane to the right of the video stores *Examples* (both positive and negative) that you have extracted from the video. Eyepatch will use these examples to build a model of the types of things that you are looking for in the video.
   - The region beneath the video pane is the *Filter Preview* pane. It shows you a representation of the filter that you are currently training, and a demonstration of how the filter is performing on the current video frame.

- The bottom right pane contains the *Filter Control Panel*. It lets you specify a filter type and save, load, and rename filters.

c) We will start by loading a video into Eyepatch. Download the following video: http://cs377s.stanford.edu/code/eyepatch-tutorial/videos/postits.avi

Select "File… Open Video" and browse to the folder where you downloaded the video. Open the video and you will see it appear in the video pane. Drag the slider back and forth to browse through the video and get a feel for its contents.

d) We will begin by training a color filter. This is one of the simplest types of filters. Go to a frame containing the blue Post-It note, and left click and drag to draw a rectangle around the note. When you release the mouse button, you will see this portion of the video frame added to the Examples pane in the "Positive Examples" category.

In the Filter Control Panel select the filter type “Color.” Click “Learn from Examples” to create a color model based on the example you provided. A representation of this model appears as a hue histogram in the Filter Preview pane. You can check the “Show Guesses” box in the filter control panel to see how this filter model performs on the current frame of video. Step through the video with “Show Guesses” enabled to see how well the filter performs at different places in the video.

Try adding a picture of the green Post-It note to the positive examples and retraining the filter by clicking “Learn from Examples” again. Notice that the color profile changes in the filter preview pane, and the filter now highlights different things.

You can adjust a filter’s threshold using the slider in the Filter Control Panel. This threshold determines how selective the filter model is. The higher the threshold, the closer the match required to make a positive identification. A lower threshold will miss fewer items, but it will also result in more false positives. A higher threshold will make fewer incorrect guesses, but may miss identifying some valid items.

You can delete an example you don’t like by dragging it outside of the example pane. Remove the green Post-It note from the examples and retrain the filter to restore the blue-only model. Now click “Save Filter” and you will see the filter added to the list of saved filters in the Filter Control Panel. To load a filter from this list, double-click its name. You can also rename a filter by highlighting it, clicking it a second time, and typing a new name.

e) The brightness filter works almost exactly the same way as the color filter, using intensity instead of hue. To try it out, we'll use another video: http://cs377s.stanford.edu/code/eyepatch-tutorial/videos/phone-light.avi

Start by discarding the examples we have collected so far, since we no longer need them.

Highlight the old examples and drag them outside the examples panel to put them in the trash (If the trash starts filling up, you can empty it with the "Empty Trash" command in the File menu).

Load the new video and switch the filter mode to "Brightness." Select the bright light and you will see the filter update with a brightness profile – in this case, mostly very bright pixels. Click "Show Guesses" and look at some different frames in the video. The light should be selected in most frames.

f) The Shape filter extracts edges and contours from objects and compares them against a model set of edge shapes. This filter works best on objects with sharp, distinct edges that stand out from the background. We'll try it out on this video:
http://cs377s.stanford.edu/code/eyepatch-tutorial/videos/shapes.avi

Open this video and select the "Shape" mode. Discard the examples we were using for the Brightness filter. Go to a frame where pen appears, and select the whole pen as an example. Go through the video and select a few of the other objects as well, such as the binder clip or the fork. Click on "Learn from Examples" and you will see a picture in the Filter Preview pane showing the strong exterior edges in the examples you chose. Click "Show Guesses" and browse through the video to see how this filter performs. Notice that it can be confused by background objects, and that rotating the objects at too much of an angle will change their edge shapes and confuse the filter. Remember that you can adjust the sensitivity of the edge matching using the threshold slider in the filter control panel.

g) You can see that matching shapes does not always work so well – it is not particularly robust to changes in angle or scale, and it can be distracted by messy backgrounds. An algorithm called SIFT (Scale Invariant Feature Transforms) solves many of these problems. It runs much slower than the methods we have tried so far, so it may not be able to track objects at interactive frame rates, but it is more robust to changes in the scene or the position of the object you are tracking. Load the following video to try it out:
http://cs377s.stanford.edu/code/eyepatch-tutorial/videos/book.avi

Discard the current set of examples, and find a frame where the book cover is upright, large, and in clear focus. Select the book cover, switch to "SIFT" filter mode, and click "Learn from Examples." You will see an image of the book cover in the filter preview pane with the SIFT features marked with arrows. If you only see a few features marked, you should try to find a better example of the book cover that contains more SIFT features.

SIFT assigns each feature a canonical orientation, so that if enough features are detected and matched, we can tell where an object is and how it has been moved or rotated. Turn on "Show Guesses" and try it out by browsing through the video. Notice that while SIFT take some time to run on each frame, it manages to perform well despite the complex background, even when the book is rotated, upside down, or partially occluded. You can adjust the filter threshold to set how many SIFT features are required to constitute a match.

h) So far the methods we have looked at work well for recognizing specific objects, but sometimes you might want to identify a general class of objects. SIFT may do a good job detecting a certain car, but what if we want to recognize *any* car? For tasks like this we can use the "Adaboost" mode, which trains a classifier using machine learning. This filter takes the most work to train, since it requires a large collection of both positive and negative examples, but it can use these examples to build a more general model of the object of interest that is more flexible than the approaches we have looked at so far. We will try to train a filter that recognizes the backs of cars. Download and open the following video clip:
http://cs377s.stanford.edu/code/eyepatch-tutorial/videos/cars1.avi

Scroll through the video and draw rectangles around the backs of cars. The training works best if you draw a tight bounding box around just the object – including portions of the background will confuse the machine learning algorithm. Continue scrolling through the video and adding examples until you have at least ten car pictures.

Unlike the other filters, Adaboost also requires negative examples. Go through the video again, and this time, right click and drag the mouse to select regions of the video that do *not* contain the object you are looking for. These regions can be as large as you like, as long as they do not contain any cars. Be careful not to include even part of a car, as this can confuse the learning algorithm. Continue adding negative examples until you have at least ten negative examples.

Now click "Learn from Examples" to begin the training. Training the model takes some time; you can interrupt the training at any time, but the longer you allow it to continue, the better the model that it will produce.

i) Once the model training has completed, click "Show Guesses" to run the filter you have trained on the current frame. Browse through the video to see how your filter performs. You will probably see some false positives (things identified as cars that are not really cars) and some false negatives (cars that are missed by the filter). You can improve the quality of the filter by telling the computer where it has made mistakes. If you see a car that was not detected by the filter, left-click and select it to add it to the positive examples. If you see a region of the image without a car that was marked as a car by the filter, right-click and select it to add it to the set of negative examples. By marking these tricky cases, you will allow the computer to learn from its mistakes. After marking some mistakes, click on "Learn from Examples" again to retrain the filter model.

If you want to continue training and testing, you can download some more car videos:
http://cs377s.stanford.edu/code/eyepatch-tutorial/videos/cars2.avi
http://cs377s.stanford.edu/code/eyepatch-tutorial/videos/cars3.avi

http://cs377s.stanford.edu/code/eyepatch-tutorial/videos/cars4.avi

j) Download this video of marbles rolling across a table:
http://cs377s.stanford.edu/code/eyepatch-tutorial/videos/marbles.avi

Try train a classifier that accurately captures the marble positions. Which detection methods worked the best? Save the classifier that gives you the best performance.

k) Now download this video of a horse and trainer:
http://cs377s.stanford.edu/code/eyepatch-tutorial/videos/horse.avi

This video presents additional challenges. Try to train classifiers that can detect the position of the horse using two different classification methods. Which methods did you choose, and why? Did one perform better than the other? If so, why do you think that this is the case? Save the classifier that gives you the best performance.

l) Saved classifiers are stored in Eyepatch's application data directory, within your user directory. In Windows XP, the classifier files can be found in the directory
`C:\Documents and Settings\USERNAME\Application Data\Eyepatch`.
In Windows Vista, the location is
`C:\Users\USERNAME\AppData\Roaming\Eyepatch`.

Each classifier consists of a separate folder, with various classifier parameters stored as files within the folder. If you are unsure which classifier is which, you can look at the file `name.dat` in the classifier folder, which stores the classifier name. Upload the classifiers you trained for the marble and the horse videos to an online location, and include the URL in your assignment hand-in.

Vision Prototyping in Eyepatch (*12 Points*)

Have you ever wondered if your roommate is drinking all of the milk in the refrigerator? Perhaps you have an idea for how to improve traffic signaling by detecting approaching cars? Now that you are comfortable with training classifiers in Eyepatch, you should be able to implement a monitoring system of this variety. Use Eyepatch, together with the development tool of your choice, to develop your own surveillance/monitoring system.

a) Begin by deciding which external development tool you will use to interface with Eyepatch. Eyepatch exports its data over a local network socket using a choice of two common protocols: OSC over UDP (on port 7000), or XML over TCP (on port 8000). You can read the data streaming from Eyepatch by creating a basic socket connection on the appropriate port. We have produced sample frameworks for that demonstrate how to establish a connection with Eyepatch using three common development tools: Flash, Processing, and Visual C#. You

can download these examples from the following locations:
http://www.stanford.edu/class/cs377s/code/eyepatch-tutorial/flash-examples/
http://www.stanford.edu/class/cs377s/code/eyepatch-tutorial/processing-examples/
http://www.stanford.edu/class/cs377s/code/eyepatch-tutorial/csharp-examples/

b) You can run Eyepatch on live video input by selecting “Run Recognizers” from the Mode menu. Double-click a classifier in the list of trained classifiers to add it to the collection of active classifiers. Double-click an output, such as XML over TCP, to add it to the collection of active outputs. Finally, click “Run on Live Video” to begin streaming data using input from a webcam, or “Run on Recorded Video” to run on a prerecorded test video. You can double-click on a classifier that is currently in the active list to select which variables it will output; different classifier types are capable of producing different types of data. All of the classifiers can send the locations of detected regions, but some can produce additional data. For example, Gesture classifiers can send the index of the detected gesture, and Motion classifiers can send the average motion direction.

c) Develop a surveillance/monitoring system by sending data from Eyepatch to the development environment you chose. Be creative! Your system should notify the user when an event of interest is occurring in the video. You can notify the user using sound, visuals, email or some other communicative/interpretive method. Here are a few sample project ideas:

- Create a drawing based on the location/trajectory of the moving objects in the video,
- Create a basic security system that sends alerts to the screen when the object in question (car, face, motion, etc.) is detected.
- Keep track of how many cars drive by on the highway, and keep a count of the color of the different cars. Create a visualization for this data.
- Create a Flash demo of a stoplight control system that changes a stoplight’s display color (red, yellow, green) based on traffic patterns.
- Create an alarm for a pot on a stove. When the pot really starts to steam, send an alert to the screen that it is about to boil.
- Keep track of the number and types of birds in a scene (e.g. ducks vs. geese).
- Create a counter to keep track of the number of times the refrigerator is opened within a given period of time.
- Capture a movie of a series of dice rolls. Keep track of the distribution of numbers that occur in the resulting rolls. Are your dice really fair?

d) Write a description of your surveillance system. Which development platform did you decide to use? What is the goal of your system? What parts worked well, and what challenges did you encounter during the implementation? Were there additional features that you would have like to see in Eyepatch that would have made the task easier for you?

e) Record a video of your system at work. Put the video online and include a link to its location in your assignment hand-in.