

CSE 398/498-010

Real-time Image Processing for Autonomous Robot Systems

Laboratory 6: 2D Pattern Recognition (Part 2)

Report Due Date: 11 Nov 04



A. Objective: Demonstrate the efficacy of the Pyramid image representation through a 2D pattern recognition application.

B. Lab Procedure:

1. This lab will rely upon the pyramid data structure to DRAMATICALLY improve framerate in the 2D normalized cross-correlation based pattern matching.
2. You will use the `myCrossCorr` and `uSigmaIm` functions developed last week
3. You will also use the `cvPyrDown` function from OpenCV to build the upper levels of the pyramid. This generates a 5x5 Gaussian smoothed version of the source image that has been subsampled (aka *decimated*). Recall that the `MILImage` class has a member `_pIplheader` that you will use in making calls to OpenCV functions.
4. Be certain to `#include "cv.h"`

5. First, recompile your code in release mode (if this works) using the larger template and note the time that was required to process for the image (for a 72x64 image, mine required 8.2 seconds on a 1.7 GHz Centrino - let me know if you beat this ☺).
6. Construct pyramids levels 1-3 for both your source image and template.
7. Run each through your `myCrossCorr` and `uSigmaIm` functions. Ensure that the proper template is found in each image.
8. Note the correlation values for each level.
9. Write a new function `matchPyramid`. This will need to take the following parameters as arguments:
 - a. Image
 - b. Template
 - c. Image size
 - d. Template size
 - e. Start point for correlation matching

It must also “return” the point of maximum correlation for the current image/template pair. Within this function, you need to call both your `myCrossCorr` and `uSigmaIm` functions. However, rather than searching the entire image, you need only search in a 3x3 neighborhood around the start point.

10. You will then follow the procedure outlined in the “Pyramid Methods” lecture
 - a. Generate the Gaussian pyramid for the images and templates
 - b. Perform the 2D pattern recognition search at the highest level of the pyramid.
 - c. The output (xMax, yMax) will be the point of maximum correlation.
 - d. Let $(xMax, yMax) = 2 \cdot (xMax, yMax)$
 - e. Feed these values, along with the next lower levels of the image and pyramid templates to `matchPyramid`.
 - f. Go to step “c” and repeat until you have processed Level 0 of the pyramid.
 - g. Plot a rectangle around the pattern on Level 0 of the image.
11. Note the time that it took to locate the target in this process (mine took 0.02 seconds, including text output and displaying the images - a speedup of over 400!).
12. Once you have completed this with your static template/image pair you need to repeat the process for a dynamic target.

- a. Pick some object with a fairly unique pattern. Place this at a distance from the camera, and generate a fixed template for the object. Template dimensions should be multiples of 8 if possible.
- b. While grabbing images with the camera, *translate* the object while maintaining the same distance from the camera as was used in template generation. Does the camera successfully track the target?
- c. Do this for a sequence of images (300 or so). What framerate could you achieve? You can include `<time.h>` and use the `clock_t` class to estimate the elapsed time.

C. Report Requirements:

1. Your code.
2. Your image template used for static trials.
3. A screen shot with all levels of your pyramid and the templates matches from the static case similar to that above.
4. Your image template used for dynamic trials.
5. Saved images during a dynamic tracking run.
6. AT A MINIMUM, answers to the following questions:
 - a. The running time (in release mode) from 4 above.
 - b. The running time from 10 above.
 - c. What was the speedup?
 - d. What were the correlation values for each level as obtained in 7 above? Explain the differences in these.
 - e. How robust was the tracking to changes in scale - the distance from which the template was generated vs. the actual distance it was tracked?
 - f. How robust was the tracking to changes in orientation?
 - g. How would you improve the process performance using time as your metric?
 - h. How could you make the tracking process more robust?
 - i. Why did we use normalized cross-correlation as our metric for the “goodness of fit”.
 - j. When would it be preferable to used SSD vs. normalized cross-correlation?