Computer Vision
CSCI-GA.2271-001
Assignment 1.

October 2, 2012

1 Introduction

This assignment works through various concepts in extracting features in images before using them to align a pair of images.

Essentially there are three main parts to the assignment.

1. Harris corner detector - Implement the Harris corner detector and evaluate it on select images, demonstrating its invariance properties.

2. Scale-invariance - Explore scale-selection using Laplacian and Difference of Gaussian operators.

3. RANSAC - Solve for an affine transformation between a pair of images using the RANSAC fitting algorithm.

2 Requirements

You should perform this assignment in Matlab. Please turn in all code and result figures/images. The code should be clearly commented, explaining what each line is doing.

The TA for the class is Bowen Zhang (`bz465@nyu.edu`). Please email him for help and assistance.

If you are not familiar with Matlab, I suggest you go through some of the tutorials posted on the course web page.

3 Harris corner detector [30%]

In this part, the objective is to write a function that implements the Harris corner detector, as described in Lectures 1 and 2. The function should take as input:

1. `im` - A gray-scale image (2D array).
2. `threshold` - "cornerness" threshold.
3. `sigma` - Standard deviation of the Gaussian used to smooth the 2nd moment matrix (typical values: 2–4).
4. `radius` - Radius of non-maximal suppression.

The function should output a $2 \times N$ matrix of corner locations on the image, as well as displaying the image with these corner locations superimposed.

There are two key equations involved in the corner detector. The first computes the second moments at each point in the image, smoothed by a function $w(u, v)$, which in our case will be a Gaussian:

$$A = \sum_u \sum_v w(u, v) \begin{pmatrix} I_x^2 & I_{xy} \\ I_{xy} & I_y^2 \end{pmatrix} \tag{1}$$

I_x^2, I_y^2 are the square of x and y derivatives, while I_{xy} is $I_x I_y$. Note that for each pixel in `im`, there will be a 2nd moment matrix A. The second computes the "cornerness" measure:

$$M = det(A) - \kappa \ trace^2(A) \tag{2}$$

where κ is 0.04. To help you, please implement the function in the following step-by-step manner:

1. First, define the filters used to compute the image derivatives: `dx = [ -1 0 1 ; -1 0 1 ; -1 0 1];` and `dy = dx';`.
2. Next, compute the image derivatives I_x,I_y of `im` using the `conv2` function, with the `'same'` option. [To get help on the `conv2` function and its syntax, type `help conv2`].

3. Generate a Gaussian smoothing filter (the $w(u, v)$ function above), using the `fspecial` function, with the `'gaussian'` filter type. The standard deviation should be `sigma`, while the size of the filter should be 6`sigma`.

4. Compute I_x^2, I_y^2, I_{xy} using I_x,I_y and `.^2` and `.*`.

5. Compute the smoothed versions of I_x^2, I_y^2, I_{xy} by using the `conv2` function, with the `'same'` option.

6. Compute the cornerness measure M. Recall that $det([a, b; b, c]) = ac - b^2$ and $trace([a, b; b, c]) = a + c$. M should be map of cornerness, the same size as `im`.

7. Perform non-maximal suppression using the `ordfilt2` function and the `radius` and `threshold` input parameters. [To see an example of how to use it, type `help ordfilt2`].

8. Find the coordinates of the corner points using `find`.

9. Display the image and superimpose the corners.

Apply this function to the `einstein.jpg` image with `threshold=1000;`, `sigma=3` and `radius=3`. Save the output figure.

Rotate the `einstein.jpg` image by 45 degrees using `imrotate` image and apply the function again. Save the output figure.

Turn in the code for your function along with the two output figures.

4 Scale-invariance [30%]

In the previous part we looked at selecting certain locations in the image but this was only done at a fixed scale. In this section we manually fix a location in the image and exhaustively search over scale-scape to show how certain operators can reliably discover an intrinsic scale to blobs in the image.

First, load in `einstein.jpg`. The fixed location we will be examining is at row 186, column 148, corresponding to the center of the tie. Note that this is a distinct dark blob, surrounded by the white of his shirt.

The basic idea is to plot the value of the Laplacian operator:

$$\nabla^2 I = (\frac{\partial^2 I}{\partial x^2} + \frac{\partial^2 I}{\partial y^2}) \tag{3}$$

and its scale-normalized version:

$$\nabla^2 I_n = \sigma^2(\frac{\partial^2 I}{\partial x^2} + \frac{\partial^2 I}{\partial y^2}) \tag{4}$$

at the fixed location as we vary the scale of the Laplacian.

To compute the Laplacian we need to first generate $\frac{\partial^2 I}{\partial x^2}$ and $\frac{\partial^2 I}{\partial y^2}$ at different scales and then combine them.

- We first need to generate a Laplacian filter of standard deviation σ. Do this by: (a) creating a 2-D Gaussian filter using `fspecial`. (b) computing ∂_{xx} and ∂_{yy} by convolving with `dxx = [1 -2 1];` and `dyy = [1 -2 1]';` respectively. Be careful to avoid edge artifacts by using the `'valid'` option in `conv2`. This will mean that the equivalent y operator is of a different size to the x operator, so some cropping may be needed. (c) Add the two filters together. Visualize it with the `mesh` command. It should look like the plot in the slide title "Blob detection in 2D" from the 1st lecture.

- Load in the Einstein image and then make a copy of it at half the scale with the `imresize` command. The idea is to apply the same Laplacian filter to both images and check the response in the center of the tie for both the functions above.

- So loop over a range of scales, varying σ from 3 to 15 in increments of 0.4. For each scale, convolve the filter with the Einstein image. Then look up the row and column (halving them for the smaller image) and in each of the two functions above ($\nabla^2 I$ and $\nabla^2 I_n$), recording their value. Hence for each scale you should have 4 numbers (full-size image, half-size image for each of the two functions above).

- Now make 2 plots. One for $\nabla^2 I$ and the other for $\nabla^2 I_n$. Plot the responses of both the full-size image and the half-size one against σ. Note that for the half-size image, the effective scale is double.

- You should find that the scale-normalized operator, as expected, yields two fairly overlapping curves, unlike the unnormalized operator.

Plot the full size image and superimpose a circle (using `imellipse`) at the fixed location, with a radius corresponding to the peak of the scale-selection

curve from the $\nabla^2 I_n$. Turn the image and circle; both plots showing the scale-selection curves and the source code.

In practice, it is quicker to successively down-sample the image and keep the Laplacian at a fixed (small) scale. For bonus points, implement this down-sampling scheme and use `tic; toc;` to time its speedup over the scheme above.

5 Image alignment [40%]

In this part of the assignment you will write a function that takes two images as input and computes the affine transformation between them. The overall scheme, as outlined in Lecture 2, is as follows:

- Find local image regions in each image
- Characterize the local appearance of the regions
- Get set of putative matches between region descriptors in each image
- Perform RANSAC to discover best transformation between images

To help you, the first two stages can be performed using David Lowe's SIFT code which can be downloaded from:
`http://www.cs.ubc.ca/spider/lowe/keypoints/siftDemoV4.zip`. If you are using a Mac, then you will need to use these libraries instead:
`http://www.vlfeat.org/overview/sift.html`.

The `sift.m` function takes an image filename as input, loads the image and runs the Difference of Gaussians blob detector to find a set of regions. It then computes SIFT descriptors for each region. The output of the function is the image, a set of 128D image descriptors and their location (x,y,scale,angle) in the image.

The two images you should match are also contained in the ZIP file: `scene.pgm` and `book.pgm`, henceforth called image 1 and 2 respectively.

The third stage, obtaining a set of putative matches T, should be done as follows: for each descriptor in image 1, compute the closest neighbor amongst the descriptors from image 2 using Euclidean distance. Spurious matches can be removed by then computing the ratio of distances between the closest and second-closest neighbor and rejecting any matches that are above a certain threshold. To test the functioning of RANSAC, we want to

have some erroneous matches in our set, thus this threshold should be set to a fairly slack value of 0.9. To check that your code is functioning correctly, plot out the two images side-by-side with lines showing the potential matches.

The final stage, running RANSAC, should be performed as follows:

- Repeat N times:
- Pick P matches at random from the total set of matches T. Since we are solving for an affine transformation which has 6 degrees of freedom, we only need to select P=3 matches.
- Construct a matrix A and vector b using the 3 pairs of points as per slide 78-79 of Lecture 4
- Solve for the unknown transformation parameters q using Matlab's `pinv` command.
- Using the transformation parameters, transform the locations of all T points in image 1. If the transformation is correct, they should lie close to their pairs in image 2.
- Count the number of inliers, inliers being defined as the number of transformed points from image 1 that lie within a radius of 10 pixels of their pair in image 2.
- If this count exceeds the best total so far, save the transformation parameters and the set of inliers.
- End repeat.
- Perform a final refit using the set of inliers belonging to the best transformation you found. This refit should use all inliers, not just 3 points chosen at random.
- Finally, transform image 1 using this final set of transformation parameters, q. This can easily be done by first forming a homography matrix `H = [ q(1) q(2) q(5) ; q(3) q(4) q(6) ; 0 0 1 ];` and then using the `imtransform` and `maketform` functions as follows: `transformed_image=imtransform(im1,maketform('affine',H));`. If you display this image you should find that the pose of the book in the scene should correspond to its pose in image 2.

Turn in the transformed image, your value of matrix H and the source code.