

Computer Vision CSCI-UA.0480-001 Assignment 1.

February 7, 2012

Introduction

This assignment explores various methods for aligning images and feature extraction. There are four parts to the assignment:

1. Image alignment by translation – Align separate R, G and B channels of to form a color image. [*15 points*].
2. Harris corner detector – Implement the Harris corner detector and evaluate it on select images, demonstrating its invariance properties. [*25 points*].
3. Scale-invariance – Explore scale-selection using Laplacian and Difference of Gaussian operators. [*30 points*].
4. Image alignment using RANSAC – Solve for an affine transformation between a pair of images using the RANSAC fitting algorithm. [*30 points*].

Please also download the `assignment1.zip` file from the course webpage as it contains images and code needed for the assignment.

Requirements

You should perform this assignment in Matlab. If you are not familiar with Matlab, I suggest you go through some of the tutorials posted on the course web page.

This assignment is due on **Wednesday February 29th** by the start class (11am). Please note that **late assignments will receive zero (0) marks**, so you are strongly encouraged to start the assignment early and don't be afraid to ask for help.

The TA for the class is Chaitanya Rudra (cr1512@nyu.edu). Please email him for help and assistance, or come to office hours (Wednesday 12.30-1.30pm). If you think the assignment is not clear, or there is an error/bug in it, please contact the TA or myself.

You are allowed to collaborate with other students in terms discussing ideas and possible solutions. However you code up the solution yourself, i.e. you must write your own code. Copying your friends code and just changing all the names of the variables is not allowed! You are not allowed to use solutions from similar assignments in courses from other institutions, or those found elsewhere on the web.

Your solutions should be emailed to me (fergus@cs.nyu.edu) as a single zip file, with the filename: `lastname_firstname_a1.zip`. This zip file should contain: (i) a PDF file `lastname_firstname_a1.pdf` with your report, showing output images for each part of the assignment and explanatory text, where appropriate; (ii) the source code used to generate the images (with code comments), along with a master script (named `master_a1.m`) that runs the code for each part of the assignment in turn.

1 Colorizing the Prokudin-Gorskii photo collection

This part of assignment is adapted, with kind permission, from Aloysha Efros¹.

Sergei Mikhailovich Prokudin-Gorskii (1863-1944) was a photographer who, between the years 1909-1915, traveled the Russian empire and took thousands of photos of everything he saw. He used an early color technology

¹http://graphics.cs.cmu.edu/courses/15-463/2010_fall/hw/proj1/



Figure 1: A sample from the Prokudin-Gorskii photo collection: three separate plates for each color channel and the resulting color image.

that involved recording three exposures of every scene onto a glass plate using a red, green, and blue filter. Back then, there was no way to print such photos, and they had to be displayed using a special projector. Prokudin-Gorskii left Russia in 1918, following the Russian revolution. His glass plate negatives survived and were purchased by the Library of Congress in 1948. Today, a digitized version of the Prokudin-Gorskii collection is available online at <http://www.loc.gov/exhibits/empire/gorskii.html>.

The goal of this assignment is to learn to work with images in Matlab by taking the digitized Prokudin-Gorskii glass plate images and automatically producing a color image with as few visual artifacts as possible. In order to do this, you will need to extract the three color channel images, place them on top of each other, and align them so that they form a single RGB color image. Some starter code is available in `part1.m`, included in the main zip file. You are welcome to use it, if you find it helpful.

The zip file contains six images (`part1_*.jpg`) that you should run your algorithm on. **Note that the filter order from top to bottom is BGR, not RGB!**

Your program should take a glass plate image as input and produce a single color image as output. The program should divide the image into

three equal parts and align the second and the third parts (G and R) to the first (B). For each image, you will need to print the (x,y) displacement vector that was used to align the parts.

The easiest way to align the parts is to exhaustively search over a window of possible displacements (say $[-15,15]$ pixels), score each one using some image matching metric, and take the displacement with the best score. There is a number of possible metrics that one could use to score how well the images match. The simplest one is just the L2 norm also known as the Sum of Squared Differences (SSD) distance which is simply `sum(sum((image1-image2).^2))`. Another is normalized cross-correlation (NCC), which is simply a dot product between two normalized vectors: `(image1./||image1|| and image2./||image2||)`. See the Matlab function `normxcorr2`.

Your report should show: (i) the output color image for each of the six images in the zip file (please be sure to avoid compressing the images unnecessarily); (ii) the displacement vectors used to align the G and R color channels for each image and (iii) a brief description of your approach.

2 Harris corner detector

In this part of the assignment, the goal is to implement the Harris corner detector, as described in Lecture 5. You should write a function that takes as input the following variables:

1. `im` - A gray-scale image (2D array).
2. `threshold` - “corneriness” 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 (i.e. each pixel in

`im`), smoothed by a function $w(u, v)$, which in our case will be a Gaussian:

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

I_x^2 , I_y^2 are the square of x and y derivatives, while I_{xy} is $I_x I_y$.

The second important equation computes the “cornerness” measure, using A :

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

where κ is 0.04.

Please implement the function in the following stages:

1. First, define the filters used to compute the image derivatives. There are a variety of choices here, but the Sobel operator is a reasonable one: $\mathbf{dx} = [-1 \ 0 \ 1 ; -2 \ 0 \ 2 ; -1 \ 0 \ 1]$; and $\mathbf{dy} = \mathbf{dx}'$;
2. Next, compute the image derivatives I_x and 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 the Gaussian smoothing filter $w(u, v)$, 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 $\operatorname{trace}([a, b; b, c]) = a + c$. 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=5e-4`; (assuming the image intensities are scaled to be $0 \dots 1$), `sigma=3` and `radius=3`.

Your report should show the Einstein image superimposed with the corners under the following transformations: (a) none, (b) rotated by 45 degrees using `imrotate`, (c) multiplying the intensities by 1.5 and (d) resizing to half the resolution (using `imresize`). Notice how the corner features are fairly invariant to rotations and the intensity scaling, but not to the resizing operation.

3 Scale-invariance

In this section we explore scale-invariant measures used in blob detectors. We manually fix a location in the image and exhaustively search over scale-space 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:

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

and its scale-normalized version:

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

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 `dx = [1 -2 1]`; and `dy = [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 5th 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.
- Loop over a range of scales, varying σ parameter in the 2-D Gaussian filter 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 $\nabla^2 I_n$, as a function of σ . Each plot should have two curves: the full-size image and the half-size images. [Note that for the half-size image, the effective scale is double.].
- You should find that the scale-normalized operator, yields two fairly overlapping curves with similar maxima, unlike the unnormalized operator. This demonstrates that we are able reliably estimate the scale of the blob, even if it is resized.

Your report should include: (i) the two plots described above; (ii) the full-size and half-size images, with circles superimposed (using `imellipse`) at the fixed location, having a radius determined by the peak in the $\nabla^2 I_n$ curve for each image.

Convolving images with large Gaussian filters is slow and inefficient. A better approach is 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.

4 Image alignment

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 7, 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

The first two stages can be performed using David Lowe's SIFT feature detector and descriptor representation. Code for this can be downloaded from:

<http://www.cs.ubc.ca/spider/lowe/keypoints/siftDemoV4.zip>.

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 described in Lecture 7.

- Solve for the unknown transformation parameters q using Matlab's `\` 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.

Your report should include: (i) the transformed image 1 and (ii) the values in the matrix H .