Mosaicing by SIFT

Report for Lab Exercise 3 for the Computer Vision Course

A.P. van Ree (10547320)
T.M. Meijers (10647023)
University of Amsterdam
The Netherlands

December 1, 2014

1 Introduction

The aim of this lab was to study how mosaicing works, using SIFT and RANSAC. This includes reading and understanding Lowe's paper "Distinctive Image Features from Scale-Invariant Keypoints" (2004), and understanding the RANSAC algorithm.
The report contains detailed explanations of both algorithms. The implementation with detailed comments can be found in the Matlab code.

Written exercises: This report contains theory on SIFT and RANSAC in the main report, and all the written answers to the theory questions in ***Mosaicing by SIFT****: Lab Exercise 3* in Appendix A.

Programming exercises: The code is commented well enough so that every step is clear, and the structure of the code is well organized. The main file (*main.m*) is organized into sections which correspond to the structure in the assignment. The helper functions we have written or adjusted (*demo_mosaic_npoints.m, projectionMatrix.m* and *ransac.m*) are commented well enough to explain their working as well. The explanation of the entire mosaicing process should be clear from all the the comments the code contains.

2 Theory

2.1 SIFT

The SIFT (Scale Invariant Feature Transform) algorithm is an algorithm that is very useful for image processing. SIFT is very good at detecting and identifying objects in images, which has a lot of applications when processing images. The special thing about this algorithm is that, in contrast to simple corner detectors, SIFT is scale and rotation invariant ánd partially illumination and viewpoint invariant. The main essence of SIFT is to find keypoints in an image, finding good description of each of these keypoints, and then finding matches in other images with these keypoints. For example an eye in an image has a certain description, that is very different from a mouth or ear. To find a valid keypoint description for an object, SIFT needs at least three distinctive features, otherwise it could match too many other objects. What the SIFT algorithm also does in a smart way is only doing costly operations on keypoints that have passed a filter.
The SIFT algorithm works in four different steps:

1. Scale-space extrema detection
2. Keypoint localization
3. Orientation assignment
4. Keypoint description

Each of these steps will be discussed in the section below:

1. Scale-space extrema detection
This part of SIFT makes sure the algorithm is scale invariant, by creating a scale space. This is done in the following way: Create a number of ocataves of the picture to be processed. The first octave of the image is a collection of $k \cdot \sigma$ blurs. The next ocatve is half the size of the original image, with the same collection of $k \cdot \sigma$ blurs applied. The next is half the size of the previous size with the same blurs, and so on. The amount of octaves and blurs are dependent on how big the image is.
These blurs can be used to find edges and corners in an image. The official way to detect corners and edges is to find the "Laplacian of Gaussian". This is very costly, so another method would be better. Here, the octaves come in handy. When the blurs in an octave are subtracted from one another we get so called "Difference of Gaussians"(DoG). Because we use different ocatves the scale invariance is still maintained. Next we need to find the keytpoints in these DoGs.

2. Keypoint localization
Finding the keypoint locations is done by comparing each pixel to the 3x3 neighbourhood in its own layer of DoG and in the one above and below. This gives $9 + 8 + 9 = 26$ points to compare to. For this reason, the top and bottom layer are discarded for comparison, because there are not enought comparison pixels.
If the pixel of interest is a maximum of minimum copmared to all 26 surrounding it, this is a keypoint. All of the points that are found by comparing to the 26 neighbours are evaluated, removing the ones that have a contrast that is below a certain threshold. These are too vague and will lead to problems later.
The problem with this method is that the real maximum or minumum could have a location that is not on a pixel. We can solve this by using the following fucntion (see Appedix A: A.8 for the details):

$$D(\hat{x}) = D + \frac{1}{2}\frac{\partial D}{\partial x}^T \hat{x}$$

In this equation $\hat{x}$ is the exact point of maxmimum or minimum and x is is pixel found with DoG comparison. If we take the derivative and set this equal to zero, we find $\hat{x}$, which is the exact maxmimum/minimum.
The last problem with this method is that it will also find edges and corners. Edges are not

helpful as the whole egde will be a extrema, but corners are helpful. Edges are removed to calculated two derivatives for every keypoint, perpendicular to each other. If both derivatives are big, the keypoint is a corner and it is a valid one. Is one is big and the other small, it is an egde so it needs to be discarded. When both are small we want to keep it too, because it is a keypoint somewhere random in the image.
Now that we have correct keypoints, the next step it to assign orientation to each keypoint.

3. Orientation assignment
This part of the SIFT algorithm makes sure that it is rotation invariant. This is done by calculating the *magnitude* and *orientation* around each keypoint. The magnitude the combination of the gradient in the x and y direction in a vector (length). It is computed by the following formula:

$$m(x, y) = \sqrt{(L(x+1, y) - L(x-1, y))^2 + (L(x, y+1) - L(x, y-1))^2}$$

The orientation is the angle in which the gradient points. This is computed by:

$$\theta(x, y) = \tan^{-1}((L(x, y+1) - L(x, y-1))/(L(x+1, y) - L(x-1, y)))$$

These orientations are stored into so-called *bins* of 10° each. The bin that has the lagrgest amount of orientations, will be leading. These bins with its orientation make SIFT rotation invariant.

4. Keypoint desciption
To describe keypoint in the best way, Lowe found that if a 16x16 window around a keypoint are the region to work with. Lowe devides these into 4x4 windows and computes the magnitude and orientaions into 8 bins of 45°. The distance to the keypoint also matters, the closer to the keypoint the more weigth it has. These 128 (4x4x8) vectors of the bins are then normalized to ensure scaling invariance.
These descriptions give an accurate view of what a keypoint looks like, and could be used for further applications.

2.2 RANSAC

The Random Sample Consensus (RANSAC) algorithm is a non-deterministic algorithm that only produces a result based on correctly chosen (specific to the problem) parameters. The algorithm needs matching points from sift as input, and will use these points to establish a model that will be fit to the other points and then evaluate how good this model is. This is done in the following steps:

1. Take n random points (n is based on problem) and use these to establish a model.
2. All other points are fit with this model, and *inliers* are points which lie within a certain error margin of this model.
3. Determine quality of the model, if good enough improve it.

1. Take random points to establish a model
The sample point correspondences that are taken are used to fit a model. In our case this is the projection matrix that fits these correspondences the best. Every time the algorithm iterates again, n random points are taken so we (probably) get a different model everytime. This model is used to determine the inliers.

2. Fit all points to this model and determine inliers
To determine the inliers, which are points that lie within a specified error margin of the model, we need to check how close they are. In our implementation we have used the euclidean (pixel) distance between the estimated and real point in the second image. We then only try to improve the model of this iteration if we have a certain amount of inliers, which is an input parameter specified by the user.

3. Determine quality and obtaining the best fit
To improve the model, we use all the correct points (the random samples plus all the inliers) to fit a new model, again a projection matrix. We now test the error (eculidean distance) of this new model on all the inliers. If this error is lower than our best error before, this becomes the new best error, and the model is saved as the best model so far.

4. Discussion on RANSAC
As stated in the introduction on RANSAC, the algorithm is non-deterministic. This poses a problem since it is not guaranteed to obtain a model that is good, or even a model at all. As Lowe himself states, when pictures don't have a lot of matching keypoints and the models don't have a lot of inliers (Lowe mentions 50%), RANSAC doesn't give good results and thus Lowe has chosen for Hough transform.
The obtained model is very much dependant on the parameters chosen by the user, which include iterations, error margin, inliers threshold. As can be seen in our experiments, when we have images that don't produce good matching keypoints (the roofs example) we need a lot of iterations to obtain a projection matrix that is only half-decent. When we have a good image such as nachtwacht or mountains, 10 to 25 iterations suffice.

3 Mosaicing

3.1 Algorithm

The code is thoroughly commented, but here is a short overview of what the algorithm does: Instead having someone pick matching points such as in *demo_mosaic.m*, the algorithm does this by itself. This is done with the help of the *VLFeat* library, that computes matching points with its SIFT-function, and self-written RANSAC algorithm that finds the best transformation matrix. This is all explained in the sections above and the comments. When the suitable matrix is found, this is used to transform one of the two images so that they can be puzzeld together like a mosaic, matching scale and rotation. Code from *demo_mosaic.m* is used to do this the right way.

3.2 Experiments

We have conducted four different experiments: Nachtwacht, Mountains, Classroom and Roof (see Matlab code and comments). Out of these four, Nachtwacht and Mountains work very well. The other two are a bit poorer in execution. This because they have very scewed angles betwee them, and they have very little good matching points.

4 Conclusion

Overall we did the coding in Matlab and the Theory questions in Appendix A quickly. These were quite easy once the understanding of Lowe's paper on SIFT was there. The most time was spent into getting to a sufficient level of understanding of SIFT.
The RANSAC code works quite well, most of the experiments give a good result when performing mosaicing, even when we used our own pictures (see Matlab code). The reason why this went so smoothly is because we are using the VLFeat implementation of SIFT. It would have been nice to code more of this algorithm to get an even better understanding of it, than there is now by only reading the paper by Lowe.

Appendices

A SIFT - Theory questions

A.1 Read Lowe (2004)

Reading: Done.
Understanding: In progress.

A.2 Why SIFT for mosaicing

Indeed SIFT is very useful for the mosaicing task. For the algorithm to perform mosaicing automatically most of the subjects described in the paper are applicable. We need to extract keypoints, match them and these need to be scale and rotation invariant.

A.3 Scale Space (Extrema)

The scale space is a collection of an image, repeatedly being convolved by a gaussian blur (every time increasing with a factor k). A set of these images is called an octave. These images are then used to produce the difference of gaussians to check for local extrema. The next octave it is dimensions are half of the original, and this image is then convolved multiple times as well.

This produces images of the original image at different scales, and also produces different extrema for the different scales.

A.4 returning to D

Equation (1), Lowe (2004)

$$\begin{aligned} D(x,y,\sigma) &= (G(x,y,k\sigma) - G(x,y,\sigma)) * I(x,y) \\ &= L(x,y,k\sigma) - L(x,y,\sigma) \end{aligned}$$

Equation (2): Scale-normalized Laplacian of Gaussian and relation to D, Lowe (2004)

$$\begin{aligned} \frac{\delta G}{\delta \sigma} &= \sigma \nabla^2 G \\ &\approx \frac{G(x,y,k\sigma) - G(x,y,\sigma)}{k\sigma - \sigma} \end{aligned}$$

According to Lowe, therefore:

$$G(x,y,k\sigma) - G(x,y,\sigma) \approx (k-1)\sigma^2 \nabla^2 G$$

We then return to the relation with D by combining (1) and (2):

$$\begin{aligned} D(x,y,\sigma) &= (G(x,y,k\sigma) - G(x,y,\sigma)) * I(x,y) \\ &\approx (k-1)\sigma^2 \nabla^2 G * I(x,y) \end{aligned}$$

A.5 Trace of the Hessian

Equation for calculating eigenvalues

$$det(A - \lambda I) = 0$$

Applying this to the Hassian gives:

$$(f_{xx} - \lambda)(f_{yy} - \lambda) - f_{xy}^2 = 0$$

In 2D, f_{xy} is always equal to 0. Applying this to the previous equation gives:

$$(f_{xx} - \lambda)(f_{yy} - \lambda) = 0$$
$$\lambda = f_{xx} \vee \lambda = f_{yy}$$

This making f_{xx} and f_{yy} the eigenvalues. So, the sum of the eigenvalues is equal to the sum of f_{xx} and f_{yy}, which is equal to the trace of the Hessian. Thereby, the trace of the Hessian is the sum of its eigenvalues.

Because the Hessian is always orthogonal (and thereby symmetric), so the Hessian is always diagonalizable.

A.6 SIFT is magic (numbers)!

Magic number are shown bold, if we probably need to modify a number for mosaicing we will explain why.

General:
Values of σ differ throughout the paper, as this corresponds to scale we will probably want to vary σ as well.
Number of bins (for orientation) for descriptors and gradient orientation. As well as size of the array are varied in the paper. Lowe states that **4 x 4 x 8** appeared to be optimal for size and orientation bins, and uses **36** bins for direction. We so no reason either to change these values as 36 direction bins feels intuitive and the size and orientation bins were obtained through experiments.

Page 11:
"If $\hat{x}$ is larger than **0.5**"
"all extrema with a value of $|D(\hat{x})|$ less than **0.03**"
We probably don't need to change these values for our mosaicing task since we use a normal image were Lowe's thresholds for extrema are probably usable."

Page 12:
"The experiments in this paper use a value of $r =$**10**"
No need to change this ratio either as we don't have an image with very large gradients in only one direction.

Page 13:
"any other local peak that is within **80%** of the highest peak"
This is used so features can be translated to multiple keypoints with different orientations, we see no reason this should be changed for our mosaicing task.

Page 16:
"thresholding the values in the unit feature vector to each be no larger than **0.2**"
We see no reason to change this since Lowe obtained this number through multiple experiments (same page).

Page 20:
"we reject all matches in which the distance ratio is greater than **0.8**"
Since this number was obtained specific for object recognition, we may need to adjust the ration specific for mosaicing.

A.7 Questions that aren't questions

No answer required.

A.8 Does Lowe make typos?!

Equation (1), Lowe (2004)

$$D(x) = D + \frac{\partial D}{\partial x}^T x + \frac{1}{2} x^T \frac{\partial^2 D}{\partial x^2} x$$

Rewriting this with $H = \frac{\partial^2 D}{\partial x^2}$ and $g = \frac{\partial D}{\partial x}$ gives:

$$D(x) = D + g^T x + \frac{1}{2} x^T H x$$

Equation (2), Lowe (2004)

$$\begin{aligned} \hat{x} &= -\frac{\partial^2 D}{\partial x^2} \frac{\partial D}{\partial x} \\ &= -H^{-1} g \end{aligned}$$

Substituting x with equation (2) in (1):

$$D(\hat{x}) = D + g^T(-H^{-1}g) + \frac{1}{2}(-H^{-1}g)^T H(-H^{-1}g)$$

Multyplying a matirx with its inverse gives the identity matrix. Also, multiplying a matrix by the identity matrix gives that same matrix. This given, the following can be said:

$$\begin{aligned} D(\hat{x}) &= D + g^T(-H^{-1}g) - \frac{1}{2} g^T(-H^{-1}g) \\ &= D + \frac{1}{2} g^T(-H^{-1}g) \\ &= D + \frac{1}{2} \frac{\partial D}{\partial x}^T \hat{x} \end{aligned}$$

This is equal to the definition given by Lowe of $D(\hat{x})$.

A.9 Trilinear interpolation

We are now in 3D, as opposed to 2D because we have an extra element in the descriptors. Ofcourse we have our coördinate system, so 2 elements of this space are still our x and y coordinates. The third element in this space is the direction of the descriptor, so the orientation.

A.10 Affine illumination

These "affine changes in illumination" describe the change in illumination on an object due to affine transformation. They keyword here is affine, as the method is invariant to linear changes in light, described in the sentences before this part. The feature vectors are normalized, so changes in illumination don't affect this between images. In a image itself, if it is illuminated in whole, ALL the pixel values change, and we are comparing pixel values (differences), so it is invariant to this as well.

A.11 No question no answer

No answer required.