University of Amsterdam

Beeldverwerken week 4

Mosaicing using SIFT and RANSAC

May 5, 2019

Tutor:
Wouter Zwerink

Student:
Tim Stolp

Lecturer:
Leo Dorst

Contents

1 Introduction

Mosaicing is a technique where two images partly containing the same information are placed on top of each other to create a smooth transition between the images. This technique can be used in various situations like video stabilization and the creation of panorama pictures. A popular way of performing mosaicing is with Scale Invariant Feature Transform (SIFT) and Random sample consensus (RANSAC). These two algorithms will be discussed and explained in this report.

2 SIFT

Scale Invariant Feature Transform (SIFT) is an algorithm to detect scale and orientation invariant features in an image. These features can be used to find matching objects in different images which has practical applications in for example object recognition or video stabilization. SIFT can be explained in five steps.

2.1 Creating the scale space

The size of an object in an image depends on how far away the object is from the viewpoint. This causes the same object to often vary in size throughout different images. To still be able to recognize objects on different scales we introduce the scale space. The scale space is a structure of Gaussian blurred images stacked on top of each other. Each Gaussian blurred image is blurred on a different scale (σ). The interval between each scale is a constant value calculated by $2^{\frac{1}{s}}$ with s being the amount of steps between each octave. An s value of 3 tends to give the best results while taking accuracy and computation time into account. Each octave is separated by an interval of $\sigma * 2$. The image size of each next octave is halved by taking each second pixel of each row and column as this does not reduce the accuracy of sampling while decreasing computation time.

2.2 Scale invariant Laplacian of Gaussian

To find features in an image a good first step is to find edges and corners. The Laplacian (second order derivative depicted as ∇^2) is good at detecting these edges and corners. Normally the Laplacian is sensitive to noise, which is why the image is first smoothed with a Gaussian to reduce noise. The result of the Laplacian of Gaussian is not yet scale invariant because the Gaussian formula uses the scale (σ^2) in its denominator. So to make it scale invariant we have to multiply the result by σ^2 giving us the formula $\sigma^2\nabla^2 G$.

Unfortunately calculating this is very computationally expensive which is why the Difference of Gaussian (DoG) is used instead. The DoG is an approximation to the scale invariant Laplacian of Gaussian. The DoG is made by taking two consecutive scales from an octave in the scale space and then subtracting these from each other. This is done for each consecutive scale pair throughout the octaves.

2.3 Finding keypoints

To find the clearest edges and corners from the DoG the extrema are sought. Firstly the pixels in the DoG are compared to their 3x3x3 neighbourhood, if the pixel is a minimum or maximum it becomes a keypoint. It is unlikely that in the 3x3x3 neighbourhood the extrema are exacly on pixels so they are more likely to be in between pixels. To find these more accurate extrema a second order polynomial is fitted through the neighbourhood of initially found keypoints after which the extrema of that polynomial is taken as final keypoint.

These final keypoints still contain some keypoints that lie on edges or have low contrast. Preferably the keypoints consist of high contrast corners as these give the best results. To check for corners both gradients of a keypoint need to be high. To find these keypoints the gradients below a certain threshold are filtered out. Afterwards the ratio between the eigenvalues is calculated. A low ratio means equally big gradients indicating a corner. The keypoints with a too big ratio are also filtered out.

2.4 Keypoint orientation

To make the features orientation invariant the information about the keypoint needs to be relative to the orientation of the keypoint. Therefore firstly the orientation needs to be calculated. To do this an area of size 1.5σ around the keypoint is taken of which the gradient magnitude and orientation is calculated for each pixel.

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

$$\theta(x,y) = tan2^{-1}((L(x,y+1) - L(x,y-1))/(L(x+1,y) - L(x-1,y))) \tag{2}$$

A histogram is created existing of 36 bins with each bin representing 10 degrees of the total 360 degrees of orientation. For each bin the gradient magnitude is summed of all the pixels with an orientation fitting in that bin. Then the orientation with the highest total gradient magnitude is taken to be the orientation of the keypoint. If there is another peak within 80% of the highest gradient magnitude this is made into a separate keypoint with a different orientation on the same place.

2.5 Creating the final feature

To create the final feature a 16x16 window around the keypoints is taken. For each location on this grid the gradient magnitude and orientation are calculated using trilinear interpolation as the keypoint itself is unlikely to be exactly on a pixel. This interpolation is done with x, y, and z with z being the height of a Gaussian laid on top of the 16x16 grid. This Gaussian makes it so that the gradient magnitude and orientation have a higher importance closer to the center of the keypoint. To make the feature orientation invariant the keypoint's orientation is substracted from all the orientations calculated in the 16x16 grid as this gives us the orientations relative to the keypoint's orientation.

The grid is then divided in sixteen 4x4 subsections. For each subsection the orientations are put in an 8 bin histogram with each bin representing 45 of the total 360 degrees of orientation. Each bin is the sum of the magnitudes of all orientations part of that bin. This gives us 8 values for each subsection of the total 4x4 subsections giving us a final feature vector of 8x4x4=128 numbers. Finally this vector is normalized to make it partly illumination invariant. Then the big numbers are thresholded to 0.2 after which the vector is again normalized to achieve complete illumination invariance.

3 Algorithm

Here the algorithm will be explained with Matlab code examples.

First of all the image values are changed to single precision floating points and RGB values are converted to gray scale values.

```
function result = sift_ransac(im1, im2)
    % make single
    im1 = im2single(im1) ;
    im2 = im2single(im2) ;

    % make grayscale
    if size(im1,3) > 1
        im1g = rgb2gray(im1) ;
    else
        im1g = im1 ;
    end

    if size(im2,3) > 1
        im2g = rgb2gray(im2) ;
    else
        im2g = im2 ;
    end
```

Afterwards SIFT is performed on both images and the resulting features are matched between the pictures. The coordinates of the matches are put into an input array (features of image 1) and output array (features of image 2) and a row of ones is added to convert them into homogeneous coordinates.

```
    % Use SIFT to find matches
    [f1,d1] = vl_sift(im1g) ;
    [f2,d2] = vl_sift(im2g) ;

    [matches, scores] = vl_ubcmatch(d1,d2) ;

    numMatches = size(matches,2) ;

    xa = f1(1,matches(1,:)) ;
    xb = f2(1,matches(2,:)) ;
    ya = f1(2,matches(1,:)) ;
    yb = f2(2,matches(2,:)) ;

    total_list = [xa; ya; xb; yb];
    input = [xa; ya; ones(size(xa))];
    output = [xb; yb; ones(size(xb))];
```

Next RANSAC is performed on these matches to find a fitting projection matrix needed to transform one image on top of another. First the parameters are chosen and calculated with which RANSAC is used. The parameters distance_threshold and outlier_probability are experimentally found.

```
    % Get RANSAC parameters
    distance_threshold = 10;
    best_projmatrix = zeros([3,3]);
    best_error = 999999999;
    outlier_probability = 0.35;
    number_points = 4;
    p = 0.99;

    % Calculate number of iterations to have a p probability to find a good
    % projection
    iterations = round(log(1-p)/log(1-(1-outlier_probability)^number_points))
```

With these parameters and feature matches from SIFT RANSAC is done with a number of iterations to give a 99 percent probability to find a fitting projection matrix.

```
% Perform RANSAC
for i = 1:iterations
    % Get 4 random matches
    random_indexes = randperm(size(total_list ,2),number_points);
    random_matches = total_list(:,random_indexes);

    % Create projection matrix
    projmatrix = estimateProjMatrix2(random_matches(1:2,:)',
random_matches(3:4,:)');

    % Project input features
    projected_output = projmatrix * input;
    projected_output = projected_output./projected_output(end,:);

    % Get error
    error = vecnorm(output - projected_output);

    % Find inliers (matches below error threshold)
    valid_indexes = find(error < distance_threshold);
    valid_matches = total_list(:,valid_indexes);

    % Check if enough inliers are found
    if size(valid_matches, 2) > round((1 - outlier_probability) *
numMatches)
        % Get projection matrix from all valid matches and project input
        projmatrix = estimateProjMatrix2(valid_matches(1:2,:)',
valid_matches(3:4,:)');
        projected_output = projmatrix * input;
        projected_output = projected_output./projected_output(end,:);

        % Get mean squared error and update if best solution
        mse = immse(output, projected_output);
        if mse < best_error
            best_projmatrix = projmatrix;
            best_error = mse;
        end
    end
end
```

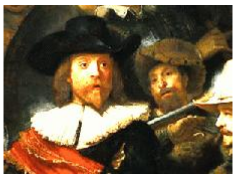
RANSAC performs a number of iteration in which it chooses four random matches and estimates a projection matrix with these matches. It then projects all features from image one using this projection matrix. The error distance is calculated between the projected features and the features of image two after which these are compared to a chosen error threshold. The feature matches below the threshold are then used to estimate a more accurate projection matrix with which again all features from image one are projected. The Mean Squared Error (MSE) is calculated between the projected features and the features of image two. The projection matrix resulting in the lowest MSE throughout the iterations is remembered and then used to project the entirety of image one resulting in a single image with a hopefully smooth transition.

```
% Project image 1 on top of image 2 with found best projection matrix
T = maketform('projective', best_projmatrix');
[x, y] = tformfwd(T,[1 size(im1,2)], [1 size(im1,1)]);

xdata = [min(1,x(1)) max(size(im2,2),x(2))];
ydata = [min(1,y(1)) max(size(im2,1),y(2))];
f12 = imtransform(im1,T,'Xdata',xdata,'YData',ydata);
f22 = imtransform(im2, maketform('affine', [1 0 0; 0 1 0; 0 0 1]), 'Xdata'
,xdata,'YData',ydata);
result = max(f12,f22);
end
```

4 Experiments

To demonstrate the algorithm it will be performed on two sets of images. Firstly two cut out pieces of the famous painting "De Nachtwacht" and secondly two top down views of a river. The runtimes of these experiments are only a matter of seconds.


(a) Nachtwacht1

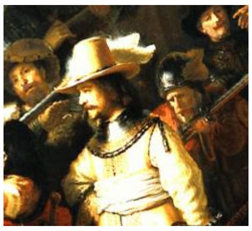

(b) Nachtwacht2

Figure 1

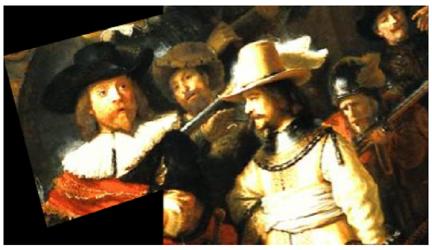
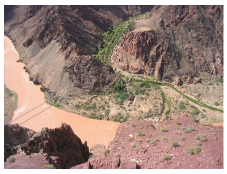


Figure 2: Nachtwacht1 projected on top of Nachtwacht2


(a) River1

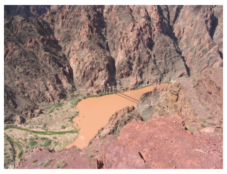

(b) River2

Figure 3

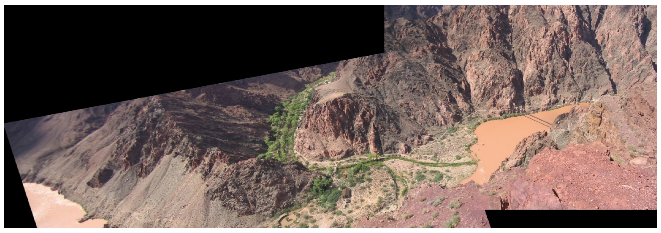


Figure 4: River1 projected on top of River2

5 Conclusion

Mosaicing is an important technique in various situations. By combining SIFT and RANSAC a very reliable algorithm can be made that is scale, rotation, and illumination invariant. This algorithm is computationally lightweight, reliable, and accurate.

Appendices

3 SIFT

3.3

The scale space is a structure of Gaussian blurred images stacked on top of each other. Each Gaussian blurred image is blurred on a different scale (σ). The interval between each scale is a constant value calculated by $2^{\frac{1}{s}}$ with s being the amount of steps between each octave. An s value of 3 tends to give the best results while taking accuracy and computation time into account. Each octave is separated by an interval of $\sigma * 2$. The image size of each next octave is halved by taking each second pixel of each row and column as this does not reduce the accuracy of sampling while decreasing computation time.

In the scale space the extrema are sought by looking at each pixel and determining if this is a minimum or maximum in its 3x3x3 neighbourhood of pixels. The scale space extrema are points that are consistently extrema throughout each Gaussian blur scale which makes them good scale invariant anchor points.

3.4

The relationship between D and $\sigma^2\nabla^2 G$ can be understood from the heat diffusion equation parameterized by σ instead of $t = \sigma^2$.

$$\frac{\partial G}{\partial \sigma} = \sigma \nabla^2 G \tag{3}$$

Using approximation we get:

$$\sigma \nabla^2 G = \frac{\partial G}{\partial \sigma} = \frac{G(x, y, k\sigma) - G(x, y, \sigma)}{k\sigma - \sigma} \tag{4}$$

and therefore,

$$G(x, y, k\sigma) - G(x, y, \sigma) = (k-1)\sigma^2 \nabla^2 G \tag{5}$$

using the definition

$$D(x, y, \sigma) = (G(x, y, k\sigma) - G(x, y, \sigma)) \star I(x, y) \tag{6}$$

we get:

$$D(x, y, \sigma) = (k-1)\sigma^2 \nabla^2 G \star I(x, y) \tag{7}$$

3.5

Using the Jordan normal form

$$P^{-1}HP = J \tag{8}$$

with J being the matrix with the eigenvalues of symmetrical matrix H on the diagonal and for symmetrical matrices (like the Hessian) using the fact that

$$tr(P^{-1}HP) = tr(HPP^{-1}) = tr(H) \tag{9}$$

we can say

$$\begin{aligned} tr(P^{-1}HP) &= tr(J) \\ tr(HPP^{-1}) &= tr(J) \\ tr(H) &= tr(J) \end{aligned} \tag{10}$$

3.6

The magic numbers are r, s, the threshold of $|D(\hat{x})|$, and the threshold of magnitude in the normalized feature vectors. s is the amount of scales in each octave. s could be higher but this would increase computation time but also might increase the accuracy depending on the task. r is the ratio threshold between the eigenvalues of edges. this could be decreased or increased to allow for more or less longer edges to be detected as key points. Lastly the threshold of $|D(\hat{x})|$ is there to remove unstable extrema with low contrast. In Lowe's paper the threshold is chosen to be 0.03. This could be increased to only allow key points with more contrast which will lower the amount of key points but might increase the average usefulness of the remaining key points. The threshold of magnitude in the normalized feature vectors is experimentally determined to be 0.2 to not be affected too much by a non-constant change in illumination. This can be decreased or increased to be more or less susceptible for illumination changes.

3.8

$$D(x) = D + \frac{\partial D^T}{\partial x} x + \frac{1}{2} x^T \frac{\partial^2 D}{\partial x^2} x \tag{11}$$

$$\hat{x} = -\frac{\partial^2 D^{-1}}{\partial x^2} \frac{\partial D}{\partial x} \tag{12}$$

Gradient is:

$$G = \frac{\partial D}{\partial x}$$

Hessian is:

$$H = \frac{\partial^2 D}{\partial x^2}$$

Then substituting for $D(\hat{x})$:

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

Rewriting with the fact that $(H^{-1})^T = (H^T)^{-1} = (H)^{-1}$ because $H^T = H$ because H is symmetrical because it is the Hessian.

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

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

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

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

The $\frac{1}{2}$ is not a typo.

3.9

Trilinear interpolation is used to calculate the magnitude and orientation between the pixels. These depend on the x and y. The magnitude is then multiplied by a Gaussian which says that the values closer to the middle have a higher importance than the ones on the outside. This Gaussian is the third dimension.

3.10

An affine change in illumination is a change in which the pixel values are changed by a constant value. Since the feature vectors are normalized increasing everything by a constant value will not have any effect on the final normalized feature vector.

3.11

See report.

3.12 Explain in your hand-in why SIFT is scale invariant and rotation invariant.

This can be found in section 1.2 and 1.5 of the report.

3.13

The size of the circles indicate the score of the matching features while the red lines connect the matching features.

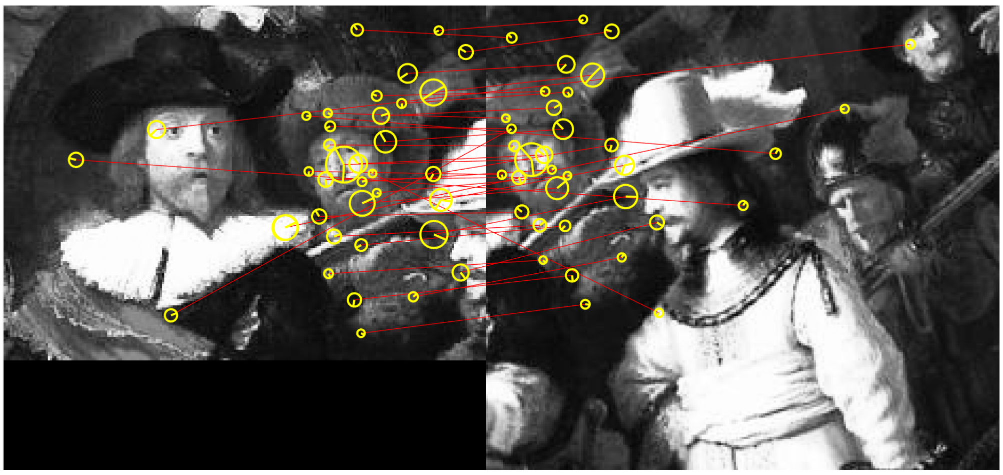


Figure .1: Feature matches using vl_sift and vl_ubcmatch

3.14

By using the four best matches to create a projection matrix and projecting the remainder of the features we get a decent result in which most points seem to be projected to the correct place. The bad features are not projected to the right place as is to be expected.

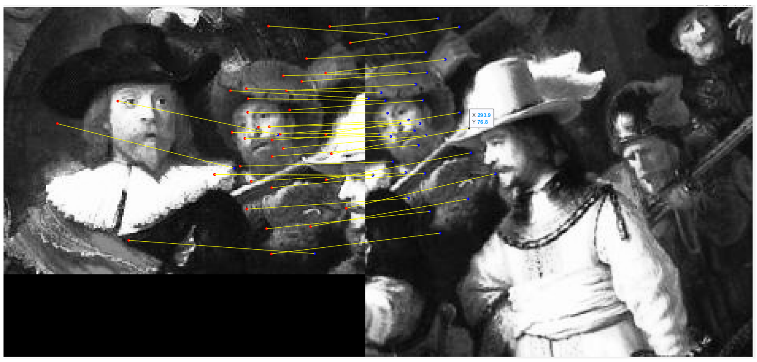


Figure .2: Features projected using projection matrix of 4 best matches

By using the four worst matches to create a projection matrix and projecting the remainder of the features we get a very bad result in which most points are roughly projected to the same spot and much further than while using the good features. This is probably because the distance between the bad matches tends to be higher than the distance between the good matches.

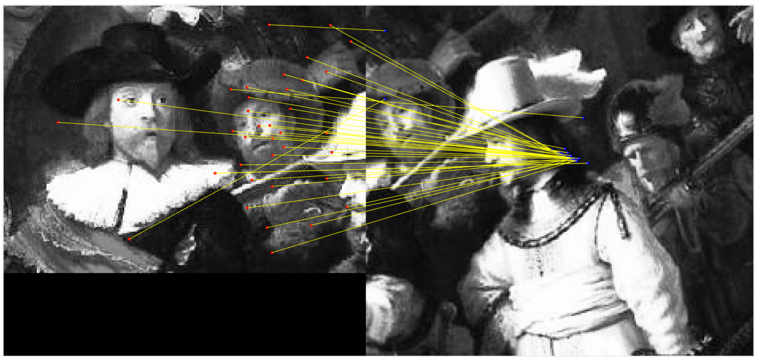


Figure .3: Features projected using projection matrix of 4 worst matches