Feature detection and matching

November 2014

Abstract

This laboratory is devoted to experiment with an implementation of the SIFT descriptor detection and matching.

Contents

1 Introduction **2**

2 Feature detection **2**

3 Feature matching **4**

3.1 First experiments 4

3.2 Linear and affine model 5

4 Panorama creation (optional) **6**

5 Practicum submission **9**

1 Introduction

In order to perform this laboratory, the code available at the following link has to be downloaded

`http://www.vlfeat.org/download/vlfeat-0.9.16-bin.tar.gz`

This code is an implementation of the SIFT descriptor detection and matching algorithm described work [1] (it is not recommended to download version 0.9.17 or higher for Linux since these versions require proprietary libraries usually not available under Linux). Although it does not correspond to the original author's implementation, the descriptors extracted by this implementation correspond nearly exactly to the ones extracted by the original SIFT.

The code has an API that can be called from MATLAB and C language. For the former case, the library already includes precompiled binaries that can be used under Windows, Linux or Mac.

In order to install the library, follow the next instructions:

1. Download the file and decompress it to a folder where you may easily change directory in MATLAB. When decompressing the folder `vlfeat-0.9.16` will be created.

2. Download from the campus the utilities files and decompress it inside the folder `vlfeat-0.9.16`.

3. Run MATLAB and change directory to the folder `vlfeat-0.9.16`.

4. In order to configure the `vlfeat` library, run the following command

   ```
   run('vlfeat-0.9.16/toolbox/vl_setup');
   ```

   MALAB may complain about and XML file: do ignore these messages.

5. To check if installation went right, run

   ```
   vl_version
   ```

2 Feature detection

We are now going to perform some experiments with the SIFT descriptor detection algorithm. For that issue run the following commands

```
I = vl_impattern('roofs1');
I = single(rgb2gray(I));
imshow(I);
[f,d] = vl_sift(I);
```

The SIFT keypoints are detected and described by means the `vl_sift` MATLAB command. This command requires a single precision gray scale image. The image range may be in [0,255] or [0,1]. In this example the image is within the range [0,1].

The `vl_sift` command returns the following information:

- The matrix `f` has a column for each keypoint. A keypoint is characterized by a center `f(1:2)`, scale `f(3)` and orientation `f(4)`.
- Each column of `d` is the descriptor of the corresponding keypoint in `f`. A descriptor is a 128-dimensional vector of class `uint8`.

We may view the descriptors by using

```
show_keypoints(I,f);
```

Note that a large number of keypoints have been detected in the image. For each keypoint, the application draws a green colored circle indicating its position, orientation and scale at which it was detected. The radius of the circle indicates the "equivalent" scale at which the keypoint has been detected. That is, a low radius is associated to a small (equivalent) patch, whereas a large radius is associated to an (equivalent) large patch.

One may view only a small subset of the previous keypoints by randomly selecting some of them

```
show_keypoints(I,random_selection(f,50));
```

Question: Observe where in the image have the keypoints been detected. Note that for the roof of the houses keypoints the radius is small. Why ? At the heaven some keypoints with large radius are detected. Why ? You may repeat the experiment with another image (such as 'river1') to understand what a significant keypoints is. Comment on your response.

The `vl_sift` command allows additional parameters to control which keypoints are detected. The SIFT detector is controlled mainly by two parameters: the peak threshold and the (non) edge threshold.

- The peak threshold filters peaks of the Difference of Gaussian D scale space that are too small (in absolute value). That is, for a given keypoint at location p one checks if $|D(p)|$ is above a given threshold.
- The edge threshold eliminates peaks of the Difference of Gaussian scale space whose curvature is too small (such peaks yield badly localized keypoints). Let H be the autocorrelation matrix defined as

$$H = \begin{pmatrix} \sum_{\mathbf{p}\in R} \left(\frac{\partial D}{\partial x}\right)^2 & \sum_{\mathbf{p}\in R} \frac{\partial D}{\partial x}\frac{\partial D}{\partial y} \\ \sum_{\mathbf{p}\in R} \frac{\partial D}{\partial x}\frac{\partial D}{\partial y} & \sum_{\mathbf{p}\in R} \left(\frac{\partial D}{\partial y}\right)^2 \end{pmatrix}$$

The principals curvatures are estimated as the eigenvalues of the previous matrix. Let r be the ratio between the largest magnitude eigenvalue and the smaller one, so that $\lambda_{max} = r\,\lambda_{min}$. Then

$$\frac{\mathrm{Tr(H)}^2}{\mathrm{Det(H)}} = \frac{(\lambda_{max} + \lambda_{min})^2}{\lambda_{max}\lambda_{min}} = \frac{(r+1)^2}{r}$$

which depends only on the ratio r rather than the individual eigenvalues. The quotient is minimum when the two eigenvalues are equal and it increases with r. Therefore, to check that the ratio of principal curvatures is below a given threshold, r, we only need to check

$$\frac{\mathrm{Tr(H)}^2}{\mathrm{Det(H)}} < \frac{(r+1)^2}{r}$$

to accept a keypoint. By default a threshold of $r = 10$ is used.

Let us see some examples. First we test the SIFT detector by using different peak threshold

```
[f,d] = vl_sift(I,'PeakThresh', 0.01);
show_keypoints(I,f);
```

Note that the small threshold is due to the fact that the range of gray-values of the input image is within [0,1].

Question: Try to slowly increase or decrease the threshold and analyze the keypoints that are accepted or discarded. What can you say about the keypoints that are discarded with the threshold ?

We now try to experiment with the edge threshold. For that you need to fix the peak threshold

```
[f,d] = vl_sift(I,'PeakThresh', 0.04, 'EdgeThres', 10);
show_keypoints(I,f);
```

Question: Comment on the effect of the edge threshold on the detection of the keypoints. What happens if you increase or decrease the threshold ? What type of keypoints are discarded with the threshold ?

3 Feature matching

3.1 First experiments

We will now use the SIFT descriptors to perform matching between two images. For that let us load two images and compute their associated descriptors

```
Ia = vl_impattern('roofs1');
Ib = vl_impattern('roofs2');
Ia = single(rgb2gray(Ia));
Ib = single(rgb2gray(Ib));
[fa, da] = vl_sift(Ia);
[fb, db] = vl_sift(Ib);
```

We compute the keypoint matching with the following command

```
[matches, scores] = vl_ubcmatch(da, db);
```

For each descriptor in `da`, `vl_ubcmatch` finds the closest descriptor in `db` (as measured by the L2 norm of the difference between them). The index of the original match and the closest descriptor is stored in each column of matches and the distance between the pair is stored in scores. The function uses the algorithm suggested by D. Lowe [1] to reject matches that are too ambiguous.

Matches also can be filtered for uniqueness by passing a third parameter to `vl_ubcmatch` which specifies a threshold. Here, the uniqueness of a pair is measured as the ratio of the distance between the best matching keypoint and the distance to the second best one (see `vl_ubcmatch` for further details).

Two view the matchings we use the following command

```
show_matches(Ia,Ib,fa,fb,matches);
```

As before, one may view a reduced set of matchings by executing

```
show_matches(Ia,Ib,fa,fb,random_selection(matches,50));
```

Note that not all the matches are correctly performed. We may improve the matching strategy by using a threshold

```
[matches, scores] = vl_ubcmatch(da, db, 2.0);
show_matches(Ia,Ib,fa,fb,matches);
```

Question: try to modify the previous threshold. What is the effect of this threshold ? Why do results improve as the threshold is increased ? Comment your response.

3.2 Linear and affine model

Once we have performed the matching between the two images we may compute the associated translational model

```
d = fb(1:2,matches(2,:))-fa(1:2,matches(1,:));
p = mean(d,2);
show_matches_linear_model(Ia,Ib,fa,fb,matches,p);
```

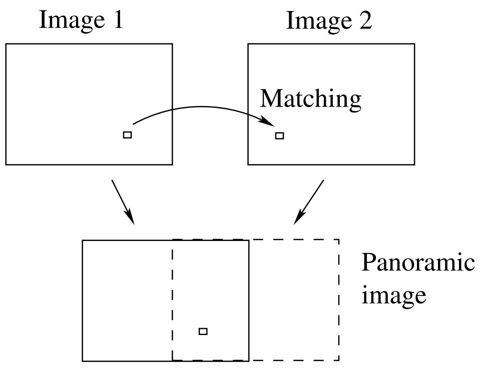

Figure 1: Panoramic image construction using SIFT descriptor matching.

Comment on the obtained result. Is the resulting model correct ? Do you think a linear model is enough to model the transformation between the two images ? Can you improve this result ?

We will now compute an affine model between the two images

```
i = size(matches, 2);
d = fb(1:2,matches(2,:))-fa(1:2,matches(1,:));
A = zeros(6,6);
b = zeros(6,1);
for j = 1:i,
    x = fa(1:2,matches(1,j));
    J = [x(1), x(2), 0, 0, 1, 0; 0, 0, x(1), x(2), 0, 1];
    A = A + J'*J;
    b = b + J'*d(1:2,j);
end
p = inv(A)*b;
show_matches_affine_model(Ia,Ib,fa,fb,matches,p);
```

Comment on the obtained result. Is the resulting model correct ? Do you think a linear model is enough to model the transformation between the two images ? Can you improve this result ?

4 Panorama creation (optional)

An interesting application of the use of SIFT descriptors is the panorama image creation. You are requested to implement a MATLAB script that constructs a panoramic image from two images, see Figure 1. For that we are going to use the RANSAC algorithm.

The objective of the algorithm you are going to implement is to automatically find the motion model between the two images and construct a panoramic image.

Note that there may be some perspective distortion between the two images. Perspective distortion is more complicated to handle than a linear or affine model and is not the purpose of this lab.

1. Assume we load two images from which we want to construct the panoramic image. Compute then the SIFT descriptors for each of both images and compute the correspondences between both images. Let xa be the matched SIFT keypoint coordinates of the first image, and xb the matched SIFT keypoint coordinates of the second image.

   ```
   Ia = imread('images/im2_ex1.jpg');
   Ib = imread('images/im1_ex1.jpg');
   Ia = single(rgb2gray(Ia));
   Ib = single(rgb2gray(Ib));
   [fa,da] = vl_sift(Ia);
   [fb,db] = vl_sift(Ib);
   [matches, scores] = vl_ubcmatch(da,db);
   numMatches = size(matches,2);
   xa = fa(1:2,matches(1,:));
   xb = fb(1:2,matches(2,:));
   ```

2. Apply the RANSAC algorithm: loop for M times, where M is a user specified parameter (e.g. $M = 100$), and perform the next task:

   (a) Select randomly some (in this example we select 10) of the previous correspondences:

   ```
   subset = vl_colsubset(1:numMatches, 10);
   ```

   (b) Assume that the two images correspond to each other according to a linear model (i.e. a translation):

   ```
   d = xb(1:2,subset) - xa(1:2,subset);
   p = mean(d,2);
   ```

   (c) Apply the previous obtained translation to all the correspondences of the first image

   ```
   xb_ = zeros(size(xb));
   for i=1:numMatches,
      xb_(:,i) = xa(:,i) + p;
   end
   ```

   (d) Check how many of these new points are near the original ones

   ```
   n = 0;
   for i=1:numMatches,
      e = xb_(:,i) - xb(:,i);
      if (norm(e) < 5), n = n + 1; end
   end
   ```

   (e) Among all randomly selected displacements, select the one with largest value of n. Use this displacement to create the panoramic image.

3. Once the best displacement has been found, construct the associated panoramic image by merging the two images. We discuss two possible solutions:

   (a) Note that the linear transformation gives us the transformation that we have to apply to a point at image `Ia` to obtain the corresponding point at (the coordinate system of) `Ib`. We may try to apply this transformation to map each point of `Ia` to (the coordinate system of) `Ib`. To construct the panoramic image we just need to paint all pixels that do not have a color assigned. This procedure is not a good idea, since an integer position of `Ia` will fall on a non-integer position of `Ib`. How do we draw a pixel on a non-integer position ? Do we just round to the nearest integer ? Just think a bit about it and you'll see that this is not a good way to proceed.

   (b) The second solution, which is usually used, is to use the inverse transform to compute the panoramic image. Assume we take the coordinate system of `Ib`. For each integer pixel of `Ib` (for which we do not know its color) we ask where from `Ia` it comes from. Again, the resulting pixel at `Ia` may be a non-integer, but this problem is easy to solve. Just use interpolation techniques.

   The algorithm is as follows. First, we take the corners of `Ia` and transform them to the coordinate system of `Ib` (which is our reference image). This allows us to compute the bounding box for the panoramic image

```
box2 = [1  size(Ia,2) size(Ia,2)  1;
        1  1          size(Ia,1)  size(Ia,1)];
box2_ = zeros(2,4);
for i=1:4,
   box2_(:,i) = box2(:,i) + p;
end
min_x = min(1,min(box2_(1,:)));
min_y = min(1,min(box2_(2,:)));
max_x = max(size(Ib,2),max(box2_(1,:)));
max_y = max(size(Ib,1),max(box2_(2,:)));
```

   We now create the panoramic image by sampling the image `Ib` and `Ia` at the corresponding points. Note that the inverse transformation is used for `Ia`.

```
ur = min_x:max_x;
vr = min_y:max_y;
[u,v] = meshgrid(ur,vr);
Ib_ = vl_imwbackward(im2double(Ib),u,v);
```

```
p_inverse = -p;
u_ = u + p_inverse(1);
v_ = v + p_inverse(2);
Ia_ = vl_imwbackward(im2double(Ia),u_,v_);

mass = ~isnan(Ib_) + ~isnan(Ia_);
Ib_(isnan(Ib_)) = 0;
Ia_(isnan(Ia_)) = 0;
panoramic = (Ib_ + Ia_) ./ mass;
imshow(panoramic,[]);
```

You are requested to deliver the associated code and the experimental results obtained with your algorithm.

5 Practicum submission

You are requested to deliver a PDF document including the experimentation performed during this first part of the lab and the second part (to be performed on November 4th). The report should included the answers to the questions proposed in this lab and the experimental results for the next lab. The PDF document should include all necessary images to fully understand your discussion.

References

[1] Lowe, “Distinctive image features from scale-invariant keypoints”, International Journal of Computer Vision, 60, 2 (2004), pp. 91-110.