Practical Session - Multiview Analysis

Abril 2016

Abstract

This lab concentrates on experimenting with several of the characteristics of multiview analysis. In order to simplify the problem we concentrate on pairs of images. The report is due for **April 19th**.

1 Previous work

Before the lab you are asked to take several photos of different objects and scenes from different point of views. The scenes should be rigid, i.e. non moving scenes. Take into account that the photos should contain textured objects (i.e. such that SIFT descriptors may be extracted from them). It is also a good idea to have objects with repetitive patterns (i.e. the windows of a building are repetitive). You also have several images included with this document.

You are also requested to read the mathematical formulation in the section before the lab session begins.

2 The Lab: projective transformation

The work described in this lab has a duration of three weeks. The report of the lab has to be delivered in one unique document (PDF is preferred) **April 19th**. You are expected to use MATLAB to do the computations.

The first exercise concentrates on learning how to compute and use the projective transformation (or homography) $\mathbf{H}$. In order to do this exercise you may either use set 1 or set 2.

2.1 Mathematical preliminaries

Recall that the projective transformation is given by

$$\begin{pmatrix} x'_1 \\ x'_2 \\ x'_3 \end{pmatrix} = \underbrace{\begin{pmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{pmatrix}}_{\mathbf{H}} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix}$$

Where $(x_1, x_2, x_3)^T$ and $(x'_1, x'_2, x'_3)^T$ are the homogenous coordinates of the points in the image. In order to transform homogenous to inhomogenous coordinates we set

$$(x, y) = (x_1/x_3, x_2/x_3)$$

For each pair of corresponding points, we have these two system of equations

$$\begin{aligned} x'(h_{31}x_1 + h_{32}x_2 + h_{33}x_3) &= h_{11}x_1 + x_{12}x_2 + h_{13}x_3 \\ y'(h_{31}x_1 + h_{32}x_2 + h_{33}x_3) &= h_{21}x_1 + x_{22}x_2 + h_{23}x_3 \end{aligned}$$

In order to completely specify the matrix $\mathbf{H}$, only 4 correspondences are needed (which gives rise to 8 equations). It is important that these 4 points in each image must be in a "general position", that is, no three points should be colinear. In order to solve the linear systems of equations the following approach can be followed.

Note that $\mathbf{x}'$ and $\mathbf{H}\,\mathbf{x}$ have the same direction but may differ in magnitude by a non-zero scale factor. This condition may be expressed in terms of the vector cross product as $\mathbf{x}' \times \mathbf{H}\,\mathbf{x} = 0$. This form enables a simple linear solution to be derived for $\mathbf{H}$.

If the j-th row of the matrix $\mathbf{H}$ is denoted by $\mathbf{h}_j^T$, then we may write

$$\mathbf{H}\,\mathbf{x} = \begin{pmatrix} \mathbf{h}_1^T\mathbf{x} \\ \mathbf{h}_2^T\mathbf{x} \\ \mathbf{h}_3^T\mathbf{x} \end{pmatrix}$$

writing $\mathbf{x}' = (x'_1, x'_2, x'_3)^T$, the cross product may then be given explicitly as

$$\mathbf{x}' \times \mathbf{H}\,\mathbf{x} = \begin{pmatrix} x'_2\mathbf{h}_3^T\mathbf{x} - x'_3\mathbf{h}_2^T\mathbf{x} \\ x'_3\mathbf{h}_1^T\mathbf{x} - x'_1\mathbf{h}_3^T\mathbf{x} \\ x'_1\mathbf{h}_2^T\mathbf{x} - x'_2\mathbf{h}_1^T\mathbf{x} \end{pmatrix}$$

Since $\mathbf{h}_j^T\mathbf{x} = \mathbf{x}^T\mathbf{h}_j$ for $j = 1 \ldots 3$ (i.e. the dot product is commutative), the equation $\mathbf{x}' \times \mathbf{H}\,\mathbf{x} = 0$ gives rise to a set of three equations in the entries of $\mathbf{H}$, which may we written in the form

$$\begin{pmatrix} \mathbf{0}^T & -x'_3\mathbf{x}^T & x'_2\mathbf{x}^T \\ x'_3\mathbf{x}^T & \mathbf{0}^T & -x'_1\mathbf{x}^T \\ -x'_2\mathbf{x}^T & x'_1\mathbf{x}^T & \mathbf{0}^T \end{pmatrix} \begin{pmatrix} \mathbf{h}_1 \\ \mathbf{h}_2 \\ \mathbf{h}_3 \end{pmatrix} = \mathbf{0} \tag{1}$$

This equation has the form $\mathbf{A}\,\mathbf{h} = \mathbf{0}$, where $\mathbf{A}$ is a 3×9 matrix, and $\mathbf{h}$ is a 9 vector made up of the entries of $\mathbf{H}$. The previous equation corresponds in fact to the equation $\mathbf{x}' \times \mathbf{H}\,\mathbf{x} = 0$ written in a more "tractable" way. Note that the previous equation is the equation for one correspondence (between $\mathbf{x}'$ and $\mathbf{x}$). The equation has to be satisfied for each of the corresponding points we have manually set. In the previous equation $\mathbf{h}$ is the vector to be solved.

Although there are 3 equations in Equation (1), only two of them are linearly independent (since the third row is obtained, up to scale, from the sum of x'_1 times the first row and x'_2 times the second). Thus, each correspondence (between $\mathbf{x}'$ and $\mathbf{x}$) gives two equations in the entries of $\mathbf{H}$. It is usual to omit the third equation in solving for $\mathbf{H}$. Then, the equation (1) can be written as

$$\begin{pmatrix} \mathbf{0}^T & -x'_3\mathbf{x}^T & x'_2\mathbf{x}^T \\ x'_3\mathbf{x}^T & \mathbf{0}^T & -x'_1\mathbf{x}^T \end{pmatrix} \begin{pmatrix} \mathbf{h}_1 \\ \mathbf{h}_2 \\ \mathbf{h}_3 \end{pmatrix} = \mathbf{0} \tag{2}$$

Please recall that the previous equation is the relationship for one correspondence (between $\mathbf{x}'$ and $\mathbf{x}$). Several relationships are obtained from equation (2) for every correspondences we have. Every correspondence (between $\mathbf{x}'$ and $\mathbf{x}$) gives rise to one set of equations as indicated in equation (2).

A simple way to solve for the linear system of equations is to set one of the elements of the matrix $\mathbf{H}$ to 1 (recall that the matrix $\mathbf{H}$ has 8 degrees of freedom). Imposing this condition is justified by the observation that the solution is determined only up to scale, and this scale can be chosen such that one of the elements is 1. For example, if the last element of $\mathbf{H}$, h_{33}, is chosen as unity then the resulting equations derived from Equation (2) are

$$\begin{pmatrix} 0 & 0 & 0 & -x'_3x_1 & -x'_3x_2 & -x'_3x_3 & x'_2x_1 & x'_2x_2 \\ x'_3x_1 & x'_3x_2 & x'_3x_3 & 0 & 0 & 0 & -x'_1x_1 & -x'_1x_2 \end{pmatrix} \tilde{\mathbf{h}} = \begin{pmatrix} -x'_2x_3 \\ x'_1x_3 \end{pmatrix}$$

where $\tilde{\mathbf{h}}$ is an 8-vector consisting of the 9 components of $\mathbf{h}$ excluding the last element. Concatenating the equations from four correspondences (between $\mathbf{x}'$ and $\mathbf{x}$) then generates a matrix equation of the form $\mathbf{M}\tilde{\mathbf{h}} = \mathbf{b}$, where $\mathbf{M}$ is 8×8 and $\mathbf{b}$ has 8 rows. Such an equation may be solved using standard techniques for solving linear equations.

The previous method works only if $h_{33} \neq 0$. If $h_{33} = 0$ is the true solution, then no multiplicative scale k can exists such that $kh_{33} = 1$. This means that the true solution cannot be reached. For this reason this method can lead to unstable results in case the chosen h_{ij} is close to zero. Consequently, this method is not recommended in general. For more information on more robust methods to compute the matrix $\mathbf{H}$ see [1], chapter 4 (which is completely devoted to the computation of $\mathbf{H}$).

2.2 Lab work

You are recommended to take several pairs of photos of the selected set and compare them using the following manual and automatic methods. The objective is to compute a projective transformation that maps one image to the other. Explain the results you have obtained in the deliverable. Include also the MATLAB code you have used.

Manual correspondence

We first follow the simplest approach: we are going to mark the corresponding points manually. Thus you do not have to worry about unreliable matchings that are usually obtained with automatic methods. In order to manually mark the image points in the two images, you may use the `ginput` MATLAB command. This command allows the user to manually select, with the mouse, a set of points in an image.

You are then requested to

1. Using the previous formulation, compute the matrix $\mathbf{H}$ between the two images.
2. Transform the first image into the second using the projective matrix $\mathbf{H}$. Rather than performing the forward projection (scanning all pixels $\mathbf{x}$ and compute $\mathbf{x}' = \mathbf{H}\,\mathbf{x}$), you are recommended to perform the inverse transformation (scanning pixels $\mathbf{x}'$ and ask where the pixels comes from with $\mathbf{x} = \mathbf{H^{-1}}\,\mathbf{x}'$). Can you explain why ?

Automatic correspondence

In order to do this part of the lab you are required to download the vlfeat library you already used in the "Computational Perception" course. The vlfeat library allows you to work with SIFT descriptors, see Section 3. Please read the latter section before continuing with the work you have do do.

You are requested to

1. Select a pair of images and compute its associated SIFT descriptors.
2. Select for instance the 8 best matches (see 3).
3. Compute the matrix $\mathbf{H}$ between the two images.
4. Again, as before, transform the first image into the second one using the method explained for the manual matching.

3 VlFeat library

This section contains the main issues you need to work with the vlfeat. It's an excerpt of the lab you did in the "Computational Perception" course.

3.1 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]. 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('toolbox/vl_setup');
   ```

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

5. To check if installation went right, run

   ```
   vl_version
   ```

3.2 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));
[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. One may view only a small subset of the previous keypoints by randomly selecting some of them

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

3.3 Feature matching

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 [2] 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);
```

4 Report

The report of these the lab have to be delivered in one unique document (PDF is preferred) latest **April 19th**. Please send it to `lluis.garrido@ub.edu`.

References

[1] R. Hartley and A. Zisserman, “Multiple view geometry in computer vision”, Cambridge.

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