Multiview Analysis

Miquel Perelló, E-mail: miquel.perello.nieto@est.fib.upc.edu

Abstract

This lab concentrates on experimenting with several of the characteris- tics of multiview analysis. In order to simplify the problem we concentrate on pairs of images.

Keywords

projective transformation, SIFT descriptors, fundamental matrix

1 Projective Transformation

We want to retrieve the frontal image of the different books in the image. To achieve that, we are going to create the projective transformation for each one to match a rectangle where each point represents one of the vertexes of the book.

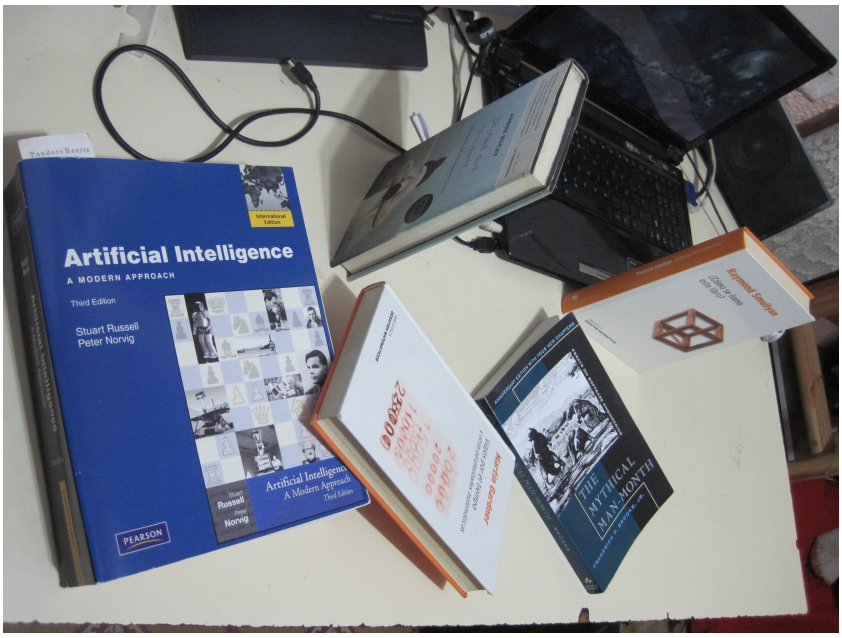

Fig. 1. - Some books on the desktop

For computing the projective transformation matrix we can solve the linear problem $Mh = b$,

First of all we get the matching points, next we can compute the values of the matrix M with the *linsolve* function implemented in *Matlab*. This function expects an equation of the type $A * X = B$ for that reason in the code $M = A$ and $B = b$

Listing 1. steps for computing the homography

```
startPoints = [y1 x1 ones(4,1)];
endPoints = [y2 x2 ones(4,1)];
[M, B] = computeMatrices(startPoints, endPoints);
h = linsolve(M, B);
```

```
h = [1 ; h];
H = reshape(h,3,3)';
```

To compute the matrix H we need to create the matrix M and the vector B where M is a $8x8$ matrix which corresponds to two rows for each two points matching, and B is a vector corresponding to the right part of the equations for the same points.

Listing 2. computeMatrices

```
function [A, B] = computeMatrices(startPoints, endPoints)
    A = [];
    B = [];
    for i = 1:size(startPoints,1)
        s = startPoints(i,:);
        e = endPoints(i,:);
        A = [A ;
            0 0 -e(3)*s(1) -e(3)*s(2) -e(3)*s(3) e(2)*s(1) e(2)*s(2) e
               (2)*s(3);
            e(3)*s(2) e(3)*s(3) 0 0 0 -e(1)*s(1) -e(1)*s(2) -e(1)*s(3)
               ];
        B = [B ; 0 ; -e(3)*s(1)];
    end
end
```

Once we have computed the matrix H we can create the new image with the initials projection. The main idea is to create a white image with the desired dimensions, and for each of the pixels of the image, we can compute the inverse transformation of this point and retrieve the corresponding pixel from the initial image. We can do that because $x' = Hx$ and $x = H^{-1}x'$.

Listing 3. creating the new image

```
hi = size(b,1);
le = size(b,2);
I2 = ones(hi,le);

index = 0;
for i = 1:hi
    for j = 1:le
        index = index+1;
        xFinal = inv(H)*[i j 1]';
        xFinal = round(xFinal/xFinal(3));

        if (xFinal(1) > 0 & xFinal(2) > 0 & xFinal(1) <= hiA & xFinal
           (2) <= leA)
            I2(i,j) = aGray(xFinal(1), xFinal(2));
        else
            I2(i,j) = 0;
        end
    end
end
```

Finally, if we execute all the steps for all the books, we can retrieve the frontal view of each one. With these images is possible to read their titles.

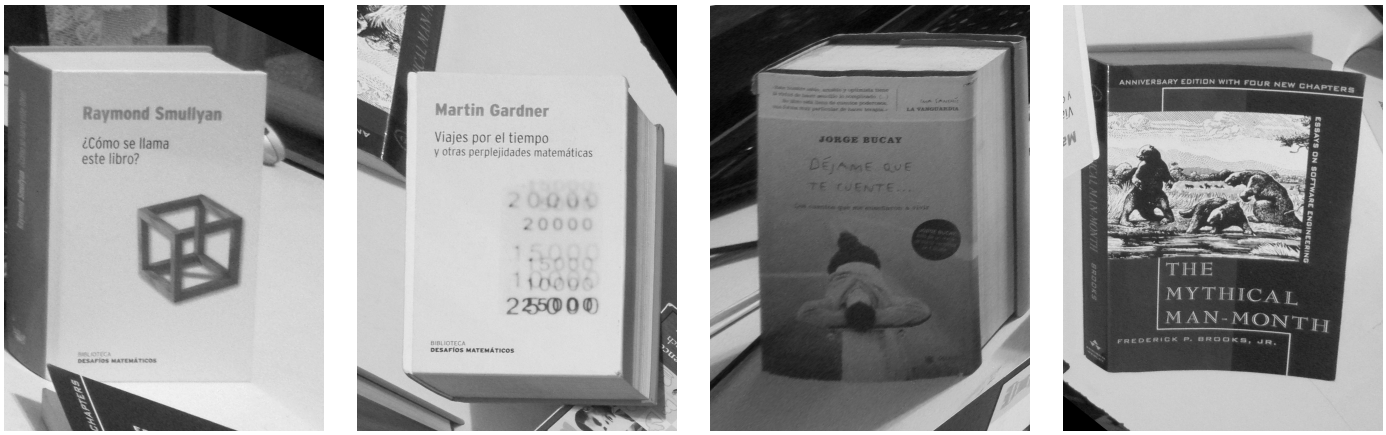

TABLE 1
Projective transformation of four different books into a rectangular distribution points

2 Fundamental Matrix

We are going to compute the fundamental matrix with the equation $x'Fx = 0$ where x' are the points of the left image and x are the right ones.

First we need to select the matching points. In that case, this is done by selecting SIFT descriptors from both images and matching them from one image to the other. For reducing the number of matches we added a threshold (depending on the image it can be smaller or bigger). This removes the matches that are not greater than $distance(D1, D2) * THRESH$ where $D1$ and $D2$ are the 128 SIFT descriptors of the two points.

Listing 4. computing sift descriptors and matches

```
[fa, da] = vl_sift(Ia);
[fb, db] = vl_sift(Ib);

[matches, scores] = vl_ubcmatch(da, db, 3.0);
```

Once we have the matching points, we can choose the desired number of points. In that case I executed with two different number of points, in one case twelve and other with thirty. Furthermore, we add a new dimension to the points with a one.

Listing 5. Selecting the best matches

```
numPoints = 12;
[value, index] = sort(scores);
startx = [fa(1:2,matches(1,index(1:numPoints))); ones(1,numPoints)];
endx = [fb(1:2,matches(2,index(1:numPoints))); ones(1,numPoints)];
```

Given that the dimensions of the image are 900 by 700 pixels, the next computations would carry a lot of problems. For that reason we can normalise these values to be in the range [-1,1].

I constructed a transformation matrix for each image, adjusting the values within the min value and max value of x and y. It must be better to adjust individually for each variable, but in this example it was not necessary.

Listing 6. Normalization of the points

```
% normalization of startx
```

```
startMin = min(min(startx(1:2,:)));
startMax = max(max(startx(1:2,:)));

scale = (startMax-startMin)^-1;
T1 = [ scale*2  0        -1  ;
       0        scale*2  -1  ;
       0        0        1   ];
startx_norm = T1*startx;

% normalization of endx
endMin   = min(min(endx(1:2,:)));
endMax   = max(max(endx(1:2,:)));

scale = (endMax-endMin)^-1;
T2 = [ scale*2  0        -1  ;
       0        scale*2  -1  ;
       0        0        1   ];
endx_norm   = T2*endx;
```

The next step is to compute the f vector. One way to compute is calculating the eigenvector with the smaller corresponding eigenvalue for $A^T A. Where A is a N x 9 matrix where each row is one of the matches.$

Listing 7. compute A matrix

```
A = coordinatesMatrix(startx_norm, endx_norm);
```

Listing 8. function coordinatesMatrix

```
function A = coordinatesMatrix(startPoints, endPoints)
    numPoints = size(startPoints,2);
    x1 = startPoints;
    x2 = endPoints;
    A = [x2(1,:)'.*x1(1,:)'   x2(1,:)'.*x1(2,:)'  x2(1,:)' ...
         x2(2,:)'.*x1(1,:)'   x2(2,:)'.*x1(2,:)'  x2(2,:)' ...
         x1(1,:)'             x1(2,:)'            ones(numPoints,1) ];
end
```

Here we compute the eigenvector and select the first that corresponds to f.

```
[V, D] = eig(A'*A);
f = V(:,1);
F = reshape(f,3,3)';
```

Once we have the fundamental matrix, we need to project into a rank 2 subspace, this is done putting a zero in the smaller value of the diagonal of their singular value decomposition.

Finally we need to add the denormalisation from the first image, to the F *and then from the* F *to the second image.*

Listing 9. projecting F onto the rank 2 subspace

```
[Q D R] = svd(F,0);
[Y I] = min(diag(D));
D(I,I) = 0;
F = Q*D*R';
```

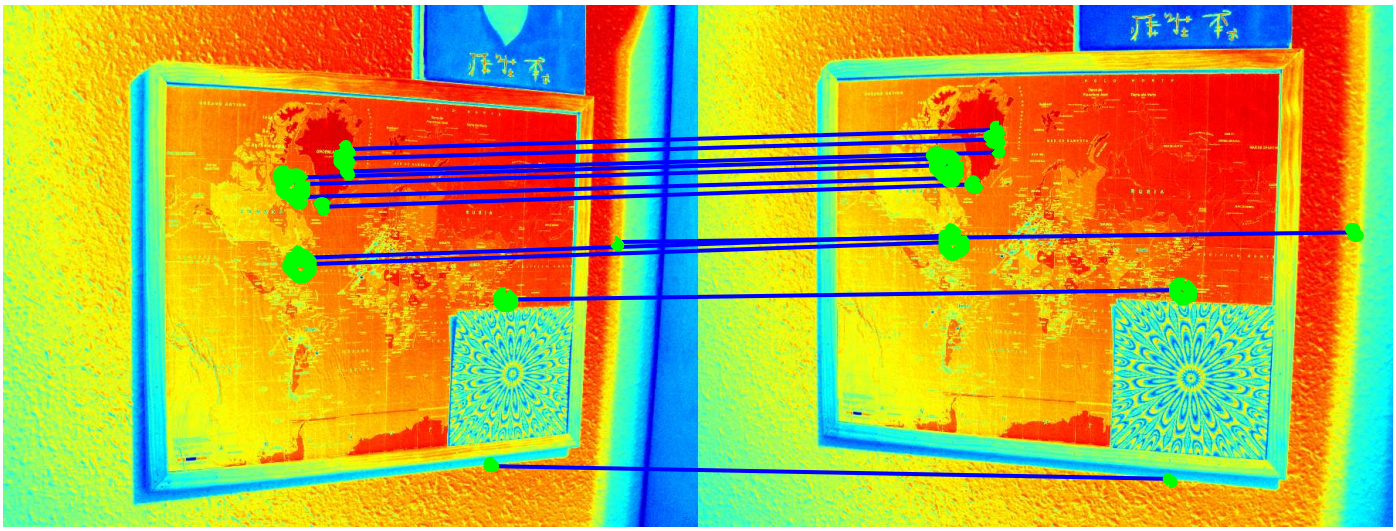

Fig. 2. Twelve matching points

```
% Denormalisation
F = T2'*F*T1;
```

With the fundamental matrix we can compute the lines for one image and the other.

Listing 10. computing the lines

```
startL = F*startx ;
startY = [-startL(3,:) ./ startL(2,:);
            -(wi * startL(1,:) + startL(3,:)) ./ startL(2,:)];

endL = F'*endx ;
endY = [-endL(3,:) ./ endL(2,:);
            -(wi * endL(1,:) + endL(3,:)) ./ endL(2,:)];
```

Here we can draw the computed lines and the matching points to see if the lines really cross the points.

Listing 11. drawing the lines and the points

```
imshow([Ia Ib],[]) ;
hold on ;

line(repmat([0 ; wi], 1, numPoints), endY, 'Color', 'green');
plot(startx(1, :)', startx(2, :)', 'ro', 'MarkerSize', 6);

line(repmat([wi ; wi*2], 1, numPoints), startY, 'Color', 'blue');
plot(wi + endx(1,:)', endx(2, :)', 'ro', 'MarkerSize', 6);
```

The result depends on the number of matching points, how can be seen in the final pictures.

REFERENCES

[1] *Low, David G. "Distinctive image features from scale-invariant keypoints." International Journal of Computer Vision 60.2 (2004): 91-110.*

[2] *Kovesi, Peter, http://www.csse.uwa.edu.au/ pk/research/matlabfns/*

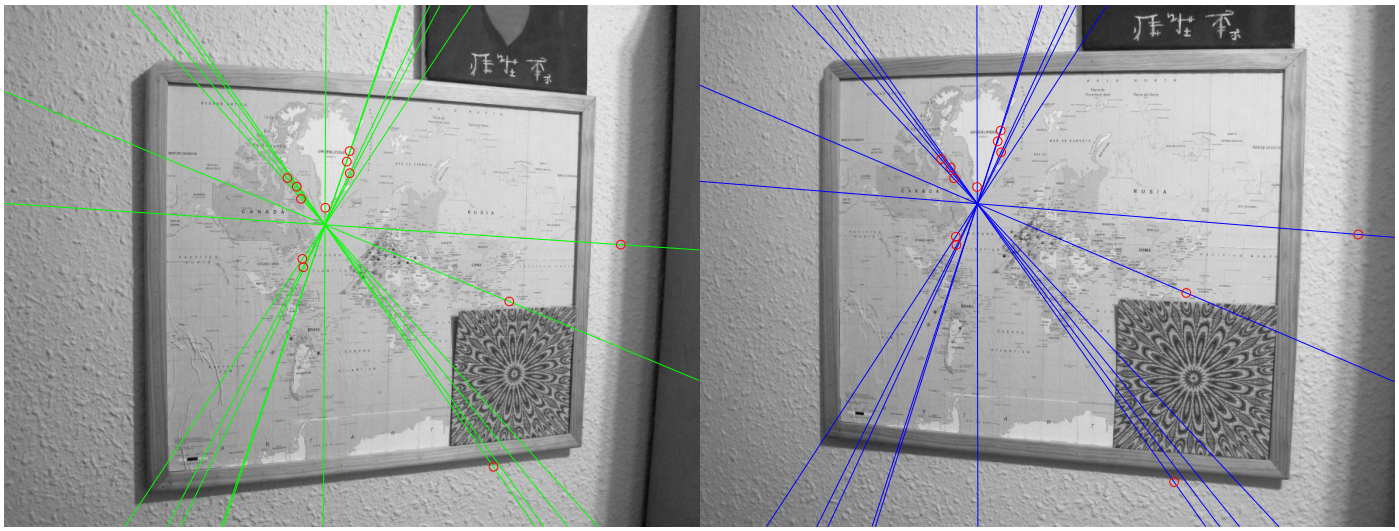

Fig. 3. Epipolar lines for twelve matching points

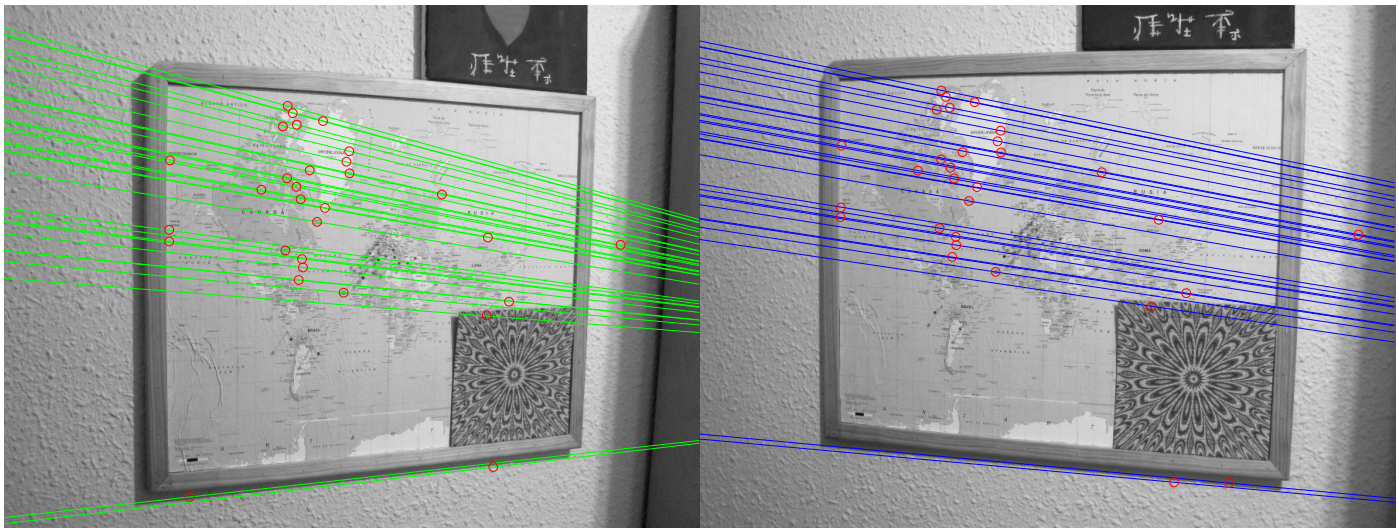

Fig. 4. Epipolar lines for thirty matching points