EEE3032 – Computer Vision and Pattern Recognition

Lab Worksheet 10 – Multiview Geometry Sheet 2
Miroslaw Bober (m.bober@surrey.ac.uk)

Task and Learning Objective: In today's lab you will experiment with camera geometry as covered in the past 2 weeks of lectures.

Resources: The exercises will be performed using Matlab. Download the Matlab code for the CVPR labs if youhaven't done already, and unzip the code into a folder.

In **Ex1-4** you will work with a pair of images taken from two different viewpoints using a single camera. This is equivalent to two cameras taking a photo of the object from different views. In fact it is simpler because the intrinsics (internal camera parameters) for both views are the same.

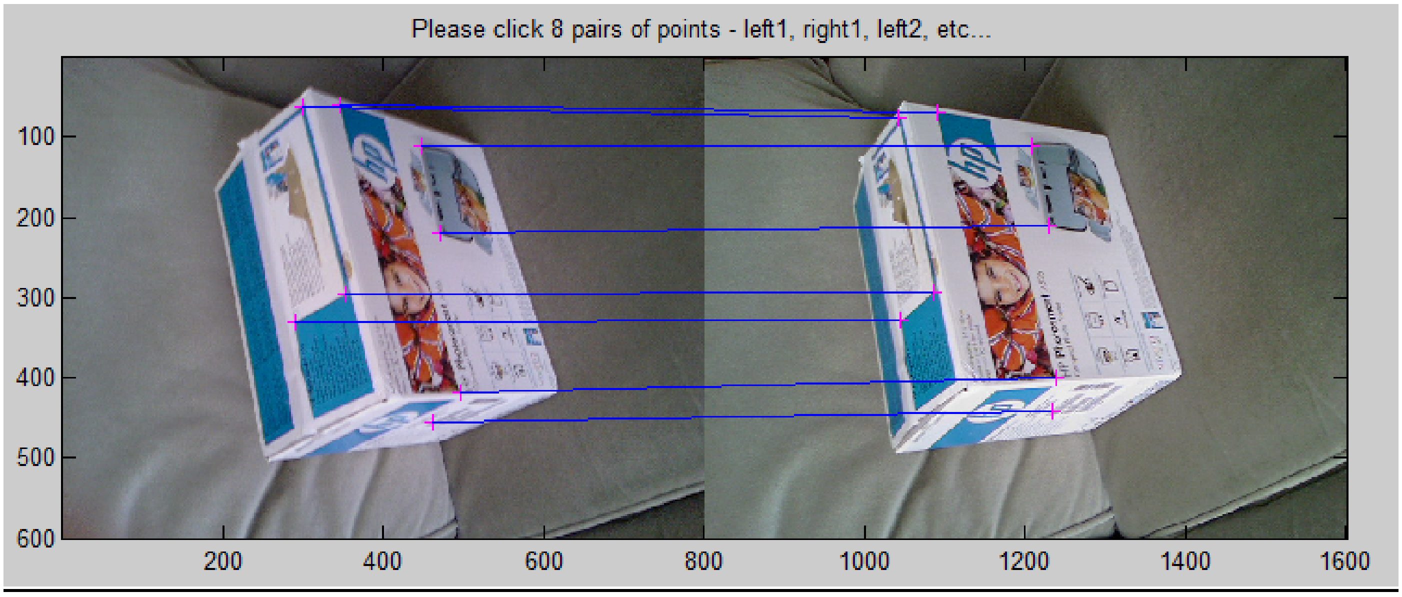
In **Ex5-6** you will work with images capture simultaneously from multiple views

Ex1: Estimating the Fundamental Matrix – The 8 Point Algorithm *(5 minutes)*

Run the program **Pick8Points.** You should manually click on 8 pairs of corresponding points in the images. It will be easiest to do this if you maximise the window containing the images. Click in this order... lefthand point 1, righthand point 1, lefthand point 2, righthand point 2 and so on. Use all sides of the cardboard box – remember that you cannot estimate the fundamental matrix using 8 points that lie on the same 3D plane (are co-planar)

When you are done, you will have a matrix **Pleft** and **Pright** containing the 2D points for each image.

Now run **EstimateF_8PointAlgorithm**. The fundamental matrix will be calculated in variable **F** from **Pleft** and **Pright**.


Please click 8 pairs of points - left1, right1, left2, etc...
100
200
300
400
500
600
200
400
600
800
1000
1200
1400
1600

If you are unable to accurately click on the points then load a pre-supplied **Pleft** and **Pright** by typing:

```
>> load eightpoints
```

Ex2: Checking the 8 Point Algorithm code *(5 minutes)*

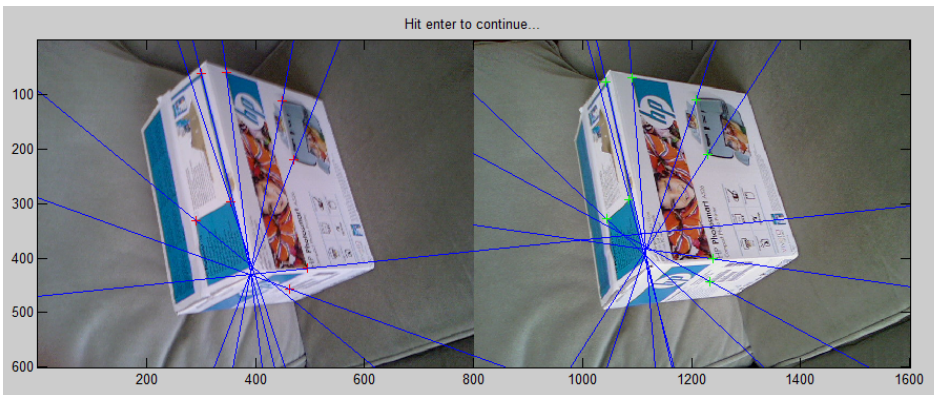
Look at the code for program **EstimateF_8PointAlgorithm** . Ensure you understand how the linear system in **A** has been derived from the mathematical relationship between two homogeneous points in two views. You can verify it by multiplying out the RHS of this equation – and obtaining expressions for α**x** and α**y** and α. By substituting α into the expression for α**x** and α**y** you should see how the factors for **A** arise in the code.

$$\begin{pmatrix} \alpha x \\ \alpha y \\ \alpha \end{pmatrix} = \begin{pmatrix} f11 & f12 & f13 \\ f21 & f22 & f23 \\ f31 & f32 & f33 \end{pmatrix} \begin{pmatrix} u \\ v \\ 1 \end{pmatrix}$$

Ex3: Visually checking the accuracy of F *(5 minutes)*

Run the program **EpipolarConstraint**. The program will visualise each pair of points in turn, with their corresponding epipolar lines. For example the left-hand point will be drawn, and its corresponding epipolar line on the right. The right-hand point will be drawn (green) and its corresponding epipolar line on the left will also be drawn. Keep pressing space/enter until all the points are drawn.

If the points you clicked in Ex.1 were accurate enough, you should see that a point in one view maps to line passing through the corresponding point in the other view. If not, repeat Ex.1 and if it still doesn't work load the presupplied points – and don't forget to calculate F again.



Now run the program **InteractiveEpipolarConstraint** and click anywhere in either view. The corresponding epipolar line will be drawn in the other view. Visually confirm that the line passes through its corresponding point.

Ex4: Mathematically checking the accuracy of F *(5 minutes)*

Recall that the fundamental matrix enforces the epipolar constraint, i.e. that

$$\begin{pmatrix} x_L & y_L & 1 \end{pmatrix} \mathbf{F} \begin{pmatrix} x_R \\ y_R \\ 1 \end{pmatrix} = 0$$

Where subscript L or R here indicates the left and right (x,y) coordinates of a pair of corresponding points.

Verify this yourself using **Pleft**, **Pright** and **F**.

First, ensure **Pleft** and **Pright** are in homogeneous form by adding a row of 1s to them

```
>>  Pleft(3,:)=1;

>>  Pright(3,:)=1;
```

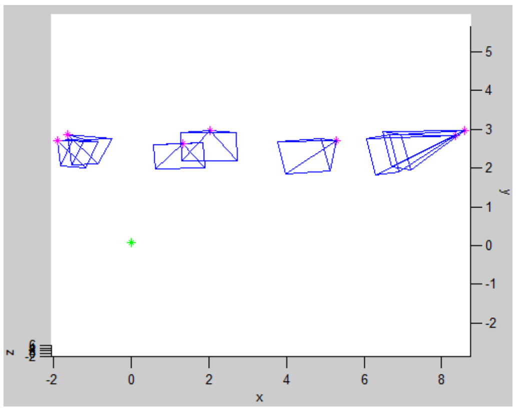
Now check a particular pair of points, for example corresponding point **pair 5** (but you can try others):

```
>> Pleft(:,5)' * F * Pright(:,5)
```

The result of the above calculation should be 0, or very close to 0 e.g . 10^{-15}

Ex5: Triangulating points *(10 minutes)*

Run the program **studio**. This will load up the intrinsic (K) and extrinsics (R,T) for 7 cameras calibrated in the Surrey Visual Media Lab. The position of the cameras is visualized. In this case, the floor of the studio is on the x-z plane and the origin of the space (0,0,0) is also visualised as a green star. Use the Matlab 3D rotation tool to explore the figure – for example, below the camera is position to look "across" the studio (recall, origin is on the floor)



You have been supplied with some observations of a 3D point in the studio, made by each of the 7 cameras. Load up a couple of the camera's observations (here we choose cameras 1 and 5) via:

```
>> pt1=loadViewpoint(1);
>> pt2=loadViewpoint(5);
```

The 2x1000 matrices **pt1** and **pt2** just created contain 1000 synchronised observations of the point, extracted from 1000 video frames. Some entries will be [-1 ; -1] if the object was hidden (occluded) from the camera at that time. If you used camera 1 and 5 above, you should find frame 47 is visible from both those views. Check this...

```
>> pt1(:,47)
>> pt2(:,47)
```

Now let's work out the light ray that passes through camera 1's Centre of Projection (COP), through the observed point, and out into the studio. This is a direct application of the equations covered in lectures **(check the Matlab code you are about to run to verify it matches the Maths you covered)**:

```
>> [A1 B1]=getLightRay(CAMS(1).R,CAMS(1).T,CAMS(1).K,pt1(:,47));
>> [A2 B2]=getLightRay(CAMS(5).R,CAMS(5).T,CAMS(5).K,pt2(:,47));
```

Now we have the parametric equations for two rays of light... in the form **P(s) = A + sB** *(where A is A1 or A2 etc).* Recall that intersecting these rays gives us the coordinates of the 3D point in the studio. Of course the rays might not exactly intersect, so we should instead find the point in the space closest to both lines.

```
>> p=intersectRays(A1,B1,A1,B2)
```

The cameras were calibrated to give coordinates in metres. You should find that p corresponds to a point around 95cm off the studio floor (i.e. up the y axis)... recall the origin is on the studio floor at (0,0,0).

Plot the point **p** in the studio (if you closed the figure, re-run **studio**)

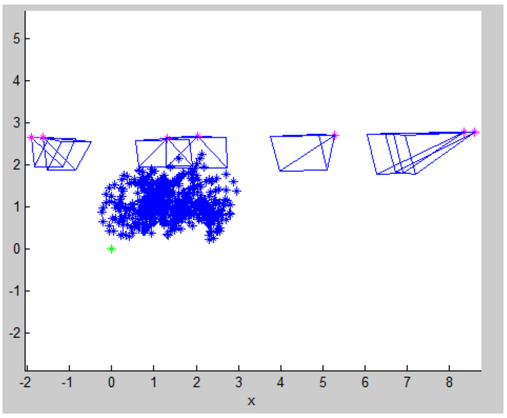
```
>> plot3(p(1),p(2),p(3),'bx');
```

Take a look at the code for **intersectRays**. Verify it matches the maths covered in lectures for interesting two rays.

You must ensure you are very familiar with the derivation of the mathematics of intersecting two rays

Ex6: Triangulate all the points *(10 minutes)*

The 3D point you just triangulated was a marker being moved around the studio by hand. Can you write a loop to plot all 1000 of the 3D positions of that object? Remember not to use the coordinates when one or both cameras couldn't observe the point i.e. the coordinates are (-1,-1). If you give up, then look at program **triangulate**



Ex7: Calculating the Homography *(5 minutes)*

First, run the program **testhomog** – and input 4 sets of corresponding points. Be careful to ensure that the 4 points you click on in the first image are exactly the points you click on in the second image – and the structure of the building in the photos is quite repetitive. You are given variables **P** and **Q** for the left and right 4 points respectively.

The program **calchomography** will calculate the homography **H** between four pairs of point correspondences.

```
>> H=calchomography(P,Q)

H =
   1.1126   0.0301   37.7303
   0.1177   1.0722   -8.4671
   0.0006   0.0000    1.0000
```

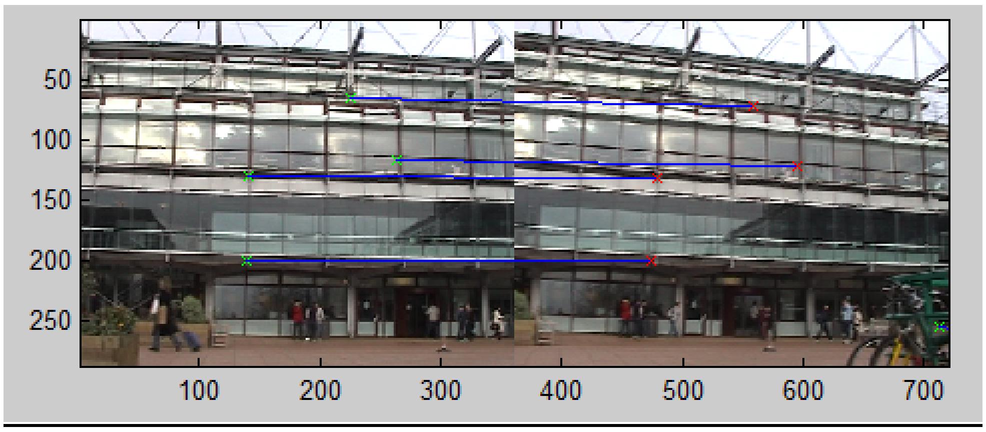
Satisfy yourself that **H** maps **P** to **Q** (and vice versa via **inv(H)**)

```
>> P(3,:)=1
>> test=H*P;
>> test(1,:)=test(1,:)./test(3,:);
>> test(2,:)=test(2,:)./test(3,:);
```

Now take a look at the code for calchomography and confirm that this matches the maths covered in lectures.
It is very important that you understand how to derive this mathematics from first principles, as in lectures.

Ex8: Visually Verifying the Homography *(5 minutes)*

Run the program **homoginteractive**. Click on points on the left-hand image – the corresponding point should appear on the righthand image. This program uses the H you just calculated. Check the code to ensure you understand how points on the left-hand image and being mapped onto the right using the matrix multiplication with H and homogeneous coordinates.



Repeat exercise **Ex.7** and **Ex.8** but choose points that:

1) All lie in a single line

or

2) Do not lie upon the same plane in the image (e.g. not all on the front of the building)

Does the homography based correspondence still work?