

# Exercise on Epipolar Geometry & Triangulation

April 2, 2019

In this exercise, we will work with a few assignments related to epipolar geometry and triangulation.

## 1 Epipolar Geometry

In this exercise, you will work on fundamental matrix estimation using the famous Eight-point algorithm. Recall the epipolar constraint formed with the fundamental matrix as

$$\mathbf{x}_2^T \mathbf{F} \mathbf{x}_1 = 0$$

where  $\mathbf{x}_1 = [x_1, y_1, 1]^T$  (homogeneous coordinate) denotes the point in image 1 with its corresponding point in image 2 being  $\mathbf{x}_2 = [x_2, y_2, 1]^T$ ,  $\mathbf{F}$  is the fundamental matrix. Expanding this equation, we will have the following linear equation:

$$[x_2, y_2, 1] \begin{bmatrix} f_{11} & f_{12} & f_{13} \\ f_{21} & f_{22} & f_{23} \\ f_{31} & f_{32} & f_{33} \end{bmatrix} \begin{bmatrix} x_1 \\ y_1 \\ 1 \end{bmatrix} = 0 \Rightarrow$$
$$[x_1 x_2 & x_1 y_2 & x_1 & y_1 x_2 & y_1 y_2 & y_1 & x_2 & y_2 & 1] \begin{bmatrix} f_{11} \\ f_{21} \\ f_{31} \\ f_{12} \\ f_{22} \\ f_{32} \\ f_{13} \\ f_{23} \\ f_{33} \end{bmatrix} = 0$$

We need at least 8 points to determine  $\mathbf{F}$ <sup>1</sup>. Stacking them together will give us a linear system:

$$\mathbf{A}\mathbf{x} = 0$$

where  $\mathbf{x} = [f_{11}, f_{21}, f_{31}, f_{12}, f_{22}, f_{32}, f_{13}, f_{23}, f_{33}]^T$ . By solving  $\mathbf{x}$  using SVD, the fundamental matrix can be estimated.

**Q1:** Implement the Eight-point algorithm use the provided data (**Fdata.mat**) as the input. **Fdata.mat** contains feature points detected and matched. You can visualize using the following code:

```
1 %% loading
2 im1 = rgb2gray(imread('000002.bmp'));
3 im2 = rgb2gray(imread('000003.bmp'));
4 load('calib.mat');
5 [im1,~] = undistortImage(im1,cameraParams);
6 [im2,~] = undistortImage(im2,cameraParams);
7
8 load('Fdata.mat');
9
10 im3 = cat(2,im1,im2);
```

<sup>1</sup> $\mathbf{F}$  is determined up to a scale, so without losing generality, we can make  $f_{33} = 1$  and then there are 8 unknowns left.

```

11 figure;imshow(im3);hold on;
12
13 plot(x1(:,1),x1(:,2),'ro');
14 plot(x2(:,1)+size(im1,2),x2(:,2),'go');
15
16 shift = size(im1,2);
17 cmap = lines(5);
18 k = 1;
19 for i = 1:size(x1,1)
20     ptdraw = [x1(i,1), x1(i,2);
21                 x2(i,1)+shift, x2(i,2)];
22     plot(ptdraw(:,1),ptdraw(:,2),'LineStyle','-', 'LineWidth',1,'Color',cmap(k,:));
23     k = mod(k+1,5);if k == 0 k = 1;end
24 end
25
26 %% here starts your code,
27 F = DLT8pt(x1',x2');
28
29 %% once you have estimated F, check your F using
30 vgg-gui.F(im1,im2,F');

```

**Q2:** Once you have estimate  $\mathbf{F}$ , you can check your result visually.

## 2 Triangulation

In this exercise, you need to implement the triangulation method you have learnt from the lecture. You start with simulated data and then work with real images.

**Q3:** Use the following code to generate simulated data for triangulation

```

1 N = 100;
2 p = rand([3,N]) * 10 - 5;% from near to far
3 p(1,:) = p(1,:) + 15;
4 C1 = [0;0;0];
5 C2 = [0;0.2;0];
6
7 rad1 = -10/57.3;
8 R1 = [cos(rad1) 0 -sin(rad1);0 1 0;sin(rad1) 0 cos(rad1)]*[0 -1 0;0 0 -1;1 0 0];
9 R2 = [cos(-rad1) 0 -sin(-rad1);0 1 0;sin(-rad1) 0 cos(-rad1)]*[0 -1 0;0 0 -1;1 0 0];
10

```



Figure 1: Check you result visually using epipolar geometry. If the  $\mathbf{F}$  is accurately estimated, then by for one point (green +) in image 1, its corresponding point in image 2 will be on the epipolar line (green line).

```

11 t1 = -R1*C1;
12 t2 = -R2*C2;
13
14 figure
15 h1 = plot3(p(1,:),p(2,:),p(3,:),'g*');hold on;
16 cam1 = plotCamera('Location',C1,'Orientation',R1,'Opacity',0,'Color',[1 0 ...
17 0],'Size',0.4,'Label','Cameral');
18 cam2 = plotCamera('Location',C2,'Orientation',R2,'Opacity',0,'Color',[0 1 ...
19 0],'Size',0.4,'Label','Camera2');
20 axis equal
21 xlabel('x: (m)');
22 ylabel('y: (m)');
23 zlabel('z: (m)');
24 title('Triangulation Simulation');
25 set(gca,'FontName','Arial','FontSize',20);
26
27 K = [500 0 320;0 500 240;0 0 1];
28
29 [uv1, in1] = proj(R1,t1, p, K);
30 [uv2, in2] = proj(R2,t2, p, K);
31
32 in = in1 & in2;
33 q1 = uv1(:,in);
34 ptrue = p(:,in);
35 q2 = uv2(:,in);
36
37 q1(1:2,:) = q1(1:2,:);
38 q2(1:2,:) = q2(1:2,:);
39
40 P1 = K*[R1 t1];
41 P2 = K*[R2 t2];
42
43 precons = zeros(3,size(q1,2));
44 for i = 1:size(q1,2)
45 %% here starts your code
46 precons(:,i) = triangulation(q1(:,i),P1,q2(:,i),P2);
47 end
48
49 %% visualization
50 figure
51 plot3(ptrue(1,:),ptrue(2,:),ptrue(3,:),'ro','MarkerSize',8);hold on;
52 plot3(precons(1,:),precons(2,:),precons(3,:),'g+','MarkerSize',8);hold on;
53 xlabel('x: (m)');
54 ylabel('y: (m)');
55 zlabel('z: (m)');
56 title('Triangulation Simulation');
57 legend({'Truth','Reconstruction'});
58 set(gca,'FontName','Arial','FontSize',20);
59 grid on;

```

**Q4:** Now you will apply your triangulation code on real images<sup>2</sup>. To save your time, parsing code is provided as follows:

```

1 baseDir = './images/';% use you own directory
2 buildingScene = imageDatastore(baseDir);
3 numImages = numel(buildingScene.Files);
4
5 load(strcat(baseDir,'viff.xy'));
6 x = viff(1:end,1:2:72); % pull out x coord of all tracks
7 y = viff(1:end,2:2:72); % pull out y coord of all tracks
8
9 %% visualization
10 num = 0;
11 for n = 1:numImages-1
12     im1 = readimage(buildingScene, n);
13     imshow(im1); hold on;
14     id = x(n,:) ~= -1 & y(n,:) ~= -1;

```

<sup>2</sup>Data source: "Dinosaur" from [www.robots.ox.ac.uk/~vgg/data/mview.html](http://www.robots.ox.ac.uk/~vgg/data/mview.html).

```

15     plot(x(n,id),y(n,id),'go');
16     num = num + sum(id);
17     hold off;
18     pause(0.1);
19 end
20
21 % load projection matrices
22 load(strcat(baseDir,'dino_Ps.mat'));
23
24
25 ptcloud = zeros(3,num);
26 k = 1;
27 for i = 1:size(x,1)-1
28     % tracked features
29     id = x(i,:) ~= -1 & y(i,:) ~= -1 & x(i+1,:) ~= -1 & y(i+1,:) ~= -1;
30     q1 = [x(i,id);y(i,id)];%
31     q2 = [x(i+1,id);y(i+1,id)];%
32 P1 = P{i};
33 P2 = P{i+1};
34
35 precons = zeros(3,size(q1,2));
36 for j = 1:size(q1,2)
37     % your code starts
38     precons(:,j) = xxxxx(q1(:,j),P1,q2(:,j),P2);
39 end
40
41 ptcloud(:,k:k+size(q1,2)-1) = precons;
42 k = k + size(q1,2);
43 end
44 figure
45 plot3(ptcloud(1,:),ptcloud(2,:),ptcloud(3,:),'k.','MarkerSize',10);hold on;
46 grid on;
47 axis equal;
48 view(3);
49 for i = 1:size(x,1)
50     P1 = P{i};
51     [K, R, t, c] = decomposeP(P1);
52     plotCamera('Location',c,'Orientation',R,'Opacity',0,'Color',[0 1 0],'Size',0.05);
53 end

```

. If your implementation is correct, you will see a dinosaur.

**Q5:** Triangulation can be made much easily if stereo vision system is used. In this exercise, you will implement the triangulation method for stereo vision. Stereo images<sup>3</sup> and the corresponding disparity are provided. The following code is provided for data parsing and visualization.

```

1 I1 = imread('im0.png');
2 I2 = imread('im1.png');
3 figure;imshow(I1);
4
5 load('disp.mat');
6 disparityRange = [0 480];
7 figure
8 imshow(disparityMap,disparityRange)
9 title('Disparity Map')
10 colormap jet
11 colorbar
12 % stereo vision intrinsics
13 f = 4841.191;
14 px = 1441.607;
15 py = 969.311;
16 % baseline: m
17 b = 170.458*1e-3;
18 % only triangulate pixel whose disparity is within [100,480].
19 id = disparityMap >= 100 & disparityMap <= 480;
20
21 % your code starts

```

<sup>3</sup>Source: "Mask" from [vision.middlebury.edu/stereo/data/2014/](http://vision.middlebury.edu/stereo/data/2014/)

```

22 ptcloud = xxxx(disparityMap, f, px, py, b, id);
23
24 % display point cloud
25 colorcloud = uint8(zeros(size(ptcloud,1),3));
26 r = I1(:,:,1); g = I1(:,:,2); b = I1(:,:,3);
27 colorcloud(:,1) = r(id);
28 colorcloud(:,2) = g(id);
29 colorcloud(:,3) = b(id);
30 ptCloud = pointCloud(ptcloud,'Color',colorcloud);
31 pcshow(ptCloud);

```

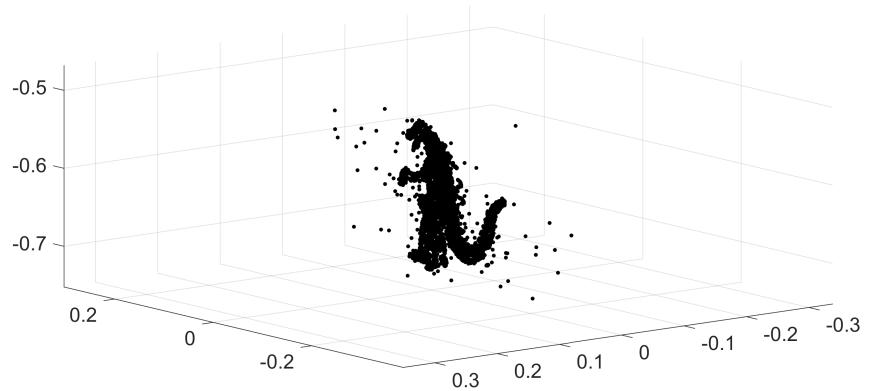


Figure 2: Dinosaur triangulated.

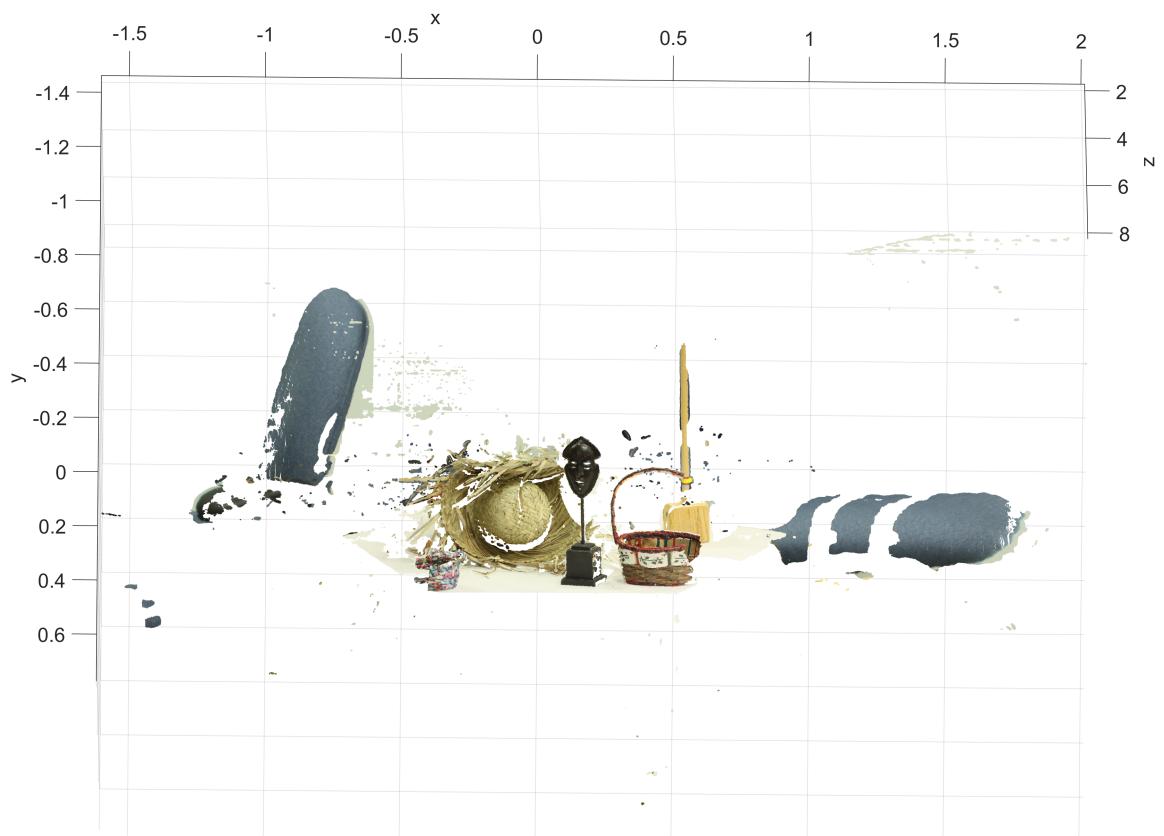


Figure 3: Result obtained by triangulating stereo images.