# Assignment 3: Epipolar Geometry and Triangulation

In this assignment, we will work with problems related to epipolar geometry and triangulation.

## 1 Fundamental Matrix

In this task you should implement an algorithm for estimation of the fundamental matrix given a number of point correspondences in a stereo image pair.

It should be recalled from the lecture slides, that the epipolar constraint imposed by the fundamental matrix is the following

$$\mathbf{x_2^T F x_1} = \mathbf{0}$$

where $\mathbf{x_1} = [x_1 \quad y_1 \quad 1]^T$ and $\mathbf{x_2} = [x_2 \quad y_2 \quad 1]^T$ are corresponding image points in a left and right image respectively. $\mathbf{F}$ is the fundamental matrix and this can be estimated by having a minimum of 8 point correspondences between the two views by Longuet-Higgins 8 point algorithm:

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

$$\Rightarrow \qquad [x_1x_2 \quad x_1y_2 \quad x_1 \quad y_1x_2 \quad y_1y_2 \quad y_1 \quad x_2 \quad y_2 \quad 1]\begin{bmatrix} f_{11} \\ f_{21} \\ f_{31} \\ f_{12} \\ f_{22} \\ f_{32} \\ f_{13} \\ f_{23} \\ f_{33} \end{bmatrix} = 0 \quad (\text{eq. } 1)$$

From the last form of the equation, we can set up a system of equations, such that

$$\mathbf{Ax} = \mathbf{0}$$

Where $\mathbf{x} = [f_{11} \quad f_{21} \quad f_{31} \quad f_{12} \quad f_{22} \quad f_{32} \quad f_{13} \quad f_{23} \quad f_{33}]^T$ and where $\mathbf{A}$ is stacked rows of point correspondences in the form shown in eq. 1.

Copy the code below in to a MATLAB script and download the referenced files from DTU Learn. The code visualizes two images of the same scene with different view-points and provides 14 point correspondences between the views in x1,x2.

```
im1 = rgb2gray(imread('000002.bmp'));
```

```
im2 = rgb2gray(imread('000003.bmp'));
load('calib.mat');
[im1,~] = undistortImage(im1,cameraParams); % Correct images for radial- and
tangential distortion
[im2,~] = undistortImage(im2,cameraParams);
load('Fdata.mat');
im3 = cat(2,im1,im2);
figure;imshow(im3);hold on;
plot(x1(:,1),x1(:,2),'ro');
plot(x2(:,1)+size(im1,2),x2(:,2),'go');
shift = size(im1,2);
cmap = lines(5);
k = 1;
for i = 1:size(x1,1)
    ptdraw = [x1(i,1), x1(i,2);
    x2(i,1)+shift, x2(i,2)];
    plot(ptdraw(:,1),ptdraw(:,2),'LineStyle',...
        '-','LineWidth',1,'Color',cmap(k,:));
    k = mod(k+1,5);
    if k == 0
        k = 1;
    end
end

% here starts your code,
F = EstimateFundamentalMatrix(x1',x2');
% once you have estimated F, check your F using
vgg_gui_F(im1,im2,F');
```

Your task is to implement a MATLAB function, “EstimateFundamentalMatrix.m”, that returns the fundamental matrix relating the two views from point correspondences. If successful, the final visualization will generate epipolar-lines is the second image from corresponding image points in the first view. See an example in the picture below. A point is selected in the left picture and the corresponding epipolar line is automatically calculated in the right figure.


*Figure 1: Illustration of how the fundamental matrix can form epipolar lines in the opposing view.*

## 2 Triangulation with simulated data

In this part of the assignment you should implement a function to perform triangulation based on simulated image point correspondences, known camera poses and the intrinsic parameters of the camera.

You should implement a Linear triangulation function in MATLAB which minimizes the algebraic error (svd-based triangulation) as this algorithm performs better than the midpoint algorithm and is scalable to multiple view triangulation.

In order to reuse your triangulation routine for a future assignment you are required to make it with the following syntax:

```
pest = triangulate_svd([q1 q2], Rs, Cs, K);
```

where q1,q2 are the image correspondence points in homogeneous coordinates. Rs is a 3-dimensional array, where the rotation assoiciated with the first view is Rs(:,:,1) =R1 and for the second view is Rs(:,:,2)=R2. Cs contains the camera perspective centers for both views in world coordinates, i.e. Cs=[C1 C2] and finally K is the intrinsic matrix for the camera.

You can check your implemented algorithm using the code below, which simulates random 3D points and project the points into a stereo image. In case that the triangulation has been correctly implemented the triangulated points should resemble the true points.

```
% Generate random 3D points
N = 100;
p = rand([3,N])* 10 - 5;% from near to far
p(1,:) = p(1,:) + 15;

% Camera position and orientations
d = 1.0;
C1 = [0;-d;0];
C2 = [0;d;0];

rad1= -10*(pi/180);
R1 = [cos(rad1) 0 -sin(rad1);0 1 0;sin(rad1) 0 cos(rad1)]*[0 -1 0;0 0 -1;1 0
0];
R2 = [cos(-rad1) 0 -sin(-rad1);0 1 0;sin(-rad1) 0 cos(-rad1)]*[0 -1 0;0 0 -
1;1 0 0];

t1 = -R1*C1;
t2 = -R2*C2;

% plot points and camera locations
figure,
h1 = plot3(p(1,:),p(2,:),p(3,:),'g*');hold on;
cam1 = plotCamera('Location',C1,'Orientation',R1,'Opacity',0,'Color',[1 0 ...
    0],'Size',0.4,'Label','Camera1');
cam2 = plotCamera('Location',C2,'Orientation',R2,'Opacity',0,'Color',[0 1 ...
    0],'Size',0.4,'Label','Camera2');
```

```
axis equal
xlabel('x:(m)');
ylabel('y:(m)');
zlabel('z:(m)');
title('Triangulation Simulation');
set(gca,'FontName','Arial','FontSize',20);

% Project 3d points into simulated images

K = [1000 0 640;0 1000 480;0 0 1];

[uv1, in1] = proj(R1,t1, p, K);
[uv2, in2] = proj(R2,t2, p, K);

in = in1 & in2;
q1 = uv1(:,in);
ptrue = p(:,in);
q2 = uv2(:,in);

Rs=zeros(3,3,2); % 3D array containing both R1 and R2.
Rs(:,:,1)=R1;
Rs(:,:,2)=R2;
Cs = [C1 C2];     % Array with camera coordinates for both views

pest = zeros(3,size(q1,2));
for i = 1:size(q1,2)
%% here starts your code
    %triangulate points based on image point correspondences
    pest(:,i) = triangulate_svd([q1(:,i) q2(:,i)], Rs, Cs, K);
end

% visualization
figure,
plot3(ptrue(1,:),ptrue(2,:),ptrue(3,:),'ro','MarkerSize',8);hold on;
plot3(pest(1,:),pest(2,:),pest(3,:),'g+','MarkerSize',8);hold on;
xlabel('x:(m)');
ylabel('y:(m)');
zlabel('z:(m)');
title('Triangulation Simulation');
legend('Truth','Reconstruction');
set(gca,'FontName','Arial','FontSize',20);
grid on;
```

If done correctly you should see a figure similar to the one below, which shows the simulated 3D points and the reconstructed (triangulated points).

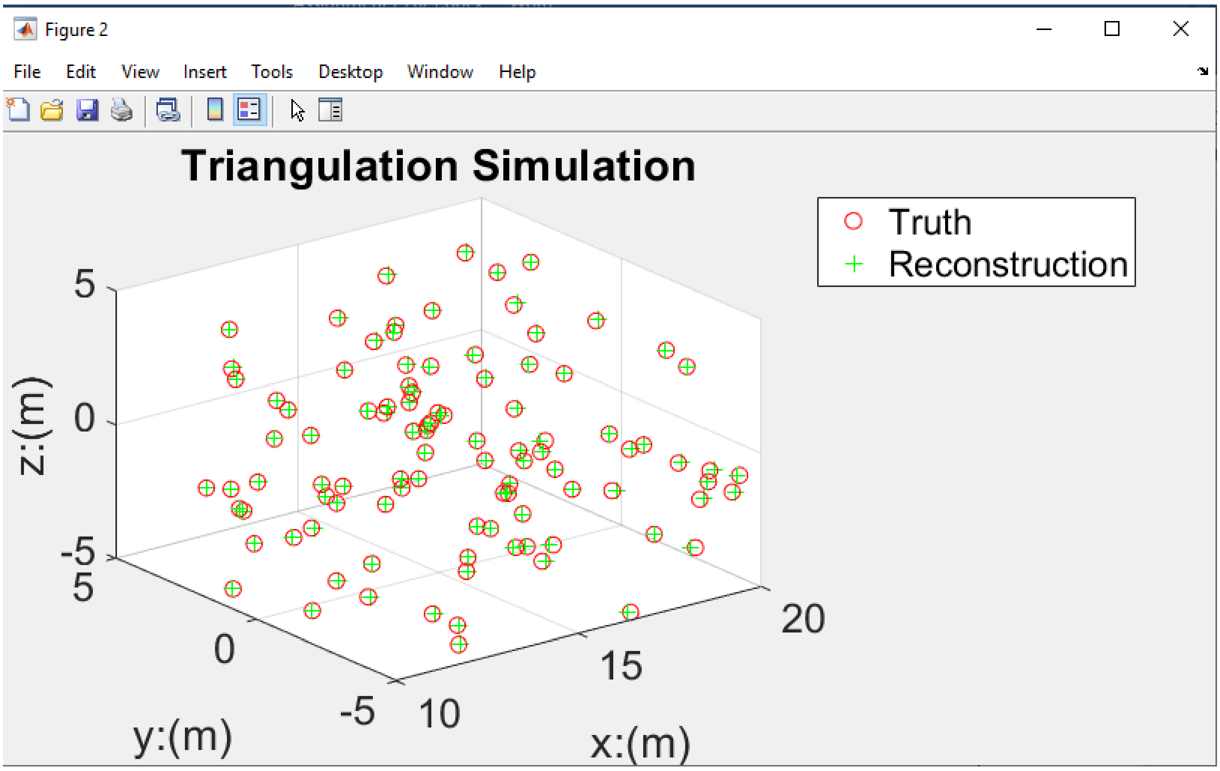


*Figure 2: Triangulation Simulation. The red circles denotes the simulated (true) 3D points and the green points shows the reconstructed points from triangulation.*

In the above code, there is a parameter d, which determines the distance between the two camera views. Try to experiment with a smaller value of d and see what happens to the reconstructed vs true 3D points. Elaborate on your findings.

## 3 Triangulation with real images

In this assignment, you should try out your triangulation method on real image data. We use a dataset of a toy dinosaur taken from a number of view-points. The images for this assignment is in the zipped folder “images” on DTU Learn.

The code provide you with stereo-correspondences from multiple viewpoints. When executing the code the sequence of dinosaur images from different viewpoints will be shown.

In case that your triangulation method from part 2 is working as expected the code will generate a 3D point cloud resembling the shape of the dinosaur.

```
baseDir = './images/';% point to the directory where the dinosaur images are
stored.

buildingScene = imageDatastore(baseDir);
numImages = numel(buildingScene.Files);

load(strcat(baseDir,'viff.xy'));
x = viff(1:end,1:2:72)'; % pull out x coord of all tracks
y = viff(1:end,2:2:72)'; % pull out y coord of all tracks

% visualization
num = 0;
for n = 1:numImages-1
im1 = readimage(buildingScene, n);
imshow(im1); hold on;
id = x(n,:) ~= -1 & y(n,:) ~= -1;
% Data source: "Dinosaur" from www.robots.ox.ac.uk/~vgg/data/data-mview.html.

plot(x(n,id),y(n,id),'go');
num = num + sum(id);
hold off;
pause(0.1);
end

% load projection matrices
load(strcat(baseDir,'dino_Ps.mat'));

ptcloud = zeros(3,num);
k = 1;
for i = 1:size(x,1)-1
    % tracked features
    id = x(i,:) ~= -1 & y(i,:) ~= -1 & x(i+1,:) ~= -1 & y(i+1,:) ~= -1;
    q1 = [x(i,id);y(i,id);];
    q2 = [x(i+1,id);y(i+1,id);];
    P1 = P{i};
    P2 = P{i+1};
    [K, R1, t1, c1] = decomposeP(P1);
    [K, R2, t2, c2] = decomposeP(P2);
    Rs=zeros(3,3,2); % 3D array containing both R1 and R2.
    Rs(:,:,1)=R1;
    Rs(:,:,2)=R2;
    Cs = [c1 c2];     % Array with camera coordinates


    precons = zeros(3,size(q1,2));
    for j = 1:size(q1,2)
    % your code starts
        precons(:,j) = triangulate_svd([[q1(:,j); 1] [q2(:,j); 1]], Rs, Cs,
K);

    end

    ptcloud(:,k:k+size(q1,2)-1) = precons;
    k = k + size(q1,2);
end
```

```
figure
plot3(ptcloud(1,:),ptcloud(2,:),ptcloud(3,:),'k.','MarkerSize',10);
hold on;
grid on;
axis equal;
view(3);
for i = 1:size(x,1)
    P1 = P{i};
    [K, R, t, c] = decomposeP(P1);
    plotCamera('Location',c,'Orientation',R,'Opacity',0,'Color',[0 1
0],'Size',0.05);
end
```

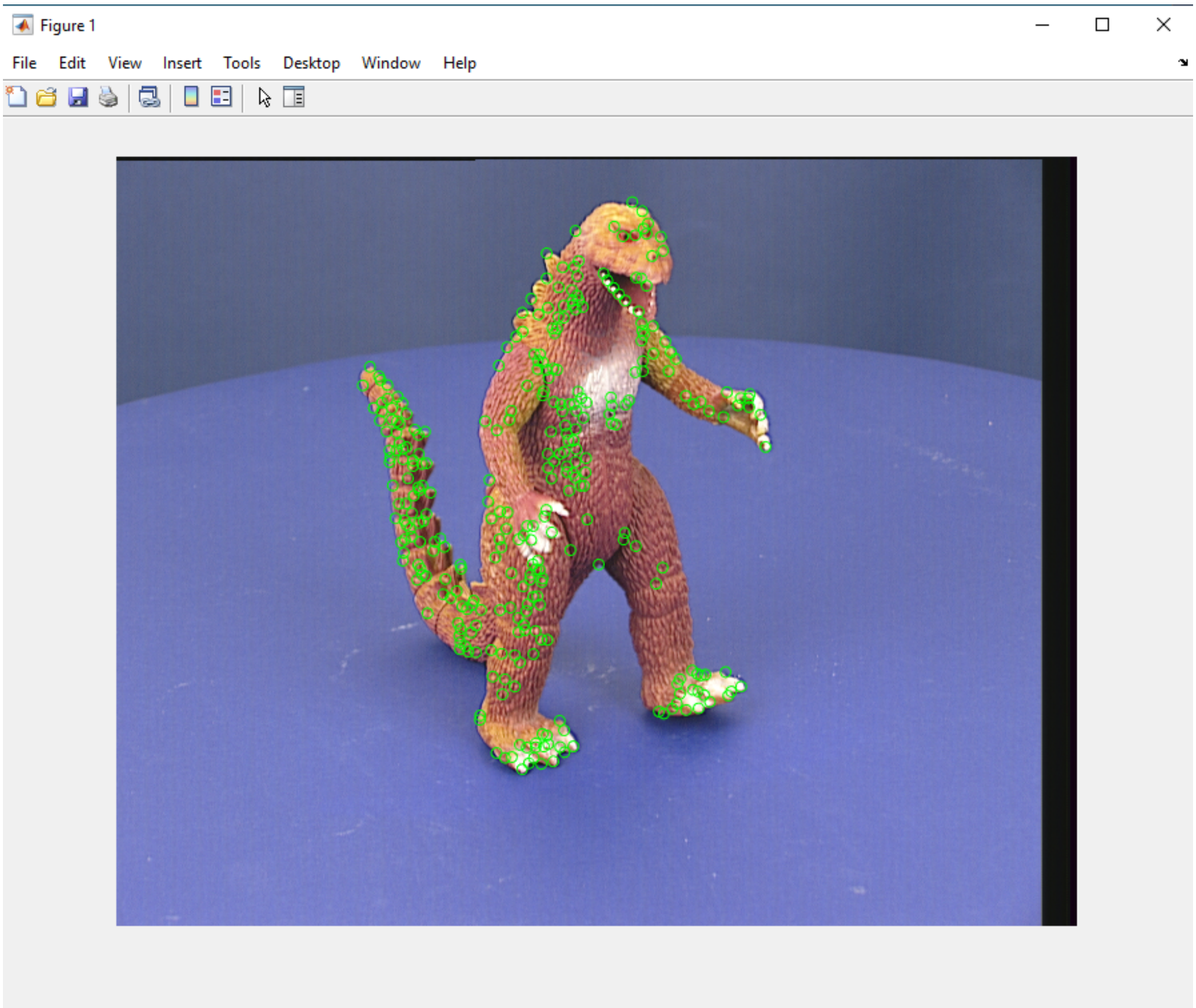


*Figure 3: A picture of the toy dinosaur. The green circles indicates image point matches between previous and current view.*

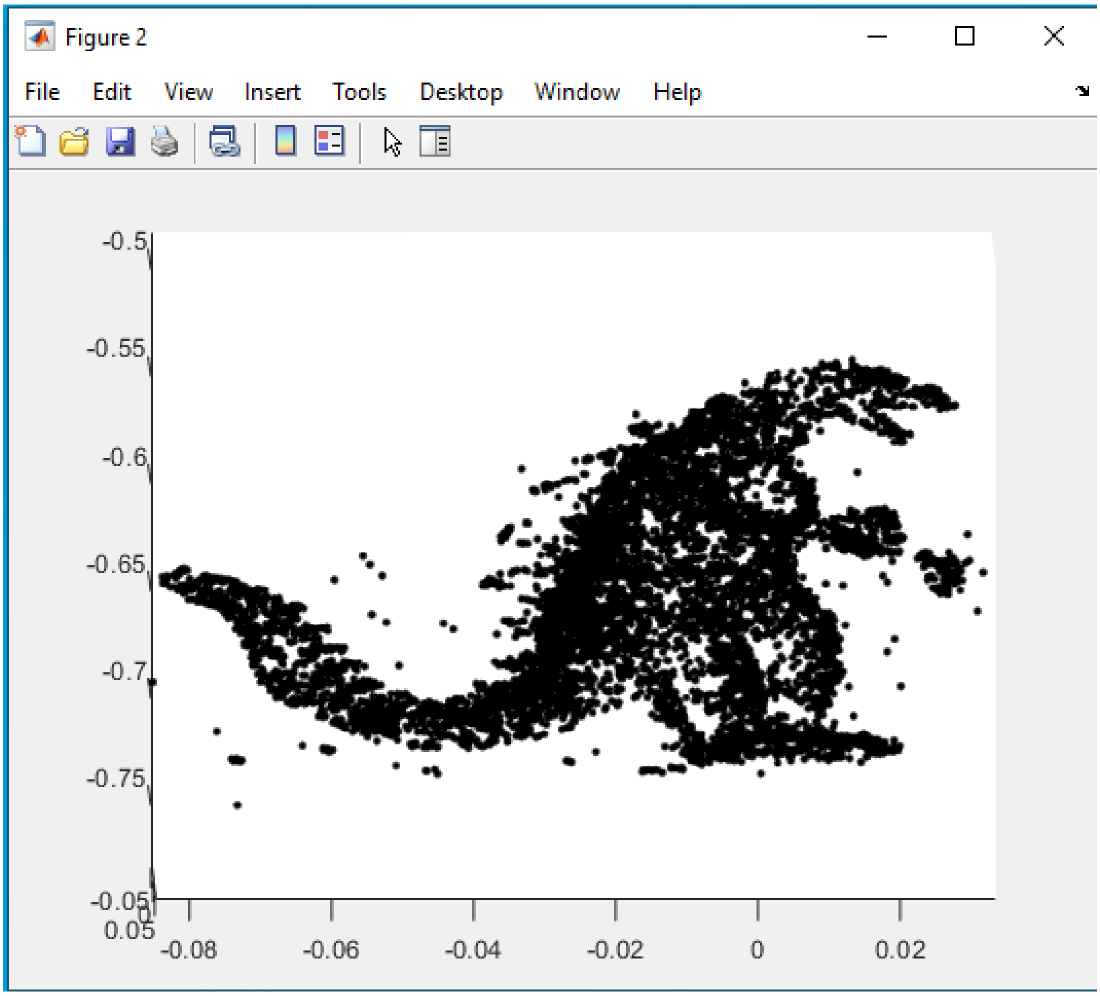


*Figure 4: 3D point cloud of the dinosaur obtained from triangulation at various viewpoints.*