Exercise on Epipolar Geometry & Triangulation

April 11, 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:

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

$$\begin{bmatrix} x_1x_2 & x_1y_2 & x_1 & y_1x_2 & y_1y_2 & y_1 & x_2 & y_2 & 1 \end{bmatrix} \begin{bmatrix} f_{11} \\ f_{21} \\ f_{31} \\ f_{12} \\ f_{22} \\ f_{32} \\ f_{13} \\ f_{23} \\ f_{33} \end{bmatrix} = 0$$

We need at least 8 points to determine $\mathbf{F}$[1]. 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:

```
%% loading
    im1 = rgb2gray(imread('000002.bmp'));
    im2 = rgb2gray(imread('000003.bmp'));
    load('calib.mat');
    [im1,~] = undistortImage(im1,cameraParams);
    [im2,~] = undistortImage(im2,cameraParams);

    load('Fdata.mat');

    im3 = cat(2,im1,im2);
```

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

```
    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 = DLT8pt(x1',x2');

%% once you have estimated F, check your F using
    vgg_gui_F(im1,im2,F');
```

Q2: Once you have estimate **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

```
    N = 100;
    p = rand([3,N]) * 10 - 5;% from near to far
    p(1,:) = p(1,:) + 15;
    C1 = [0;0;0];
    C2 = [0;0.2;0];

    rad1 = -10/57.3;
    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];

```

Figure 1: Check you result visually using epipolar geometry. If the **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).

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

    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);

    K = [500 0 320;0 500 240;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);

    q1(1:2,:) = q1(1:2,:);
    q2(1:2,:) = q2(1:2,:);

    P1 = K*[R1 t1];
    P2 = K*[R2 t2];

    precons = zeros(3,size(q1,2));
    for i = 1:size(q1,2)
     %% here starts your code
        precons(:,i) = triangulation(q1(:,i),P1,q2(:,i),P2);
    end

    %% visualization
    figure
    plot3(ptrue(1,:),ptrue(2,:),ptrue(3,:),'ro','MarkerSize',8);hold on;
    plot3(precons(1,:),precons(2,:),precons(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;
```

Q4: Now you will apply your triangulation code on real images[2]. To save your time, parsing code is provided as follows:

```
    baseDir = './images/';% use you own directory
    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;
```

[2]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};

        precons = zeros(3,size(q1,2));
        for j = 1:size(q1,2)
            % your code starts
            precons(:,j) = xxxxx(q1(:,j),P1,q2(:,j),P2);
        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
```

. 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[3] and the corresponding disparity are provided. The following code is provided for data parsing and visualization.

```
    I1 = imread('im0.png');
    I2 = imread('im1.png');
    figure;imshow(I1);

    load('disp.mat');
    disparityRange = [0 480];
    figure
    imshow(disparityMap,disparityRange)
    title('Disparity Map')
    colormap jet
    colorbar
    % stereo vision intrinsics
    f = 4841.191;
    px = 1441.607;
    py = 969.311;
    % baseline: m
    b = 170.458*1e-3;
    % only triangulate pixel whose disparity is within [100,480].
    id = disparityMap >= 100 & disparityMap <= 480;

    % your code starts
```

[3]Source: "Mask" from 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);
```

References

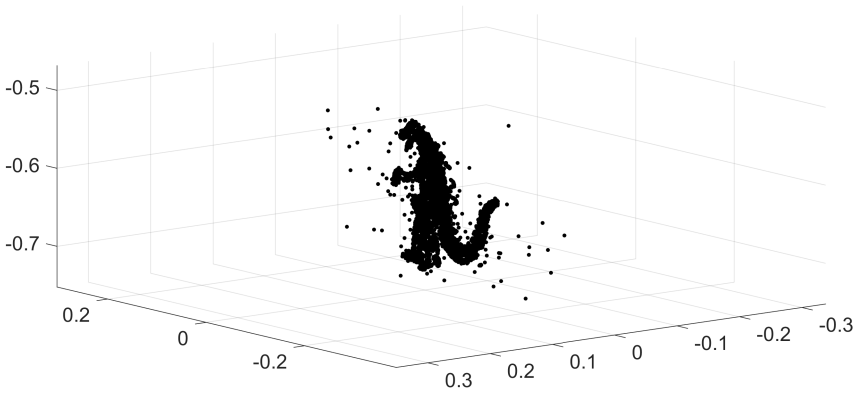

Figure 2: Dinosaur triangulated.

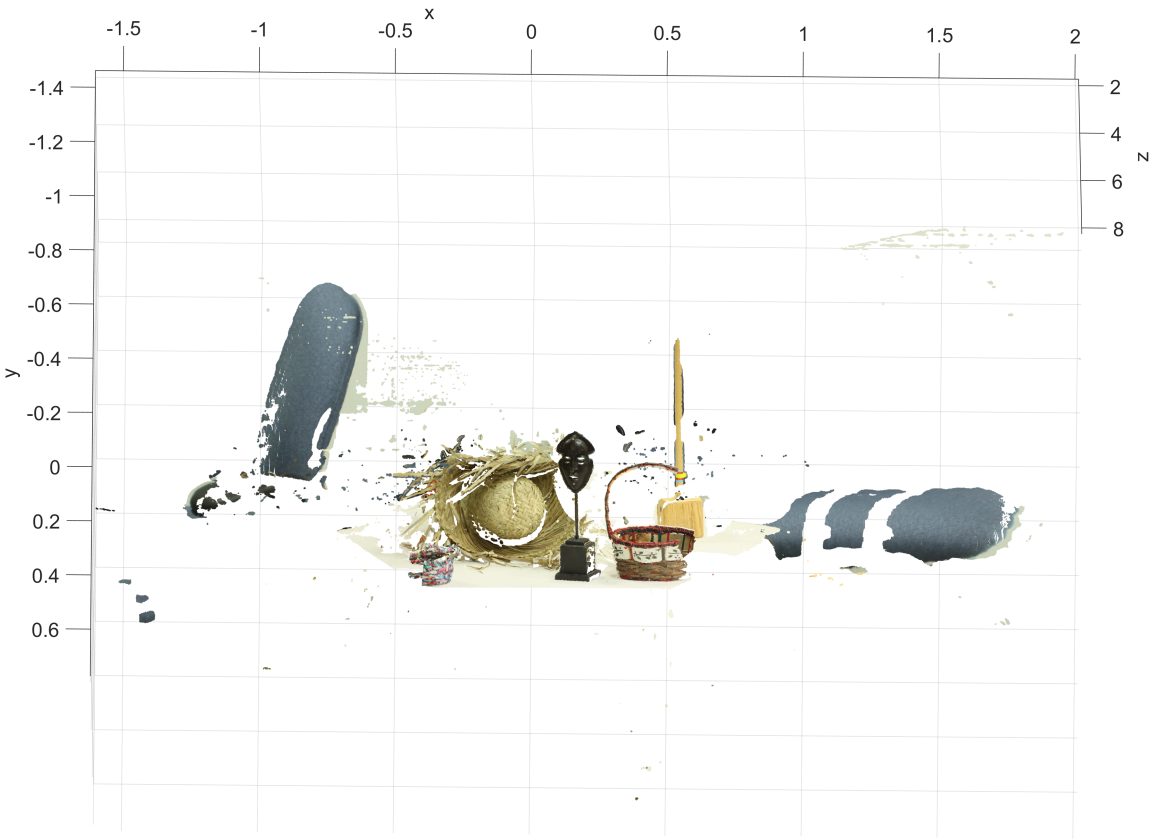

Figure 3: Result obtained by triangulating stereo images.