EE 417 Computer Vision

Berkant Deniz Aktaş

19515

Post-Laboratory Report

December 17, 2018

Sabancı University

Faculty of Engineering and Natural Sciences

Computer Science and Engineering

Introduction

In this lab, we tried to recover the pose of the camera from the given images of a cube which is taken from different viewpoints. In order to achieve desired goal we used 8 point algorithm with computing essential matrix and using epipolar geometry. We assumed the camera is already calibrated which means we know the intrinsic parameters of the camera.

Followed Method

First, we did an transformation of pixel coordinates by multiplying by inverse of the camera matrix K.

```
b1 =  inv(K) * u1;
b2 =  inv(K) * u2;
```

Then we created our X matrix by selecting a's created from point pairs. In order to avoid degenerate pose recover, one should select points from different planes. We will see the wrong estimation with the points on the same plane in the following sections.

```
X =[];
for i=1:1:9
a = transpose( [b1(1,i)*b2(1,i) b1(1,i)*b2(2,i) b1(1,i)*b2(3,i)
b1(2,i)*b2(1,i) b1(2,i)*b2(2,i) b1(2,i)*b2(3,i) b1(3,i)*b2(1,i)
b1(3,i)*b2(2,i) b1(3,i)*b2(3,i)  ] );
X = [X;transpose(a)];
end
```

$$a = [x_1x_2 \quad x_1y_2 \quad x_1z_2 \quad y_1x_2 \quad y_1y_2 \quad y_1z_2 \quad z_1x_2 \quad z_1y_2 \quad z_1z_2]^T$$

$$X = [a_1^T;\ a_2^T;\ \ldots\ a_8^T]$$

Note that above formula only has 8 point pairs and code has 9 point pairs. After computing X matrix, we can use SVD to find E stacked. Note that we need to use svd on Xtranspose*X to achieve correct result.

```
[U S V] = svd(transpose(X)*X);
Estacked = V(:,9);
E = reshape(Estacked, 3, 3);
```

We can succesfully computed an essential matrix but there is a problem about it. According to terminology, the E we computed is ill and we need to cure it by adding an S which has 1,1,0 in its diagonal.

Theorem 1a (Essential Matrix Characterization)
A non-zero matrix E is an essential matrix iff its SVD: $E = U\Sigma V^T$ satisfies: $\Sigma = diag([\sigma_1, \sigma_2, \sigma_3])$ with $\sigma_1 = \sigma_2 \neq 0$ and $\sigma_3 = 0$ and $U, V \in SO(3)$

```
[U S V] = svd(E);
diag_110 = [1 0 0; 0 1 0; 0 0 0];
newE = U*diag_110*transpose(V);
```

After curing E, we can continue to compute our epipoles and following lines with:

```
e1 = null(newE);
e2 = null(transpose(newE));

firstline = transpose(newE)*b2;
secondline = newE*b1;
```

formulas are taken from lecture slides as follows:

$$l_1 \sim E^T \mathbf{x}_2 \qquad l_i^T \mathbf{x}_i = 0 \qquad l_2 \sim E\mathbf{x}_1$$

$$E\mathbf{e}_1 = 0 \qquad l_i^T \mathbf{e}_i = 0 \qquad \mathbf{e}_2 E^T = 0$$

```
firstline10 = transpose(firstline)*b1;
firstline11 = transpose(firstline)*e1;

secondline20 = transpose(secondline)*b2;
secondline21 = transpose(secondline)*e2;
```

`Then we can verify our lines.`

After, all these steps we can recover pose of the camera by following formulas which are stated in the lecture slides.

$$E = U\Sigma V^T$$

$$\begin{aligned}(\widehat{T}_1, R_1) = &\ (UR_Z(+\tfrac{\pi}{2})\Sigma U^T, UR_Z^T(+\tfrac{\pi}{2})V^T)\\ (\widehat{T}_2, R_2) = &\ (UR_Z(-\tfrac{\pi}{2})\Sigma U^T, UR_Z^T(-\tfrac{\pi}{2})V^T)\end{aligned}$$

```
T1skewed = U*Rz*S*U';
R1 = U*Rz'*V';
```

Using above code will give us the rotation estimations and skewed matrix of transformation. Note that we can get the values from skewed matrix by just looking:

```
estT1 = [T1skewed(2,3); T1skewed(3,1); T1skewed(1,2)];
```

Points on Same Plane

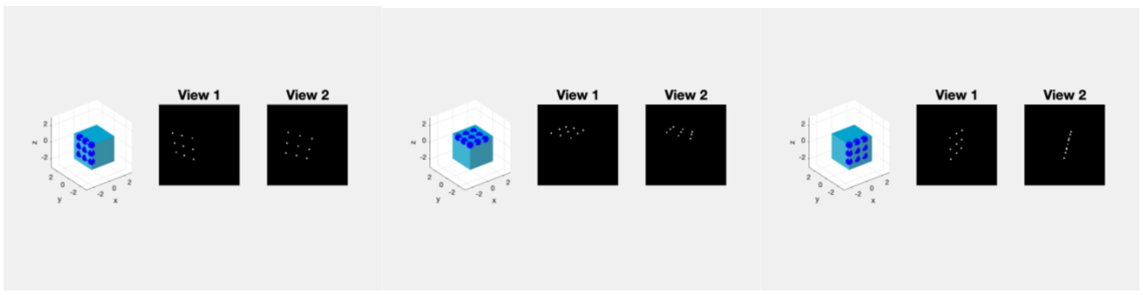
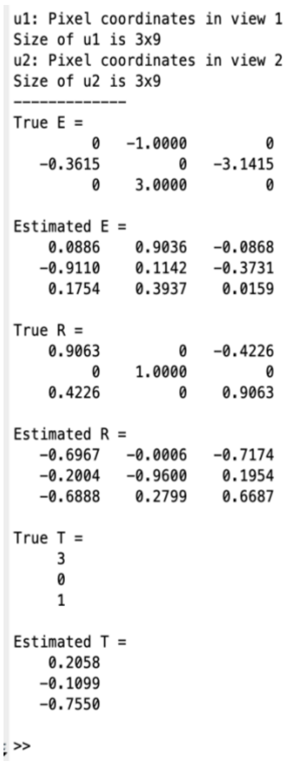
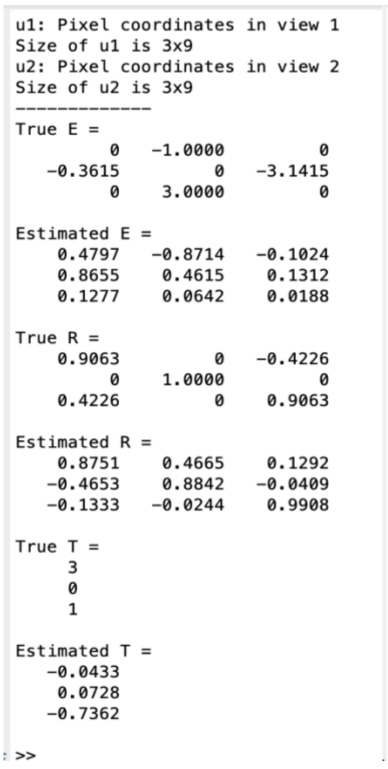


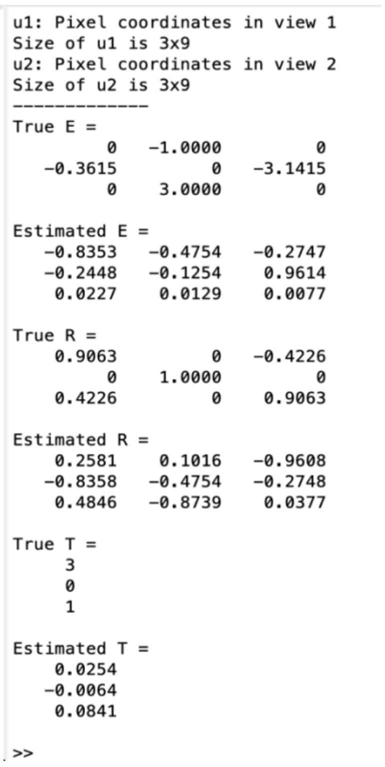
Figure 1, 2, 3. Planes



```
u1: Pixel coordinates in view 1
Size of u1 is 3x9
u2: Pixel coordinates in view 2
Size of u2 is 3x9
-------------
True E =
         0   -1.0000         0
   -0.3615         0   -3.1415
         0    3.0000         0

Estimated E =
    0.0886    0.9036   -0.0868
   -0.9110    0.1142   -0.3731
    0.1754    0.3937    0.0159

True R =
    0.9063         0   -0.4226
         0    1.0000         0
    0.4226         0    0.9063

Estimated R =
   -0.6967   -0.0006   -0.7174
   -0.2004   -0.9600    0.1954
   -0.6888    0.2799    0.6687

True T =
     3
     0
     1

Estimated T =
    0.2058
   -0.1099
   -0.7550

>>
```



```
u1: Pixel coordinates in view 1
Size of u1 is 3x9
u2: Pixel coordinates in view 2
Size of u2 is 3x9
-------------
True E =
         0   -1.0000         0
   -0.3615         0   -3.1415
         0    3.0000         0

Estimated E =
    0.4797   -0.8714   -0.1024
    0.8655    0.4615    0.1312
    0.1277    0.0642    0.0188

True R =
    0.9063         0   -0.4226
         0    1.0000         0
    0.4226         0    0.9063

Estimated R =
    0.8751    0.4665    0.1292
   -0.4653    0.8842   -0.0409
   -0.1333   -0.0244    0.9908

True T =
     3
     0
     1

Estimated T =
   -0.0433
    0.0728
   -0.7362

>>
```



```
u1: Pixel coordinates in view 1
Size of u1 is 3x9
u2: Pixel coordinates in view 2
Size of u2 is 3x9
-------------
True E =
         0   -1.0000         0
   -0.3615         0   -3.1415
         0    3.0000         0

Estimated E =
   -0.8353   -0.4754   -0.2747
   -0.2448   -0.1254    0.9614
    0.0227    0.0129    0.0077

True R =
    0.9063         0   -0.4226
         0    1.0000         0
    0.4226         0    0.9063

Estimated R =
    0.2581    0.1016   -0.9608
   -0.8358   -0.4754   -0.2748
    0.4846   -0.8739    0.0377

True T =
     3
     0
     1

Estimated T =
    0.0254
   -0.0064
    0.0841

>>
```

Figure 3, 4, 5 Estimated values

After following above steps, we can see that we can recover pose of the camera but it strongly depends on the number of the points and the orientation of the points. Algorithm's name concludes that we need at least 8 points. It is because of the DoF of Essential Matrix but, if we choose the points on the same plane our results are not very close to real results as its shown on the Figure 3 4 and 5. I believe that it is because we are losing one dimension variable when we choose the points from the same plane. In order to achieve better results we can usee more points and check the estimation accuracy.

All planes & Discussion

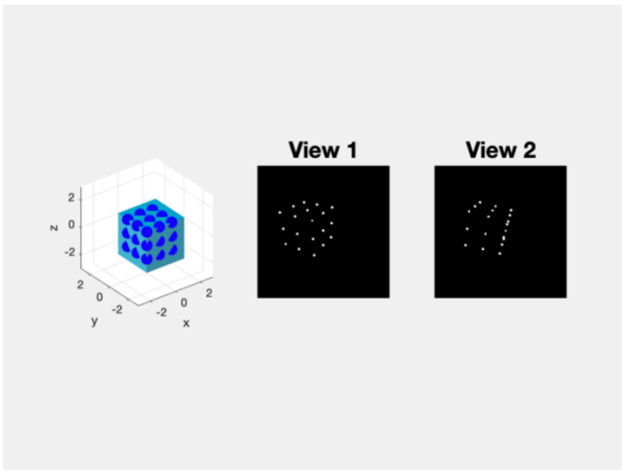
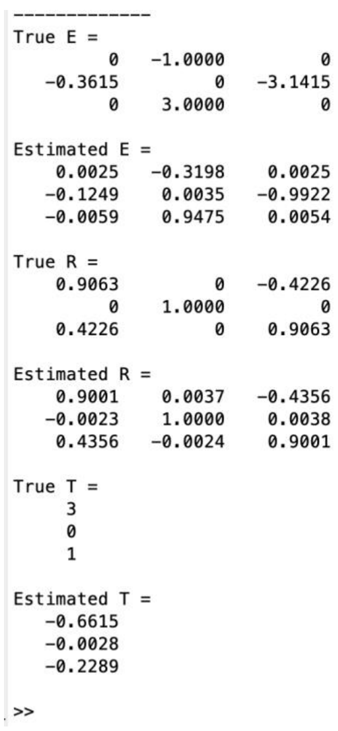


Figure 6. all planes



```
-------------
True E =
         0   -1.0000         0
   -0.3615         0   -3.1415
         0    3.0000         0

Estimated E =
    0.0025   -0.3198    0.0025
   -0.1249    0.0035   -0.9922
   -0.0059    0.9475    0.0054

True R =
    0.9063         0   -0.4226
         0    1.0000         0
    0.4226         0    0.9063

Estimated R =
    0.9001    0.0037   -0.4356
   -0.0023    1.0000    0.0038
    0.4356   -0.0024    0.9001

True T =
     3
     0
     1

Estimated T =
   -0.6615
   -0.0028
   -0.2289

>>
```

Figure 7. Estimation with all planes

As its shown in the Figure 7 we estimated rotation with high accuracy but due to nature of the problem we can only estimate the transformation up to a scale. We can see that true values are equal to three times of the estimated values. Also we can estimate essential matrix up to a scale and these scales are matching with eachother. Thus, we can say that we need more than one plane for estimation and atleast 8 points fort his estimation. If we increase the number of the points the results will be closer to true values. But interesting points is, during the lab I used 8 points from different planes and the estimation was not that robust so I used all 19 points to achieve a correct result. Furtherwork should be related with computing without calibrating camera (don't know camera instrinsic parameters) with using Fundamental Matrix as its stated in the lectures. To sum up, I would choose the pose recovered from using the points from three planes.

Code

```
clear all; close all; clc;
%% Definitions
rng(1);
L = 300;
I1 = zeros(L,L);

f=L;
u0 = L/2;
v0 = L/2;

K = [f  0  u0;
     0  f  v0;
     0  0  1];

DEG_TO_RAD = pi/180;

% select 8 of them
%% World Coordinates
P_W=[
     0    2    0    1;
     0    1    0    1;
     0    0    0    1;
%      0  2   -1    1;
%      0  1   -1    1;
%      0  0   -1    1;
%      0  2   -2    1;
%      0  1   -2    1;
%      0  0   -2    1;
     1    0    0    1;
     2  0    0    1;
%     1 0    -1   1;
%     2 0    -1   1;
%     1 0    -2   1;
%     2 0    -2   1;
    1    1    0    1;
    2    1    0    1;
    1    2    0    1;Z%
    2    2    0    1
];

% P_W=[0     2    0    1;
%
%     0 0    0    1;
%
%     0 1    -1   1;
%
%     0 2    -2   1;
%
%     0 0    -2   1;
%
%     2 0    0    1;
%
%     2 0    -1   1;
%
%
%
%     2 2    0    1
%];


P_W = P_W';
NPTS = size(P_W,2); %Number of points

%% Visualization
figure;
subplot(1,3,1)
wally = meshgrid(0:0.1:3);
wallz = meshgrid(-3:0.1:0);
wallx = 0*ones(size(wallz,1));
surf(wallx, wally, wallz','FaceColor',(1/255)*[97 178
205],'EdgeColor','none')
hold on
wallx = meshgrid(0:0.1:3);
wallz = meshgrid(-3:0.1:0);
wally = 0*ones(size(wallz,1));
surf(wallx, wally, wallz','FaceColor',(1/255)*[77 137
157],'EdgeColor','none')
wallx = meshgrid(0:0.1:3);
wally = meshgrid(0:0.1:3);
wallz = zeros(size(wally,1)); % Generate z data
surf(wallx, wally', wallz,'FaceColor',(1/255)*[45 162
200],'EdgeColor','none')
plot3(P_W(1,:),P_W(2,:),P_W(3,:),'b.','MarkerSize',36);
axis equal;
grid on
axis vis3d;
```

```
axis([-3 3 -3 3 -3 3])
grid on
xlabel('x')
ylabel('y')
zlabel('z')

%% Camera Transformation for View 1
ax = 120 * DEG_TO_RAD;
ay = 0 *DEG_TO_RAD;
az = 60 * DEG_TO_RAD;

Rx = [1 0 0;
      0 cos(ax) -sin(ax);
      0 sin(ax) cos(ax)];
Ry = [cos(ay)  0  sin(ay);
           0   1     0;
      -sin(ay) 0  cos(ay)];
Rz = [cos(az) -sin(az) 0;
      sin(az) cos(az)  0;
      0          0     1];

Rc1 = Rx*Ry*Rz;
Tc1 = [0;0;5];
M = [Rc1 Tc1];

p1 = K*(M * P_W);
noise1 = 4*rand(3,NPTS)-2;
noise1(3,:)=1;
p1 = p1 + noise1;

u1(1,:) = p1(1,:) ./ p1(3,:);
u1(2,:) = p1(2,:) ./ p1(3,:);
u1(3,:) = p1(3,:) ./ p1(3,:);

for i=1:length(u1)
    x = round(u1(1,i)); y=round(u1(2,i));
    I1(y-2:y+2, x-2:x+2) = 255;
end

subplot(1,3,2), imshow(I1, []), title('View 1', 'FontSize',20);

%% Camera Transformation for View 2
ax = 0 * DEG_TO_RAD;
ay = -25 *DEG_TO_RAD;
az = 0 * DEG_TO_RAD;

Rx = [1 0 0;
      0 cos(ax) -sin(ax);
      0 sin(ax) cos(ax)];
Ry = [cos(ay)  0  sin(ay);
           0   1     0;
      -sin(ay) 0  cos(ay)];
Rz = [cos(az) -sin(az) 0;
      sin(az) cos(az)  0;
      0          0     1];
```

```
Rc2c1 = Rx*Ry*Rz;
TrueR = Rc2c1;
Tc2c1 = [3;0;1];
TrueT = Tc2c1;
Hc1 = [Rc1 Tc1; 0 0 0 1];
Hc2c1 = [Rc2c1 Tc2c1; 0 0 0 1];
Hc2 = Hc2c1*Hc1;

Rc2 = Hc2(1:3,1:3);
Tc2 = Hc2(1:3,4);

M = [Rc2 Tc2];

I2 = zeros(L,L);
p2 = K*(M*P_W);

noise2 = 4*rand(3,NPTS)-2;
noise2(3,:)=1;
p2 = p2 + noise2;

u2(1,:) = p2(1,:) ./ p2(3,:);
u2(2,:) = p2(2,:) ./ p2(3,:);
u2(3,:) = p2(3,:) ./ p2(3,:);

for i=1:length(u2)
    x = round(u2(1,i)); y=round(u2(2,i));
    I2(y-2:y+2, x-2:x+2) = 255;
end

subplot(1,3,3), imshow(I2, []), title('View 2', 'FontSize',20);

t = Tc2c1;
T_skew = [0 -t(3) t(2); t(3) 0 -t(1); -t(2) t(1) 0];
Etrue = T_skew*Rc2c1;

%% Displaying the information
disp('u1: Pixel coordinates in view 1')
u1info = ['Size of u1 is ' num2str(size(u1,1)) 'x' num2str(size(u1,2))];
disp(u1info)
disp('u2: Pixel coordinates in view 2')
u2info = ['Size of u2 is ' num2str(size(u2,1)) 'x' num2str(size(u2,2))];
disp(u2info)
disp('-------------')
%% Lab#8 Assignment starts here.
%% Transform pixel coordinates and construct X matrix using Equations 1 and
2

 %ut = u1(:,1).*inv(K);

 b1 =  inv(K) * u1;
 b2 =  inv(K) * u2;
 X =[];
 for i=1:1:9
 a = transpose( [b1(1,i)*b2(1,i) b1(1,i)*b2(2,i) b1(1,i)*b2(3,i)
b1(2,i)*b2(1,i) b1(2,i)*b2(2,i) b1(2,i)*b2(3,i) b1(3,i)*b2(1,i)
b1(3,i)*b2(2,i) b1(3,i)*b2(3,i)  ] );
```

```
 X = [X;transpose(a)];
 end
%% Estimate E, cure it and check for Essential Matrix Characterization
[U S V] = svd(transpose(X)*X);
%[min_val, min_index] = min(diag(S(1:9,1:9)));
%Estacked = V(1:9, min_index);

% pick the eigenvector corresponding to the smallest eigenvalue
Estacked = V(:,9);
%e = (round(1.0e+10*e))*(1.0e-10);
% essential matrix
%E = reshape(e, 3, 3);

E = reshape(Estacked, 3, 3);
%E = [Estacked(1,1) Estacked(4,1) Estacked(7,1); Estacked(2,1)
Estacked(5,1) Estacked(8,1); Estacked(3,1) Estacked(6,1) Estacked(9,1) ];
%E = [Estacked(1,1) Estacked(2,1) Estacked(3,1); Estacked(4,1)
Estacked(5,1) Estacked(6,1); Estacked(7,1) Estacked(8,1) Estacked(9,1) ];
%res = null(X);
%res2 = X*res;
%
[U S V] = svd(E);
diag_110 = [1 0 0; 0 1 0; 0 0 0];
newE = U*diag_110*transpose(V);



%[U,S,V] = svd(E); %Perform second decompose to get S=diag(1,1,0)


% eigens = eig(transpose(X)*X);
% mineigen = min(eigens);

%% Find epipoles and epipolar lines
e1 = null(newE);
e2 = null(transpose(newE));

firstline = transpose(newE)*b2;
secondline = newE*b1;

%% Verify epipoles and epipolar lines

firstline10 = transpose(firstline)*b1;
firstline11 = transpose(firstline)*e1;

secondline20 = transpose(secondline)*b2;
secondline21 = transpose(secondline)*e2;


%% Recover the rotation and the translation

az = 90 * DEG_TO_RAD;

Rz = [cos(az) -sin(az) 0;
      sin(az) cos(az)  0;
      0          0     1];

T1skewed = U*Rz*S*U';
R1 = U*Rz'*V';

az = 270 * DEG_TO_RAD;
Rz = [cos(az) -sin(az) 0;
      sin(az) cos(az)  0;
      0          0     1];
T2skewed = U*Rz*S*U';
R2 = U*Rz'*V';

estT1 = [T1skewed(2,3); T1skewed(3,1); T1skewed(1,2)];

estT2 = [T2skewed(2,3); T2skewed(3,1); T2skewed(1,2)];

%% Compare your results with ground truth
disp('True E =')
disp(Etrue)
disp('Estimated E = ')
% disp(your estimated E variable)
disp(newE)

disp('True R =')
disp(TrueR)
disp('Estimated R = ')
% disp(your estimated R variable)
disp(R1)

disp('True T =')
disp(TrueT)
disp('Estimated T = ')
% disp(your estimated T variable)
disp(estT1)
```