POSE RECOVERY OF A CALIBRATED CAMERA THROUGH ESSENTIAL MATRIX USING 8-POINT ALGORITHM

DECEMBER 21, 2021

TA's: Muhammed Zemzemoğlu and Mehmet Emin Mumcuoğlu

Name of Student: Moses Chuka Ebere
Student Number: 31491
Course: Computer Vision (EE 417) Lab

TABLE OF CONTENTS

INTRODUCTION

The notion of two-view geometry is guided by the fact that:
Given two views of a scene, is it possible to recover the unknown camera displacement and reconstruct the 3D scene?
The two views here could be obtained from a stereo system or a monocular moving camera.

In this lab, the focus is on applying the 8-point algorithm in tandem with the essential matrix to recover the pose of a calibrated camera.

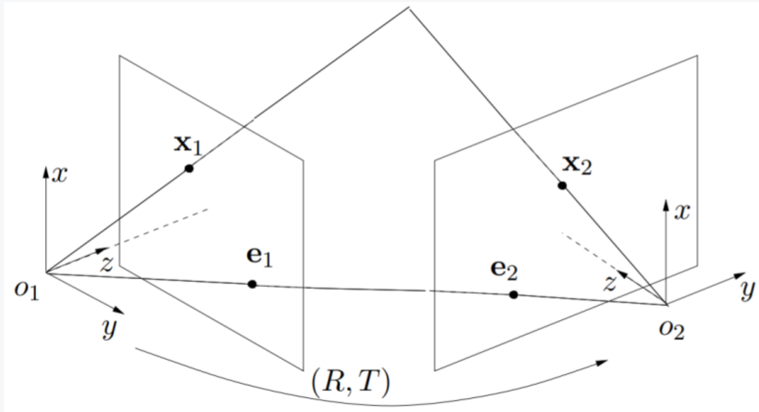
The general concept for this is as follows:

- Images of corresponding points (in the two views) are related by the epipolar constraint, which involves the unknown relative pose between the cameras. Therefore, given a number of corresponding points, we could use the epipolar constraints to try to recover camera pose. A simple closed-form solution to this problem consists of two steps: First a matrix E is recovered from a number of epipolar constraints, then relative translation and orientation is extracted from E. However, since the matrix E recovered using correspondence data in the epipolar constraint may not be an essential matrix it needs to be projected into the space of essential matrices prior to applying the formula of equation.

EXPLANATION OF METHODS

8-POINT ALGORITHM

Given the two views:



Corresponding points can be related by the following equation such that there's rigid body transformation between the points:

$$\lambda_2 \mathbf{x}_2 = R\lambda_1 \mathbf{x}_1 + T.$$

Where x_i's are normalized image coordinates λ_i's are the depths. s λ2x2 = Rλ1x1 + T. In order to eliminate the depths λ_i's in the preceding equation, we multiply both sides by a skew-symmetric matrix obtained from T. After further algebraic elimination, we obtain the epipolar constraint.

$$\boxed{\mathbf{x}_2^T \widehat{T} R \mathbf{x}_1 = 0.}$$

Where $E \doteq \widehat{T}R \in \mathbb{R}^{3\times 3}$ is the essential matrix that captures the orientation between the two cameras (or successive camera centers in the case of a monocular camera).

$$E = \begin{bmatrix} e_1 & e_2 & e_3 \\ e_4 & e_5 & e_6 \\ e_7 & e_8 & e_9 \end{bmatrix}$$

To properly estimate this essential matrix, the 8-point algorithm is used.

- First, we come up with normalized coordinates of corresponding points.
- We take the Kronecker product of each pair to obtain a vector of the form:

$$\mathbf{a} = [x_2x_1, x_2y_1, x_2z_1, y_2x_1, y_2y_1, y_2z_1, z_2x_1, z_2y_1, z_2z_1]^T \in \mathbb{R}^9.$$

- This is repeated for all 8 pairs and the results are stacked to obtain an 8x9 matrix called X.

$$X = [a_1^T;\ a_2^T;\ \dots\ a_8^T] \qquad XE^s = 0$$

- This leaves us with a homogenous equation of the form on the right.
- E^s in the above equation is simply a stacked version of the essential matrix, E.
- Now, we find the eigenvector corresponding to the minimum eigenvalue of X^TX. This obtained by taking the last column of the right orthogonal matrix after performing a singular value decomposition on X^TX. We obtain a matrix, F here.

$$F = U diag\{\lambda_1, \lambda_2, \lambda_3\} V^T$$

- At this point, there is no guarantee that the F-matrix is actually the desired essential matrix. So, we project it onto the essential manifold and enforce the rank constraint. In theory, we are trying to minimize the Frobenius norm between F and the essential matrix, E.

$$\|E - F\|_f^2 \ \textit{is given by}\ E = U diag\{\sigma, \sigma, 0\} V^T$$

- In our case, we enforce the rank constraint by making the first two singular values equal to 1 and the third one equal to 0.
- We can now confirm that E is an essential matrix by checking whether U and V belong to the special orthogonal group of 3x3 matrix (i.e., they must have a determinant of +1).

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

- Using the estimated essential matrix, we can now obtain the epipoles and epipolar lines of any set of corresponding points by the following relations:

$$\begin{array}{lll} l_1 \sim E^T \mathbf{x}_2 & l_i^T \mathbf{x}_i = 0 & l_2 \sim E\mathbf{x}_1 \\ E\mathbf{e}_1 = 0 & l_i^T \mathbf{e}_i = 0 & \mathbf{e}_2 E^T = 0 \end{array}$$

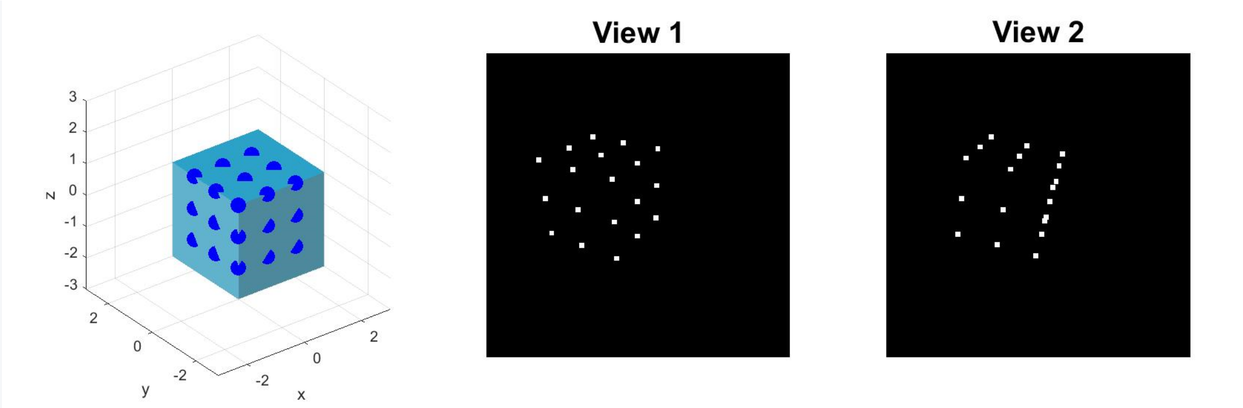
- Finally, two solutions (two for +E; two for -E) can be obtained for the rotation and translation:

$$R = UR_Z^T\left(\pm\frac{\pi}{2}\right)V^T, \quad \widehat{T} = UR_Z\left(\pm\frac{\pi}{2}\right)\Sigma U^T.$$

- From the above, the translation vector could be obtained from the components of the translation matrix:

T = [t(3,2) t(1,3) t(2,1)];

IN-LAB RESULTS




```
u1: Pixel coordinates in view 1
Size of u1 is 3x19
u2: Pixel coordinates in view 2
Size of u2 is 3x19
-------------
True E =
         0   -1.0000         0
   -0.3615         0   -3.1415
         0    3.0000         0

Estimated E =
    0.0040   -0.1936   -0.0221
   -0.2107    0.0197    0.9770
    0.0091    0.9809   -0.0231
```

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

Estimated R1 & R2 :
R1_est =
    0.9987    0.0472   -0.0182
    0.0472   -0.9989   -0.0005
   -0.0182   -0.0003   -0.9998

-------------
R2_est =
    0.9184   -0.0090   -0.3955
    0.0052    0.9999   -0.0108
    0.3956    0.0078    0.9184
```

```
True T =
     3
     0
     1

Estimated T1 & T2 :
T1_est =
   -0.9808
   -0.0268
   -0.1931

-------------
T2_est =
    0.9808
    0.0268
    0.1931
```

T2 and R2 are much closer to the true values; hence we chose them as our solution.

verify_epipole1	-1.3878e-17
verify_epipole2	4.3368e-18
verify_point1	-0.3280
verify_point2	-0.3280

The epipoles and points are confirmed to lie on the epipolar line.

SO3_U_	1.0000
SO3_V_	1.0000

The left and right matrices after performing SVD on E are found to belong to SO3.

MULTIPLE RESULTS AND POST-LAB QUESTIONS

X-Y Plane

```
u1: Pixel coordinates in view 1
Size of u1 is 3x19
u2: Pixel coordinates in view 2
Size of u2 is 3x19
-------------
True E =
         0   -1.0000         0
   -0.3615         0   -3.1415
         0    3.0000         0

Estimated E =
   -0.8707   -0.0345    0.3395
    0.0723   -0.9522   -0.1672
   -0.3222   -0.2723    0.0878
```

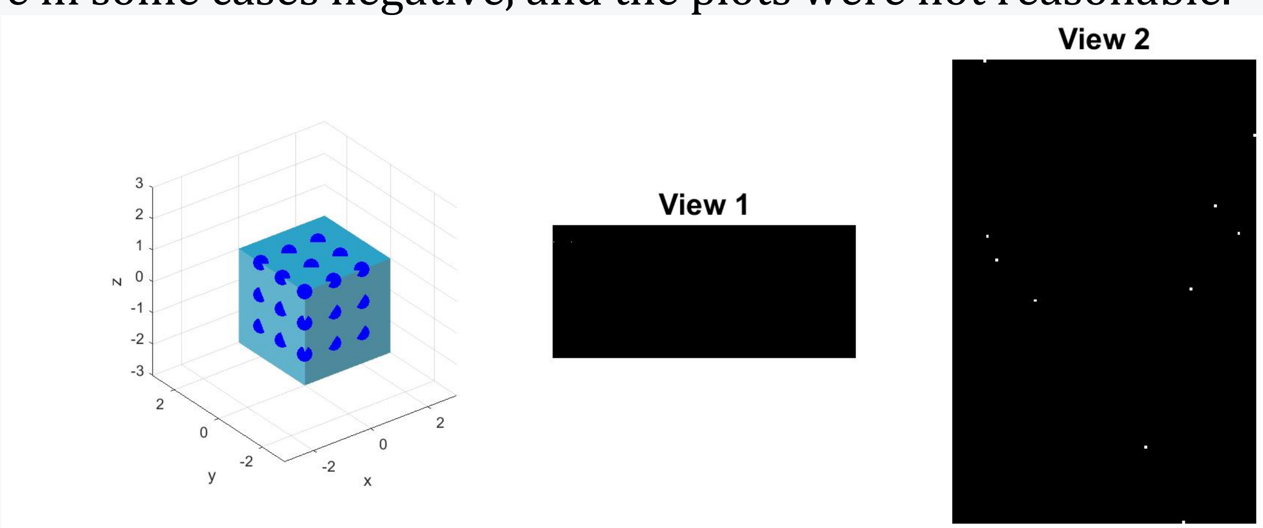
```
True R =
    0.9063         0   -0.4226
         0    1.0000         0
    0.4226         0    0.9063

Estimated R1 & R2 :
R1_est =
    0.9081    0.4179    0.0282
   -0.4066    0.8957   -0.1799
   -0.1004    0.1519    0.9833

-------------
R2_est =
   -0.6866   -0.2545   -0.6811
    0.5599   -0.7826   -0.2720
   -0.4638   -0.5681    0.6798
```

```
True T =
     3
     0
     1

Estimated T1 & T2 :
T1_est =
   -0.3543
   -0.2452
    0.9024

-------------
T2_est =
    0.3543
    0.2452
   -0.9024
```

For the images, I tried to modify the code used to plot the true position and orientation to obtain the new pose by creating the projection matrix (M) from the multiplication of the intrinsic parameter matrix, K, by the concatenation of the estimated rotation matrix and translation vector. After this, I multiplied the results by the world points, P_W, on the right. *The full code is in the appendix*.

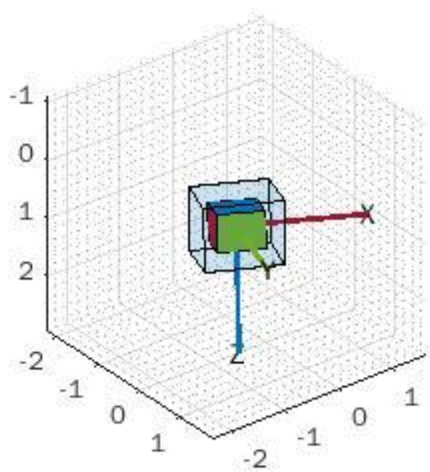
```
M2_ = [R2 T2];
p2_ = K*(M2_*P_W);
```

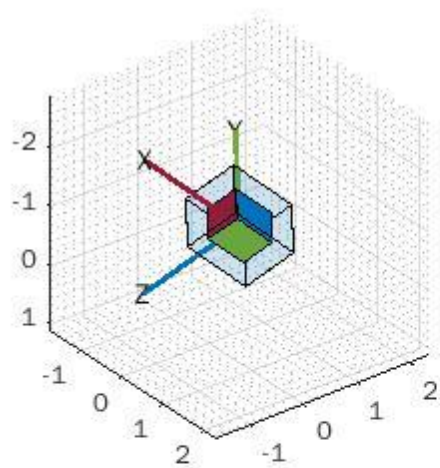
However, I ran into some issues because my final points for the plots were in some cases negative, and the plots were not reasonable.



To solve the above issue, I simply used the **poseplot function available in MATLAB 2021b version to obtain the follow results**:
Code: poseplot(R1, T1):

(R1, T1) (R2, T2)





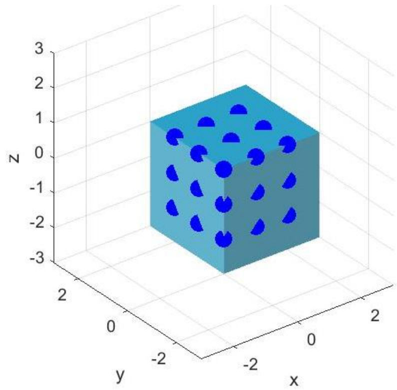
X-Z Plane

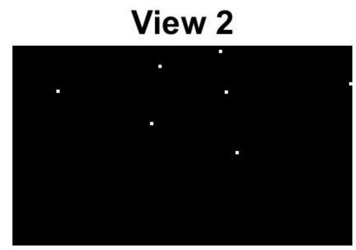
```
u1: Pixel coordinates in view 1
Size of u1 is 3x19
u2: Pixel coordinates in view 2
Size of u2 is 3x19
--------------
True E =
         0   -1.0000         0
   -0.3615         0   -3.1415
         0    3.0000         0

Estimated E =
    0.2279   -0.1099   -0.9539
    0.9725    0.0734    0.2164
   -0.0061   -0.0213   -0.1657
```

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

Estimated R1 & R2 :
R1_est =
   -0.0702    0.2114   -0.9749
    0.4015   -0.8887   -0.2216
   -0.9132   -0.4069   -0.0225

-------------
R2_est =
    0.3513   -0.0583    0.9344
   -0.4782    0.8469    0.2326
   -0.8049   -0.5285    0.2697
```

```
True T =
     3
     0
     1

Estimated T1 & T2 :
T1_est =
    0.1613
   -0.0440
   -0.9859

-------------
T2_est =
   -0.1613
    0.0440
    0.9859
```

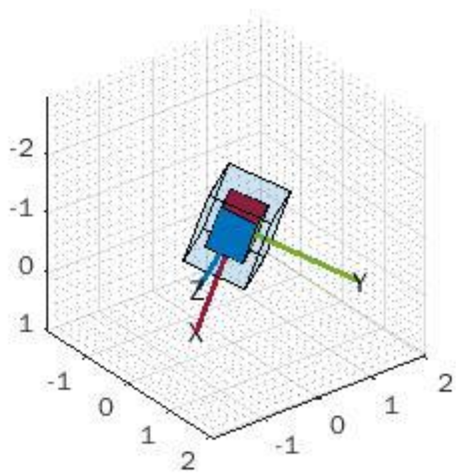
The same explanation from the before is also applicable here.



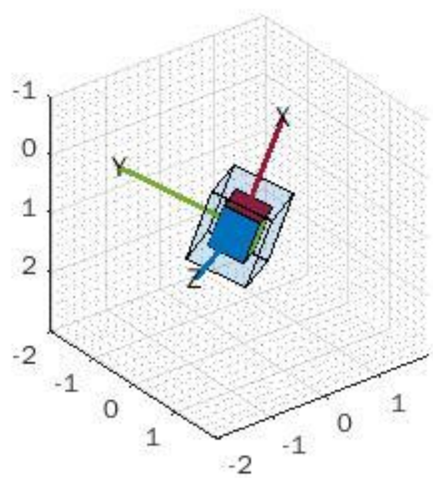


Using the poseplot function,

(R1, T1)



(R2, T2)

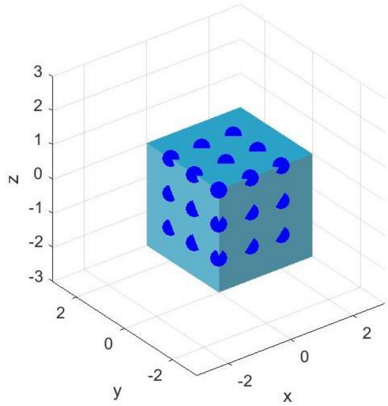


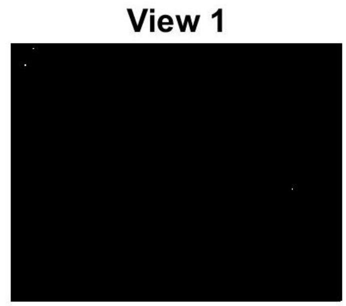
Y-Z Plane

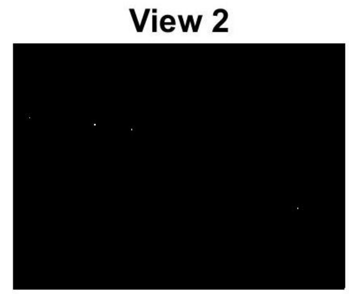
```
u1: Pixel coordinates in view 1
Size of u1 is 3x19
u2: Pixel coordinates in view 2
Size of u2 is 3x19
--------------
True E =
         0   -1.0000         0
   -0.3615         0   -3.1415
         0    3.0000         0

Estimated E =
    0.6755   -0.6969    0.1403
    0.6905    0.6902    0.2043
    0.0845   -0.1901    0.0131
```

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

Estimated R1 & R2 :
R1_est =
    0.9909   -0.1244    0.0507
    0.1228    0.9919    0.0334
   -0.0544   -0.0269    0.9982

-------------
R2_est =
   -0.8977    0.0973   -0.4298
   -0.1570   -0.9819    0.1057
   -0.4117    0.1624    0.8967
```

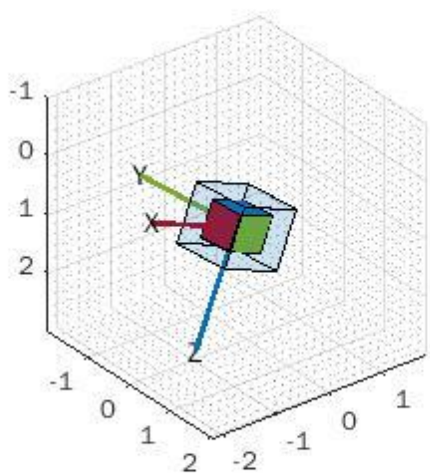
```
True T =
     3
     0
     1

Estimated T1 & T2 :
T1_est =
   -0.1957
    0.0718
    0.9780

-------------
T2_est =
    0.1957
   -0.0718
   -0.9780
```



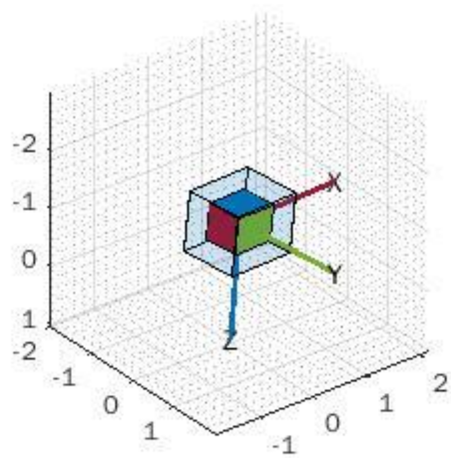




(R1, T1)



(R2, T2)

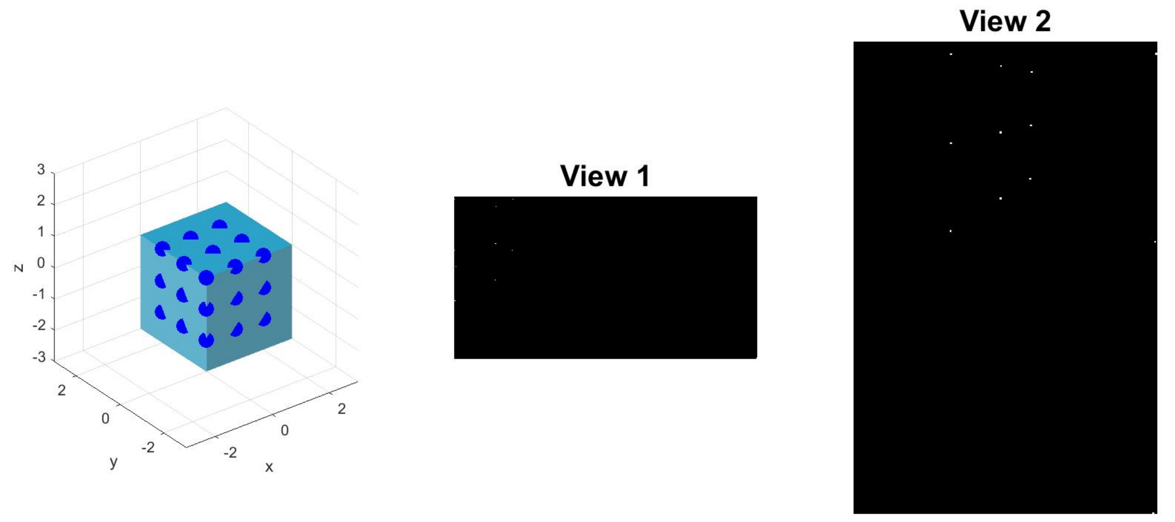


Using all the Points

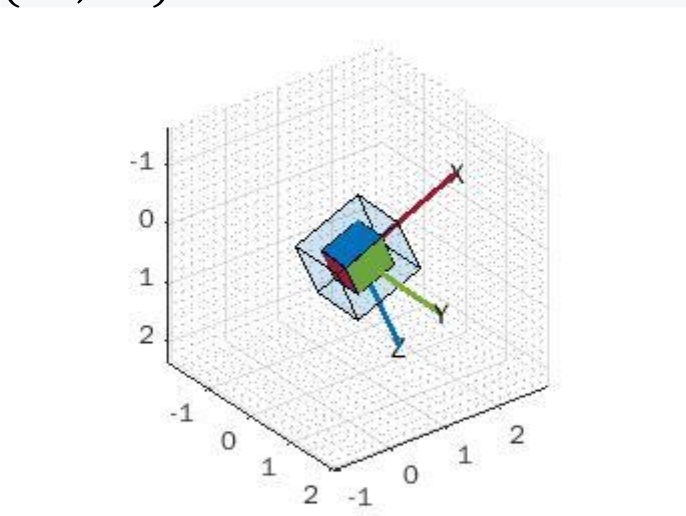
```
u1: Pixel coordinates in view 1
Size of u1 is 3x19
u2: Pixel coordinates in view 2
Size of u2 is 3x19
-------------
True E =
         0   -1.0000         0
   -0.3615         0   -3.1415
         0    3.0000         0

Estimated E =
   -0.1553   -0.2094   -0.2105
   -0.4499   -0.4425    0.7758
    0.4377    0.5900    0.5900
```

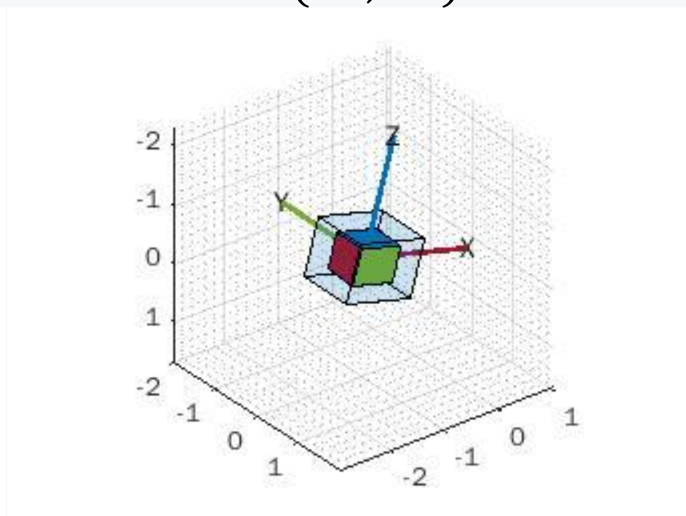
```
True R =
    0.9063         0   -0.4226
         0    1.0000         0
    0.4226         0    0.9063

Estimated R1 & R2 :
R1_est =
    0.9139    0.0054   -0.4058
    0.0028    0.9998    0.0197
    0.4058   -0.0192    0.9137

-------------
R2_est =
    0.9650   -0.0064    0.2623
   -0.0012   -0.9998   -0.0199
    0.2623    0.0188   -0.9648
```

```
True T =
     3
     0
     1

Estimated T1 & T2 :
T1_est =
    0.9422
    0.0008
    0.3351

-------------
T2_est =
   -0.9422
   -0.0008
   -0.3351
```



(R1, T1) (R2, T2)





From the above results, the second recovered pose obtained when all the points are used seems like the closest with the true pose, so I would choose this; however, looking at the results, the first pose obtained when all the points were used has a rotation very similar with the true rotation, and the translation vector has positive scaled components, so this should be the choice ideally.

I would definitely not choose the poses recovered when the points are only taken from one plane. This is due to the inevitable degeneracy that arises in the rank condition. For planar points, we would have to estimate the **homography**, not the essential matrix.

Verifications

1. Points from x-y plane

SO3 confirmed

SO3_U_	1.0000
SO3_V_	1

Epipoles and epipolar lines confirmed

verify_epipole1	-5.5511e-17
verify_epipole2	1.1102e-16
verify_point1	0.5061
verify_point2	0.5061

2. Points from x-z plane

SO3 confirmed

SO3_U_	1.0000
SO3_V_	1.0000

Epipoles and epipolar lines confirmed

verify_epipole1	1.1102e-16
verify_epipole2	-1.9429e-16
verify_point1	-0.1775
verify_point2	-0.1775

3. Points from y-z plane

SO3 confirmed

SO3_U_	1.0000
SO3_V_	1

Epipoles and epipolar lines confirmed

verify_epipole1	8.3267e-17
verify_epipole2	5.5511e-17
verify_point1	-0.1173
verify_point2	-0.1173

4. Using all points

SO3 confirmed

SO3_U_	1
SO3_V_	1.0000

Epipoles and epipolar lines confirmed

verify_epipole1	-1.1102e-16
verify_epipole2	1.1102e-16
verify_point1	0.6643
verify_point2	0.6643

Observations:

- Due to the random noise used in the simulation, we would always end up with slightly different results any time we rerun the code. In some cases, the determined of one of the left and right matrices from the SVD process would be negative one. Therefore, we need keep rerunning the code until SO3 is confirmed for both matrices.
- The epipoles are verified to a very high degree of accuracy. E.g., $-1\text{x}10^{-17}$; however, the points are verified within the range from 0 to about 0.8 in the worst case. In the ideal case, these should be exactly zero according to the following relations:

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

  My results were very close to zero, so I believe they are acceptable.

DISCUSSION

- The points we choose for the 8-point algorithm have to be a general position. This is to avoid degenerate conditions that may arise from planar points or quadratic surfaces.
- The 8-point algorithm provides 4 solutions in total (2 for +E and 2 for -E). The final choice of a plausible solution is determined using the positive depth constraint. In succinct terms, the solution that places the object in front of the camera is the right solution.

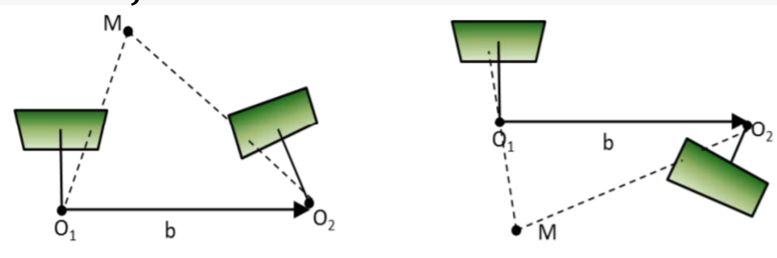


Figure 3. Camera orientation with respect to the observed points for "positive" baseline direction.

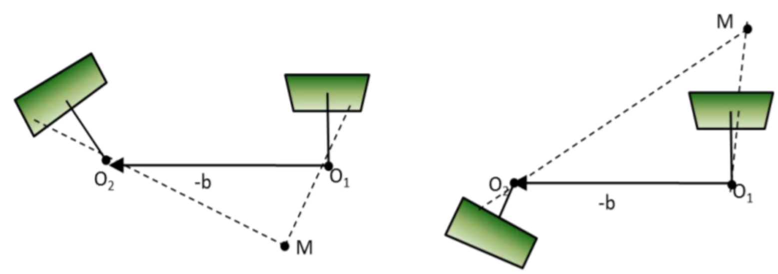


Figure 4. Camera orientation with respect to the observed points for "negative" baseline direction.

- The matrix E (as a function of (R, T)) has only a total of five degrees of freedom: three for rotation and two for translation (up to a scalar factor), so the number of points, eight, assumed by the -point algorithm is mostly for simplicity and convenience.
- This is why there are 5-point algorithms available in the literature; albeit nonlinear.
- Generally, in the recovered pose, the rotation should be very similar to that of the original object while the translation obtained would be a scaled version of the original one.
- For planar points, we estimate the **homography**, not the essential matrix.

REFERENCES

Reinhard Klette. 2014. Concise Computer Vision: An Introduction into Theory and Algorithms. Springer Publishing Company, Incorporated.

https://cseweb.ucsd.edu/classes/sp03/cse252/MaSKS_Ch5.pdf

https://www.plymouth.ac.uk/uploads/production/document/path/8/8593/Relative_Pose_-_George_Terzakis.pdf

APPENDIX

POSE RECOVERY

In-Lab Code

Lab9.m

```
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;

%% World Coordinates
% we need to select 8 points
since min 8 points is needed
to estimate the
% essential matrix E
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;
    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)*[4
5 162
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,:);
yxy=[];xyx =[];
for i=1:length(u2)
    x = round(u2(1,i));
y=round(u2(2,i));
    yxy=[yxy; y];xyx =[xyx;
x];
    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

XX = [];
p11 = []; p22 = [];
for ii =
[1,6,7,9,14,16,17,10]
   pp1 = K\u1(:,ii);
   pp2 = K\u2(:,ii);

   % Create a matrix for the
normalized image
coordinates. Points from
here
   % will be used to find
the epipoles and epipolar
lines and also verify
   % them.
   p11 = [p11; pp1']; p22 =
[p22; pp2'];
   aa = [pp1(1)*pp2;
pp1(2)*pp2; pp1(3)*pp2];
   XX = [XX; aa'];
end

%% Estimate E, cure it and
check for Essential Matrix
Characterization

[U,S,V] = svd(XX'*XX);

E = V(:,end);
```

```
E = [E(1:3) E(4:6) E(7:9)];

[U_,S_,V_] = svd(E);

S_(1,1) = 1; S_(2,2) = 1;
S_(3,3) = 0;

E_est = U_*S_*V_;

SO3_U_ = det(U_);
SO3_V_ = det(V_);

%% Find epipoles and
epipolar lines

e1 = null(E_est);
e2 = null(E_est');

L1 = E_est'*p22(1,:)';
L2 = E_est*p11(1,:)';

%% Verify epipoles and
epipolar lines

verify_epipole1 = L1'*e1;
verify_point1 =
L1'*p11(1,:)';
verify_epipole2 = L2'*e2;
verify_point2 =
L2'*p22(1,:)';

%% Recover the rotation and
the translation

ar = pi/2;
Rz1 = [0 -1 0;
       1  0 0;
       0  0 1];

Rz2 = [0  1 0;
       -1 0 0;
       0  0 1];

T1_1 = U_*Rz1*S_*U_';
R1 = U_*Rz1'*V_';

T2_2 = U_*Rz2*S_*U_';
R2 = U_*Rz2'*V_';

T1 = [T1_1(3,2); T1_1(1,3);
T1_1(2,1)];
T2 = [T2_2(3,2); T2_2(1,3);
T2_2(2,1)];

%% Compare your results with
ground truth
disp('True E =')
disp(Etrue)
disp('Estimated E = ')
disp(E_est)
disp('-------------')
disp('-------------')

% R should be exactly
similar, but one of them
only since the other means
% the case when the camera
is behind the view
disp('True R =')
disp(TrueR)
disp('Estimated R1 & R2 :')
disp('R1_est = ')
disp(R1)
disp('-------------')
disp('R2_est = ')
disp(R2)
disp('-------------')
disp('-------------')

% T should be scaled version
of True T, since we cannot
find a unique T and
% it is always up to scale
disp('True T =')
disp(TrueT)
```

```
disp('Estimated T1 & T2 :')
disp('T1_est = ')
disp(T1)
disp('-------------')
disp('T2_est = ')
disp(T2)
disp('-------------')
```

Post-Lab Codes

Lab9_x_y_plane.m

```
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;

%% World Coordinates
% we need to select 8 points
since min 8 points is needed
to estimate the
% essential matrix E
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;
    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)*[4
5 162
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

X = [];
p11 = []; p22 = [];
% The 8 points are chosen
from the x-y plane
for ii =
[1,2,10,11,16,17,18,19]
   pp1 = K\u1(:,ii);
   pp2 = K\u2(:,ii);

   % Create a matrix for the
normalized image
coordinates. Points from
here
   % will be used to find
the epipoles and epipolar
lines and also verify
   % them.
   p11 = [p11; pp1']; p22 =
[p22; pp2'];
   aa = [pp1(1)*pp2;
pp1(2)*pp2; pp1(3)*pp2];
```

```
    X = [X; aa'];
end

%% Estimate E, cure it and
check for Essential Matrix
Characterization

[U,S,V] = svd(X'*X);

E = V(:,end);

E = [E(1:3) E(4:6) E(7:9)];

[U_,S_,V_] = svd(E);

S_(1,1) = 1; S_(2,2) = 1;
S_(3,3) = 0;

E_est = U_*S_*V_;

SO3_U_ = det(U_);
SO3_V_ = det(V_);

%% Find epipoles and
epipolar lines

e1 = null(E_est);
e2 = null(E_est');

L1 = E_est'*p22(1,:)';
L2 = E_est*p11(1,:)';

%% Verify epipoles and
epipolar lines

verify_epipole1 = L1'*e1;
verify_point1 =
L1'*p11(1,:)';
verify_epipole2 = L2'*e2;
verify_point2 =
L2'*p22(1,:)';

%% Recover the rotation and
the translation

ar = pi/2;
Rz1 = [0 -1 0;
       1  0 0;
       0  0 1];

Rz2 = [0  1 0;
       -1 0 0;
       0  0 1];

T1_1 = U_*Rz1*S_*U_';
R1 = U_*Rz1'*V_';

T2_2 = U_*Rz2*S_*U_';
R2 = U_*Rz2'*V_';

T1 = [T1_1(3,2); T1_1(1,3);
T1_1(2,1)];
T2 = [T2_2(3,2); T2_2(1,3);
T2_2(2,1)];

%% Compare your results with
ground truth
disp('True E =')
disp(Etrue)
disp('Estimated E = ')
disp(E_est)
disp('-------------')
disp('-------------')

% R should be exactly
similar, but one of them
only since the other means
% the case when the camera
is behind the view
disp('True R =')
disp(TrueR)
disp('Estimated R1 & R2 :')
disp('R1_est = ')
disp(R1)
disp('-------------')
disp('R2_est = ')
```

```
disp(R2)
disp('-------------')
disp('-------------')

% T should be scaled version
of True T, since we cannot
find a unique T and
% it is always up to scale
disp('True T =')
disp(TrueT)
disp('Estimated T1 & T2 :')
disp('T1_est = ')
disp(T1)
disp('-------------')
disp('T2_est = ')
disp(T2)
disp('-------------')

%% Post-lab Image Plots

I1_ = zeros(L,L);
I2_ = zeros(L,L);

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)*[4
5 162
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')

M1_ = [R1 T1];

I1_ = zeros(L,L);
p1_ = K*(M1_*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 =1:length(u1_)
    x1_ = round(u1_(1,i_1));
y1_=round(u1_(2,i_1));
    if x1_ > 2 && y1_ > 2
    I1_(y1_-2:y1_+2, x1_-
2:x1_+2) = 255;
```

```
    end
end

subplot(1,3,2), imshow(I1_,
[]), title('View 1',
'FontSize',20);

M2_ = [R2 T2];

I2_ = zeros(L,L);
p2_ = K*(M2_*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_)));
    if x_ > 2 && y_ > 2
    I2_(y_-2:y_+2, x_-
2:x_+2) = 255;
    end
end

subplot(1,3,3), imshow(I2_,
[]), title('View 2',
'FontSize',20);
```

Lab9_x_z_plane.m

```
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;

%% World Coordinates
% we need to select 8 points
since min 8 points is needed
to estimate the
% essential matrix E

P_W=[0   2   0   1; %1
    0   1   0   1; %2
    0   0   0   1; %3
    0   2   -1  1; %4
    0   1   -1  1; %5
    0   0   -1  1; %6
    0   2   -2  1; %7
    0   1   -2  1; %8
    0   0   -2  1; %9
    1   0   0   1; %10
    2   0   0   1; %11
    1   0   -1  1; %12
    2   0   -1  1; %13
    1   0   -2  1; %14
    2   0   -2  1; %15
    1   1   0   1; %16
    2   1   0   1; %17
    1   2   0   1; %18
    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)*[4
5 162
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

X = [];
p11 = []; p22 = [];
% The 8 points are chosen
from the x-z plane
for ii =
[3,9,10,11,12,13,14,15]
   pp1 = K\u1(:,ii);
   pp2 = K\u2(:,ii);

   % Create a matrix for the
normalized image
coordinates. Points from
here
   % will be used to find
the epipoles and epipolar
lines and also verify
   % them.
   p11 = [p11; pp1']; p22 =
[p22; pp2'];
   aa = [pp1(1)*pp2;
pp1(2)*pp2; pp1(3)*pp2];
   X = [X; aa'];
end

%% Estimate E, cure it and
check for Essential Matrix
Characterization

[U,S,V] = svd(X'*X);

E = V(:,end);

E = [E(1:3) E(4:6) E(7:9)];

[U_,S_,V_] = svd(E);

S_(1,1) = 1; S_(2,2) = 1;
S_(3,3) = 0;

E_est = U_*S_*V_;

SO3_U_ = det(U_);
SO3_V_ = det(V_);

%% Find epipoles and
epipolar lines

e1 = null(E_est);
e2 = null(E_est');

L1 = E_est'*p22(1,:)';
L2 = E_est*p11(1,:)';

%% Verify epipoles and
epipolar lines

verify_epipole1 = L1'*e1;
verify_point1 =
L1'*p11(1,:)';
verify_epipole2 = L2'*e2;
verify_point2 =
L2'*p22(1,:)';

%% Recover the rotation and
the translation

ar = pi/2;
Rz1 = [0 -1 0;
       1  0 0;
       0  0 1];

Rz2 = [0  1 0;
       -1 0 0;
       0  0 1];

T1_1 = U_*Rz1*S_*U_';
R1 = U_*Rz1'*V_';

T2_2 = U_*Rz2*S_*U_';
R2 = U_*Rz2'*V_';

T1 = [T1_1(3,2); T1_1(1,3);
T1_1(2,1)];
```

```
T2 = [T2_2(3,2); T2_2(1,3);
T2_2(2,1)];

%% Compare your results with
ground truth
disp('True E =')
disp(Etrue)
disp('Estimated E = ')
disp(E_est)
disp('-------------')
disp('-------------')

% R should be exactly
similar, but one of them
only since the other means
% the case when the camera
is behind the view
disp('True R =')
disp(TrueR)
disp('Estimated R1 & R2 :')
disp('R1_est = ')
disp(R1)
disp('-------------')
disp('R2_est = ')
disp(R2)
disp('-------------')
disp('-------------')

% T should be scaled version
of True T, since we cannot
find a unique T and
% it is always up to scale
disp('True T =')
disp(TrueT)
disp('Estimated T1 & T2 :')
disp('T1_est = ')
disp(T1)
disp('-------------')
disp('T2_est = ')
disp(T2)
disp('-------------')
```

```
%% Post-lab Image Plots

I1_ = zeros(L,L);
I2_ = zeros(L,L);

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)*[4
5 162
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')

M1_ = [R1 T1];

I1_ = zeros(L,L);
p1_ = K*(M1_*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 =1:length(u1_)
    x1_ = round(u1_(1,i_1));
y1_=round(u1_(2,i_1));
    if x1_ > 2 && y1_ > 2
    I1_(y1_-2:y1_+2, x1_-
2:x1_+2) = 255;
    end
end

subplot(1,3,2), imshow(I1_,
[]), title('View 1',
'FontSize',20);

M2_ = [R2 T2];

I2_ = zeros(L,L);
p2_ = K*(M2_*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_)));
    if x_ > 2 && y_ > 2
    I2_(y_-2:y_+2, x_-
2:x_+2) = 255;
    end
end

subplot(1,3,3), imshow(I2_,
[]), title('View 2',
'FontSize',20);
```

Lab9_y_z_plane.m

```
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;
```

```
%% World Coordinates
% we need to select 8 points
since min 8 points is needed
to estimate the
% essential matrix E
P_W=[0  2   0   1; %1
     0   1   0   1; %2
     0   0   0   1; %3
     0   2   -1  1; %4
     0   1   -1  1; %5
     0   0   -1  1; %6
     0   2   -2  1; %7
     0   1   -2  1; %8
     0   0   -2  1; %9
     1   0   0   1; %10
     2   0   0   1; %11
     1   0   -1  1; %12
     2   0   -1  1; %13
     1   0   -2  1; %14
     2   0   -2  1; %15
     1   1   0   1; %16
     2   1   0   1; %17
     1   2   0   1; %18
     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)*[4
5 162
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

X = [];
p11 = []; p22 = [];
% The 8 points are chosen
from the y-z plane
for ii = [1,2,3,4,5,6,7,8]
   pp1 = K\u1(:,ii);
   pp2 = K\u2(:,ii);

   % Create a matrix for the
normalized image
coordinates. Points from
here
   % will be used to find
the epipoles and epipolar
lines and also verify
   % them.
   p11 = [p11; pp1']; p22 =
[p22; pp2'];
   aa = [pp1(1)*pp2;
pp1(2)*pp2; pp1(3)*pp2];
   X = [X; aa'];
end

%% Estimate E, cure it and
check for Essential Matrix
Characterization

[U,S,V] = svd(X'*X);

E = V(:,end);

E = [E(1:3) E(4:6) E(7:9)];

[U_,S_,V_] = svd(E);

S_(1,1) = 1; S_(2,2) = 1;
S_(3,3) = 0;

E_est = U_*S_*V_;

SO3_U_ = det(U_);
SO3_V_ = det(V_);

%% Find epipoles and
epipolar lines

e1 = null(E_est);
e2 = null(E_est');

L1 = E_est'*p22(1,:)';
L2 = E_est*p11(1,:)';
```

```
%% Verify epipoles and
epipolar lines

verify_epipole1 = L1'*e1;
verify_point1 =
L1'*p11(1,:)';
verify_epipole2 = L2'*e2;
verify_point2 =
L2'*p22(1,:)';

%% Recover the rotation and
the translation

ar = pi/2;
Rz1 = [0 -1 0;
       1  0 0;
       0  0 1];

Rz2 = [0  1 0;
       -1 0 0;
       0  0 1];

T1_1 = U_*Rz1*S_*U_';
R1 = U_*Rz1'*V_';

T2_2 = U_*Rz2*S_*U_';
R2 = U_*Rz2'*V_';

T1 = [T1_1(3,2); T1_1(1,3);
T1_1(2,1)];
T2 = [T2_2(3,2); T2_2(1,3);
T2_2(2,1)];

%% Compare your results with
ground truth
disp('True E =')
disp(Etrue)
disp('Estimated E = ')
disp(E_est)
disp('-------------')
disp('-------------')
```

```
% R should be exactly
similar, but one of them
only since the other means
% the case when the camera
is behind the view
disp('True R =')
disp(TrueR)
disp('Estimated R1 & R2 :')
disp('R1_est = ')
disp(R1)
disp('-------------')
disp('R2_est = ')
disp(R2)
disp('-------------')
disp('-------------')

% T should be scaled version
of True T, since we cannot
find a unique T and
% it is always up to scale
disp('True T =')
disp(TrueT)
disp('Estimated T1 & T2 :')
disp('T1_est = ')
disp(T1)
disp('-------------')
disp('T2_est = ')
disp(T2)
disp('-------------')

%% Post-lab Image Plots

I1_ = zeros(L,L);
I2_ = zeros(L,L);

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)*[4
5 162
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')

M1_ = [R1 T1];

I1_ = zeros(L,L);
p1_ = K*(M1_*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 =1:length(u1_)
    x1_ = round(u1_(1,i_1));
y1_=round(u1_(2,i_1));
    if x1_ > 2 && y1_ > 2
    I1_(y1_-2:y1_+2, x1_-
2:x1_+2) = 255;
    end
end

subplot(1,3,2), imshow(I1_,
[]), title('View 1',
'FontSize',20);

Hc2_ = [R1 T1; 0 0 0 1];
Hc2c2_ = [R2 T2; 0 0 0 1];
H2_ = Hc2c2_*Hc2_;

R2_ = H2_(1:3,1:3);
T2_ = H2_(1:3,4);
M2_ = [R2_ T2_];

I2_ = zeros(L,L);
p2_ = K*(M2_*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_)));
    if x_ > 2 && y_ > 2
    I2_(y_-2:y_+2, x_-
2:x_+2) = 255;
    end
end

subplot(1,3,3), imshow(I2_,
[]), title('View 2',
'FontSize',20);
```

Lab9_all_points.m

```
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;

%% World Coordinates
% we need to select 8 points
since min 8 points is needed
to estimate the
% essential matrix E
P_W=[0  2   0   1; %1
    0   1   0   1; %2
    0   0   0   1; %3
    0   2   -1  1; %4
    0   1   -1  1; %5
    0   0   -1  1; %6
    0   2   -2  1; %7
    0   1   -2  1; %8
    0   0   -2  1; %9
    1   0   0   1; %10
    2   0   0   1; %11
    1   0   -1  1; %12
    2   0   -1  1; %13
    1   0   -2  1; %14
    2   0   -2  1; %15
    1   1   0   1; %16
    2   1   0   1; %17
    1   2   0   1; %18
    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)*[4
5 162
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,:);
yxy=[];xyx =[];
for i=1:length(u1)
    x = round(u1(1,i));
y=round(u1(2,i));
    yxy=[yxy; y];xyx =[xyx;
x];
    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,:);
yxy1=[];xyx1 =[];
for i=1:length(u2)
    x = round(u2(1,i));
y=round(u2(2,i));
    yxy1=[yxy1; y];xyx1
=[xyx1; x];
    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

X = [];
p11 = []; p22 = [];
% The 8 points are chosen
from the y-z plane
for ii = 19:-1:1
   pp1 = K\u1(:,ii);
   pp2 = K\u2(:,ii);

   % Create a matrix for the
normalized image
coordinates. Points from
here
   % will be used to find
the epipoles and epipolar
lines, and also verify
```

```
    % them.
    p11 = [p11; pp1']; p22 =
[p22; pp2'];
    aa = [pp1(1)*pp2;
pp1(2)*pp2; pp1(3)*pp2];
    X = [X; aa'];
end
rankin = rank(X'*X);

%% Estimate E, cure it and
check for Essential Matrix
Characterization

[U,S,V] = svd(X'*X);

E = V(:,end);

E = [E(1:3) E(4:6) E(7:9)];

[U_,S_,V_] = svd(E);

S_(1,1) = 1; S_(2,2) = 1;
S_(3,3) = 0;

E_est = U_*S_*V_;

SO3_U_ = det(U_);
SO3_V_ = det(V_);

%% Find epipoles and
epipolar lines

e1 = null(E_est);
e2 = null(E_est');

L1 = E_est'*p22(5,:)';
L2 = E_est*p11(5,:)';

%% Verify epipoles and
epipolar lines

verify_epipole1 = L1'*e1;
verify_point1 =
L1'*p11(5,:)';
verify_epipole2 = L2'*e2;
verify_point2 =
L2'*p22(5,:)';

%% Recover the rotation and
the translation

ar = pi/2;
Rz1 = [0 -1 0;
       1  0 0;
       0  0 1];

Rz2 = [0  1 0;
       -1 0 0;
       0  0 1];

T1_1 = U_*Rz1*S_*U_';
R1 = U_*Rz1'*V_';

T2_2 = U_*Rz2*S_*U_';
R2 = U_*Rz2'*V_';

T1 = [T1_1(3,2); T1_1(1,3);
T1_1(2,1)];
T2 = [T2_2(3,2); T2_2(1,3);
T2_2(2,1)];

%% Compare your results with
ground truth
disp('True E =')
disp(Etrue)
disp('Estimated E = ')
disp(E_est)
disp('-------------')
disp('-------------')

% R should be exactly
similar, but one of them
only since the other means
```

```
% the case when the camera
is behind the view
disp('True R =')
disp(TrueR)
disp('Estimated R1 & R2 :')
disp('R1_est = ')
disp(R1)
disp('-------------')
disp('R2_est = ')
disp(R2)
disp('-------------')
disp('-------------')

% T should be scaled version
of True T, since we cannot
find a unique T and
% it is always up to scale
disp('True T =')
disp(TrueT)
disp('Estimated T1 & T2 :')
disp('T1_est = ')
disp(T1)
disp('-------------')
disp('T2_est = ')
disp(T2)
disp('-------------')

%% Post-lab Image Plots

I1_ = zeros(L,L);
I2_ = zeros(L,L);

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)*[4
5 162
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')

M1_ = [R1 T1];

I1_ = zeros(L,L);
p1_ = K*(M1_*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,:);
yxy_=[];xyx_ =[];
for i_1 =1:length(u1_)
    x1_ = round(u1_(1,i_1));
y1_=round(u1_(2,i_1));
    yxy_=[yxy_; y1_];xyx_
=[xyx_; x1_];
    if x1_ > 2 && y1_ > 2
    I1_(y1_-2:y1_+2, x1_-
2:x1_+2) = 255;
    end
end

subplot(1,3,2), imshow(I1_,
[]), title('View 1',
'FontSize',20);

M2_ = [R2 T2];

I2_ = zeros(L,L);
p2_ = K*(M2_*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,:);
yxy_1=[];xyx_1 =[];
for i_=1:length(u2_)
    x_ = (round(u2_(1,i_)));
y_= (round(u2_(2,i_)));
    yxy_1=[yxy_1; y_];xyx_1
=[xyx_1; x_];
    if x_ > 2 && y_ > 2
    I2_(y_-2:y_+2, x_-
2:x_+2) = 255;
    end
end

subplot(1,3,3), imshow(I2_,
[]), title('View 2',
'FontSize',20);
```