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**

- 1. Introduction
- 2. Explanation of Methods
- 3. Results
- 4. Multiple Results and Post-lab Questions
- 5. Discussion
- 6. References
- 7. Appendix

# 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.

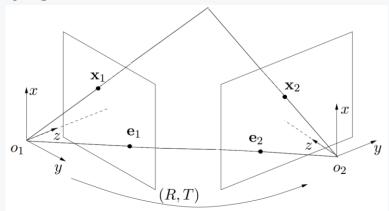
The general concept for this is as follows:

- 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  $\lambda_i$ 's are the depths. s  $\lambda 2x2 = R\lambda 1x1 + T$ . In order to eliminate the depths  $\lambda_i$ '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.

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

The epipolar constraint  $\mathbf{x}_2^T\widehat{T}R\mathbf{x}_1=0.$  Where  $E=\widehat{T}R\in\mathbb{R}^{3 imes 3}$  is the essential matrix that captures the orientation  $E=\begin{bmatrix}e_1&e_2&e_3\\e_4&e_5&e_6\\e_7&e_8&e_9\end{bmatrix}$ 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.

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

- This leaves us with a homogenous equation of the form on the right.
- Es 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<sup>T</sup>X. This obtained by taking the last column of the right orthogonal matrix after performing a singular value decomposition on X<sup>T</sup>X. We obtain a matrix, F here.

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

• 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.

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

- 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:

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

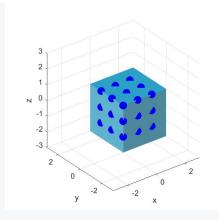
• Finally, two solutions (two for +E; two for -E) can be obtained for the rotation and translation:

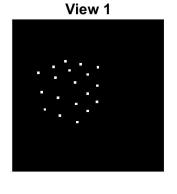
$$R = UR_Z^T \left(\pm \frac{\pi}{2}\right) V^T, \quad \widehat{T} = UR_Z \left(\pm \frac{\pi}{2}\right) \Sigma U^T.$$

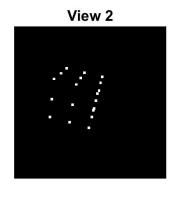
• From the above, the translation vector could be obtained from the components of the translation matrix:

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

# **IN-LAB RESULTS**







ul: Pixel coordinates in view 1 Size of ul is 3x19 u2: Pixel coordinates in view 2 Size of u2 is 3x19

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

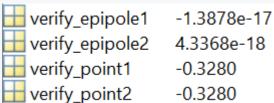
True R =							
0.9063	0	-0.4226					
0	1.0000	0					
0.4226	0	0.9063					
Estimated R1	s D2 •						
	α ΚΖ .						
R1_est =							
0.9987	0.0472	-0.0182					
0.0472	-0.9989	-0.0005					
-0.0182	-0.0003	-0.9998					
R2_est =							
0.9184	-0.0090	-0.3955					
0.0052	0.9999	-0.0108					
0.3956	0.0078	0.9184					

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.



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

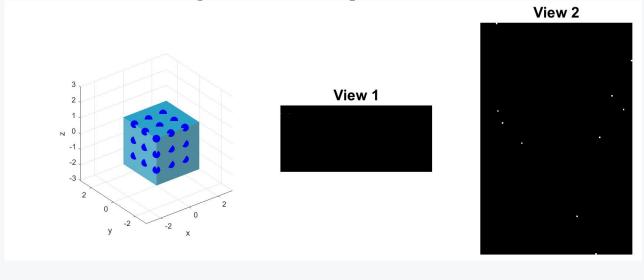
#### X-Y Plane

ul: Pixel coordinates in view 1 Size of ul is 3x19 u2: Pixel coordinates in view 2 Size of u2 is 3x19	0.9063 0 -0.4226	True T = 3 0 1
True E =  0 -1.0000 0  -0.3615 0 -3.1415 0 3.0000 0	Estimated R1 & R2: R1_est =     0.9081    0.4179    0.0282     -0.4066    0.8957    -0.1799     -0.1004    0.1519    0.9833	Estimated T1 & T2: T1_est =     -0.3543     -0.2452     0.9024
Estimated E =  -0.8707 -0.0345 0.3395  0.0723 -0.9522 -0.1672  -0.3222 -0.2723 0.0878	R2_est = -0.6866 -0.2545 -0.6811 0.5599 -0.7826 -0.2720 -0.4638 -0.5681 0.6798	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*.

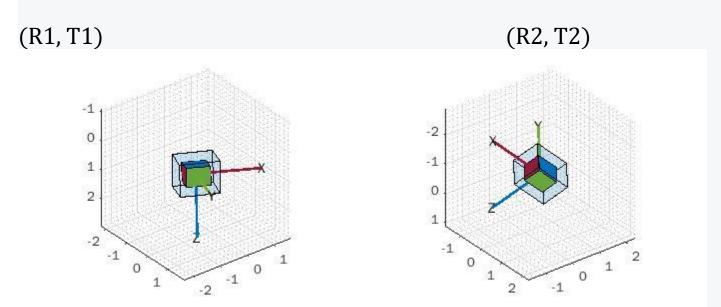
$$M2_{-} = [R2 T2];$$
  
 $p2 = K*(M2 *P W);$ 

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

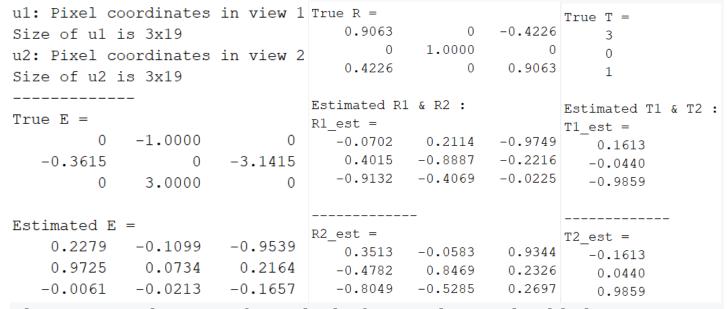


To solve the above issue, I simply used the **poseplot function available** in MATLAB 2021b version to obtain the follow results:

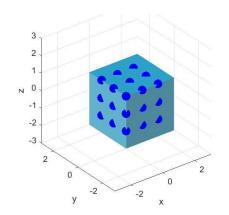
Code: poseplot(R1, T1):



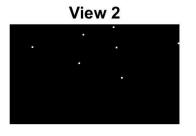
#### X-Z Plane



The same explanation from the before is also applicable here.

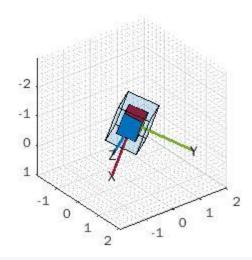


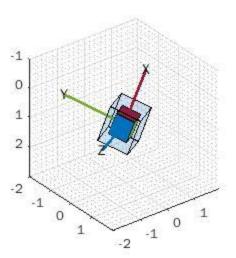
# View 1



#### Using the poseplot function,

(R1, T1) (R2, T2)





#### Y-Z Plane

ul: Pixel coordinates in view 1
Size of ul is 3x19

u2: Pixel coordinates in view 2
Size of u2 is 3x19

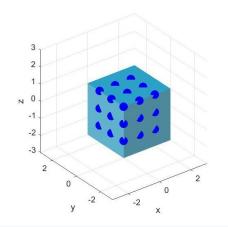
0.906

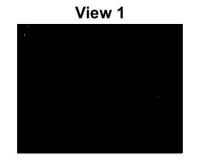
-----

		True E =
0	-1.0000	0
-3.1415	0	-0.3615
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 = True T = 0.9063 0 -0.4226 3 0 0.4226 0 0.9063 1

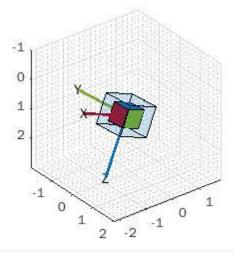
Estimated R1 & R2: Estimated T1 & T2: R1\_est = T1\_est = 0.9909 -0.1244 0.0507 -0.1957 0.1228 0.9919 0.0334 0.0718 -0.0544 -0.0269 0.9982 0.9780



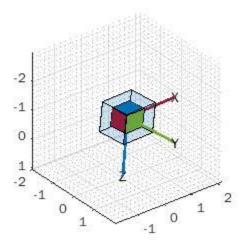




(R1, T1)

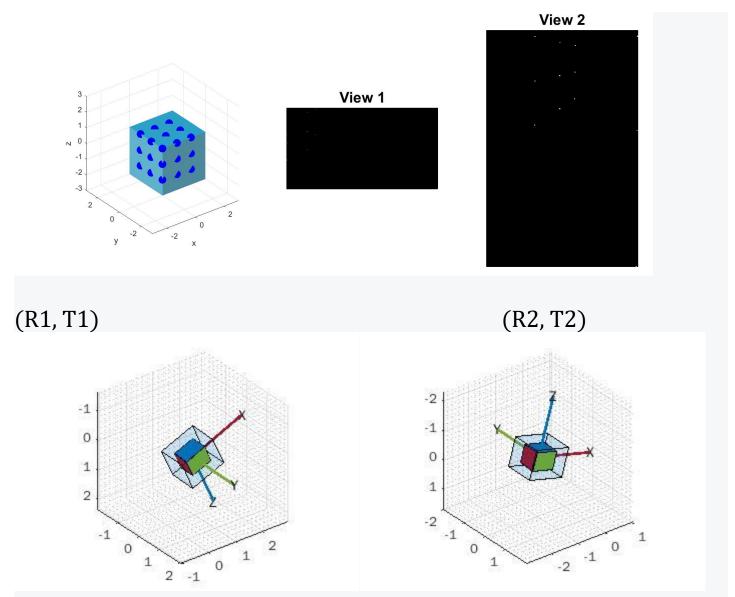


(R2, T2)



#### Using all the Points

ul: Pixel co	s 3x19		0.9063	0 1.0000	-0.4226 0	True T = 3 0
u2: Pixel co Size of u2 i		in view 2	0.4226	0	0.9063	1
True E = 0 -0.3615	-1.0000 0 3.0000	0 -3.1415 0	Estimated R1 R1_est = 0.9139 0.0028 0.4058	<pre>&amp; R2 :     0.0054     0.9998 -0.0192</pre>	-0.4058 0.0197 0.9137	Estimated T1 & T2: T1_est =     0.9422     0.0008     0.3351
Estimated E -0.1553 -0.4499 0.4377	-0.2094 -0.4425 0.5900	-0.2105 0.7758 0.5900	R2_est = 0.9650 -0.0012 0.2623	-0.0064 -0.9998 0.0188		-0.0008



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 Epipoles and epipolar lines confirmed

verify\_epipole1 1.1102e-16
verify\_epipole2 -1.9429e-16
SO3\_U\_ 1.0000 verify\_point1 -0.1775
SO3\_V\_ 1.0000 verify\_point2 -0.1775

#### 3. Points from y-z plane

#### SO3 confirmed 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

SO3 V

SO3\_U\_ 1 SO3\_V\_ 1.0000

#### Epipoles and epipolar lines confirmed

1.0000

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., -1x10<sup>-17</sup>; 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:

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

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.

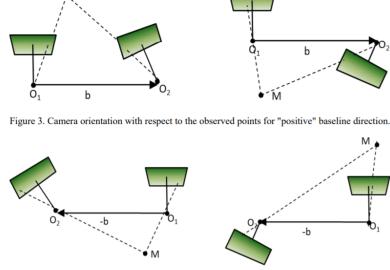


Figure 4. Camera orientation with respect to the observed points for "negative" baseline direction.

- 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/spo3/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 11;
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
    0
                 1;
    0
        0
           0
                1;
    0
        2 -1 1;
        1
    0
            -1
                1;
    ()
        0
          -1
                1;
        2
    0
           -2
                1;
        1
    0
            -2
                 1;
    ()
        0
          -2
                 1;
    1
        0
                 1;
            0
    2
        0
            0
                 1;
    1
        0
          -1
                1;
    2
            -1
                 1;
    1
          -2
        0
                 1;
    2
        0
            -2
                1;
    1
        1
             0
                 1;
    2
        1
             \cap
                 1:
```

```
1
        2
           0
                 1;
    2
                 1
            \cap
    ];
PW = PW';
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);
wallv =
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, wallv',
wallz, 'FaceColor', (1/255) * [4
5 162
200], 'EdgeColor', 'none')
```

```
plot3(P W(1,:), P W(2,:), P W(
                                   for i=1:length(u1)
3,:),'b.','MarkerSize',36);
                                        x = round(u1(1,i));
axis equal;
                                   y=round(u1(2,i));
grid on
                                        I1(y-2:y+2, x-2:x+2) =
axis vis3d;
                                   255;
axis([-3 \ 3 \ -3 \ 3 \ -3 \ 3])
                                   end
grid on
xlabel('x')
                                   subplot(1,3,2), imshow(I1,
ylabel('y')
                                   []), title('View 1',
zlabel('z')
                                   'FontSize',20);
%% Camera Transformation for
                                   %% Camera Transformation for
View 1
                                   View 2
ax = 120 * DEG TO RAD;
                                   ax = 0 * DEG TO RAD;
ay = 0 *DEG TO RAD;
                                   ay = -25 * DEG TO RAD;
az = 60 * DEG TO RAD;
                                   az = 0 * DEG \overline{TO} \overline{RAD};
Rx = [1 \ 0 \ 0;
                                   Rx = [1 \ 0 \ 0;
      0 \cos(ax) - \sin(ax);
                                         0 \cos(ax) - \sin(ax);
      0 sin(ax) cos(ax)];
                                         0 sin(ax) cos(ax)];
Ry = [\cos(ay) \quad 0 \quad \sin(ay);
                                   Ry = [\cos(ay) \quad 0 \quad \sin(ay);
            0 1 0;
                                               0 1 0;
      -sin(ay) 0 cos(ay)];
                                          -\sin(ay) \ 0 \ \cos(ay)];
Rz = [\cos(az) - \sin(az) 0;
                                   Rz = [\cos(az) - \sin(az) 0;
      sin(az) cos(az) 0;
                                          sin(az) cos(az) 0;
                  ()
                         11;
                                                      0
                                                             1];
Rc1 = Rx*Ry*Rz;
                                   Rc2c1 = Rx*Ry*Rz;
Tc1 = [0;0;5];
                                   TrueR = Rc2c1;
M = [Rc1 Tc1];
                                   Tc2c1 = [3;0;1];
                                   TrueT = Tc2c1;
p1 = K*(M * P W);
                                   Hc1 = [Rc1 Tc1; 0 0 0 1];
noise1 = 4*rand(3, NPTS) - 2;
                                   Hc2c1 = [Rc2c1 Tc2c1; 0 0 0]
noise1(3,:)=1;
                                   11;
p1 = p1 + noise1;
                                   Hc2 = Hc2c1*Hc1;
u1(1,:) = p1(1,:)./
                                   Rc2 = Hc2(1:3,1:3);
p1(3,:);
                                   Tc2 = Hc2(1:3,4);
u1(2,:) = p1(2,:) ./
p1(3,:);
                                   M = [Rc2 Tc2];
u1(3,:) = p1(3,:) ./
p1(3,:);
```

```
I2 = zeros(L,L);
                                  u2info = ['Size of u2 is '
p2 = K*(M*P W);
                                  num2str(size(u2,1)) 'x'
                                  num2str(size(u2,2))];
noise2 = 4*rand(3, NPTS)-2;
                                  disp(u2info)
noise2(3,:)=1;
                                  disp('----')
p2 = p2 + noise2;
                                  %% Lab#8 Assignment starts
                                  here.
u2(1,:) = p2(1,:)./
                                  %% Transform pixel
p2(3,:);
                                  coordinates and construct X
u2(2,:) = p2(2,:) ./
                                  matrix using Equations 1 and
p2(3,:);
u2(3,:) = p2(3,:) ./
p2(3,:);
                                  XX = [];
\forall x \forall = []; x \forall x = [];
                                  p11 = []; p22 = [];
for i=1:length(u2)
                                  for ii =
    x = round(u2(1,i));
                                  [1,6,7,9,14,16,17,10]
y=round(u2(2,i));
                                     pp1 = K \setminus u1(:,ii);
    yxy=[yxy; y];xyx = [xyx;
                                     pp2 = K \setminus u2(:,ii);
x];
    12(y-2:y+2, x-2:x+2) =
                                     % Create a matrix for the
255;
                                  normalized image
                                  coordinates. Points from
end
                                  here
subplot(1,3,3), imshow(I2,
                                     % will be used to find
[]), title('View 2',
                                  the epipoles and epipolar
'FontSize', 20);
                                  lines and also verify
                                      % them.
t = Tc2c1;
                                     p11 = [p11; pp1']; p22 =
T \text{ skew} = [0 - t(3) t(2); t(3)]
                                  [p22; pp2'];
0 - t(1); - t(2) t(1) 0];
                                     aa = [pp1(1)*pp2;
Etrue = T skew*Rc2c1;
                                  pp1(2)*pp2; pp1(3)*pp2];
                                     XX = [XX; aa'];
%% Displaying the
                                  end
information
disp('u1: Pixel coordinates
                                  %% Estimate E, cure it and
in view 1')
                                  check for Essential Matrix
ulinfo = ['Size of ul is '
                                  Characterization
num2str(size(u1,1)) 'x'
num2str(size(u1,2))];
                                  [U,S,V] = svd(XX'*XX);
disp(ulinfo)
disp('u2: Pixel coordinates E = V(:,end);
in view 2')
```

```
T1_1 = U_*Rz1*S_*U_';
R1 = U_*Rz1'*V_';
E = [E(1:3) E(4:6) E(7:9)];
[U,S,V] = svd(E);
                                T2 2 = U *Rz2*S_*U_';
S(1,1) = 1; S(2,2) = 1;
                                R2 = U * \overline{R}z2'*V ;
S(3,3) = 0;
                                T1 = [T1 \ 1(3,2); \ T1 \ 1(1,3);
E est = U_*S_*V_;
                                T1 1 (2,1) ];
                                T2 = [T2_2(3,2); T2_2(1,3);
SO3 U = det(U);
                                T2 2(2,1)];
SO3V = det(V);
                                %% Compare your results with
%% Find epipoles and
                                 ground truth
epipolar lines
                                disp('True E =')
                                disp(Etrue)
e1 = null(E est);
                                disp('Estimated E = ')
e2 = null(E est');
                                disp(E est)
                                disp('----')
L1 = E est'*p22(1,:)';
                                disp('----')
L2 = E est*p11(1,:)';
                                % R should be exactly
%% Verify epipoles and
                                 similar, but one of them
epipolar lines
                                only since the other means
verify epipole1 = L1'*e1;
                                 % the case when the camera
verify point1 =
                                 is behind the view
L1'*p11(1,:)';
                                 disp('True R =')
verify epipole2 = L2'*e2;
                                 disp(TrueR)
verify point2 =
                                disp('Estimated R1 & R2 :')
                                disp('R1 est = ')
L2'*p22(1,:)';
                                disp(R1)
%% Recover the rotation and
                                disp('----')
the translation
                                disp('R2 est = ')
                                disp(R2)
ar = pi/2;
                                disp('----')
Rz1 = [0 -1 0;
                                disp('----')
       1 0 0;
       0 0 11;
                                 % T should be scaled version
                                of True T, since we cannot
Rz2 = [0 \ 1 \ 0;
                                find a unique T and
       -1 0 0;
                                 % it is always up to scale
       0 0 11;
                                disp('True T =')
                                disp(TrueT)
```

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

#### **Post-Lab Codes**

#### Lab9\_x\_y\_plane.m

```
-1 1;
clear all; close all; clc;
                                     1
                                         0 -1 1;
0 -2 1;
%% Definitions
                                     2
                                     1
%rng(1);
                                         0 -2 1;
L = 300;
                                     2
                                         1 0 1;
1 0 1;
I1 = zeros(L,L);
                                     1
                                     2
                                         2 0
f=L;
                                     1
                                                  1;
u0 = L/2;
                                     2
                                          2
                                             \cap
                                                  1
v0 = L/2;
                                     ];
K = [f \ 0 \ u0;
                                 PW = PW';
    0 f v0;
                                 NPTS = size(P W,2); %Number
    0 0 11;
                                 of points
DEG TO RAD = pi/180;
                                 %% Visualization
                                 figure;
%% World Coordinates
                                 subplot(1,3,1)
% we need to select 8 points
                                 wally = meshgrid(0:0.1:3);
since min 8 points is needed
                                 wallz = meshgrid(-3:0.1:0);
to estimate the
                                 wallx =
% essential matrix E
                                 0*ones(size(wallz,1));
P_W = [0 \ 2 \ 0 \ 1;
                                 surf(wallx, wally,
        1 0
    0
                1;
                                 wallz','FaceColor',(1/255)*[
        0 0 1;
2 -1 1;
    0
                                 97 178
    0
                                 205], 'EdgeColor', 'none')
       1 -1 1;
    0
                                 hold on
        0 -1 1;
    0
                                 wallx = meshgrid(0:0.1:3);
        2 -2 1;
    0
                                 wallz = meshgrid(-3:0.1:0);
        1 -2 1;
    0
                                 wallv =
    0
           -2
                1;
                                 0*ones(size(wallz,1));
        0
    1
            0
                1;
                                 surf(wallx, wally,
    2.
                1;
                                 wallz','FaceColor',(1/255)*[
```

```
77 137
                                  p1 = K*(M * P W);
157], 'EdgeColor', 'none')
                                  noise1 = 4*rand(3, NPTS)-2;
wallx = meshgrid(0:0.1:3);
                                  noise1(3,:)=1;
wally = meshgrid(0:0.1:3);
                                  p1 = p1 + noise1;
wallz =
                                  u1(1,:) = p1(1,:)./
zeros(size(wally,1)); %
                                  p1(3,:);
Generate z data
                                  u1(2,:) = p1(2,:)./
surf(wallx, wally',
                                  p1(3,:);
wallz, 'FaceColor', (1/255) * [4
                                  u1(3,:) = p1(3,:) ./
5 162
                                  p1(3,:);
200], 'EdgeColor', 'none')
plot3(PW(1,:),PW(2,:),PW(
                                   for i=1:length(u1)
3,:),'b.','MarkerSize',36);
                                       x = round(u1(1,i));
axis equal;
                                   y=round(u1(2,i));
grid on
                                       I1(y-2:y+2, x-2:x+2) =
axis vis3d;
                                   255;
axis([-3 \ 3 \ -3 \ 3 \ -3 \ 3])
                                   end
grid on
xlabel('x')
                                   subplot(1,3,2), imshow(I1,
ylabel('y')
                                   []), title('View 1',
zlabel('z')
                                   'FontSize',20);
%% Camera Transformation for
                                   %% Camera Transformation for
View 1
                                  View 2
ax = 120 * DEG_TO_RAD;
                                  ax = 0 * DEG_TO_RAD;
ay = 0 *DEG TO RAD;
                                   ay = -25 * DEG TO RAD;
az = 60 * DEG TO RAD;
                                   az = 0 * DEG TO RAD;
Rx = [1 \ 0 \ 0;
                                  Rx = [1 \ 0 \ 0;
      0 \cos(ax) - \sin(ax);
                                        0 \cos(ax) - \sin(ax);
      0 sin(ax) cos(ax)];
                                         0 sin(ax) cos(ax)];
Ry = [\cos(ay) \quad 0 \quad \sin(ay);
                                  Ry = [\cos(ay) \quad 0 \quad \sin(ay);
               1
                   0;
                                               0 1
                                                         0;
      -\sin(ay) \ 0 \ \cos(ay)];
                                         -\sin(ay) \ 0 \ \cos(ay)];
Rz = [\cos(az) - \sin(az) 0;
                                  Rz = [\cos(az) - \sin(az) 0;
      sin(az) cos(az) 0;
                                         sin(az) cos(az) 0;
      0
                  0
                         11;
                                                     0
                                                            11;
Rc1 = Rx*Ry*Rz;
                                  Rc2c1 = Rx*Ry*Rz;
Tc1 = [0;0;5];
                                   TrueR = Rc2c1;
M = [Rc1 Tc1];
                                   Tc2c1 = [3;0;1];
```

```
disp('u1: Pixel coordinates
TrueT = Tc2c1;
Hc1 = [Rc1 Tc1; 0 0 0 1];
                                  in view 1')
Hc2c1 = [Rc2c1 Tc2c1; 0 0 0]
                                  ulinfo = ['Size of ul is '
                                  num2str(size(u1,1)) 'x'
11;
                                  num2str(size(u1,2))];
Hc2 = Hc2c1*Hc1;
                                  disp(ulinfo)
Rc2 = Hc2(1:3,1:3);
                                  disp('u2: Pixel coordinates
Tc2 = Hc2(1:3,4);
                                  in view 2')
                                  u2info = ['Size of u2 is '
M = [Rc2 Tc2];
                                  num2str(size(u2,1)) 'x'
                                  num2str(size(u2,2))];
I2 = zeros(L, L);
                                  disp(u2info)
p2 = K*(M*P W);
                                  disp('----')
                                  %% Lab#8 Assignment starts
noise2 = 4*rand(3, NPTS)-2;
                                  here.
noise2(3,:)=1;
                                  %% Transform pixel
p2 = p2 + noise2;
                                  coordinates and construct X
                                 matrix using Equations 1 and
u2(1,:) = p2(1,:) ./
p2(3,:);
u2(2,:) = p2(2,:) ./
                                  X = [];
p2(3,:);
                                 p11 = []; p22 = [];
u2(3,:) = p2(3,:) ./
                                  % The 8 points are chosen
p2(3,:);
                                  from the x-y plane
for i=1:length(u2)
                                  for ii =
    x = round(u2(1,i));
                                  [1,2,10,11,16,17,18,19]
                                     pp1 = K \setminus u1(:,ii);
y=round(u2(2,i));
    12(y-2:y+2, x-2:x+2) =
                                     pp2 = K \setminus u2(:,ii);
255;
                                     % Create a matrix for the
end
                                  normalized image
subplot(1,3,3), imshow(I2,
                                  coordinates. Points from
[]), title('View 2',
                                  here
'FontSize', 20);
                                     % will be used to find
                                  the epipoles and epipolar
t = Tc2c1;
                                  lines and also verify
T \text{ skew} = [0 - t(3) t(2); t(3)]
                                     % them.
0^{-}-t(1); -t(2) t(1) 0];
                                     p11 = [p11; pp1']; p22 =
Etrue = T skew*Rc2c1;
                                  [p22; pp2'];
                                     aa = [pp1(1)*pp2;
%% Displaying the
                                  pp1(2)*pp2; pp1(3)*pp2];
information
```

```
X = [X; aa'];
                                 ar = pi/2;
end
                                Rz1 = [0 -1 0;
%% Estimate E, cure it and
                                        1 0 0;
check for Essential Matrix
                                        0 0 1];
Characterization
                                Rz2 = [0 \ 1 \ 0;
                                       -1 0 0;
[U,S,V] = svd(X'*X);
                                        0 0 11;
E = V(:,end);
                                T1 1 = U *Rz1*S_*U_';
E = [E(1:3) E(4:6) E(7:9)];
                                R1 = U *Rz1'*V ;
[U,S,V] = svd(E);
                                T2 2 = U *Rz2*S *U';
                                R2 = U *Rz2'*V ;
S_{1}(1,1) = 1; S_{2}(2,2) = 1;
S(3,3) = 0;
                                T1 = [T1 \ 1(3,2); \ T1 \ 1(1,3);
                                T1 1(2,1)];
E est = U_*S_*V_;
                                 T2 = [T2 \ 2(3,2); \ T2_2(1,3);
                                T2 2(2,1)];
SO3 U = det(U_);
SO3 V = det(V);
                                 %% Compare your results with
                                 ground truth
%% Find epipoles and
                                disp('True E =')
epipolar lines
                                disp(Etrue)
                                 disp('Estimated E = ')
e1 = null(E est);
                                 disp(E est)
e2 = null(E est');
                                disp('----')
                                disp('----')
L1 = E est'*p22(1,:)';
L2 = E est*p11(1,:)';
                                % R should be exactly
                                 similar, but one of them
%% Verify epipoles and
epipolar lines
                                 only since the other means
                                 % the case when the camera
verify epipole1 = L1'*e1;
                                 is behind the view
verify point1 =
                                 disp('True R =')
L1'*p11(1,:)';
                                 disp(TrueR)
verify epipole2 = L2'*e2;
                                 disp('Estimated R1 & R2 :')
verify point2 =
                                 disp('R1 est = ')
L2'*p22(1,:)';
                                 disp(R1)
                                 disp('----')
%% Recover the rotation and
                                disp('R2 est = ')
the translation
```

```
disp(R2)
                                 wallx = meshgrid(0:0.1:3);
disp('----')
                                 wally = meshgrid(0:0.1:3);
disp('----')
                                 wallz =
                                 zeros(size(wally,1)); %
% T should be scaled version
                                 Generate z data
of True T, since we cannot
                                 surf(wallx, wally',
find a unique T and
                                 wallz, 'FaceColor', (1/255) * [4
% it is always up to scale
                                 5 162
disp('True T =')
                                 200], 'EdgeColor', 'none')
disp(TrueT)
                                 plot3(PW(1,:),PW(2,:),PW(
disp('Estimated T1 & T2 :')
                                 3,:),'b.','MarkerSize',36);
disp('T1 est = ')
                                 axis equal;
disp(T1)
                                 grid on
disp('----')
                                 axis vis3d;
disp('T2 est = ')
                                 axis([-3 \ 3 \ -3 \ 3 \ -3 \ 3])
disp(T2)
                                 grid on
disp('----')
                                 xlabel('x')
                                 ylabel('y')
%% Post-lab Image Plots
                                 zlabel('z')
I1 = zeros(L,L);
                                 M1 = [R1 T1];
I2 = zeros(L, L);
                                 I1 = zeros(L, L);
figure;
                                 p1 = K*(M1 *P W);
subplot(1,3,1)
wally = meshgrid(0:0.1:3);
                                 noise1 = 4*rand(3, NPTS)-2;
wallz = meshgrid(-3:0.1:0);
                                 noise1(3,:)=1;
wallx =
                                 p1 = p1 + noise1;
0*ones(size(wallz,1));
surf(wallx, wally,
                                 u1 (1,:) = p1 (1,:) ./
wallz', 'FaceColor', (1/255) * [
                                 p1 (3,:);
                                 u1 (2,:) = p1 (2,:) ./
97 178
                                 p1_(3,:);
205], 'EdgeColor', 'none')
hold on
                                 u1 (3,:) = p1 (3,:) ./
                                 p1 (3,:);
wallx = meshgrid(0:0.1:3);
wallz = meshgrid(-3:0.1:0);
                                 for i 1 =1:length(u1)
wally =
                                     x1 = round(u1 (1, i 1));
0*ones(size(wallz,1));
                                 y1 = round(u1 (2, i 1));
surf(wallx, wally,
                                     if x1_ > 2 && y1_ > 2
wallz', 'FaceColor', (1/255) * [
                                     I1 (y\overline{1} -2:y1_+2, x1_-
77 137
                                 2:x1_{+2}) = 255;
157], 'EdgeColor', 'none')
```

```
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 11;
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]
           0 1; %1
           0 1; %2
    0
        0
               1; %3
    0
        2 -1 1; %4
    0
        1 -1 1; %5
0 -1 1; %6
    ()
    \cap
        2 -2 1; %7
    0
        1
           -2 1; %8
    \Omega
        0 -2 1; %9
    ()
           0 1; %10
    1
        0
    2
        0
               1; %11
           -1 1; %12
    1
        0
    2
        0
            -1 1; %13
           -2 1; %14
    1
        0
        0 -2 1; %15
    2
        1
    1
           0
               1; %16
    2
        1 0 1; %17
           0
    1
        2
               1; %18
        2
    2
           \cap
                1
    ];
```

```
ylabel('y')
PW = PW';
NPTS = size(P W, 2); %Number
                                  zlabel('z')
of points
                                  %% Camera Transformation for
%% Visualization
                                  View 1
                                  ax = 120 * DEG TO RAD;
figure;
subplot(1,3,1)
                                  ay = 0 *DEG TO RAD;
wally = meshgrid(0:0.1:3);
                                  az = 60 * DEG TO RAD;
wallz = meshgrid(-3:0.1:0);
                                  Rx = [1 \ 0 \ 0;
wallx =
                                        0 \cos(ax) - \sin(ax);
0*ones(size(wallz,1));
                                        0 sin(ax) cos(ax)];
surf(wallx, wally,
                                  Ry = [\cos(ay) \quad 0 \quad \sin(ay);
wallz', 'FaceColor', (1/255) * [
                                              0
                                                  1
                                                         0;
97 178
                                         -\sin(ay) 0 \cos(ay);
205], 'EdgeColor', 'none')
                                  Rz = [\cos(az) - \sin(az) 0;
hold on
                                         sin(az) cos(az) 0;
wallx = meshgrid(0:0.1:3);
                                                           1];
                                                    0
wallz = meshgrid(-3:0.1:0);
wally =
                                  Rc1 = Rx*Ry*Rz;
0*ones(size(wallz,1));
                                  Tc1 = [0;0;5];
surf(wallx, wally,
                                  M = [Rc1 Tc1];
wallz','FaceColor',(1/255)*[
77 137
                                  p1 = K*(M * P W);
157], 'EdgeColor', 'none')
                                  noise1 = 4*rand(3, NPTS)-2;
wallx = meshgrid(0:0.1:3);
                                  noise1(3,:)=1;
wally = meshgrid(0:0.1:3);
                                  p1 = p1 + noise1;
wallz =
zeros(size(wally,1)); %
                                  u1(1,:) = p1(1,:)./
Generate z data
                                  p1(3,:);
surf(wallx, wally',
                                  u1(2,:) = p1(2,:) ./
wallz, 'FaceColor', (1/255) * [4
                                  p1(3,:);
5 162
                                  u1(3,:) = p1(3,:) ./
200], 'EdgeColor', 'none')
                                  p1(3,:);
plot3(P W(1,:),P W(2,:),P W(
3,:),'b.','MarkerSize',36);
                                  for i=1:length(u1)
axis equal;
                                      x = round(u1(1,i));
grid on
                                  y=round(u1(2,i));
                                       I1(y-2:y+2, x-2:x+2) =
axis vis3d;
axis([-3 \ 3 \ -3 \ 3 \ -3 \ 3])
                                  255;
grid on
                                  end
xlabel('x')
```

```
subplot(1,3,2), imshow(I1,
                                 u2(1,:) = p2(1,:)./
[]), title('View 1',
                                 p2(3,:);
                                 u2(2,:) = p2(2,:) ./
'FontSize', 20);
                                 p2(3,:);
%% Camera Transformation for
                                 u2(3,:) = p2(3,:) ./
View 2
                                 p2(3,:);
ax = 0 * DEG_TO_RAD;
ay = -25 * DEG TO RAD;
                                  for i=1:length(u2)
az = 0 * DEG TO RAD;
                                      x = round(u2(1,i));
                                  y=round(u2(2,i));
Rx = [1 \ 0 \ 0;
                                      12(y-2:y+2, x-2:x+2) =
     0 \cos(ax) - \sin(ax);
                                  255;
      0 sin(ax) cos(ax)];
                                 end
Ry = [\cos(ay) \quad 0 \quad \sin(ay);
           0 1 0;
                                  subplot(1,3,3), imshow(I2,
                                  []), title('View 2',
     -\sin(ay) 0 \cos(ay);
Rz = [\cos(az) - \sin(az) 0;
                                  'FontSize', 20);
      sin(az) cos(az) 0;
                                 t = Tc2c1;
      0
                  ()
                        1];
                                 T \text{ skew} = [0 - t(3) t(2); t(3)]
                                 0 - t(1); - t(2) t(1) 0];
Rc2c1 = Rx*Ry*Rz;
                                 Etrue = T skew*Rc2c1;
TrueR = Rc2c1;
Tc2c1 = [3;0;1];
                                  %% Displaying the
TrueT = Tc2c1;
                                  information
Hc1 = [Rc1 Tc1; 0 0 0 1];
                                 disp('u1: Pixel coordinates
Hc2c1 = [Rc2c1 Tc2c1; 0 0 0]
                                 in view 1')
11;
                                 ulinfo = ['Size of ul is '
Hc2 = Hc2c1*Hc1;
                                 num2str(size(u1,1)) 'x'
                                 num2str(size(u1,2))];
Rc2 = Hc2(1:3,1:3);
Tc2 = Hc2(1:3,4);
                                  disp(ulinfo)
                                 disp('u2: Pixel coordinates
M = [Rc2 Tc2];
                                 in view 2')
                                 u2info = ['Size of u2 is '
I2 = zeros(L, L);
                                 num2str(size(u2,1)) 'x'
p2 = K*(M*P W);
                                 num2str(size(u2,2))];
                                 disp(u2info)
                                 disp('----')
noise2 = 4*rand(3, NPTS)-2;
noise2(3,:)=1;
                                 %% Lab#8 Assignment starts
p2 = p2 + noise2;
                                 here.
                                  %% Transform pixel
                                  coordinates and construct X
```

```
matrix using Equations 1 and
                                 SO3 U = det(U);
2
                                 SO3_V_ = det(V);
X = [];
                                 %% Find epipoles and
p11 = []; p22 = [];
                                 epipolar lines
% The 8 points are chosen
from the x-z plane
                                 e1 = null(E est);
for ii =
                                 e2 = null(E est');
[3, 9, 10, 11, 12, 13, 14, 15]
   pp1 = K \setminus u1(:,ii);
                                 L1 = E est'*p22(1,:)';
   pp2 = K \setminus u2(:,ii);
                                 L2 = E est*p11(1,:)';
   % Create a matrix for the
                                 %% Verify epipoles and
normalized image
                                 epipolar lines
coordinates. Points from
here
                                 verify epipole1 = L1'*e1;
   % will be used to find
                                 verify point1 =
the epipoles and epipolar
                                 L1'*p11(1,:)';
lines and also verify
                                 verify_epipole2 = L2'*e2;
   % them.
                                 verify_point2 =
   p11 = [p11; pp1']; p22 =
                                 L2'*p22(1,:)';
[p22; pp2'];
   aa = [pp1(1)*pp2;
                                 %% Recover the rotation and
pp1(2)*pp2; pp1(3)*pp2];
                                 the translation
   X = [X; aa'];
                                  ar = pi/2;
end
                                 Rz1 = [0 -1 0;
%% Estimate E, cure it and
                                         1 0 0;
check for Essential Matrix
                                         0 0 1];
Characterization
                                 Rz2 = [0 \ 1 \ 0;
[U,S,V] = svd(X'*X);
                                         -1 0 0;
                                         0 0 11;
E = V(:,end);
                                 T1 1 = U *Rz1*S_*U_';
E = [E(1:3) E(4:6) E(7:9)];
                                 R1 = U *Rz1 *V ;
[U,S,V] = svd(E);
                                 T2 2 = U *Rz2*S *U ';
                                 R2 = U *Rz2'*V ;
S(1,1) = 1; S(2,2) = 1;
S(3,3) = 0;
                                 T1 = [T1_1(3,2); T1_1(1,3);
                                 T1 1(2,1);
E = U *S *V ;
```

```
T2 = [T2 \ 2(3,2); T2 \ 2(1,3);
                                %% Post-lab Image Plots
T2 2(2,1)];
                                I1 = zeros(L, L);
%% Compare your results with
                                I2 = zeros(L,L);
ground truth
                                figure;
disp('True E =')
                                subplot(1,3,1)
disp(Etrue)
                                wally = meshgrid(0:0.1:3);
disp('Estimated E = ')
                                wallz = meshgrid(-3:0.1:0);
disp(E est)
disp('----')
                                wallx =
                                0*ones(size(wallz,1));
disp('----')
                                surf(wallx, wally,
                                wallz','FaceColor',(1/255)*[
% R should be exactly
                                97 178
similar, but one of them
only since the other means
                                205], 'EdgeColor', 'none')
% the case when the camera
                                hold on
is behind the view
                                wallx = meshgrid(0:0.1:3);
disp('True R =')
                                wallz = meshgrid(-3:0.1:0);
disp(TrueR)
                                wally =
disp('Estimated R1 & R2 :')
                                0*ones(size(wallz,1));
disp('R1 est = ')
                                surf(wallx, wally,
                                wallz','FaceColor',(1/255)*[
disp(R1)
disp('----')
                                77 137
disp('R2 est = ')
                                157], 'EdgeColor', 'none')
                                wallx = meshgrid(0:0.1:3);
disp(R2)
disp('----')
                                wally = meshgrid(0:0.1:3);
disp('----')
                                wallz =
                                zeros(size(wally,1)); %
% T should be scaled version
                                Generate z data
of True T, since we cannot
                                surf(wallx, wally',
find a unique T and
                                wallz, 'FaceColor', (1/255) * [4]
% it is always up to scale
                                5 162
disp('True T =')
                                200], 'EdgeColor', 'none')
disp(TrueT)
                                plot3(PW(1,:),PW(2,:),PW(
disp('Estimated T1 & T2 :')
                                3,:),'b.','MarkerSize',36);
disp('T1 est = ')
                                axis equal;
disp(T1)
                                grid on
disp('----')
                                axis vis3d;
disp('T2_est = ')
                                axis([-3 \ 3 \ -3 \ 3 \ -3 \ 3])
disp(T2)
                                grid on
disp('----')
                                xlabel('x')
```

```
ylabel('v')
                                  M2 = [R2 T2];
zlabel('z')
                                  I2 = zeros(L,L);
M1 = [R1 T1];
                                  p2 = K* (M2 *P W);
I1 = zeros(L, L);
                                  noise2 = 4*rand(3, NPTS)-2;
p1 = K*(M1 *P W);
                                  noise2(3,:)=1;
                                 p2_{-} = p2_{-} + noise2;
noise1 = 4*rand(3, NPTS)-2;
noise1(3,:)=1;
                                 u2 (1,:) = p2 (1,:) ./
p1 = p1 + noise1;
                                  p2 (3,:);
u1 (1,:) = p1 (1,:) ./
                                 u2_(2,:) = p2_(2,:) ./
                                 p2 (3,:);
p1 (3,:);
u1 (2,:) = p1 (2,:) ./
                                 u2 (3,:) = p2 (3,:) ./
p1 (3,:);
                                 p2 (3,:);
u1^{-}(3,:) = p1 (3,:) ./
                                  for i =1:length(u2)
p1 (3,:);
                                      x = (round(u2 (1,i_)));
                                  y = (round(u2_(2,i_)));
for i 1 =1:length(u1)
    x1 = round(u1 (1, i 1));
                                      if x > 2 \&\& y > 2
                                      12_{y_--2:y_+-2, x_-}
y1 = round(u1 (2, i 1));
    if x1 > 2 && y1 > 2
                                  2:x +2) = 255;
    I1 (y\overline{1} - 2:y1 + 2, x1 -
                                      end
2:x1 +2) = 255;
                                  end
    end
                                  subplot(1,3,3), imshow(I2,
end
                                  []), title('View 2',
                                  'FontSize',20);
subplot(1,3,2), imshow(I1,
[]), title('View 1',
'FontSize', 20);
Lab9 y z plane.m
clear all; close all; clc; v0 = L/2;
%% Definitions
                                  K = [f \ 0 \ u0;
%rng(1);
                                      0 f v0;
L = 300;
                                      0 0 1];
I1 = zeros(L, L);
                                  DEG TO RAD = pi/180;
f=L;
u0 = L/2;
```

```
%% World Coordinates
                                  hold on
% we need to select 8 points
                                  wallx = meshgrid(0:0.1:3);
                                  wallz = meshgrid(-3:0.1:0);
since min 8 points is needed
to estimate the
                                  wallv =
                                  0*ones(size(wallz,1));
% essential matrix E
P W = [0]
        2
                 1; %1
                                  surf(wallx, wally,
            0
                 1; %2
                                  wallz','FaceColor',(1/255)*[
    0
        1
            0
    0
        0
                1; %3
                                  77 137
        2
           -1 1; %4
                                  157], 'EdgeColor', 'none')
    0
                                  wallx = meshgrid(0:0.1:3);
    0
        1
           -1 1; %5
            -1 1; %6
                                  wally = meshgrid(0:0.1:3);
    0
        0
        2
    0
           -2
                1; %7
                                  wallz =
    0
        1
           -2
                1; %8
                                  zeros(size(wally,1)); %
            -2
                1; %9
                                  Generate z data
    0
    1
        0
            0
                1; %10
                                  surf(wallx, wally',
    2
                                  wallz, 'FaceColor', (1/255) * [4
                1; %11
        0
            0
    1
                1; %12
        0
            -1
                                  5 162
            -1 1; %13
                                  200], 'EdgeColor', 'none')
    2
        0
                                  plot3(P W(1,:),P_W(2,:),P_W(
    1
        0
            -2 1; %14
                                  3,:),'b.','MarkerSize',36);
    2
        0
           -2
                1; %15
    1
        1
                1; %16
                                  axis equal;
            0
        1
    2
                1; %17
                                  grid on
            0
            0
    1
        2
                1; %18
                                  axis vis3d;
    2
                                  axis([-3 \ 3 \ -3 \ 3 \ -3 \ 3])
            0
                 1
                                  arid on
    ];
                                  xlabel('x')
PW = PW';
                                  ylabel('y')
NPTS = size(P W,2); %Number
                                  zlabel('z')
of points
                                  %% Camera Transformation for
%% Visualization
                                  View 1
figure;
                                  ax = 120 * DEG_TO_RAD;
subplot(1,3,1)
                                  ay = 0 *DEG TO RAD;
wally = meshgrid(0:0.1:3);
                                  az = 60 * DEG TO RAD;
wallz = meshgrid(-3:0.1:0);
wallx =
                                  Rx = [1 \ 0 \ 0;
0*ones(size(wallz,1));
                                       0 \cos(ax) - \sin(ax);
surf(wallx, wally,
                                        0 sin(ax) cos(ax)];
wallz','FaceColor',(1/255)*[
                                  Ry = [\cos(ay) \quad 0 \quad \sin(ay);
97 178
                                             0 1
                                                        0;
205], 'EdgeColor', 'none')
                                        -sin(ay) 0
                                                     cos(ay)];
```

```
-\sin(ay) \ 0 \ \cos(ay)];
Rz = [\cos(az) - \sin(az) 0;
                                 Rz = [\cos(az) - \sin(az) 0;
      sin(az) cos(az)
                        0;
                                        sin(az) cos(az) 0;
                  0
                        11;
                                                   0 1];
Rc1 = Rx*Ry*Rz;
Tc1 = [0;0;5];
                                  Rc2c1 = Rx*Ry*Rz;
M = [Rc1 Tc1];
                                  TrueR = Rc2c1;
                                  Tc2c1 = [3;0;1];
p1 = K*(M * P W);
                                  TrueT = Tc2c1;
noise1 = 4*rand(3,NPTS)-2;
                                  Hc1 = [Rc1 Tc1; 0 0 0 1];
noise1(3,:)=1;
                                  Hc2c1 = [Rc2c1 Tc2c1; 0 0 0]
p1 = p1 + noise1;
                                  11;
                                 Hc2 = Hc2c1*Hc1;
u1(1,:) = p1(1,:)./
p1(3,:);
                                 Rc2 = Hc2(1:3,1:3);
u1(2,:) = p1(2,:) ./
                                  Tc2 = Hc2(1:3,4);
p1(3,:);
u1(3,:) = p1(3,:) ./
                                 M = [Rc2 Tc2];
p1(3,:);
                                  I2 = zeros(L,L);
                                 p2 = K*(M*P W);
for i=1:length(u1)
    x = round(u1(1,i));
                                 noise2 = 4*rand(3, NPTS)-2;
y=round(u1(2,i));
                                  noise2(3,:)=1;
    I1(y-2:y+2, x-2:x+2) =
                                 p2 = p2 + noise2;
255;
end
                                 u2(1,:) = p2(1,:)./
                                 p2(3,:);
subplot(1,3,2), imshow(I1,
                                 u2(2,:) = p2(2,:) ./
[]), title('View 1',
                                  p2(3,:);
'FontSize',20);
                                 u2(3,:) = p2(3,:) ./
%% Camera Transformation for
                                 p2(3,:);
View 2
                                  for i=1:length(u2)
ax = 0 * DEG TO RAD;
                                      x = round(u2(1,i));
ay = -25 * DEG TO RAD;
                                  y=round(u2(2,i));
az = 0 * DEG TO RAD;
                                      12(y-2:y+2, x-2:x+2) =
Rx = [1 \ 0 \ 0;
                                  255;
      0 \cos(ax) - \sin(ax);
                                  end
      0 sin(ax) cos(ax)];
Ry = [\cos(ay) \quad 0 \quad \sin(ay);
           0
                1
                      0;
```

```
subplot(1,3,3), imshow(I2,
                                  coordinates. Points from
[]), title('View 2',
                                  here
'FontSize', 20);
                                     % will be used to find
                                  the epipoles and epipolar
t = Tc2c1;
                                  lines and also verify
T \text{ skew} = [0 - t(3) t(2); t(3)]
                                     % them.
0 - t(1); -t(2) t(1) 0];
                                     p11 = [p11; pp1']; p22 =
Etrue = T skew*Rc2c1;
                                  [p22; pp2'];
                                     aa = [pp1(1)*pp2;
%% Displaying the
                                  pp1(2)*pp2; pp1(3)*pp2];
information
                                     X = [X; aa'];
disp('u1: Pixel coordinates
                                  end
in view 1')
ulinfo = ['Size of ul is '
                                  %% Estimate E, cure it and
num2str(size(u1,1)) 'x'
                                  check for Essential Matrix
num2str(size(u1,2));
                                  Characterization
disp(ulinfo)
disp('u2: Pixel coordinates
                                  [U,S,V] = svd(X'*X);
in view 2')
                                  E = V(:,end);
u2info = ['Size of u2 is '
num2str(size(u2,1)) 'x'
                                  E = [E(1:3) E(4:6) E(7:9)];
num2str(size(u2,2))];
disp(u2info)
                                  [U_{,}S_{,}V_{]} = svd(E);
disp('----')
%% Lab#8 Assignment starts
                                  S(1,1) = 1; S(2,2) = 1;
here.
                                  S(3,3) = 0;
%% Transform pixel
coordinates and construct X
                                  E est = U *S *V ;
matrix using Equations 1 and
                                  SO3 U = det(U);
2
                                  SO3 V = det(V);
X = [];
                                  %% Find epipoles and
p11 = []; p22 = [];
                                  epipolar lines
% The 8 points are chosen
from the y-z plane
                                  e1 = null(E est);
for ii = [1, 2, 3, 4, 5, 6, 7, 8]
                                  e2 = null(E est');
   pp1 = K \setminus u1(:,ii);
   pp2 = K \setminus u2(:,ii);
                                  L1 = E est'*p22(1,:)';
                                  L2 = E est*p11(1,:)';
   % Create a matrix for the
normalized image
```

```
% R should be exactly
%% Verify epipoles and
                                similar, but one of them
epipolar lines
                                only since the other means
verify epipole1 = L1'*e1;
                                % the case when the camera
verify point1 =
                                is behind the view
L1'*p11(1,:)';
                                disp('True R =')
verify epipole2 = L2'*e2;
                                disp(TrueR)
verify point2 =
                                disp('Estimated R1 & R2 :')
L2'*p22(1,:)';
                                disp('R1 est = ')
                                disp(R1)
%% Recover the rotation and
                                disp('----')
the translation
                                disp('R2 est = ')
                                disp(R2)
ar = pi/2;
                                disp('----')
Rz1 = [0 -1 0;
                                disp('----')
      1 0 0;
       0 0 1];
                                % T should be scaled version
                                of True T, since we cannot
Rz2 = [0 \ 1 \ 0;
                                find a unique T and
      -1 0 0;
                                % it is always up to scale
       0 0 11;
                                disp('True T =')
                                disp(TrueT)
T1 1 = U *Rz1*S *U ';
                                disp('Estimated T1 & T2 :')
R1 = U *Rz1'*V';
                                disp('T1 est = ')
T2 2 = U_*Rz2*S_*U_';
                                disp(T1)
R2 = U *Rz2'*V ';
                                disp('----')
                                disp('T2 est = ')
T1 = [T1 \ 1(3,2); \ T1 \ 1(1,3);
                                disp(T2)
T1 1(2,1);
                                disp('----')
T2 = [T2 \ 2(3,2); T2 \ 2(1,3);
T2 \ 2(2,1);
                                %% Post-lab Image Plots
                                I1 = zeros(L,L);
%% Compare your results with
                                I2 = zeros(L,L);
ground truth
disp('True E =')
                                figure;
disp(Etrue)
                                subplot(1,3,1)
disp('Estimated E = ')
                                wally = meshgrid(0:0.1:3);
disp(E est)
disp('----')
                               wallz = meshgrid(-3:0.1:0);
                                wallx =
disp('----')
                                0*ones(size(wallz,1));
```

```
u1 (1,:) = p1_{(1,:)} ./
surf(wallx, wally,
wallz','FaceColor', (1/255)*[
                                 p1 (3,:);
                                  u1 (2,:) = p1 (2,:) ./
97 178
                                  p1 (3,:);
205], 'EdgeColor', 'none')
                                  u1 (3,:) = p1 (3,:) ./
hold on
wallx = meshgrid(0:0.1:3);
                                 p1 (3,:);
wallz = meshgrid(-3:0.1:0);
                                  for i 1 =1:length(u1)
wally =
                                      x1 = round(u1 (1, i 1));
0*ones(size(wallz,1));
                                  y1 = round(u1 (2, i 1));
surf(wallx, wally,
                                      if x1 > 2 && y1 > 2
wallz','FaceColor',(1/255)*[
                                      I1 (y\overline{1} -2:y1_+2, x1_-
77 137
                                  2:x1_{+2}) = 255;
157], 'EdgeColor', 'none')
                                      end
wallx = meshgrid(0:0.1:3);
wally = meshgrid(0:0.1:3);
                                  end
wallz =
                                  subplot(1,3,2), imshow(I1,
zeros(size(wally,1)); %
                                  []), title('View 1',
Generate z data
                                  'FontSize', 20);
surf(wallx, wally',
wallz, 'FaceColor', (1/255) * [4]
                                  Hc2 = [R1 T1; 0 0 0 1];
5 162
                                  Hc2c2 = [R2 T2; 0 0 0 1];
200], 'EdgeColor', 'none')
                                  H2 = Hc2c2 *Hc2;
plot3(PW(1,:),PW(2,:),PW(
3,:),'b.','MarkerSize',36);
                                  R2 = H2 (1:3,1:3);
axis equal;
                                     = H2 (1:3,4);
grid on
                                  M2 = [R2 T2];
axis vis3d;
axis([-3 \ 3 \ -3 \ 3 \ -3 \ 3])
                                  I2 = zeros(L,L);
                                  p2 = K* (M2 *P W);
grid on
xlabel('x')
                                  noise2 = 4*rand(3, NPTS)-2;
ylabel('y')
                                  noise2(3,:)=1;
zlabel('z')
                                  p2 = p2 + noise2;
M1 = [R1 T1];
                                 u2 (1,:) = p2 (1,:) ./
I1 = zeros(L,L);
                                  p2 (3,:);
                                  u2^{-}(2,:) = p2_{-}(2,:) ./
p1 = K*(M1 *P W);
                                 p2_(3,:);
noise1 = 4*rand(3, NPTS)-2;
                                 u2 (3,:) = p2 (3,:) ./
noise1(3,:)=1;
                                 p2 (3,:);
p1 = p1 + noise1;
```

```
for i =1:length(u2)
                                     end
    x = (round(u2 (1,i)));
                                 end
y = (round(u2_(2,i_)));
                                 subplot(1,3,3), imshow(I2,
    if x > 2 \&\& y > 2
                                 []), title('View 2',
    12_{(y_--2:y_+2,x_--
                                 'FontSize', 20);
2:x +2) = 255;
Lab9_all_points.m
clear all; close all; clc;
                                     1
                                             -1 1; %12
                                           -1 1; %13
%% Definitions
                                         0 -2 1; %14
                                     1
%rng(1);
                                     2.
                                            -2 1; %15
L = 300;
                                         0
                                     1
                                         1 0 1; %16
I1 = zeros(L,L);
                                     2
                                         1
                                            0
                                                1; %17
                                            0 1; %18
f=L;
                                     1
                                         2
u0 = L/2;
                                         2. 0
                                     2
                                                 1
v0 = L/2;
                                     ];
K = [f \ 0 \ u0;
                                PW = PW';
    0 f v0;
                                 NPTS = size(P W, 2); %Number
    0 0 11;
                                 of points
DEG TO RAD = pi/180;
                                 %% Visualization
                                 figure;
%% World Coordinates
                                 subplot(1,3,1)
% we need to select 8 points
                                 wally = meshgrid(0:0.1:3);
since min 8 points is needed
                                 wallz = meshgrid(-3:0.1:0);
to estimate the
                                 wallx =
% essential matrix E
                                 0*ones(size(wallz,1));
P W = [0]
      2 0 1; %1
                                 surf(wallx, wally,
        1 0 1; %2
    0
                                 wallz','FaceColor',(1/255)*[
    0
                1; %3
                                 97 178
        2 -1 1; %4
    ()
                                 205], 'EdgeColor', 'none')
           -1 1; %5
    0
        1
                                 hold on
        0 -1 1; %6
    ()
                                 wallx = meshgrid(0:0.1:3);
        2 -2 1; %7
1 -2 1; %8
    ()
                                 wallz = meshgrid(-3:0.1:0);
    0
                                 wally =
    ()
        0 -2 1; %9
                                 0*ones(size(wallz,1));
                1; %10
    1
            0
                                 surf(wallx, wally,
                1; %11
                                 wallz','FaceColor',(1/255)*[
```

```
77 137
                                   p1 = K*(M * P W);
157], 'EdgeColor', 'none')
                                   noise1 = 4*rand(3, NPTS)-2;
wallx = meshgrid(0:0.1:3);
                                   noise1(3,:)=1;
wally = meshgrid(0:0.1:3);
                                   p1 = p1 + noise1;
wallz =
                                   u1(1,:) = p1(1,:)./
zeros(size(wally,1)); %
                                   p1(3,:);
Generate z data
                                   u1(2,:) = p1(2,:) ./
surf(wallx, wally',
                                   p1(3,:);
wallz, 'FaceColor', (1/255) * [4
                                   u1(3,:) = p1(3,:) ./
5 162
                                   p1(3,:);
200], 'EdgeColor', 'none')
                                   yxy=[];xyx=[];
plot3(PW(1,:),PW(2,:),PW(
                                   for i=1:length(u1)
3,:),'b.','MarkerSize',36);
                                       x = round(u1(1,i));
axis equal;
                                   y=round(u1(2,i));
grid on
                                       yxy=[yxy; y];xyx = [xyx;
axis vis3d;
                                   x];
axis([-3 \ 3 \ -3 \ 3 \ -3 \ 3])
                                       I1(y-2:y+2, x-2:x+2) =
grid on
                                   255;
xlabel('x')
                                   end
ylabel('y')
zlabel('z')
                                   subplot(1,3,2), imshow(I1,
                                   []), title('View 1',
%% Camera Transformation for
                                   'FontSize',20);
View 1
ax = 120 * DEG TO RAD;
                                   %% Camera Transformation for
ay = 0 *DEG TO RAD;
                                   View 2
az = 60 * DEG TO RAD;
                                   ax = 0 * DEG_TO_RAD;
                                   ay = -25 * DEG TO RAD;
Rx = [1 \ 0 \ 0;
                                   az = 0 * DEG TO RAD;
      0 \cos(ax) - \sin(ax);
      0 \sin(ax) \cos(ax);
                                   Rx = [1 \ 0 \ 0;
Ry = [\cos(ay) \quad 0 \quad \sin(ay);
                                         0 \cos(ax) - \sin(ax);
               1
                       0;
                                         0 sin(ax) cos(ax)];
      -\sin(ay) \ 0 \ \cos(ay)];
                                   Rv = [\cos(ay) \quad 0 \quad \sin(ay);
Rz = [\cos(az) - \sin(az) 0;
                                               0 1
                                                          0;
      sin(az) cos(az)
                         0;
                                         -\sin(ay) \ 0 \ \cos(ay)];
      0
                  0
                         1];
                                   Rz = [\cos(az) - \sin(az) 0;
                                         sin(az) cos(az) 0;
Rc1 = Rx*Ry*Rz;
                                                      0
                                         0
                                                            11;
Tc1 = [0;0;5];
M = [Rc1 Tc1];
                                   Rc2c1 = Rx*Ry*Rz;
```

```
T \text{ skew} = [0 - t(3) t(2); t(3)]
TrueR = Rc2c1;
                                  0 - t(1); -t(2) t(1) 0];
Tc2c1 = [3;0;1];
                                 Etrue = T skew*Rc2c1;
TrueT = Tc2c1;
Hc1 = [Rc1 Tc1; 0 0 0 1];
                                 %% Displaying the
Hc2c1 = [Rc2c1 Tc2c1; 0 0 0]
                                  information
11;
                                 disp('u1: Pixel coordinates
Hc2 = Hc2c1*Hc1;
                                  in view 1')
                                 ulinfo = ['Size of ul is '
Rc2 = Hc2(1:3,1:3);
                                 num2str(size(u1,1)) 'x'
Tc2 = Hc2(1:3,4);
                                 num2str(size(u1,2))];
M = [Rc2 Tc2];
                                 disp(ulinfo)
                                 disp('u2: Pixel coordinates
I2 = zeros(L,L);
                                  in view 2')
p2 = K*(M*P W);
                                 u2info = ['Size of u2 is '
                                 num2str(size(u2,1)) 'x'
noise2 = 4*rand(3,NPTS)-2;
                                 num2str(size(u2,2))];
noise2(3,:)=1;
                                 disp(u2info)
p2 = p2 + noise2;
                                 disp('----')
                                 %% Lab#8 Assignment starts
u2(1,:) = p2(1,:)./
                                 here.
p2(3,:);
                                 %% Transform pixel
u2(2,:) = p2(2,:)./
                                 coordinates and construct X
p2(3,:);
                                 matrix using Equations 1 and
u2(3,:) = p2(3,:) ./
p2(3,:);
yxy1=[];xyx1 =[];
                                 X = [];
for i=1:length(u2)
                                 p11 = []; p22 = [];
    x = round(u2(1,i));
                                 % The 8 points are chosen
y=round(u2(2,i));
                                 from the y-z plane
    yxy1=[yxy1; y];xyx1
                                  for ii = 19:-1:1
=[xyx1; x];
                                     pp1 = K \setminus u1(:,ii);
    12(y-2:y+2, x-2:x+2) =
                                    pp2 = K \setminus u2(:,ii);
255;
end
                                     % Create a matrix for the
                                  normalized image
subplot(1,3,3), imshow(I2,
                                  coordinates. Points from
[]), title('View 2',
                                 here
'FontSize', 20);
                                     % will be used to find
                                  the epipoles and epipolar
t = Tc2c1;
                                  lines, and also verify
```

```
verify_point1 =
   % them.
  p11 = [p11; pp1']; p22 =
                                L1'*p11(5,:)';
[p22; pp2'];
                                verify epipole2 = L2'*e2;
                                verify point2 =
   aa = [pp1(1)*pp2;
pp1(2)*pp2; pp1(3)*pp2];
                                L2'*p22(5,:)';
  X = [X; aa'];
                                %% Recover the rotation and
end
                                the translation
rankin = rank(X'*X);
                                ar = pi/2;
%% Estimate E, cure it and
                                Rz1 = [0 -1 0;
check for Essential Matrix
                                       1 0 0;
Characterization
                                        0 0 1];
[U,S,V] = svd(X'*X);
                                Rz2 = [0 \ 1 \ 0;
                                       -1 0 0;
E = V(:,end);
                                        0 0 1];
E = [E(1:3) E(4:6) E(7:9)];
                                T1 1 = U *Rz1*S *U ';
[U,S,V] = svd(E);
                                R1 = U *Rz1'*V ';
S(1,1) = 1; S_(2,2) = 1;
                                T2 2 = U_*Rz2*S_*U_';
S(3,3) = 0;
                                R2 = U *Rz2'*V ';
E = U *S *V ;
                                T1 = [T1 \ 1(3,2); \ T1 \ 1(1,3);
                                T1 1(2,1)];
SO3 U = det(U);
                                T2 = [T2 \ 2(3,2); T2 \ 2(1,3);
SO3_V^- = det(V_);
                                T2 2(2,1)];
%% Find epipoles and
                                %% Compare your results with
epipolar lines
                                ground truth
                                disp('True E =')
e1 = null(E est);
                                disp(Etrue)
e2 = null(E est');
                                disp('Estimated E = ')
                                disp(E est)
L1 = E est'*p22(5,:)';
                                disp('----')
L2 = E est*p11(5,:)';
                                disp('----')
%% Verify epipoles and
                                % R should be exactly
epipolar lines
                                similar, but one of them
verify epipole1 = L1'*e1;
                                only since the other means
```

```
% the case when the camera
                                 hold on
is behind the view
                                 wallx = meshgrid(0:0.1:3);
disp('True R =')
                                 wallz = meshgrid(-3:0.1:0);
disp(TrueR)
                                 wallv =
disp('Estimated R1 & R2 :')
                                 0*ones(size(wallz,1));
disp('R1 est = ')
                                 surf(wallx, wally,
                                 wallz','FaceColor',(1/255)*[
disp(R1)
disp('----
                                 77 137
disp('R2 est = ')
                                 157], 'EdgeColor', 'none')
                                 wallx = meshgrid(0:0.1:3);
disp(R2)
disp('----')
                                 wally = meshgrid(0:0.1:3);
disp('----')
                                 wallz =
                                 zeros(size(wally,1)); %
% T should be scaled version
                                 Generate z data
of True T, since we cannot
                                 surf(wallx, wally',
find a unique T and
                                 wallz, 'FaceColor', (1/255) * [4]
% it is always up to scale
                                 5 162
disp('True T =')
                                 200], 'EdgeColor', 'none')
disp(TrueT)
                                 plot3(P W(1,:),P W(2,:),P W(
disp('Estimated T1 & T2 :')
                                 3,:),'b.','MarkerSize',36);
disp('T1 est = ')
                                 axis equal;
disp(T1)
                                 grid on
disp('----')
                                 axis vis3d;
disp('T2 est = ')
                                 axis([-3 \ 3 \ -3 \ 3 \ -3 \ 3])
disp(T2)
                                 grid on
disp('----')
                                 xlabel('x')
                                 ylabel('y')
%% Post-lab Image Plots
                                 zlabel('z')
I1 = zeros(L, L);
                                 M1 = [R1 T1];
I2 = zeros(L, L);
                                 I1_{\underline{}} = zeros(L,L);
figure;
                                 p1 = K*(M1 *P W);
subplot(1,3,1)
wally = meshgrid(0:0.1:3);
                                 noise1 = 4*rand(3, NPTS)-2;
wallz = meshgrid(-3:0.1:0);
                                 noise1(3,:)=1;
wallx =
                                 p1 = p1 + noise1;
0*ones(size(wallz,1));
surf(wallx, wally,
                                 u1 (1,:) = p1 (1,:) ./
wallz','FaceColor',(1/255)*[
                                 p1_(3,:);
                                 u1 (2,:) = p1 (2,:) ./
97 178
205], 'EdgeColor', 'none')
                                 p1 (3,:);
```

```
u1_(3,:) = p1_(3,:) ./
                                   noise2(3,:)=1;
                                   p2 = p2 + noise2;
p1 (3,:);
yxy = []; xyx = [];
                                   u2 (1,:) = p2 (1,:) ./
for i 1 =1:length(u1)
                                   p2_(3,:);
    x\overline{1} = round(u1 (\overline{1}, i 1));
                                   u2 (2,:) = p2 (2,:) ./
y1_=round(u1_(2,i_1));
                                   p2 (3,:);
    yxy_=[yxy_; y\overline{1}_];xyx
                                   u2^{-}(3,:) = p2_{-}(3,:) ./
=[xyx_{;}x1_{]};
                                   p2 (3,:);
    if x1 > 2 && y1 > 2
    I1 (y\overline{1} -2:y1_+2, x1_-
                                   yxy_1 = []; xyx 1 = [];
                                   for i =1:length(u2)
2:x1 +2) = 255;
                                       x = (round(u2 (1,i)));
    end
                                   y = (round(u2 (2,i)));
end
                                       yxy 1=[yxy 1; y];xyx 1
subplot(1,3,2), imshow(I1,
                                   =[xyx 1; x];
                                       if x > 2 && y > 2
[]), title('View 1',
                                       12 (y -2:y +2, x -
'FontSize',20);
                                   2:x +2) = 255;
M2 = [R2 T2];
                                       end
                                   end
I2 = zeros(L, L);
p2 = K*(M2 *P W);
                                   subplot(1,3,3), imshow(I2_,
                                   []), title('View 2',
noise2 = 4*rand(3,NPTS)-2;
                                   'FontSize', 20);
```