CSCE 643 Multi-View Geometry CV Project V

I. Problem Definition

The 3D reconstruction problem we are going to discuss about in this paper can be defined as follows:

Suppose that we have two views of a real-world scene (e.g., two photos taken from different perspective that contain overlapping scenes), in which a set of points correspondences $\mathbf{x}_i \leftrightarrow \mathbf{x}'_i$ could be found among the two views. The 3D reconstruction task is to find two camera matrices $\mathbf{P}$ and $\mathbf{P}'$, as well as the real-world point set $\mathbf{X}_i$ so that:

$$\forall i, \mathbf{x}_i = \mathbf{P}\mathbf{X}_i, \mathbf{x}'_i = \mathbf{P}'\mathbf{X}_i \tag{1}$$

In this project, we are given two photos taken from different angles and share overlapped scenaries, so the first step is to find out sufficient point correspondences among two photos.

II. Point Correspondences in Original Images

Similar to 2D rectification, in 3D reconstruction, we also need to choose points on parallel lines so that we could bring the image back to affinity by leveraging the properties of vanishing point and infinity plane. However, different from the 2D cases where we only choose points and parallel lines on the same plane in one image and rectify the entire image targeting at it, here in the 3D reconstruction we have to choose point correspondences from two images and on both planes.

With that, we begin our point selection process. As we are required to select at least 20 points, in this paper, we choose to select 10 points on both the plane of wall and plane of floor, respectively. Detailed point correspondences selection we adopted here can be summarized as follows:

- Wall Plane
  - 8 points (4 on each side) for the rectangle formed by the wall that surrounded elevator door.
  - 2 points on the small gap (which is actually the middle line of elevator door that cuts the elevator door in two halfs).
- Floor Plane
  - 8 points (4 on each side) from the cross points of tiles on the floor that formed a rectangle (starting from the foot of two sides of elevator door).
  - 2 points randomly selected outside of the rectangle we selected in the last step.

For better demonstration, we visualize the point correspondences as shown in Figure 1. The numerical expressions of point correspondences we selected can be also found in Section VII.

III. Methodologies

A. Overview of Reconstruction Method

According to the textbook, the 3D reconstruction method from two-view can be summarized as follows:

- Compute the fundamental matrix from point correspondences.
- Compute the camera matrices from the fundamental matrix.
- For each point correspondence $\mathbf{x}_i \leftrightarrow \mathbf{x}'_i$, compute the point in space that projects to these two image points.

The first step regarding the computation of fundamental matrix and camera matrices are not neccessary in some cases while in this paper we assume no priori knowledge about the camera (i.e., the camera is not calibrated), therefore we have to conduct all steps mentioned above.

B. Computing Fundamental Matrix

1) Problem Specification: After we have selected and obtained a set of correspondences $\mathbf{x} \leftrightarrow \mathbf{x}'_i$ in two images as provided in the project, the fundamental matrix $\mathbf{F}$ satisfies the condition:

$$\forall i, \mathbf{x}'_i\mathbf{F}\mathbf{x}_i = 0 \tag{2}$$

As we have already learnt the coordinates of 20 corresponding points, the equation is linear in the (unknown) entries of the matrix $\mathbf{F}$. Moreover, each point correspondence generates one linear equation in the entries of $\mathbf{F}$. If we have at least 8 point correspondences, it is possible to solve linearly for the entries of $\mathbf{F}$ up to scale. In our case, we have 20 correspondences, which is much more than 8 equations, we can find a least-squares solution simply through SVD.

With the definition of $\mathbf{F}$ in the previous discussion, any pair of point correspondences $\mathbf{x}_i \leftrightarrow \mathbf{x}'_i$ in two images can provide a linear equation as shown in . Combining all the linear equations we came up, the unknown matrix $\mathbf{F}$ can be computed. For any specific linear equation provided by a pair of point, we can further expand the equation in Eq. 2 as follows:

$$\begin{split} x'xf_{11} + x'yf_{12} + x'f_{13} + y'xf_{21} + y'yf_{22} \\ + y'f_{23} + xf_{31} + yf_{32} + f_{33} = 0 \end{split} \tag{3}$$

where f_{ij} is the element in ith row and jth column of $\mathbf{F}$.

If we denote by $\mathbf{f}$ the 9-vector made up of the entries of $\mathbf{F}$ in row-major order. Then Eq. 3 can be expressed as a vector inner product as follows:

$$(x'x, x'y, x', y'x, y'y, y', x, y, 1)\mathbf{f} = 0 \tag{4}$$

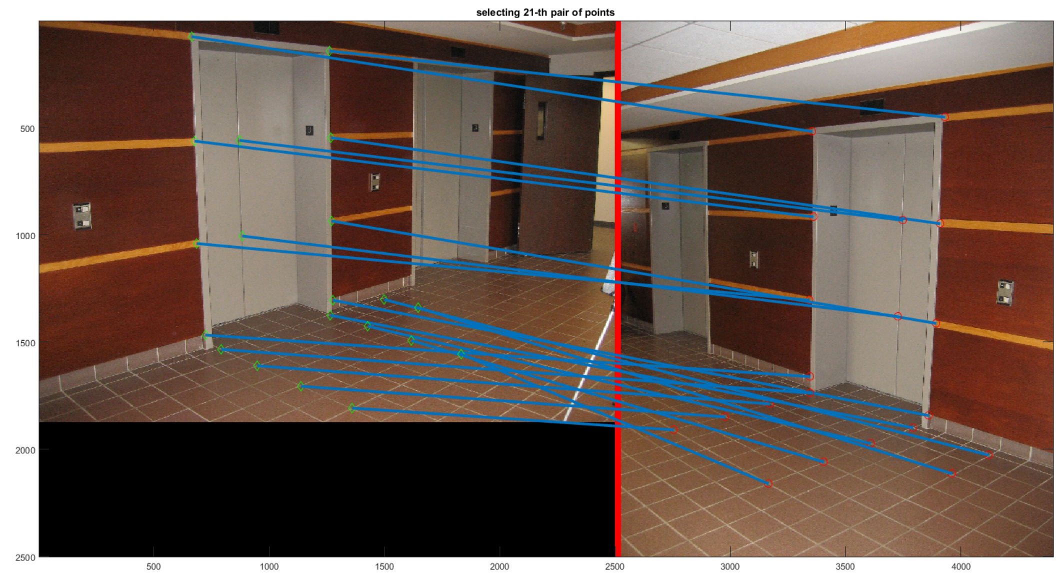


Fig. 1. Visualizing Point Correspondences Selected for 3D Reconstruction

After obtaining n set of point correspondences, we can construct matrix $\mathbf{A}$ for solving $\mathbf{f}$ shown in part I of section VII. Then Eq. 2 can be rewritten into the following form:

$$\mathbf{Af} = \mathbf{0} \tag{5}$$

Be noticed that we can only obtain $\mathbf{f}$ up to scale, and matrix $\mathbf{A}$ must have a rank at most 8 so that the above equation is solvable. If the rank is exactly 8, then the solution is unique (up to scale), and can be found by linear methods.

2) Normalized 8-Point Algorithm: Different from the exact solution case, in reality, the data is not necessarily exact, due to the existence of noises in photos taken from cameras. Under such circumstances, the rank of $\mathbf{A}$ may be greater than 8 and we can only find a least-square solution. In this paper, we adopted the same approaches from the estimation algorithm in DLT for finding the least-square solution of the system. The only difference is that here we have different formations of equation sets now. Recall from the estimation problem that the least-squares solution for $\mathbf{f}$ is the singular vector corresponding to the smallest singular value of $\mathbf{A}$, i.e., the last column of V in the $SVD(A) = UDV^T$. The solution vector $\mathbf{f}$ yielded by SVD minimizes $\|Af\|$ subject to the condition $\|f\| = 1$. The algorithm just described is the essence of *linear solution* part in the 8-point algorithm for computation of the fundamental matrix.

So far, we have been able to solve $\mathbf{f}$ (also $\mathbf{F}$) that minimizes $\mathbf{Af}$ as closer as possible to $\mathbf{0}$ through SVD. Problem still exists in that the solution of $\mathbf{F}$ we have now doesn't necessarily have a rank of 2, which is an important property of the fundamental matrix and is heavily relied for many applications of fundamental matrix. To correct the rank discrepancy in our result, the simplest way is to correct $\mathbf{F}$, and then do another minimization to bring corrected matrix closer to the original $\mathbf{F}$. Specifically, suppose $\mathbf{F} = UDV^T$ is the SVD of $\mathbf{F}$ (the result we got after first SVD), we know that D is a diagonal matrix $diag(r, s, t)$. Then $\mathbf{F}' = Udiag(r, s, 0)V^T$ should minimize $\|\mathbf{F} - \mathbf{F}'\|$. This part of work corrects $\mathbf{F}$ we get from first SVD to be of rank 2 and is known as *constraint enforcement* in normalized 8-point algorithm.

C. Camera Matrices from Fundamental Matrix

Given the fundamental matrix we computed from last step, we can further extract camera matrices $\mathbf{P}, \mathbf{P}'$ for both images based on it (if and only if $\mathbf{P}'^T\mathbf{FP}$ is skew-symmetric) as discussed in Chap. 9 of the textbook. We ignore the inductions for extracting camera matrices as they are provided in the textbook and jump directly to the result we are going to use:

$$\mathbf{P} = \left[\mathbf{I}|\mathbf{0}\right], \mathbf{P}' = [[e']_\times \mathbf{F}|e'] \tag{6}$$

where e' is the epipole such that $e'^T\mathbf{F} = \mathbf{0}$ and assume $\mathbf{P}'$ so defined is a valid rank 3 camera matrix.

D. Triangulation

Given the camera matrices $\mathbf{P}$ and $\mathbf{P}'$, let $\mathbf{x}$ and $\mathbf{x}'$ be two points in the two images that satisfy the epipolar constraint, $\mathbf{x}^T\mathbf{F}\mathbf{x} = 0$. Such a constraint may be interpreted geometrically in terms of the rays in space corresponding to the two image points. More specifically, it means that $\mathbf{x}$ lies on the epipolar line $\mathbf{Fx}$, from which we further learn that the two rays back-projected from points $\mathbf{x}, \mathbf{x}'$ on the image plane actually lie in the same epipolar plane (which is also the plane that contains the two camera centers) and they will intersect in some point. The 3D point $\mathbf{X}$ is projected to the points $\mathbf{x}$ and $\mathbf{x}'$ in the two images, respectively through two cameras, as shown in Figure 2.

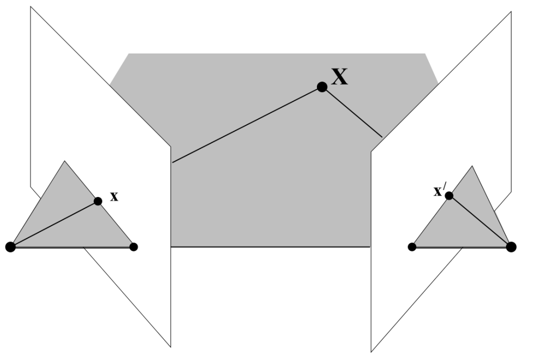


Fig. 2. Two-view Triangulation

1) Errors in Real Measurements: However, the triangulation mentioned above assumed that there are no errors at all in the points on image plane, which is not even remotely possible when we are using an actual camera.

In reality, since there are unevitable errors in the measured points $\mathbf{x}$ and $\mathbf{x}'$, the rays back-projected from these points to their counterparts in real world coordinate are skew. This means that there will not be a point $\mathbf{X}$ that exactly satisfies $\mathbf{x} = \mathbf{PX}, \mathbf{x}' = \mathbf{PX}'$. Moreover, it's not likely that the image points will exactly satisfy the epipolar constraint $\mathbf{x'TFx} = 0$. A figure illustrating the skew problem we encountered due to errors in real measurements is shown in Figure 3.

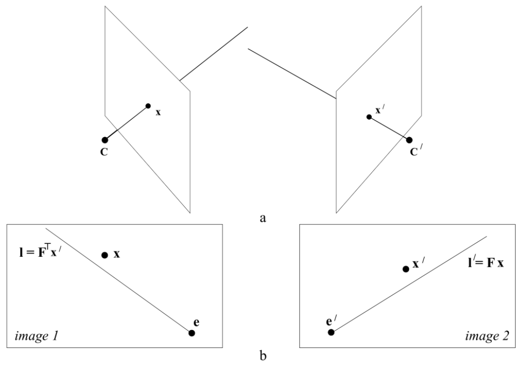


Fig. 3. Skew of Epipolar Line Due to Measurement Errors

2) Linear Triangulation Methods: Assuming that there are some measurement errors, i.e., the estimated point does not exactly satisfy the geometric relations, and is not an optimal estimate. The linear triangulation method is the direct analogue of the DLT method. In each image we have a measurement $\mathbf{x} = \mathbf{PX}, \mathbf{x}' = \mathbf{PX}'$, and these equations can be combined into a form $\mathbf{AX} = \mathbf{0}$, which is an equation linear in $\mathbf{X}$. First the homogeneous scale factor is eliminated by a cross product to give three equations for each image point, of which two are linearly independent. For example for the first image, $\mathbf{x} \times (\mathbf{PX}) = \mathbf{0}$, which can be expanded to obtain the following equations:

$$\begin{aligned} x(\mathbf{p^3}^T\mathbf{x} - (\mathbf{p^1}^T\mathbf{X}) &= 0 \\ y(\mathbf{p^3}^T\mathbf{x}) - (\mathbf{p^2}^T\mathbf{X}) &= 0 \\ x(\mathbf{p^2}^T\mathbf{X}) - y(\mathbf{p^1}^T\mathbf{X}) &= 0 \end{aligned} \tag{7}$$

where $\mathbf{p^i}^T$ are the rows of $\mathbf{P}$. An equation of form $\mathbf{AX} = \mathbf{0}$ can be similarly composed, where:

$$\mathbf{A} = \begin{bmatrix} x\mathbf{p^3}^T - \mathbf{p^1}^T \\ y\mathbf{p^3}^T - \mathbf{p^2}^T \\ x'\mathbf{p'^3}^T - \mathbf{p'^1}^T \\ y'\mathbf{p'^3}^T - \mathbf{p'^2}^T \end{bmatrix} \tag{8}$$

the above two equations have been included from each image, giving a total of four equa- tions in four homogeneous unknowns. This is a redundant set of equations, since the solution is determined only up to scale.

As we have established the equation set, we need to do minimization to find out the closest estimation. Here similar to other estimation problems like when finding fundamental matrix and in DLT, we adopted SVD in this paper for minimizing those left sides of equations to zero. The detailed steps of doing SVD is ignored here for brevity as it's just another repetition of what we have discussed in section III.

E. Back to Affinity

For this part of work, we try to find the affinity transformation that can bring image back to the affined space, as we are literally using the same affinity rectification approach as we did in Project 1, we attached the approach of affinity rectification from Project 1 in the next section for reference.

IV. Rectifying to Affinity

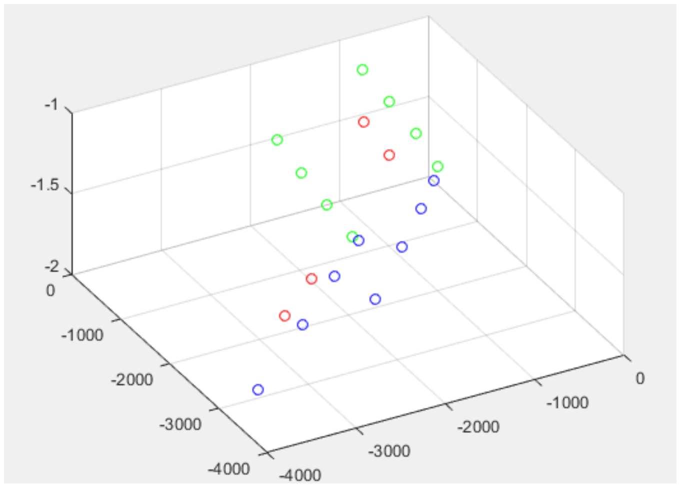
The key of using parallel lines in projective space to recover affine properties from images is the infinite line. In the affinity space, the infinite line is a fixed line $l_\infty = (0, 0, 1)^T$, however a projective transformation might maps l_∞ from the fixed line at infinity to a finite line l on the space after projection. Then, say we have the infinite line $l = (l_1, l_2, l_3)^T$ in a projective space, where $l_3 \neq 0$, and the homography of this current projection $\mathbf{H}$ can be divided as:

$$\mathbf{H} = \mathbf{H}_A\mathbf{H}_P \tag{9}$$

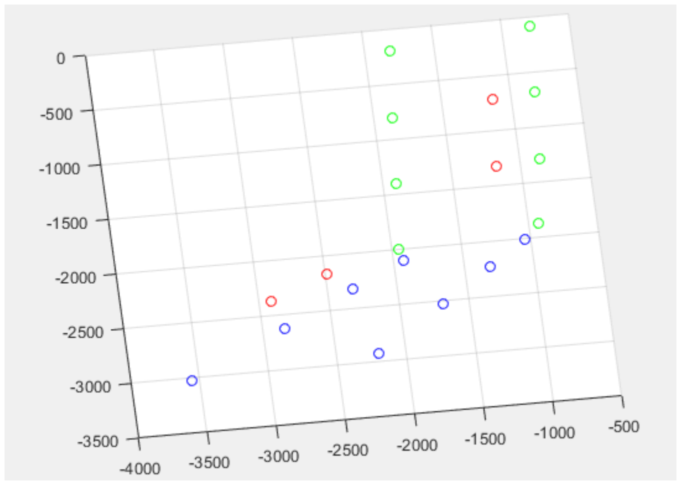
where $\mathbf{H}_A$ is the affine homography and the last matrix $\mathbf{H}_P$ is the homography for transformation from affine space to current projective space:

$$\mathbf{H}_P = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ l_1 & l_2 & l_3 \end{bmatrix} \tag{10}$$

That is to say, the current projective transformation can be decomposed into two parts, one is the transformation to affine space and the other one is the transformation from affinity to current projective space, and the later one can be directly calculated if the infinite line is given.



(a) Reconstruction Results in 3D



(b) Reconstruction Results Projected in 2D Plane

Fig. 4. 3D Reconstruction Results

Now that we figured out the infinite line could help us back to affinity, we can start to work on the details to calculate the infinite line. We know that in Euclidean space, two parallel line will intersect at an ideal point on infinite line, and if we can get two ideal points then we can easily calculate the infinite line as two points determine a line. Intuitively, we can identify two pairs of parallel lines from the distorted picture and calculate two ideal points through them to form the infinite line and then we can get back to affinity based on our discussion above.

Suppose we have four points p_1, p_2, p_3, p_4 and they form a rectangle similar to what we have in question 1, through those we can simply get two pairs of parallel lines:

$$\begin{aligned} \vec{l_1} &= p_1 \times p_2 \\ \vec{l_2} &= p_3 \times p_4 \\ \vec{m_1} &= p_1 \times p_3 \\ \vec{m_2} &= p_2 \times p_4 \end{aligned} \tag{11}$$

in which we have $l_1 /\!/ l_2$ and $m_1 /\!/ m_2$. Through those pairs of parallel lines, we can further compute two points at the infinite line as follows:

$$\begin{aligned} v_1 &= \vec{l_1} \times \vec{l_2} \\ v_2 &= \vec{m_1} \times \vec{m_2} \end{aligned} \tag{12}$$

And finally we can acquire the line at infinity $\vec{l_\infty} = (l_1, l_2, l_3)$ which can be calculated by:

$$\vec{l_\infty} = v_1 \times v_2 \tag{13}$$

According to our discussion above, now we can form a new matrix same as given in equation 10 based on the infinite line, and the new $\mathbf{H}_P$ can handle the transformation between the picture space and affinity.

To summarize, the following steps are needed when we are doing the affine rectification:

1) Find two physically parallel line pairs in the picture plane. (here we can simply use those points we picked for the rectangle in our first question, since those four apexes naturally provide two parallel line pairs)
2) Through those parallel lines we got, solve two intercepts of those parallel lines in picture plane, which are ideal points that should lie on the line of infinity in the world plane.
3) From the two ideal points we can now get line of infinity in the picture plane.
4) Since we know the coordinates of infinite line in the world plane, we can now solve the homography transformation from picture plane to affinity.

V. Graphical Illustration of Reconstructed Planes

The reconstructed planes of wall and floor are shown in Figure 4. For clarity of demonstration, we have marked the points unused in reconstruction (two points on elevator gap and other two outside of the rectangle on the floor plane) in red color, points used for reconstruction on the wall plane in green and points used for reconstruction on the floor plane in blue. With little tolerance of skews on parallel lines, we can apparently see that the parallelism in both planes is recovered and the four points not used for reconstruction lie in proper places. Be noticed between wall plane and floor plane shown in Figure 4 are not exactly perpendicular to each other, which is okay since after applying our reconstruction approach they are actually in the affined space, though the parallelism property is recovered, the angles might not be correct.

VI. Discussions

A. Point Selection

The first and foremost lessons I learnt from the point selection process is that we need to make sure the correspondence between points in a pair. Previously in the panorama project, I have already made a mistake due to forgetting the correct point selection sequence thereby causing mismatches in two point sets, which led to a long time of debugging. This time, to make sure that the point pairs are selected correctly, I referred to and modified a MATLAB code segment for selecting point pairs. This time when using MATLAB to choose point pairs, I contatenated two images together (one on the left side and the other on the right side, with a bound in between them for discrimination), and then use code to control the point selection sequence. For example, if I selected point on the left side (1st image) last time, the program will enforce me to select point from right side this time. After any point pair is choose, it will also draw a line between them for us to clearly see the point correspondence relations which is especially useful when we are selecting a lot of correspondences.

B. *Fundamental Matrix*

Based on the definition of fundamental matrix, it can actually be regarded as a combined matrix that contains two transformation, one from the image in the first view to a standard space, the other from the standard space to the second view. For simplicity, we can also regard the first view as the standard space, so that the transformation from first view to standard is a simple diagonal matrix and the transformation from standard to second view is the same transformation from first to second. This way, the fundamental matrix actually "degenerated" to one matrix that contains only one transformation. I hope my interpretation is correct in this part though...

C. *Measurement Error and Minimization*

The influence of measurement error and importance of minimization have been highlighted for numerous times in the class. However, in this project, I felt them to be underscored more strongly than ever. Due to measurement errors:

- the fundamental matrix cannot be exactly solved.
- there are skews in the back-projected ray from points in image space

And the solution to such problems is now really familiar to me, that is to use minimization approaches. A common step is first to build numerical relations based on the relationship we can infer from some properties, then set up linear equations (ideally with 0 at right side). Last, we can build up matrixes and use SVD to do the minimization.

VII. Numerical Results

A. A Matrix for Fundamental Matrix

The $\mathbf{A}$ matrix we constructed through point pairs when solving $\mathbf{F}$:

$$\mathbf{A} = \begin{bmatrix} x_1' x_1 & x_1' y_1 & x_1' & y_1' x_1 & y_1' y_1 & y_1' & x_1 & y_1 & 1 \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ x_n' x_n & x_n' y_n & x_n' & y_n' x_n & y_n' y_n & y_n' & x_n & y_n & 1 \end{bmatrix} \tag{14}$$

B. Numerical Representation of Point Correspondences

The numerical expression of points choosed in the first image is shown as follows:

$$\begin{pmatrix} 664.3 & 75.34 \\ 676.2 & 562.6 \\ 685.0 & 1040.0 \\ 723.5 & 1467.0 \\ 1262.0 & 142.2 \\ 1268.0 & 546.7 \\ 1271.0 & 935.2 \\ 1276.0 & 1305.0 \\ 883.2 & 1005.0 \\ 871.3 & 559.4 \\ 791.5 & 1534.0 \\ 948.2 & 1610.0 \\ 1137.0 & 1706.0 \\ 1359.0 & 1808.0 \\ 1265.0 & 1375.0 \\ 1427.0 & 1426.0 \\ 1616.0 & 1493.0 \\ 1832.0 & 1556.0 \\ 1498.0 & 1301.0 \\ 1646.0 & 1340.0 \end{pmatrix} \tag{15}$$

The numerical expressions of point correspondances in the second image are:

$$\begin{pmatrix} 830.2 & 518.0 \\ 836.1 & 916.1 \\ 818.4 & 1305.0 \\ 821.3 & 1661.0 \\ 1407.0 & 451.1 \\ 1386.0 & 948.0 \\ 1368.0 & 1413.0 \\ 1336.0 & 1843.0 \\ 1203.0 & 1381.0 \\ 1223.0 & 932.0 \\ 821.3 & 1738.0 \\ 646.9 & 1789.0 \\ 451.7 & 1849.0 \\ 229.9 & 1910.0 \\ 1268.0 & 1900.0 \\ 1085.0 & 1973.0 \\ 880.5 & 2059.0 \\ 641.0 & 2161.0 \\ 1596.0 & 2028.0 \\ 1436.0 & 2114.0 \end{pmatrix} \tag{16}$$

C. Intermediate Results

The fundamental matrix $\mathbf{F}$:

$$\mathbf{F} = \begin{pmatrix} 6.524 \cdot 10^{-9} & 2.236 \cdot 10^{-7} & -0.0001097 \\ 2.134 \cdot 10^{-7} & -5.942 \cdot 10^{-9} & -0.0008351 \\ -0.0001682 & 0.0003871 & 0.5147 \end{pmatrix} \tag{17}$$

The plane at infinity $\mathbf{P}_{inf}$ is:

$$\mathbf{P}_{inf} = \begin{pmatrix} 0.0002381 \\ 3.178 \cdot 10^{-5} \\ -1.0 \\ -9.22 \cdot 10^{-17} \end{pmatrix} \tag{18}$$

The homography matrix $\mathbf{H}$ used for transformation to affinity:

$$\mathbf{H} = \begin{pmatrix} 1.0 & 0 & 0 & 0 \\ 0 & 1.0 & 0 & 0 \\ 0 & 0 & 1.0 & 0 \\ 0.0002381 & 3.178 \cdot 10^{-5} & -1.0 & -9.22 \cdot 10^{-17} \end{pmatrix} \tag{19}$$

Camera matrix $\mathbf{P}$ for the first camera (that produced the first figure in our paper):

$$\mathbf{P} = \begin{pmatrix} 1.0 & 0 & 0 & 0 \\ 0 & 1.0 & 0 & 0 \\ 0 & 0 & 1.0 & 0 \end{pmatrix} \tag{20}$$

Camera matrix $\mathbf{P}'$ for the second camera that produced the second figure:

$$\mathbf{P}' = \begin{pmatrix} -7.425 \cdot 10^{-5} & 0.0001709 & 0.2272 & -0.8973 \\ -0.0001509 & 0.0003474 & 0.4618 & 0.4414 \\ -1.943 \cdot 10^{-7} & -9.337 \cdot 10^{-8} & 0.0007978 & 0.000525 \end{pmatrix} \tag{21}$$

Numerical representation of the 3D wall plane $\mathbf{P}_{wall}$:

$$\mathbf{P}_{wall} = \begin{pmatrix} -0.0001684 & -2.247 \cdot 10^{-5} & 0.7071 & 0.7071 \end{pmatrix} \tag{22}$$

Numerical representation of the 3D floor plane $\mathbf{P}_{floor}$:

$$\mathbf{P}_{floor} = \begin{pmatrix} -0.0001684 & -2.247 \cdot 10^{-5} & 0.7071 & 0.7071 \end{pmatrix} \tag{23}$$

Appendix

A. *Fundamental Matrix Estimation*

```
function [F] = estimate_Fundamental_Matrix(x1,x2,T1,T2)
 n_correspondences = size(x1,1);
 % Normalize the correspondences
 x1_n = (T1*x1')';
 x2_n = (T2*x2')';
 % Stacking up the equations for all the correspondences
 A = zeros(n_correspondences,9);
 for i=1:n_correspondences
 A(i,:) = [x2_n(i,1)*x1_n(i,1) x2_n(i,1)*x1_n(i,2) x2_n(i,1) ...
 x2_n(i,2)*x1_n(i,1) x2_n(i,2)*x1_n(i,2) x2_n(i,2) ...
 x1_n(i,1) x1_n(i,2) 1];
 end
 % Solving for initial estimate of Fundamental Matrix using Linear Least
 % squares by finding SVD of A and choosing the last eigenvector
 [U_lls D_lls V_lls] = svd(A);
 f = V_lls(:,end);
 F_unconditioned = reshape(f,3,3)';
 % Conditioning the Fundamental Matrix to enforce the rank constraint
 [U D V] = svd(F_unconditioned)
 D(end,end) = 0;
 F_conditioned = U*D*V';
 % Denormalizing the Fundamental Matrix
 F = T2'*F_conditioned*T1;
end
```

B. *Point Selection with GUI*

```
close all
clear all
clc

im1=imread('pics/pic1.jpg');
im2=imread('pics/pic2.jpg');
seperate_line_width=25;
seperate_line=zeros(2500,seperate_line_width,3);
seperate_line(:,:,1)=255;
im1(1876:2500, 1:2500) = zeros((2500 - 1875), 2500);
im_com=[im1 seperate_line im2];

figure_handler=figure(1);
image(im_com)
hold on
set(gcf,'Units','centimeters', 'Position',[0 3.5 50 15],'Renderer','painters','color'
    ,'white');
% all odd points we selected are supposed to be in the first figure
pts_img_1 = [];
% all even points we selected are supposed to be in the second figure
pts_img_2 = [];
if exist('points1.mat','file')>0
    load('points1.mat');
    for j=1:length(x1)
        im1(floor(y1(j))-10:floor(y1(j))+10,floor(x1(j))-10:floor(x1(j))+10,1:2)=255;
        im1(floor(y1(j))-10:floor(y1(j))+10,floor(x1(j))-10:floor(x1(j))+10,3)=0;
        im2(floor(y2(j))-10:floor(y2(j))+10,floor(x2(j))-10:floor(x2(j))+10,1)=0;
        im2(floor(y2(j))-10:floor(y2(j))+10,floor(x2(j))-10:floor(x2(j))+10,2)=250;
        im2(floor(y2(j))-10:floor(y2(j))+10,floor(x2(j))-10:floor(x2(j))+10,3)=150;
        A(2*j-1:2*j,:)=[   0,     0,     0, -x1(j), -y1(j), -1,  y2(j)*x1(j),  y2(j)*y1(
            j),  y2(j);
                         x1(j), y1(j), 1,    0,       0,      0, -x2(j)*x1(j), -x2(j)*y1(
                             j), -x2(j)];
        plot(x1(j),y1(j),'gd','MarkerSize',8);
        plot(x2(j)+(2500+seperate_line_width),y2(j),'ro','MarkerSize',8);
        line([x1(j),x2(j)+(2500+seperate_line_width)],[y1(j),y2(j)],'linewidth',3);
    end
else
    i=0;
    while i<40
        title( ['selecting the ',num2str(floor(i/2)+1),'-th pair of points'])
        j=floor(i/2)+1;
        if mod(i,2)==0
            h=rectangle('Position',[0.01 0.01 25 18.75]*100,'EdgeColor','g','
                LineWidth',4);
            [x1(j),y1(j)]=ginput(1);
            if x1(j)<=2500
                plot(x1(j),y1(j),'gd','MarkerSize',8)
                pts_img_1 = [pts_img_1;
                    x1(j), y1(j)]
                i=i+1;
            else
                disp('please select point inside the green box');
            end
        else
            h=rectangle('Position',[25+seperate_line_width/100 0.01 18.74 24.99]*100,
                'EdgeColor','g','LineWidth',4);
            [x2(j),y2(j)]=ginput(1);
            x2(j)=x2(j)-(2500+seperate_line_width);
            if x2(j)>=0
                plot(x2(j)+(2500+seperate_line_width),y2(j),'ro','MarkerSize',8);
                line([x1(j),x2(j)+(2500+seperate_line_width)],[y1(j),y2(j)],'
                    linewidth',3);
                pts_img_2 = [pts_img_2;
                    x2(j), y2(j)]
                i=i+1;
            else
                disp('please select point inside the green box');
            end
        end
    % im_com=[im1 seperate_line im2];
        delete(h);
        %im_com=[im1 seperate_line im2];
        %image(im_com)
        title( ['selecting ',num2str(floor(i/2)+1),'-th pair of points'])
    end
end

%save('points.mat', 'pts_img_1', 'pts_img_2');
```

C. Main Function for 3D Reconstruction

```
% triangulation based approach
% vanish point in two figures, compute 3D coordinate through triangulation
% find 3 vanishing points to determine a infinity plane
close all; clear all; clc;

% load point correspondences into variable pts_img_1 and pts_img_2
load('points.mat');
% establish the normalization transformation
T = [2/2500,0,-1;
    0,2/1875,-1;
    0,0,1];
T2=[2/1875,0,-1;
     0,2/2500,-1;
     0,0,1];
% do normalization for both iamges
norm_img1=(T*[pts_img_1';ones(1,20)])';
norm_img2=(T2*[pts_img_2';ones(1,20)])';

% linear solution
% build A matrix for computing fundamental
for i=1:20
     A(i,:)=[norm_img2(i,1)*pts_img_1(i,1),norm_img2(i,1)*norm_img1(i,2),norm_img2(i
         ,1),norm_img2(i,2)*norm_img1(i,1),norm_img2(i,2)*norm_img1(i,2),norm_img2(i,2)
         ,norm_img1(i,1),norm_img1(i,2),1];
end
% SVD to obtain least square solution
[U,S,V]=svd(A);
null(A');
F=V(:,9);
F=(reshape(F,[3,3]))';
% constraint enforcement
% do svd on F=UDV^T
% D = s = diag(r, s, t)
[u,s,v]=svd(F);
% construct diag(r, s, 0)
s(3,3)=0;
% F' = Udiag(r, s, 0)V^T
F2=u*s*v';
% denormalize
F=T2'*F2*T;
% or use approaches from purdue
%[F] = estimate_Fundamental_Matrix(x1,x2,T,T2);

% epipole
e_2=null(F');
% [e']_x
skew_e_2=[0,-e_2(3),e_2(2);e_2(3),0,-e_2(1);-e_2(2),e_2(1),0];
% skew'
% camera matrix P' for figure 2
% [[e']_x F | e']
P_2=[skew_e_2*F,e_2];
% just I
% camera matrix P for figure 1
P_1=[1,0,0,0;0,1,0,0;0,0,1,0];

% estimate the actual 3d points
% by minimizing Ax through SVD
for i=1:20
```

```
    A_X(1,:)=pts_img_1(i,1)*P_1(3,:)-P_1(1,:);
    A_X(2,:)=pts_img_1(i,2)*P_1(3,:)-P_1(2,:);
    A_X(3,:)=pts_img_2(i,1)*P_2(3,:)-P_2(1,:);
    A_X(4,:)=pts_img_2(i,2)*P_2(3,:)-P_2(2,:);
    [u2,s2,v2]=svd(A_X);
    X(:,i)=v2(:,4)';
end

% compute 3 vanishing point in the actual 3d space
% 1st vanishing point in 3d space
line1_1=cross([pts_img_1(1,:),1],[pts_img_1(4,:),1]);
line1_2=cross([pts_img_1(5,:),1],[pts_img_1(8,:),1]);
vanish1_1=cross(line1_1,line1_2);
vanish2_1=F*vanish1_1';
line2_1=cross([pts_img_2(1,:),1],[pts_img_2(4,:),1]);
vanish3_1=cross(vanish2_1,line2_1);
A_matrix(1,:)=vanish1_1(1)/vanish1_1(3)*P_1(3,:)-P_1(1,:);
A_matrix(2,:)=vanish1_1(2)/vanish1_1(3)*P_1(3,:)-P_1(2,:);
A_matrix(3,:)=vanish3_1(1)/vanish3_1(3)*P_2(3,:)-P_2(1,:);
A_matrix(4,:)=vanish3_1(2)/vanish3_1(3)*P_2(3,:)-P_2(2,:);
% 3d coordinates for vanish point
vanishPt3d_1=null(A_matrix);
% 2nd vanish point in 3d space
line1_3=cross([pts_img_1(1,:),1],[pts_img_1(5,:),1]);
line1_4=cross([pts_img_1(4,:),1],[pts_img_1(8,:),1]);
vanish1_2=cross(line1_3,line1_4);
vanish2_2=F*vanish1_2';
line2_2=cross([pts_img_2(1,:),1],[pts_img_2(5,:),1]);
vanish3_2=cross(vanish2_2,line2_2);
A_matrix(1,:)=vanish1_2(1)/vanish1_2(3)*P_1(3,:)-P_1(1,:);
A_matrix(2,:)=vanish1_2(2)/vanish1_2(3)*P_1(3,:)-P_1(2,:);
A_matrix(3,:)=vanish3_2(1)/vanish3_2(3)*P_2(3,:)-P_2(1,:);
A_matrix(4,:)=vanish3_2(2)/vanish3_2(3)*P_2(3,:)-P_2(2,:);
% 3d coordinates for vanish point
vanishPt3d_2=null(A_matrix);
% 3rd vanish point in 3d space
line1_5=cross([pts_img_1(11,:),1],[pts_img_1(14,:),1]);
line1_6=cross([pts_img_1(15,:),1],[pts_img_1(18,:),1]);
vanish1_3=cross(line1_3,line1_4);
vanish2_3=F*vanish1_3';
line2_3=cross([pts_img_2(11,:),1],[pts_img_2(14,:),1]);
vanish3_3=cross(vanish2_3,line2_3);
A_matrix(1,:)=vanish1_3(1)/vanish1_3(3)*P_1(3,:)-P_1(1,:);
A_matrix(2,:)=vanish1_3(2)/vanish1_3(3)*P_1(3,:)-P_1(2,:);
A_matrix(3,:)=vanish3_3(1)/vanish3_3(3)*P_2(3,:)-P_2(1,:);
A_matrix(4,:)=vanish3_3(2)/vanish3_3(3)*P_2(3,:)-P_2(2,:);
% 3d coordinates for vanish point
vanishPt3d_3=null(A_matrix);

% infinity plane from 3 vanishing point
inf_plane=null([vanishPt3d_1';vanishPt3d_2';vanishPt3d_3']);
% build the homography used for affinity rectification
H=[1,0,0,0;0,1,0,0;0,0,1,0;inf_plane'];
% apply homography on estimated 3d pts
X = H * X;
for i = 1 : 4
```

```
    X(i, :) = X(i, :) ./ X(4, :);
end

% scatter the pts on the wall using green markers
scatter3(X(1, 1:8), X(2, 1:8), X(3, 1:8), 'g')
hold on;
% scatter the pts on the wall (but not used for reconstruction) using red
% markers
scatter3(X(1, 9 : 10), X(2, 9 : 10), X(3, 9 : 10), 'r')
hold on;
% scatter the pts on the floor (that are not used for reconstruction) using
% red markers
scatter3(X(1, 19 : 20),X(2, 19 : 20),X(3, 19 : 20), 'r')
hold on;
% scatter the pts on the floor (that are used for recons) using blue
% markers
scatter3(X(1, 11 : 18),X(2, 11 : 18),X(3, 11 : 18), 'b')
hold on;
% compute the numerical representation of the two planes
plane1=null([X(:, 1)';
    X(:, 4)';
    X(:, 5)']);
plane2=null([X(:, 11)';
    X(:, 14)';
    X(:, 15)']);
plane1 = plane1';
plane2 = plane2';
```