Assignment 1

1 Projection Model, Homogeneous coordinates

1. (a) * Suppose you have two parallel lines in 3-D space, one passing through the point$(100, 100, 1000)$, the other through $(200, 200, 1100)$. The lines are parallel to the vector $(1, 2, 1)$. The lines are observed by a unit focal length camera at the origin (i.e. the camera reference frame and the world reference frame are identical). All coordinates are in camera coordinates. What is their point of intersection in the image? (Hint: the point of infinity along ℓ_i in 3-D space is given by $\lim_{\lambda_i \to \infty} \ell_i = \begin{pmatrix} X_i \\ Y_i \\ Z_i \end{pmatrix} + \lambda_i \begin{pmatrix} a \\ b \\ c \end{pmatrix}$, where $\begin{pmatrix} X_i \\ Y_i \\ Z_i \end{pmatrix}$ and $\begin{pmatrix} a \\ b \\ c \end{pmatrix}$ are a point on the line and its direction respectively)

Answer:

Denote the two parallel lines as follows:

$$\ell_a = \begin{pmatrix} 100 \\ 100 \\ 1000 \end{pmatrix} + \lambda_a \begin{pmatrix} 1 \\ 2 \\ 1 \end{pmatrix}, \ell_b = \begin{pmatrix} 200 \\ 200 \\ 1100 \end{pmatrix} + \lambda_b \begin{pmatrix} 1 \\ 2 \\ 1 \end{pmatrix}$$

Set $\lambda_a = 100$ and $\lambda_a = 200$ respectively to get two points on ℓ_a:

$$P_{1a} = \begin{pmatrix} 200 \\ 300 \\ 1100 \end{pmatrix}, P_{2a} = \begin{pmatrix} 300 \\ 500 \\ 1200 \end{pmatrix}$$

Using the linear mapping of the perspective projection:

$$\begin{pmatrix} x \\ y \\ w \end{pmatrix} = \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{pmatrix} X_c \\ Y_c \\ Z_c \\ 1 \end{pmatrix} \tag{1}$$

The corresponding points (in $\mathcal{P}^2$) of $\mathbf{P}_{1a}$ and $\mathbf{P}_{2a}$ on the image are:

$$P_{1a}^{'} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{pmatrix} 200 \\ 300 \\ 1100 \\ 1 \end{pmatrix} = \begin{pmatrix} 200 \\ 300 \\ 1100 \end{pmatrix}$$

$$P_{2a}^{'} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{pmatrix} 300 \\ 500 \\ 1200 \\ 1 \end{pmatrix} = \begin{pmatrix} 300 \\ 500 \\ 1200 \end{pmatrix}$$

The corresponding line (in $\mathcal{P}^2$) of ℓ_a on the image are formed by two points above:

$$\ell_a^{'} = P_{1a}^{'} \times P_{2a}^{'} = \begin{pmatrix} -190000 \\ 90000 \\ 10000 \end{pmatrix}$$

Similarly, the corresponding line (in $\mathcal{P}^2$) of ℓ_b on the image can be obtained:

$$\ell_b^{'} = \begin{pmatrix} -200000 \\ 90000 \\ 20000 \end{pmatrix}$$

Therefore, the intersection point (in $\mathcal{P}^2$) of the two parallel lines (in $\mathcal{N}^3$) on the image is:

$$P = \ell_a^{'} \times \ell_b^{'} = \begin{pmatrix} 9 \times 10^8 \\ 1.8 \times 10^9 \\ 9 \times 10^8 \end{pmatrix}$$

Converting back to $\mathcal{N}^2$, $P = \begin{pmatrix} 1 \\ 2 \end{pmatrix}$ is the intersection point on the image.

(b) Given the 3D coordinates of several corresponding points P_i and $P_i^{'}$ in two views, you are required to find the 3D rotation $\mathbf{R}$ and translation $\mathbf{T}$ that relate the two views ($P_i^{'} = \mathbf{R}P_i + \mathbf{T}$). Formulate a linear least squares algorithm (of the form $\mathbf{Ax} = \mathbf{b}$) that ignores the orthogonality constraint associated with $\mathbf{R}$ (that is, it is okay if the solution for $\mathbf{R}$ returned by your formulation is not orthogonal). State the entries of the matrix $\mathbf{A}$, and the vectors $\mathbf{x}$ and $\mathbf{b}$.

Answer:

Denote $P_i = \begin{pmatrix} x_i \\ y_i \\ z_i \end{pmatrix}$, $P_i^{'} = \begin{pmatrix} x_i^{'} \\ y_i^{'} \\ z_i^{'} \end{pmatrix}$, $\mathbf{R} = \begin{bmatrix} R_{11} & R_{12} & R_{13} \\ R_{21} & R_{22} & R_{23} \\ R_{31} & R_{32} & R_{33} \end{bmatrix}$, $\mathbf{T} = \begin{pmatrix} t_1 \\ t_2 \\ t_3 \end{pmatrix}$.

The relationship between points in two views is determined by $P_i^{'} = \mathbf{R}P_i + \mathbf{T}$, which can also be written as a linear mapping in the homogeneous coordinate:

$$\begin{aligned} \begin{pmatrix} x_i^{'} \\ y_i^{'} \\ z_i^{'} \\ 1 \end{pmatrix} &= \begin{bmatrix} \mathbf{R} & \mathbf{T} \\ \mathbf{0}^T & 1 \end{bmatrix} \begin{pmatrix} x_i \\ y_i \\ z_i \\ 1 \end{pmatrix} \\ &= \begin{bmatrix} R_{11} & R_{12} & R_{13} & t_1 \\ R_{21} & R_{22} & R_{23} & t_2 \\ R_{31} & R_{32} & R_{33} & t_3 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{pmatrix} x_i \\ y_i \\ z_i \\ 1 \end{pmatrix} \end{aligned} \tag{2}$$

Expanding Eq. (2), three independent equations can be obtained:

$$\begin{aligned} x_i^{'} &= R_{11}x_i + R_{12}y_i + R_{13}z_i + t_1 + \cdots, \\ y_i^{'} &= R_{21}x_i + R_{22}y_i + R_{23}z_i + t_2 + \cdots, \\ z_i^{'} &= R_{31}x_i + R_{32}y_i + R_{33}z_i + t_3 + \cdots, \end{aligned}$$

where "$\cdots$" means padding zeros to complete the expressions of the form $r^T x \in \mathcal{R}$ ($r, x \in \mathcal{R}^{12}$).

Hence, in the linear least squares formulation, the unknown $\mathbf{x} = \begin{pmatrix} R_{11} \\ R_{12} \\ R_{13} \\ R_{21} \\ \vdots \\ R_{33} \\ t_1 \\ t_2 \\ t_3 \end{pmatrix} \in \mathcal{R}^{12}$.

There are three equations to impose constraints to solve for different parts of entries of $\mathbf{x}$:

$$\mathbf{A}_1\mathbf{x} = \mathbf{b}_1,\ \mathbf{A}_2\mathbf{x} = \mathbf{b}_2,\ \mathbf{A}_3\mathbf{x} = \mathbf{b}_3,\ \text{where}$$

$$\mathbf{A}_1 = \begin{bmatrix} | & | & | & | & & | & | & | & | \\ x_i & y_i & z_i & 0 & \cdots & 0 & 1 & 0 & 0 \\ | & | & | & | & & | & | & | & | \end{bmatrix},\ \mathbf{b}_1 = \begin{pmatrix} | \\ x_i^{'} \\ | \end{pmatrix}$$

$$\mathbf{A}_2 = \begin{bmatrix} | & | & | & | & | & | & | & & | & | & | \\ 0 & 0 & 0 & x_i & y_i & z_i & 0 & \cdots & 0 & 1 & 0 \\ | & | & | & | & | & | & | & & | & | & | \end{bmatrix},\ \mathbf{b}_2 = \begin{pmatrix} | \\ y_i^{'} \\ | \end{pmatrix}$$

$$\mathbf{A}_3 = \begin{bmatrix} | & & | & | & | & | & | & | & | \\ 0 & \cdots & 0 & x_i & y_i & z_i & 0 & 0 & 1 \\ | & & | & | & | & | & | & | & | \end{bmatrix},\ \mathbf{b}_3 = \begin{pmatrix} | \\ z_i^{'} \\ | \end{pmatrix}$$

To be more compact, we could also write them as one equation:

$$\mathbf{A}\mathbf{x} = \mathbf{b},\ \text{where}$$

$$\mathbf{A} = \begin{bmatrix} \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ x_i & y_i & z_i & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & x_i & y_i & z_i & 0 & 0 & 0 & 0 & 1 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & 0 & 0 & x_i & y_i & z_i & 0 & 0 & 1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \end{bmatrix},\ \mathbf{b} = \begin{pmatrix} \vdots \\ x_i^{'} \\ \vdots \\ \vdots \\ y_i^{'} \\ \vdots \\ \vdots \\ z_i^{'} \\ \vdots \end{pmatrix}$$

(c) You are now given two sets of corresponding point clouds P_i and $P_i^{'}$, in the files `pts.txt` and `pts_prime.txt` respectively. Each row is a 3D coordinate, possibly contaminated with some noise. Write a Matlab routine to estimate the optimal values of $\mathbf{R}$ and $\mathbf{T}$ in the least squares sense via SVD, using the formulation in (b). Compute the determinant of $\mathbf{R}$ using the Matlab function "det", and comment on the resulting values.

Answer:
My Matlab commands run as follows:

```
P = importdata('pts.txt');
P_prime = importdata('pts_prime.txt');
pad = zeros([200, 3]);
A = [P;pad;pad];
A = [A [pad;P;pad]];
A = [A [pad;pad;P]];
last_cols = [ones([200,1]) zeros([200,1]) zeros([200,1]); [zeros([200,1]) ...
    ones([200,1]) zeros([200,1])]; [zeros([200,1]) zeros([200,1]) ones([200,1])]];
```

```
A = [A last_cols];
b = [P_prime(:,1); P_prime(:,2); P_prime(:,3)];
[U,S,V]=svd(A);
s=diag(S);
disp('singular values:'), disp(s')
bt=U'*b;
y=bt(1:12)./s;
x=V*y;
disp('x using SVD, all singular values:'), disp(x')
disp('||Ax-b||:'), disp(norm(A*x-b))
R = reshape(x(1:9), [3,3])';
det(R)
```

For the results, the unknown is solved to be:

$$x^T = (0.9782, -0.0084, 0.0094, 0.0202, -0.0046, 0.9979, -0.0016, \\ -0.9940, -0.0007, -0.0365, -4.0009, 4.0021)$$

using SVD in Matlab. And the residual $||Ax - b|| = 3.5757$. The rotation matrix

$$R = \begin{bmatrix} 0.9782 & -0.0084 & 0.0094 \\ 0.0202 & -0.0046 & 0.9979 \\ -0.0016 & -0.9940 & -0.0007 \end{bmatrix}$$

The determinant $det(R) = 0.9701$, which is close to 1, showing that the resulted rotation matrix is reasonable since it should not change rigid body's volume. However, as we can see it is a bit off due to som noises.

(d) If the world homogeneous coordinates are (X_w, Y_w, Z_w, W_w), the image plane homogeneous coordinates are (u, v, w), and they are related by: $\begin{pmatrix} u \\ v \\ w \end{pmatrix} \sim M \begin{pmatrix} X_w \\ Y_w \\ Z_w \\ W_w \end{pmatrix}$.

i. Find the 3×4 matrix M as a function of α, h, according to Figure 1 below (in the left diagram, the axis X and X_w are pointing directly out of the paper). Assume that the only intrinsic camera parameter is the focal length f (given in pixel unit). Use $P_c = RP_w + t$ to relate the camera coordinates P_c and the world coordinates P_w. R is the orientation of the world with respect to the camera; t is the world origin expressed in the camera frame.

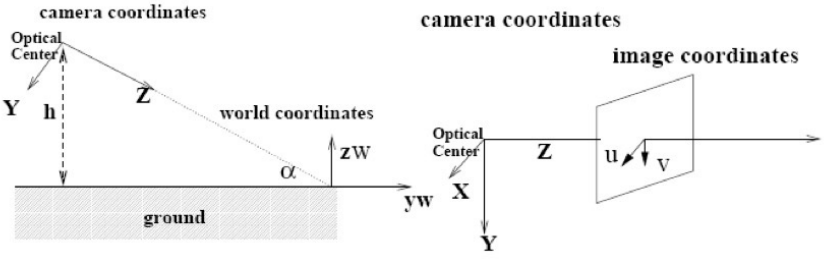


Figure 1: Coordinate System

Answer:
First of all, find the rotation matrix R that maps points in world coordinate to the same points in camera coordinate. Since the rotation is about x axis, it can be described by the matrix (θ is defined using right-hand rule):

$$\begin{aligned} R_x(\theta) &= \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta \\ 0 & \sin\theta & \cos\theta \end{bmatrix} \\ &= \begin{bmatrix} 1 & 0 & 0 \\ 0 & -\sin\alpha & -\cos\alpha \\ 0 & \cos\alpha & -\sin\alpha \end{bmatrix}, \ \theta = \frac{\pi}{2} + \alpha \end{aligned} \tag{3}$$

Next, the translation vector is defined by the world origin expressed in the camera frame:

$$t = \begin{pmatrix} 0 \\ 0 \\ \frac{h}{\sin\alpha} \end{pmatrix}$$

Composite the Euclidean transformation described in Eq. (2) and the perspective projection described in Eq. (1):

$$\begin{aligned} M &= \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & -\sin\alpha & -\cos\alpha & 0 \\ 0 & \cos\alpha & -\sin\alpha & \frac{h}{\sin\alpha} \\ 0 & 0 & 0 & 1 \end{bmatrix} \\ &= \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & -f\sin\alpha & -f\cos\alpha & 0 \\ 0 & \cos\alpha & -\sin\alpha & \frac{h}{\sin\alpha} \end{bmatrix} \end{aligned}$$

With this,

ii. * Find the 3×3 matrix for the linear transformation that maps points on the world plane $Z_w = d$ to the image plane (u, v, w).

Answer:
Borrowing the result from part (ii):

$$\begin{aligned} \begin{pmatrix} u \\ v \\ w \end{pmatrix} &= M \begin{pmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{pmatrix} \\ &= \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & -f\sin\alpha & -f\cos\alpha & 0 \\ 0 & \cos\alpha & -\sin\alpha & \frac{h}{\sin\alpha} \end{bmatrix} \begin{pmatrix} X_w \\ Y_w \\ d \\ 1 \end{pmatrix} \\ &= \begin{pmatrix} fX_w \\ -fY_w\sin\alpha - fd\cos\alpha \\ Y_w\cos\alpha - d\sin\alpha + \frac{h}{\sin\alpha} \end{pmatrix} \end{aligned}$$

We need to convert this into a $M_{3\times3}^{'}\begin{pmatrix} X_w \\ Y_w \\ 1 \end{pmatrix}$ form:

$$\begin{pmatrix} u \\ v \\ w \end{pmatrix} = \begin{bmatrix} f & 0 & 0 \\ 0 & -f\sin\alpha & -fd\cos\alpha \\ 0 & \cos\alpha & \frac{h}{\sin\alpha} - d\sin\alpha \end{bmatrix} \begin{pmatrix} X_w \\ Y_w \\ 1 \end{pmatrix}$$

2 Singular Value Decomposition for Image Compression and PCA

1. Many thousands of outdoor cameras are currently connected to the Internet. They can be used to measure plant growth (e.g. in response to recent warming trends), survey animal populations (e.g. in national parks), monitor surf conditions, and security, etc. In this question, you are provided with a 150-frame time-lapse video of a city scene taken between 6.30 - 7.30pm. Each frame is of the dimension 161x261 pixels. Watch the video, and you can see that clouds are moving, and planes take off and land from a nearby airport.

   (a) Using Matlab, load the first image in the video sequence provided with this question and convert it to an appropriate data type (`I=im2double(imread(image001.png));`).

   (b) Do a singular value decomposition using the command svd: `[U S V]=svd(I);` This will give you the singular values and the singular vectors. The singular values in S have been sorted in descending order. Plot the singular value spectrum using the command: `plot(diag(S),'b.')`. **Submit this plot. What do you notice in this plot?**

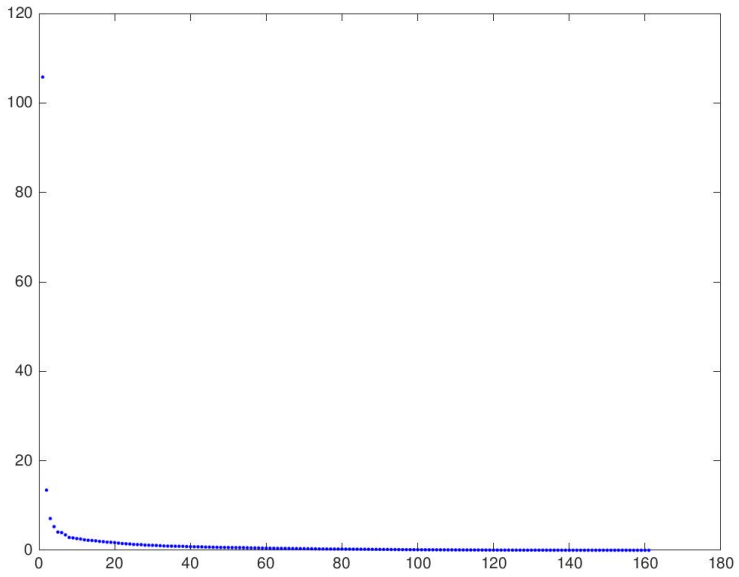


Figure 2: Singular values

Answer:
All the singular values are plotted in Fig. 2. As shown, the first singular value is very large (>100), while the following singular values are approximately between $[5, 20]$. When the index gets even larger, the singular values approaches zero (become very small). The condition number, which is

$$cond(I) = \frac{\sigma_{max}}{\sigma_{min}} \tag{4}$$

gets very large as a result. This suggests that it is ill-conditioned.

   (c) Let K=20, Extract the first K singular values and their corresponding vectors in U and V:

```
K=20;
Sk=S(1:K,1:K);
Uk=U(:,1:K);
Vk=V(:,1:K);
```

Uk, Vk, Sk contain the compressed version of the image. To see this, form the compressed image:

```
Imk=Uk*Sk*Vk';
imshow(Imk)
```

Print out a copy of this compressed image and submit it.

Answer:
The compressed image using the first 20 singular values is shown in Fig. 3 below.

Figure 3: Compressed image (K=20)

(d) Repeat question (c) for K=40,60,80. **Submit the compressed images for the different values of K. Compare the 4 compressed images. Briefly describe what you notice.**

Answer:
The resultant images are shown in Fig. 4 - Fig. 6. As we take more singular values into computation (larger K), the quality of the compressed image improves.

$$\tilde{I} = s_1 u_1 v_1^T + s_2 u_2 v_2^T + \cdots + s_K u_K v_K^T \tag{5}$$

As shown in the equation above, larger K will give better approximation.

Figure 4: Compressed image (K=40)

Figure 5: Compressed image (K=60)

Figure 6: Compressed image (K=80)

(e) Thus, in image transmission, instead of transmitting the original image you can transmit Uk, Vk, Sk, which should be much less data than the original. **Is it worth transmitting when K=100** (i.e. do you save any bits in transmission when K=100)? Explain your answer.

Answer:
The total storage of the compressed image I_K is expressed by:

$$S = K(m + n + 1) \tag{6}$$

For this question, $m = 161, n = 261$, so if we take $K = 100$, the total storage we need for the transmission is $100 \times (161 + 261 + 1) = 42300$. This is even larger than the storage for just transmitting original image ($161 \times 261 = 42021$). Thus, transmitting compressed image with K=100 is **NOT** worth.

(f) ** Find an expression that bounds the per-pixel error (the difference between the original image and the compressed image for a particular pixel (i, j)) in terms of K, the singular values and the elements in U and V with the largest absolute magnitude (u_{max}, v_{max} respectively).

Answer:
The completely re-constructed image (suppose N non-zero singular values) and the compressed image using first K singular values are expressed by:

$$\begin{aligned} I &= s_1 u_1 v_1^T + s_2 u_2 v_2^T + \cdots + s_K u_K v_K^T + \cdots + s_n u_n v_n^T \\ \tilde{I} &= s_1 u_1 v_1^T + s_2 u_2 v_2^T + \cdots + s_K u_K v_K^T \end{aligned} \tag{7}$$

So the error term is their difference:

$$\varepsilon = |\sum_{n=K+1}^{N} s_n u_n v_n^T| \tag{8}$$

For a single pixel ($I_{i,j}$), the error is:

$$\begin{aligned}\epsilon &= |\sum_{n=K+1}^{N} s_n u_{i,n}(v^T)_{n,j}| \\ &\leqslant \sum_{n=K+1}^{N} |s_n u_{i,n}(v^T)_{n,j}| \\ &\leqslant \sum_{n=K+1}^{N} s_n |u_{i,n}||(v^T)_{n,j}| \\ &\leqslant \sum_{n=K+1}^{N} s_n u_{max} v_{max}\end{aligned} \tag{9}$$

(g) ** SVD is intimately related to PCA (Principal components analysis). Some of you might have learned about PCA in the EE3731C course, but it is not a pre-requisite for this question. Basically, finding the principal components of a matrix X amounts to finding an orthonormal basis that spans the column space of X (these are the column vectors u_i in the matrix U). Here we create the data matrix X by first vectorizing each of the 150 images into a column vector (by scanning in either row-major order or column- major order), and then stacking these vectors together into a matrix of size 42021×150. In effect, we have captured the entire video sequence into the 2D matrix X.

Before we proceed further, mean subtraction (a.k.a. "mean centering") to center the data at the origin is necessary for performing PCA. That is, the images are mean centered by subtracting the mean image vector from each image vector. This is to ensure that the first principal component u_1 really describes the direction of maximum variance. Again, if you do not have background in PCA, it is okay; just take it as a preprocessing step (or you can do some independent learning).

Apply SVD to the resultant mean-centered matrix. Take only the first 10 principal components and reconstruct the image sequence. [NB: you can use the Matlab `reshape` command to convert a matrix to a vector and vice versa]. Observe the dynamics in the reconstructed video, comment on what you find, and explain why. Remember to add back this mean image vector when you are displaying your reconstructed results.

[Non-mandatory]: You can also explore other possibilities, like investigating what each principal component means, and their implications for processing and editing of outdoor webcam imagery.

Answer:
In principal component analysis we find the directions in the data with the most variation, i.e. the eigenvectors corresponding to the largest eigenvalues of the covariance matrix, and project the data onto these directions [3].
Let's denote the elements in the data matrix X as $X_{j,\alpha} = x_j^\alpha$. The mean vector is:

$$\langle \mathbf{x} \rangle \equiv \frac{1}{m} \sum_{\alpha=1}^{m} \mathbf{x}^\alpha \tag{10}$$

and the coariance matrix is:

$$C \equiv \frac{1}{m} XX^T, \tag{11}$$

where we have removed the mean of the data: $X_{j,\alpha} := X_{j,\alpha} - \langle x_j \rangle$.
Apply the SVD to X and substitute X in Eq. (11):

$$\begin{aligned} C &= \frac{1}{m}(U\Sigma V^T)(U\Sigma V^T)^T \\ &= \frac{1}{m}U\Sigma V^T V\Sigma^T U^T \\ &= \frac{1}{m}U\Sigma^2 U^T \end{aligned} \tag{12}$$

Clearly, the result in Eq. (12) demonstrates the Eigendecomposition of the real-valued symmetric matrix mC. Hence, the columns of U are the eigenvectors of matrix mC and the diagonal elements of Σ^2 are the eigenvalues. Therefore, the matrix U after SVD of X actually contains what we want for PCA.
Furthermore, if we denote the matrix of eigenvectors **sorted** according to eigenvalue by $\tilde{U}$, we can then use PCA to transform the data to be $Y = \tilde{U}^T X$. The eigenvectors are called the principal components. By selecting only the first d rows of Y, we have projected the data from n down to d dimensions.

In Matlab, to load all the images and construct the data matrix X, and do the mean subtraction:

```
X = [];
Files=dir('*.png');
for k=1:length(Files)
   FileNames=Files(k).name;
   I = im2double(imread(FileNames));
   I = I(:);
   X = [X I];
end
mean_X = mean(X, 2);
de_mean_X = X - mean_X;
```

Then, to calculate the SVD of matrix X and transform the data into the first 10 principle components' space:

```
[U S V] = svd(de_mean_X,0); % to produce "economy size"
X_PCAed = U(:,1:10) * S(1:10, :) * V';
X_reconstructed = X_PCAed + meanX;
for i=1:150
    Mat = X_reconstructed(:,i);
    Name = strcat('img_', int2str(i), '.png');
    imwrite(reshape(Mat, [161, 261]) ,Name,'png');
end
```

Findings
By viewing the reconstructed video, I found that the only the light generated by planes sliding on the ground remains, while the light during the landing and taking off processes are nearly "removed out". Besides, the light also gets blurry after PCA operation. This is because the operation only takes the first ten largest singular values out as valid information, and forms the orthogonal basis using these corresponding eigen-vectors. Every single image (column) is reconstructed using the principle components of all the images thus only the "common"

information remains; and between the frames (row) they are also "smoothed out" thus things get blurry.

3 Projection Model, Homogeneous coordinates

1. (a) An affine camera is a simplification of the full perspective camera but is more complicated than the scaled orthographic model. It has a projection relationship given by the following equations: $\begin{pmatrix} x \\ y \end{pmatrix} = A \begin{pmatrix} X \\ Y \\ Z \end{pmatrix} + \mathbf{b}$, where A is a 2×3 matrix, and $\mathbf{b}$ is a vector in $\mathcal{R}^2$. If the world point $(X, Y, Z)^T$ and the image point $(x, y)^T$ are represented by homogeneous vectors, write down the matrix representing the linear mapping between their homogeneous coordinates.

   * Show that the point at infinity in space $(X, Y, Z, 0)^T$ is mapped to point of infinity in the image plane. What does this result imply about the projection of parallel lines in space onto the image plane?

   Answer:
   Similar to Eq. (2), in the homogeneous coordinate, affine transformation is described by:

$$\begin{aligned} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} &= \begin{bmatrix} A & \mathbf{b} \\ \mathbf{0}^T & 1 \end{bmatrix} \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} \\ &= H_A \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} \end{aligned} \tag{13}$$

   To show the required statement, just apply the affine transformation H_A to $(X, Y, Z, 0)^T$, we obtain $(x', y', 0)$ in the image plane. The result indicates that this point is at infinity in the image plane.
   We know that the intersection of two parallel lines is at infinity in the 3-D space. So this result implies that after affine transformation, the intersection point is still at the infinity in the image plane. This is because affine transformation is parallelity-invariant.

   (b) Given an image of a scene containing a cube as shown in the following.

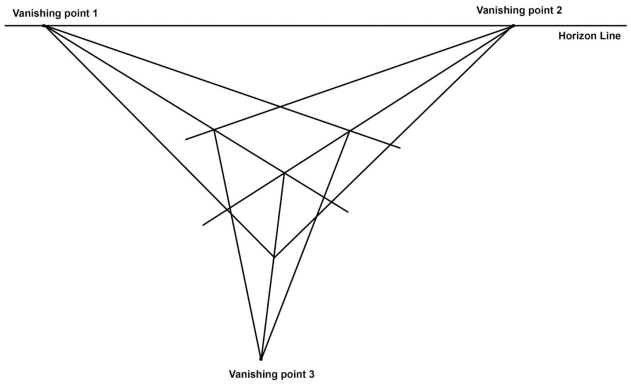


Figure 7: Cube

Each vanishing point is the point on the image corresponding to a point in 3D space at infinity of the form $(\mathbf{d}, 0)^T$, where $\mathbf{d}$ is a Euclidean vector with 3 coordinates expressing the direction of a cube edge.

i. ** Show that the coordinates of a vanishing point $\mathbf{v}$ can be expressed as $\mathbf{v} = KR\mathbf{d}$, where K is the intrinsic calibration matrix and R is the rotation matrix between the camera and world coordinate system. Hence express an edge direction $\mathbf{d}$ as a function of K, R and $\mathbf{v}$. (Hint: Start from the 3×4 projection matrix equation in Ch 1)

Answer:
Let $\mathbf{d} = (a, b, c)^T$, and apply the projection matrix to point at infinity in 3D space $x_\infty = (a, b, c, 0)^T$:

$$\begin{aligned} \mathbf{v} = Mx_\infty &= K \begin{bmatrix} R & T \end{bmatrix} \begin{pmatrix} a \\ b \\ c \\ 0 \end{pmatrix} \\ &= KR \begin{pmatrix} a \\ b \\ c \end{pmatrix} \\ &= KR\mathbf{d} \end{aligned} \tag{14}$$

Hence, the direction vector is:

$$\mathbf{d} = \frac{(KR)^{-1}\mathbf{v}}{\|(KR)^{-1}\mathbf{v}\|} \tag{15}$$

ii. ** The 3 directions of the cube edges are mutually perpendicular; therefore the dot product between any two directions is zero. Show that this condition leads to an equation in terms of the vanishing points and the unknown calibration matrix K. Note that such an equation can be written for each pair of the 3 vanishing points.

Answer:
First, the angle between any two directions in the 3D space is given by the cosine rule:

$$\begin{aligned} \cos\theta &= \frac{\mathbf{d}_1 \cdot \mathbf{d}_2}{\|\mathbf{d}_1\| \, \|\mathbf{d}_2\|} \\ &= \frac{v_1^T \omega v_2}{\sqrt{v_1^T \omega v_1}\sqrt{v_2^T \omega v_2}} \end{aligned} \tag{16}$$

where $\omega = \left((KR)(KR)^T\right)^{-1} = \left(KK^T\right)^{-1}$.
As given, all the edges are mutually perpendicular, therefore $\cos\theta = 0$, and the equations given by these three vanishing points are:

$$v_1^T \omega v_2 = v_1^T \omega v_3 = v_3^T \omega v_3 = 0 \tag{17}$$

iii. *** Note that the derived equation above can be used to solve for K, but you are not required to explicitly propose a scheme for solving this equation. However, doing so will earn you bonus point.

Answer:
In this setup, we have three constraints, but the unknown K has five degrees of freedom. Hence, we could make assumption that the camera has zero-skew and square pixels to add two more constraints on it.
Our original intrinsic matrix is given by:

$$K = \begin{bmatrix} \frac{-1}{s_x} & k & o_x \\ 0 & \frac{-1}{S_y} & o_y \\ 0 & 0 & 1 \end{bmatrix} \tag{18}$$

By the additional constraints, it becomes:

$$K = \begin{bmatrix} \alpha & 0 & o_x \\ 0 & \alpha & o_y \\ 0 & 0 & 1 \end{bmatrix}$$

Therefore, with three equations, to solve three unknowns becomes possible. And by doing this we calibrate the camera with a single image!

(c) The projection of a scene point in world coordinates to pixel coordinates in an image can be represented using a camera projection matrix P as follows:

$$\begin{pmatrix} s_u \\ s_v \\ s \end{pmatrix} = \begin{bmatrix} p_{11} & p_{12} & p_{13} & p_{14} \\ p_{21} & p_{22} & p_{23} & p_{24} \\ p_{31} & p_{32} & p_{33} & p_{34} \end{bmatrix} \begin{pmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{pmatrix}$$

i. * Given a set of lines in a scene that are all parallel to the world X-axis, what is the vanishing point, (u, v), of these lines in the image? Can you conclude whether an infinite line in 3D space always yield an infinite line in the 2D image plane?

Answer:
Let a point at infinity in 3D space $x_\infty = (1, 0, 0, 0)^T$. Apply the projection matrix P, we obtain the corresponding image point:

$$\begin{pmatrix} s_u \\ s_v \\ s \end{pmatrix} = \begin{pmatrix} p_{11} \\ p_{21} \\ p_{31} \end{pmatrix}$$

As we can observe, an infinite line in 3D space dons **not** necessarily yield an infinite line in 2D image plane, since the s component may be non-zero. For example, if the projection is an affine transformation, the image line will be at infinity; however, is the projection is an projective transformation, the image line will not be at infinity.

ii. What is the significance of the image point (represented as a homogeneous 3-vector) given by the last column (p_{14}, p_{24}, p_{34}) of P? That is,which world point gives rise to (p_{14}, p_{24}, p_{34}) ?

Answer:
If the world point is $(0, 0, 0, 1)^T$, the corresponding image point will be (p_{14}, p_{24}, p_{34}). Actually is the image point of the plane at infinity $\Pi_\infty = (0, 0, 0, 1)^T$, and it's also referred as **horizon line**.

iii. ** Consider the 1-dimensional right null-space of P, i.e., the 4-vector $\mathbf{C}$ such that $P\mathbf{C} = 0$. In this case, the image point of $\mathbf{C}$ is $(0, 0, 0)^T$ which is not defined. What is the point $\mathbf{C}$ which possesses this property? Explain.

Answer:
Point $\mathbf{C}$ should be the camera center.

Proof. Let $\mathbf{A}$ be an arbitrary point in the 3D space. A line can be formed by connecting point $\mathbf{A}$ and point $\mathbf{C}$. Points on this line can be described by:

$$\mathbf{X}(\lambda) = \lambda\mathbf{A} + (1-\lambda)\mathbf{C}, -\infty < \lambda < \infty \tag{19}$$

Under the projection $\mathbf{x} = P\mathbf{X}$, the image point on this line can be described by:

$$\mathbf{x} = P\mathbf{X}(\lambda) = \lambda \mathrm{P}\mathbf{A} + (1-\lambda)\underbrace{P\mathbf{C}}_{=0} = \lambda \mathrm{P}\mathbf{A}, -\infty < \lambda < \infty \tag{20}$$

Eq. (20) shows that all the points on this 3D line are mapped to the same point. Thus this line must go through the camera center. Additionally, since point $\mathbf{A}$ is arbitrary, to make sure the line goes through the camera center, point $\mathbf{C}$ must be the camera center. ■

(d) The equation of a conics in inhomogeneous coordinates is given by $ax^2 + bxy + cy^2 + dx + ey + f = 0$.

i. Homogenize this equation by the replacement $x \to \frac{x_1}{x_3}$, $y \to \frac{x_2}{x_3}$, and write down the homogeneous form. Finally, express this homogeneous form in the matrix form $\mathbf{x}^T \mathrm{C}\mathbf{x} = 0$, where C is symmetric. Write down the elements of the symmetric matrix C.

Answer:
In homogeneous coordinate, the conics equation becomes:

$$ax_1^2 + bx_1x_2 + cx_2^2 + dx_1x_3 + ex_2x_3 + fx_3^2 = 0 \tag{21}$$

This can be written in matrix form:

$$\begin{pmatrix} x_1 & x_2 & x_3 \end{pmatrix} \begin{bmatrix} a & b/2 & d/2 \\ b/2 & c & e/2 \\ d/2 & e/2 & f \end{bmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = 0 \tag{22}$$

So the symmetric conics matrix $\mathrm{C} = \begin{bmatrix} a & b/2 & d/2 \\ b/2 & c & e/2 \\ d/2 & e/2 & f \end{bmatrix}$.

ii. ** If C has the special form $\mathrm{C} = \mathbf{l}\mathbf{m}^T + \mathbf{m}\mathbf{l}^T$, clearly C is symmetric and therefore represents a conic. Show that the vector $\mathbf{x} = \mathbf{l} \times \mathbf{m}$ is the null vector of C. Since a null vector exists, C is a degenerate conic. Show in this degenerate case, C contains two lines $\mathbf{l}$ and $\mathbf{m}$, that is, show that the points on the lines $\mathbf{l}$ and $\mathbf{m}$ satisfy $\mathbf{x}^T \mathrm{C}\mathbf{x} = 0$.

Answer:
We know that in homogeneous coordinate, $\mathbf{x} = \mathbf{l} \times \mathbf{m}$ means that x is the intersection of lines $\mathbf{m}$ and $\mathbf{l}$. Thus we have:

$$\mathbf{m}^T\mathbf{x} = 0, \quad \mathbf{l}^T\mathbf{x} = 0 \tag{23}$$

Therefore, to show that $\mathbf{x}$ is the null vector of C:

$$\mathrm{C}\mathbf{x} = \mathbf{l}\mathbf{m}^T\mathbf{x} + \mathbf{m}\mathbf{l}^T\mathbf{x} = \mathbf{0} \tag{24}$$

Since conic C has a null vector, the matrix C is not of full rank, and it is a degenerate conic. To show line $\mathbf{l}$ is on this conic, let $\mathbf{p}$ be any arbitrary point on line $\mathbf{l}$, we have $\mathbf{l}^T\mathbf{p} = \mathbf{p}^T\mathbf{l} = 0$. To evaluate $\mathbf{p}^T\mathrm{C}\mathbf{p}$:

$$\mathbf{p}^T\mathrm{C}\mathbf{p} = \left(\mathbf{p}^T\mathbf{l}\right)\left(\mathbf{m}^T\mathbf{p}\right) + \left(\mathbf{p}^T\mathbf{m}\right)\left(\mathbf{l}^T\mathbf{p}\right) = 0 \tag{25}$$

This shows $\mathbf{p}$ is on this conic. Since $\mathbf{p}$ is any arbitrary point on line $\mathbf{l}$, line $\mathbf{l}$ is on this conic. Using the similar steps for line $\mathbf{m}$ ($\mathbf{m}^T\mathbf{p} = \mathbf{p}^T\mathbf{m} = 0$), we can get the same result as Eq. (25).
Thus, we can conclude that this degenerate conic contains two lines $\mathbf{l}$ and $\mathbf{m}$, and their intersection is the null vector for conic matrix C.

(e) Given a set of two-dimensional points (x, y) as follows (not listed here, can be found in *conics.txt*), write a Matlab routine to find the conics best (in the least squares sense) represented by these points, in terms of the parameters a, b, c, d, e, and f, as formulated in Question 3(d).

Answer:
Any point (x_i, y_i) on the conic should satisfy the conic equation:

$$ax_i^2 + bx_iy_i + cy_i^2 + dx_i + ey_i + f = 0 \tag{26}$$

This can be written as:

$$\begin{pmatrix} x_i^2 & x_iy_i & y_i^2 & x_i & y_i & 1 \end{pmatrix}\mathbf{c} = 0 \tag{27}$$

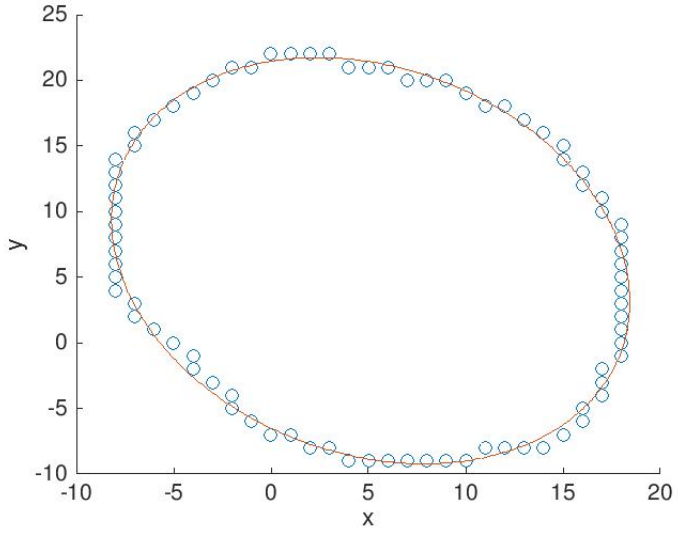
where $\mathbf{c} = (a, b, c, d, e, f)^T$ is the conic C as a 6-vector. This becomes a least square problem of the form $Ax = 0$. The Matlab routine as below will find the "best" conic:

```
load conics.txt
c = fit_conic(conics);
x = conics(:,1);
y = conics(:,2);
scatter(x,y)
hold on
fimplicit(@(x,y) c(1)*x^2 + c(2)*x*y + c(3)*y^2 + c(4)*x + c(5)*y + c(6))
xlabel('x')
ylabel('y')

function c = fit_conic(pts)
    x = pts(:,1);
    y = pts(:,2);
    A = [x.^2, x.*y, y.^2, x, y, ones(size(x))];
    [U, S, V] = svd(A);
    c = V(:,length(V)); % last column is the null vector
end
```

The result is verified in by plotting the figure below.

Assignment 1



4 Estimate Fundamental Matrix F using 8-Point Algorithm

1. **Introduction**

   The fundamental matrix F completely describes the epipolar geometry of a pair of images. 8-Point Algorithm is the simplest method for estimating the fundamental matrix. In this assignment, you will implement the 8-Point Algorithm to estimate the fundamental matrix given a pair of images. You would also need to implement the normalization so as to improve the stability (outlined in steps 4.2.2 and 4.2.4 below). Finally, we have also provided a package to help you reconstruct and visualize the depths (see point 6 in 4.3; this step is optional but will earn you bonus point). Please read this document completely before starting work. A list of reading materials you may need to solve this problem is given in Appendix I.

2. **Tasks**

   You are given 2 pairs of images (inria1.tif, inria2.tif, frc1.tif, frc2.tif) for this assignment. The steps for this assignment are:

   1. Using one of the given pairs of images, establish $n(n \geq 8)$ point correspondences.
   2. Normalize the coordinates of the correspondences.
   3. Using the normalized coordinates of the correspondences, compute the fundamental matrix.
   4. Perform denormalization so as to find the fundamental matrix corresponding to the original data.

   Details of each step are given in the following sections. Some useful Matlab commands are given in Appendix II.

   (a) [Establish Point Correspondences]
   In this step, you need to establish point correspondences between the two images. Using Matlab, manually or automatically mark and record at least 8 pairs of corresponding points (preferably much more than 8 pairs for robustness!).

   Answer:
   After researching online, I used feature matching function provided in Matlab Computer Vision System toolbox based on **Harris Features** to **automatically** extract feature-matched points:

```
I1 = imread('frc1.jpg');
I2 = imread('frc2.jpg');

points1 = detectHarrisFeatures(I1, 'MinQuality', 0.001);
points2 = detectHarrisFeatures(I2, 'MinQuality', 0.001);

[features1,valid_points1] = extractFeatures(I1,points1);
[features2,valid_points2] = extractFeatures(I2,points2);

indexPairs = matchFeatures(features1,features2);

matchedPoints1 = valid_points1(indexPairs(:,1),:);
matchedPoints2 = valid_points2(indexPairs(:,2),:);


rand1 = randsample(32, 8);
grp1a = matchedPoints1(rand1);
grp1b = matchedPoints2(rand1);
grp1a = grp1a.Location;
grp1b = grp1b.Location;
```

```
21
22  rand1 = randsample(32, 8);
23  grp2a = matchedPoints1(rand1);
24  grp2b = matchedPoints2(rand1);
25  grp2a = grp2a.Location;
26  grp2b = grp2b.Location;
27
28  rand1 = randsample(32, 8);
29  grp3a = matchedPoints1(rand1);
30  grp3b = matchedPoints2(rand1);
31  grp3a = grp3a.Location;
32  grp3b = grp3b.Location;
```

Running the scripts above, I obtained three sets of points, each of which contains 8 point correspondences. As required by the question, I repeated the process of computing the fundamental matrix to check for the accuracy.

The Figure $9-11$ below shows three different sets of matched points. These points are all corresponding **corners** on both images. They are chosen as the feature points to do the matching since they are less ambiguous compared to plain areas or edges.

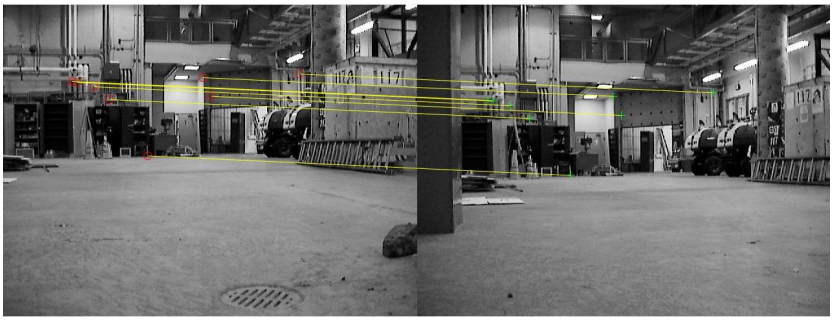


Figure 8: 1^{st} set of point correspondences

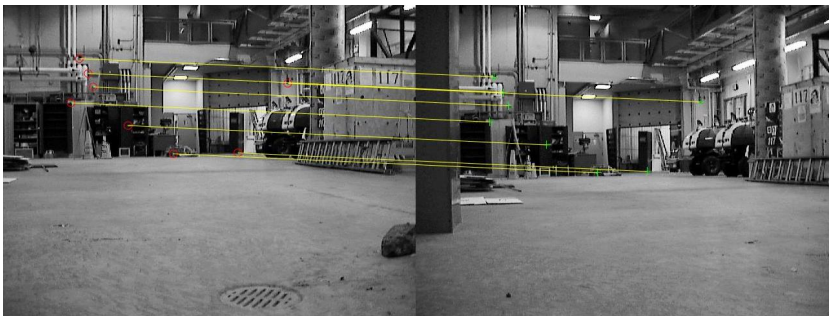


Figure 9: 2^{nd} set of point correspondences

Figure 10: 3^{rd} set of point correspondences

(b) [Normalization of the data]
In 8-point algorithm, the stability of the results can be greatly improved by a simple normalization (translation and scaling) of the coordinates of the correspondences (originally presented in [Hartley-1997] and also in [Hartley and Zisserman-2000]). For this normalization, you should find a similarity transformation T , consisting of a translation and scaling, that takes points x_i to a new set of points $\hat{x}_i$ ($\hat{x}_i = Tx_i$)such that the centroid of the points $\hat{x}_i$ is the coordinates origin, and their average distance (Root Mean Squares) from the origin is 2 pixels . Similarly, find a similarity transformation T' that takes points x'_i to points x_i ($\hat{x}'_i = T'x_i$). Note: Homogeneous coordinates are used in this step.

Answer:
For one image, the centroid of all points is defined by:

$$(x_c, y_c) = (\frac{1}{N}\sum_{i=1}^{N} x_i, \frac{1}{N}\sum_{i=1}^{N} y_i) \tag{28}$$

Every point is translated such that the centroid of the points becomes the coordinate origin. The translation is determined by the centroid's coordinates:

$$\begin{aligned} t_x = -x_c, \quad t_y = -y_c \\ x_i := x_i + t_x \\ y_i := y_i + t_y \end{aligned} \tag{29}$$

After the translation, all the points are scaled such that their Root Mean Square distance from the origin is $\sqrt{2}$, which is equivalent to be scaled by s:

$$s = \sqrt{\frac{2}{\frac{1}{N}\sum_{i=1}^{N}(x_i^2 + y_i^2)}} \tag{30}$$

Hence, to write the transformation $\hat{x}_i = s(x_i + t_x)$ and $\hat{y}_i = s(y_i + t_y)$ in matrix form:

$$T_s = \begin{bmatrix} s & 0 & st_x \\ 0 & s & st_y \\ 0 & 0 & 1 \end{bmatrix} \tag{31}$$

The Matlab codes are listed in the next sectioon together with the codes for F estimation.

(c) [Computation of Fundamental Matrix]
The computation of the fundamental matrix is described in detail in your textbook or in Section 10.1 of [Hartley and Zisserman-2000]. In summary the steps are as follows:

i. Find the solution for f
From a set of n point correspondences, we obtain a set of linear equations of the form:

$$Af = \begin{bmatrix} x'_1x_1 & x'_1y_1 & x'_1 & y'_1x_1 & y'_1y_1 & y'_1 & x_1 & y_1 & 1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ x'_nx_n & x'_ny_n & x'_n & y'_nx_n & y'_ny_n & y'_n & x_n & y_n & 1 \end{bmatrix} f = 0 \tag{32}$$

A is an $n \times 9$ matrix ($n \geq 8$)and f is the 9-vector made up of the entries of F in row-major order. If $rank(A) \leq 8$, non-trivial solution for the system above exists, and if $rank(A) = 8$, the solution is unique (up to scale). The solution for f should be the last column of V in the $svd(A) = U\Sigma V^T$.

ii. Enforce singularity constraint
Fundamental Matrix is singular, in fact of rank 2. The matrix F found by the above method will not in general have rank 2 and steps should be taken to enforce this singularity constraint. One easy way to do this is: let $F = U\Sigma V^T$ be the SVD of F, where Σ is a diagonal matrix $diag(r, s, t)$ satisfying $r \geq s \geq t$. Let $F' = Udiag(r, s, 0)V^T$. According to the so-called Frobenius norm, F' is the closest singular matrix to F. Thus replace F by F'. Note that you should use the normalized data.

Answer:
The codes are listed as follows:

```
% construct A; assume normalized points are stored in n_pts1 and n_pts2
A = [n_pts2(:,1).*n_pts1(:,1) n_pts2(:,1).*n_pts1(:,2) n_pts2(:,1) ...
     n_pts2(:,2).*n_pts1(:,1) n_pts2(:,2).*n_pts1(:,2) n_pts2(:,2) ...
     n_pts1(:,1) n_pts1(:,2) ones(size(n_pts1(:,1)))];
% from svd(A), get F
[U, S, V] = svd(A);
F = V(:,length(V));
F = reshape(F, [3,3])';
% enforce F singularity
[U_f, S_f, V_f] = svd(F);
S_f(3,3) = 0;
F_prime = U_f * S_f * V_f';
```

iii. Denormalization
Till this step, you have obtained the fundamental matrix corresponding to the normalized data $\hat{x}_i \leftrightarrow \hat{x}'_i$. This needs to be denormalized. Set $F = T_s'^T F' T_s$ (T_s and T'_s is obtained in the normalization step), then F is the fundamental matrix corresponding to the original data $x_i \leftrightarrow x'_i$

Answer:
The Matlab command simply runs as follows:

```
F = T_s2' * F_prime * T_s1;
```

And the result for the fundamental matrix is calculated as:

$$F_1 = \begin{bmatrix} -5.4958e-07 & -4.9610e-06 & 2.3606e-03 \\ 3.1570e-06 & -1.7717e-06 & 2.0223e-03 \\ -1.6578e-03 & -8.4793e-04 & -1.4653e-01 \end{bmatrix} \tag{33}$$

During the testing, I found that the epipolar lines do not correspond to the points as shown in the figure below.

Figure 11: Epipolar lines found by $l_r = Fp_l$ (1^{st} set)

For the second round, I think it will be better if I include more than 8 points. Hence, I concatenated the remaining two sets of data and computed the fundamental matrix again. The calculated fundamental matrix and testing result is shown in the figure below:

$$F_2 = \begin{bmatrix} 4.6784\text{e}-10 & -9.2670\text{e}-07 & 1.3625\text{e}-04 \\ 7.0476\text{e}-07 & 1.2601\text{e}-07 & -1.7650\text{e}-03 \\ -1.2534\text{e}-04 & 1.7892\text{e}-03 & 4.5285\text{e}-02 \end{bmatrix} \tag{34}$$

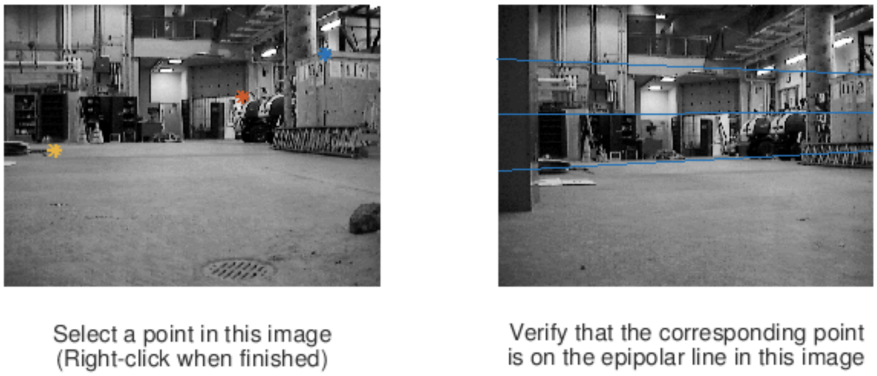


Figure 12: Epipolar lines found by $l_r = Fp_l$ ($2^{nd}, 3^{rd}$ sets)

As shown, the results get much better than before in the sense that the corresponding epipolar lines go through the matched points. However, it is a bit strange that the position

of the epipole now lies on the right of the second image.
After these experiments, I think that the accuracy of the eight-point algorithm could be quite low if the matching points are not accurately selected. As pointed out in by Hartley [1], the inaccuracies in the matching point measurement will result in A not full rank. Consequently, we can only choose the least eigenvector of A^TA to be our solution, and this introduce some errors. Although I used Harris feature detector to do the feature matching, there's still inaccuracy in the result. Besides the problem with matching points, this algorithm itself is also not very good. Instead, we can use some iterative algorithm like RANSAC to improve the results.

(d) **Final results and structured codes**
To get more accurate results, I used Harris feature detector with more strict requirement and it outputs 17 points for me only. There are supposed to be best matched points and I used these data as input to calculate the fundamental matrix. My final result is shown below:

$$F = \begin{bmatrix} -3.9335\text{e}-09 & 6.2600\text{e}-07 & -1.1718\text{e}-04 \\ -7.6470\text{e}-07 & -1.7108\text{e}-08 & -1.7488\text{e}-03 \\ 1.5567\text{e}-04 & 1.7997\text{e}-03 & 4.5633\text{e}-02 \end{bmatrix} \tag{35}$$

The Matlab source codes is attached together with this report, but for your convenience the complete codes for normalization and estimation are listed as follow:

```
I1 = imread('frc1.tif');
I2 = imread('frc2.tif');
% first computation
displayEpipolarF(I1,I2,estimateF(grp1a', grp1b'))   % transpose since the required ...
    input is 2 $\times$ n matrix

% second computation
displayEpipolarF(I1,I2,estimateF([grp2a; grp3a]', [grp2b; grp3b]'))

% final computation using better data omitted...

function [ normalized_pts, T_s ] = normalize( pts )
%   Input: points on an image
%   Output: normalized points for the use of eight_point_algo
    n = size(pts, 1);
    centroid = sum(pts, 1) / n;
    disp('The centroid is');
    disp(centroid);

    pts_demean = pts - centroid;

    squared_distance_sum = sum(sum(pts_demean.^2));

    scale_factor = sqrt(mean(squared_distance_sum) / 2);    % scale to sqrt(2)

    normalized_pts = pts_demean / scale_factor;

    disp('Check for the RMS average distance:');
    disp(sqrt(mean(sum(sum(normalized_pts.^2)))));

    % x' = s * (x + t_x) ; y' = s * (y + t_y)
    t_x = -centroid(1);
    t_y = -centroid(2);
    s = 1 / scale_factor;

```

```
35      T_s = [s 0 s*t_x; 0 s s*t_y; 0 0 1];
36      disp('The similarity transformation is');
37      disp(T_s);
38  end
39
40
41  function F = estimateF(x1, x2)
42      % x1 and x2 are each 2 x n matrices with the
43      % columns corresponding to the coordinates in the first
44      % and the second image respectively
45      pts1 = x1';
46      pts2 = x2';
47
48      [n_pts1, T_s1] = normalize(pts1);
49      [n_pts2, T_s2] = normalize(pts2);
50
51      % construct A
52      A = [n_pts2(:,1).*n_pts1(:,1) n_pts2(:,1).*n_pts1(:,2) n_pts2(:,1) ...
             n_pts2(:,2).*n_pts1(:,1) n_pts2(:,2).*n_pts1(:,2) n_pts2(:,2) n_pts1(:,1) ...
             n_pts1(:,2) ones(size(n_pts1(:,1)))];
53      % from svd(A), get F
54      [U, S, V] = svd(A);
55      F_temp = V(:,length(V));
56      F_temp = reshape(F_temp, [3,3])';
57      % enforce F singularity
58      [U_f, S_f, V_f] = svd(F_temp);
59      S_f(3,3) = 0;
60      F_prime = U_f * S_f * V_f';
61
62      %denormalization
63      F = T_s2' * F_prime * T_s1;
64  end
```

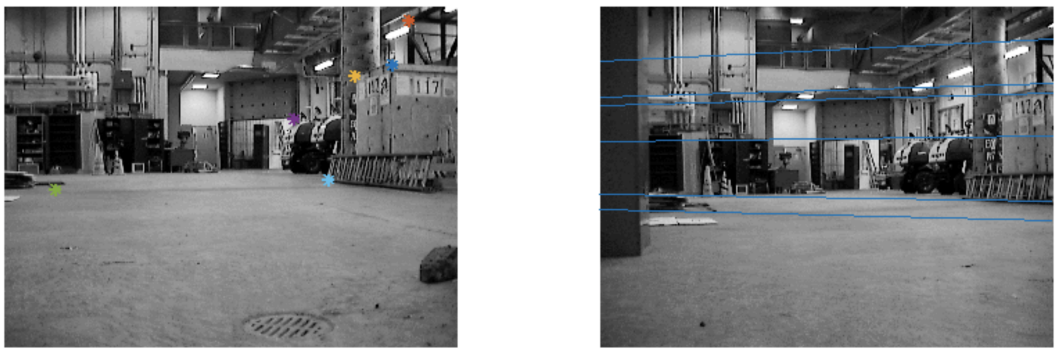


Figure 13: Final (best) result

The figure above shows that for every point selected in the left image, the fundamental matrix can effectively find the corresponding epipolar line in the right image. This is obtained use the most accurate matched points I can get.

5 Motion Perception

1. In the figure below, the two squares are translating horizontally in the opposite directions as indicated by the arrows. Indicate the respective perceived motions if you are looking through the apertures 1, 2, and 3. Explain your answers.

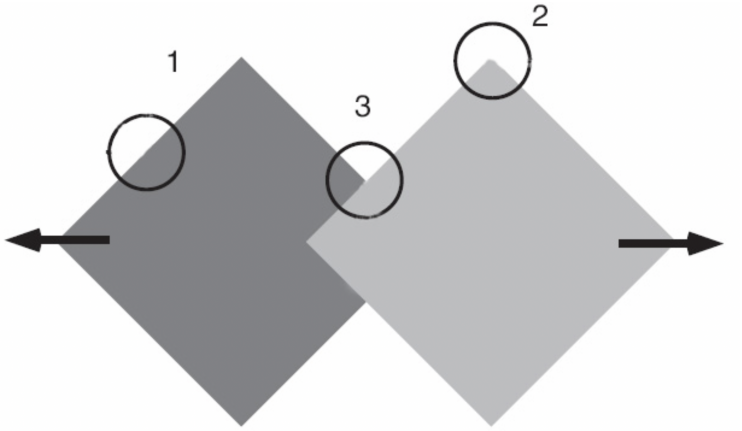


Figure 14: Two squares translating horizontally in the opposite directions.

Answer:

Based on Brightness Constancy Equation assumption, we humans can observe the normal flow of an object within an aperture.
Donald Hoffman has explained this "aperture problem" as the choice of our visual system to construct the smallest motion. This choice is motion orthogonal to the line that moves [2].
At **aperture 1**, the perceived motion will be pointing to the upper left direction that is perpendicular to the edge. This is because without context, we can only see the movement of the edge (the sharp change of intensity) and conclude that the object is moving perpendicular to its edge.
At **aperture 2**, the perceived motion will be pointing vertically downwards, with the assumption that the two squares are moving with the same speed in the opposite direction. Due to symmetry, the corner formed by two edges of two squares will moving downwards. And because of it, we will perceive that the object is moving downwards since the corner is the motion clue we can see through this aperture.
At **aperture 3**, the perceived motion will be pointing to the right. The reason is similar to aperture 2, with the corner giving us clues about the object's motion.

2. Barber-pole with Occlusion: The figure below illustrates various barber-pole configurations, with occluders placed along either vertical or horizontal sides of the barber-pole. The arrows indicate the perceived direction of barber-pole motion. Explain why the perceived motion tends to be biased in the direction orthogonal to the occlusion boundary.

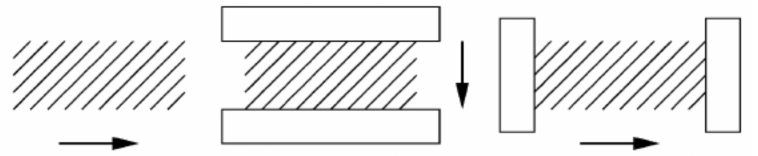
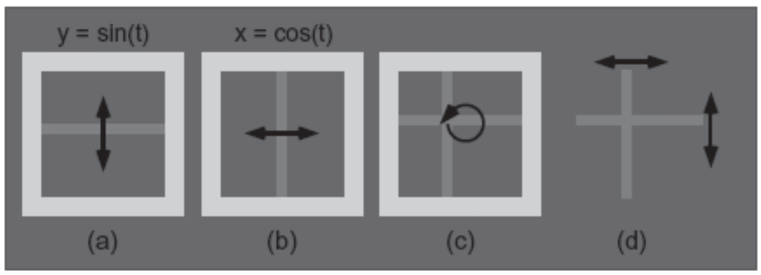


Figure 15: Barber-pole with occlusion. Arrows indicate the perceived motion. Left: Barber-pole with no occluders. Middle: Occluders placed on the top and bottom cause the perceived motion to be mostly vertical. Right: Occluders placed on the left and right sides of the barber-pole bias the perceived motion toward horizontal.

Answer:

For a barber-pole, the diagonally oriented lines are spinning inside the occluder. However, he eyes use the visual cues where the stripes end at the sides of the pole to override any visual depth cues, and therefore the stripes appear to move vertically or horizontally rather than spin.
In addition, the line terminators aligned with the occluders are considered as **extrinsic** terminators, therefore their motion signals tend to become more ambiguous and have less influence on the perceived motion [4]. This is the reason for perceiving the middle scenario as moving downwards. And for the right scenario, it is obvious that more number of intrinsic terminators are moving horizontally so that we perceive it moving rightwards.

3. * Referring to the diagram below, the stimulus consists of two orthogonal bars that move sinusoidally, 90 degree out of phase (Figure a and b). When presented together within an occluding aperture (Figure c), the bars perceptually cohere and appear to move in a circle as a solid cross. However, when presented alone (Figure d), they appear to move separately (the horizontal bar translates vertically and the vertical bar translates horizontally), even though the image motion is unchanged in Figures c and d. In either stimulus condition, both percepts are legitimate interpretations of the image motion. Yet a single interpretation is predominantly seen in each case. Why?



Answer:

In the configuration (d), the bar endpoints are clearly seen by viewers. It can provide unambiguous two dimensional motion signals, which are believed to determine the motion percept. The endpoints move linearly, and each bar follows along.
However, when the frame is present, the T-junctions are formed at the bar endpoints. These junctions provide a cue that the endpoint motions are the spurious result of occlusion. Thus, the motion at T-junctions will have less influence on our perception. With this, the circular motion of the bar

intersection determines the motion percept, as all the local motions in the stimulus apart from those of the endpoints are consistent with such a circular motion.

4. * For a purely rotational motion, the equation that relates the optical flow (u, v) to its rotational parameters $(\omega_x, \omega_y, \omega_z)$ is of the conic form explored in Questions 3*d* and 3*e*:

$$\begin{aligned} u &= \omega_x xy - \omega_y \left(x^2 + 1\right) + \omega_z y \\ v &= \omega_x \left(y^2 + 1\right) - \omega_y xy - \omega_z x \end{aligned} \tag{36}$$

Assume you are given enough optical flow measurements at different (x, y) locations, solve $(\omega_x, \omega_y, \omega_z)$ in the least squares sense by writing down the least squares equation of the form $Ax = b, (b \neq 0)$. Explain why in this case, the equation is of the non-homogeneous form, whereas that in Question 3e is of the homogeneous form $Ax = 0$, and explain whether it is appropriate or inappropriate to change the formulation of this question to the $Ax = 0$ form.

Answer:

The least squares equation is formed as:

$$\begin{bmatrix} xy & -(x^2+1) & y \\ y^2+1 & -xy & -x \end{bmatrix} \begin{pmatrix} \omega_x \\ \omega_y \\ \omega_z \end{pmatrix} = \begin{pmatrix} u \\ v \end{pmatrix} \tag{37}$$

In this question, there are three unknowns and two equations. The matrix is rank-deficient and thus we'd like to use SVD and compute the psedo-inverse. It is not suitable to change to $Ax = 0$ (homogeneous) form because the null vector will not be unique. But in question 3e, which is an over-determined question, we just take the eigenvector of $A^T A$ corresponding to the smallest singular value.

References

[1] R. I. Hartley, *In defense of the eight-point algorithm*, IEEE Transactions on Pattern Analysis and Machine Intelligence, 19 (1997), pp. 580–593.

[2] D. Hoffman, *Visual Intelligence: How We Create What We See*, W. W. Norton, 2000.

[3] R. E. Madsen, L. K. Hansen, and O. Winther, *Singular value decomposition and principal component analysis*, (2004).

[4] P. Sajda and K. Baek, *Integration of form and motion within a generative model of visual cortex*, Neural networks : the official journal of the International Neural Network Society, 17 (2004), pp. 809–21.