Zhu, Zhongjian
University of California, San Diego

March 30, 2018

HOMEWORK 5

Problem 1. Point on line closest to the origin

Given a line $\mathbf{l} = (a, b, c)^\top$, show that the point on $\mathbf{l}$ that is closest to the origin is the point $\mathbf{x} = (-ac, -bc, a^2 + b^2)^\top$.

Solution
The point that on the line and closest to the origin is the intersection between $\mathbf{l}$ and the line that passes the origin and orthogonal to $\mathbf{l}$. The 3D homogeneous coordinate of the origin is $\mathbf{c} = (0, 0, 1)^\top$. So the line that passes the origin and orthogonal to $\mathbf{l}$ is $\mathbf{l}_\perp = (-b, a, 0)^\top$. Their intersection is

$$\mathbf{x} = \mathbf{l} \times \mathbf{l}_\perp = \begin{bmatrix} -ac \\ -bc \\ a^2 + b^2 \end{bmatrix}$$

Results
The point on $\mathbf{l}$ that is closest to the origin is the point $\mathbf{x} = (-ac, -bc, a^2 + b^2)^\top$.

Problem 2. Automatic estimation of the planar projective transformation

(a) **Feature detection**
Calculate an image where each pixel value is the minor eigenvalue of the gradient matrix. Calculate the gradient images using the five-point central difference operator. Set resulting values that are below a specified threshold value to zero. Apply an operation that suppresses local nonmaximum pixel values in the minor eigenvalue image. Determine the subpixel feature coordinate.

Solution
First, in order to get the gradient matrix, we need to filter the image in x- and y-direction. The convolution kernel is $\mathbf{k} = (-1, 8, 0, -8, 1)^\top/12$. So the x-derivative image, I_x, is to filter each row with $\mathbf{k}^\top$ and the y-derivative image, I_y, is to filter each column with $\mathbf{k}$.
Then use the following equation to get the minor eigenvalue for each pixel.

$$\mathbf{N} = \begin{bmatrix} \sum_w I_x^2 & \sum_w I_x I_y \\ \sum_w I_x I_y & \sum_w I_y^2 \end{bmatrix}.$$

$$\lambda_{min} = \frac{Tr(\mathbf{N} - \sqrt{Tr(\mathbf{N})^2 - 4det(\mathbf{N})})}{2}$$

where w is the window about the pixel, and I_x and I_y are the gradient images in the x and y direction, respectively. Set resulting values that are below a specified threshold value to zero. Now we get the minor eigenvalue matrix.

Next step is to apply the non-maxium suppression. For each pixel, we set a window on it and calculate the maximum value in the window, the local maximum. If the value of the center pixel of the window is less than the local maximum, set it to 0, otherwise, keep the value. This step is actually to find the corner pixels and set the value of non-corner pixels to 0.

Since what we need is not the corner pixel but rather the corner coordinates, we need to use the Föstner corner point operator to find the subpixels.

$$\begin{bmatrix} \sum_w I_x^2 & \sum_w I_x I_y \\ \sum_w I_x I_y & \sum_w I_y^2 \end{bmatrix} \begin{bmatrix} x_{corner} \\ y_{corner} \end{bmatrix} = \begin{bmatrix} \sum_w (x I_x^2 + y I_x I_y \\ \sum_w (x I_x I_y + y I_y^2 \end{bmatrix}$$

x_{corner} and y_{corner} are the coordinates of the subpixel.

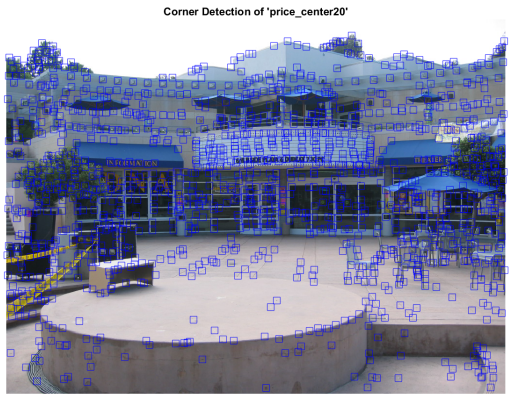
Result

The size of the feature detection window is 9×9, the minor eigenvalue threshold value is 5.57, the size of the local non-maximum suppression window is 9×9.

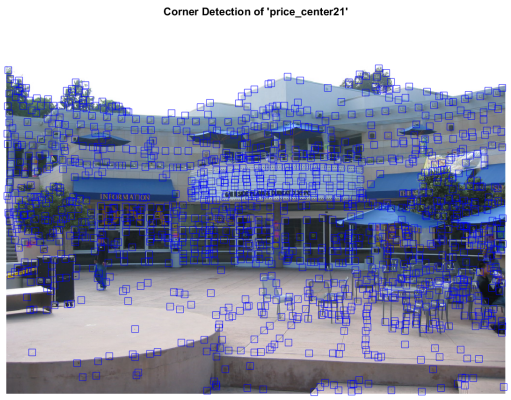
The number of features detected in *IMG_5030* is 1399, and the number of features detected in *IMG_5031* is 1350.

(a) (b)

Figure 1: Input images. (a) IMG_5030. (b) IMG_5031.



(a)



(b)

Figure 2: Detected corners. (a) IMG_5030. (b) IMG_5031.

(b) **Feature matching**

Determine the set of one-to-one putative feature correspondences by performing a brute-force search for the greatest correlation coefficient value between windows centered about the detected features in image 1 and windows centered about the detected features in image 2. Only allow matches that are above a specified correlation coefficient threshold value. Further, only allow matches that are above a specified distance ratio threshold value, where distance is measured to the next best match for a given feature. Vary these parameters such that around 300 putative feature correspondences are established. Optional: constrain the search to coordinates in image 2 that are within a proximity of the detected feature coordinates in image 1.

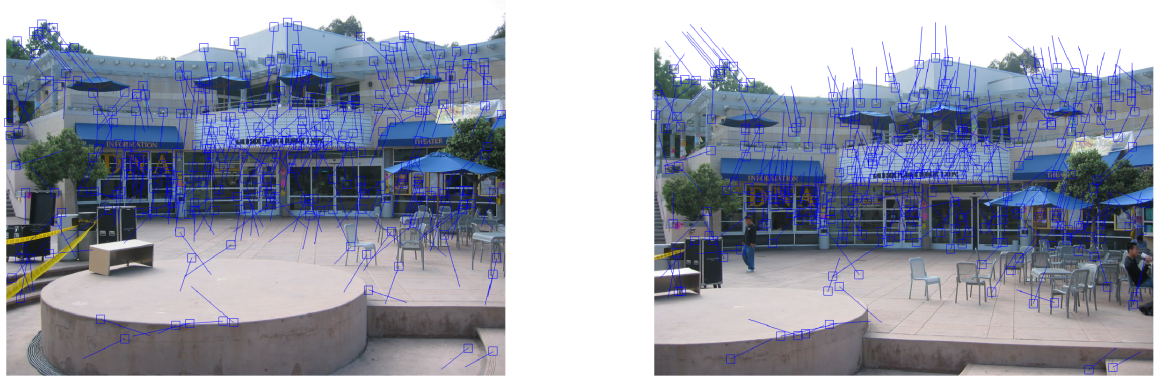
Solution

First we need to calculate the correlation coefficient between a given window in image 1 and all windows in image 2. Here since the subpixel value do not exist in image coordinate, we need to use interpolation.

Next step is the one-to-one matching. The algorithm is as follow:

```
1  Find indices of element with maximum value
2    If similarity threshold < maximum value
3      Store the best match value
4      Set element to -1
5      Find next best match value as
6      max(next best match value in row, next best match value in column)
7      If (1-best match value)<(1-next best match value)*distance ratio threshold
8        Store feature value
9      else
10       Match is not unique enough
11     Set row and column to -1
```

Result
The size of the proximity window is 120, the size of the matching window is 11×11, the correlation coefficient threshold value is 0.6, the distance ratio threshold value is 0.7, and the resulting number of putative feature correspondences is 300.



(a) (b)

Figure 3: Matched features. (a) IMG_5030. (b) IMG_5031.

(c) **Outlier rejection**
The resulting set of putative point correspondences contain both inlier and outlier correspondences. Determine the set of inlier point correspondences using the M-estimator Sample Consensus (MSAC) algorithm, where the maximum number of attempts to find a consensus set is determined adaptively. Use the 4-point algorithm to estimate the planar projective transformation from the 2D points in image 1 to the 2D points in image 2. Calculate the Sampson error as a first order approximation to the geometric error.

Solution
The adaptive number of trials MSAC algorithm can be concluded as follow:

```
consensus_min_cost = inf
max_trials = inf
for(trials=0;trials<max_trials && consensus_min_cost>threshold;++trials)
{
  select a random sample
  calculate the model
  calculate the error
  calculate the cost
  if(consensus_min_cost<consensus_min_cost)
  {
```

```
11        consensus_min_cost = consensus_cost
12        consensus_min_cost_model = model
13        number of inliers
14        w = # of inliers / # of data points
15        max_trials = log(1-p) / log(1-w^s)
16      }
17    }
```

Since the degree of freedom of $\mathbf{F}$ is 7, to calculate the model, we need 7 points. Randomly select 7 points and do data normalization

$$\mathbf{T} = \begin{bmatrix} s & 0 & -\mu_{\tilde{\mathbf{x}}} s \\ 0 & s & -\mu_{\tilde{\mathbf{y}}} s \\ 0 & 0 & 1 \end{bmatrix},$$

where

$$\sigma^2 = \sigma_{\tilde{\mathbf{x}}}^2 + \sigma_{\tilde{\mathbf{y}}}^2, s = \frac{\sqrt{2}}{\sigma}.$$

Then solve the following equation for $\mathbf{f}$

$$\begin{bmatrix} \mathbf{x}_1^{'\top} \otimes \mathbf{x}_1^{\top} \\ \mathbf{x}_2^{'\top} \otimes \mathbf{x}_2^{\top} \\ \vdots \\ \mathbf{x}_7^{'\top} \otimes \mathbf{x}_7^{\top} \end{bmatrix} \mathbf{f} = \mathbf{A}\mathbf{f} = \mathbf{0},$$

where $\mathbf{f} = vec(\mathbf{F}^{\top})$.
Since the fundamental matrix has an rank-2 constrain, we can use the SVD algorithm to find two least-squares solution, $\mathbf{F_1}$ and $\mathbf{F_2}$. And set $\mathbf{F} = \alpha\mathbf{F_1} + \mathbf{F_2}$. Then set $det(\mathbf{F}) = 0$ and solve for α. It will have 1 or 3 solutions. Substitute one real solution into $\mathbf{F} = \alpha\mathbf{F_1} + \mathbf{F_2}$ and get $\hat{\mathbf{F}}$. This is the normalized fundamental matrix since we use normalized data. So we need to denormalize it.

$$\mathbf{F} = \mathbf{T}^{'\top}\hat{\mathbf{F}}\mathbf{T}.$$

The error used here is the Sampson error which is the first order approximation of thee geometric error.

$$\|\delta_{\mathbf{x}}\|^2 = \epsilon(\mathbf{J}\mathbf{J}^{\top})^{-1}\epsilon = \frac{(\mathbf{x}^{'\top}\mathbf{F}\mathbf{x})^2}{(\mathbf{x}^{'\top}\mathbf{f}_1)^2 + (\mathbf{x}^{'\top}\mathbf{f}_2)^2 + (\mathbf{f}^{1\top}\mathbf{x})^2 + \mathbf{f}^{2\top}\mathbf{x})^2}$$

where

$$\mathbf{J} = \frac{\partial C_{\mathbf{F}}(\mathbf{x})}{\partial \tilde{\mathbf{x}}} = (\tilde{x}'f_{11} + \tilde{y}'f_{21} + f_{31}, \tilde{x}'f_{12} + \tilde{y}'f_{22} + f_{32}, \tilde{x}f_{11} + \tilde{y}f_{12} + f_{13}, \tilde{x}f_{21} + \tilde{y}f_{22} + f_{23})$$

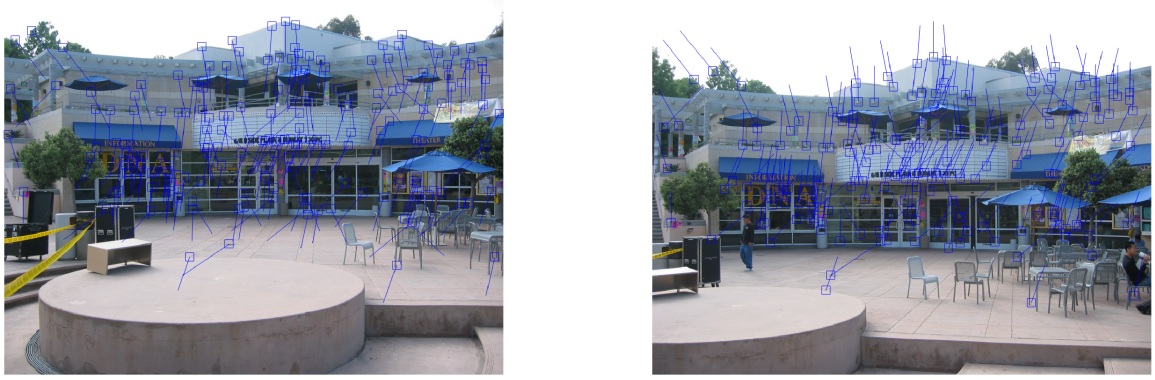
and

$$\epsilon = C_{\mathbf{F}}(\mathbf{x}) = \tilde{x}\tilde{x}'f_{11} + \tilde{x}\tilde{y}'f_{21} + \tilde{x}f_{31} + \tilde{y}\tilde{x}'f_{12} + \tilde{y}\tilde{y}'f_{22} + \tilde{y}f_{32} + \tilde{x}'f_{13} + \tilde{y}f_{23} + f_{33}$$

is the cost associated with $\mathbf{x}$.

Next step is to calculate the cost. First we set a tolerance. Here we use $t^2 = F_m^{-1}(\alpha)$ where t^2 is the mean squared distance threshold and $F_m^{-1}(\alpha)$ is the inverse chi-squared cumulative distribution function. We assume the probability that a data point is an inlier is 0.95, $\alpha = 0.95$, and the variance of the measurement error is 1, $\sigma^2 = 1$. The codimension $m = 1$. For points whose error is less or equal to the tolerance, we add the error to the cost and for those whose error is greater than the tolerance, we add the tolerance to the cost. The points whose error are less or equal to the tolerance are inliers and the others are outliers. Then if the cost is less than the previous cost, we keep the cost, the camera pose and the number of inliers. Then update the maximum number of trials.

Result
The total number of the inliers are dependent on the choose of the points used to calculate the model, so we will have different number of inliers at each trial. Here we assumed that the probability p that at least one of the random samples does not contain any outliers is 0.99, the probability α that a given data point is an inlier is 0.95 and the variance σ^2 of the measurement error is 1. In the code file, I keep one random seed which leads to 189 inliers. The number of maximum trials is 114.5947, so it runs 115 times to find the consensus set.



(a) (b)

Figure 4: Matched features of inliers. (a) IMG_5030. (b) IMG_5031.

Since the projection of points in image 1 is a line in image 2 under the fundamental matrix, the error can be considered as the distance between the points in image 2 and the line. This constrain is not strict. So, there will exist some outliers in the image even after outlier rejection as we can see from figure 4.

(d) **Linear estimation**
Estimate the fundamental matrix $\mathbf{F}_{DLT}$ from the resulting set of inlier correspondences using the direct linear transformation (DLT) algorithm (with data normalization).

Solution
For DLT algorithm, we need first normalize data point by

$$\mathbf{x} = \begin{bmatrix} s & 0 & -\mu_{\tilde{\mathbf{x}}} s \\ 0 & s & -\mu_{\tilde{\mathbf{y}}} s \\ 0 & 0 & 1 \end{bmatrix} \mathbf{x}$$

where

$$\sigma^2 = \sigma_{\tilde{\mathbf{x}}}^2 + \sigma_{\tilde{\mathbf{y}}}^2, s = \frac{\sqrt{2}}{\sigma}$$

Then we need to solve the equation below:

$$\begin{bmatrix} \mathbf{x}_1^{'\top} \otimes \mathbf{x}_1^\top \\ \mathbf{x}_2^{'\top} \otimes \mathbf{x}_2^\top \\ \vdots \\ \mathbf{x}_n^{'\top} \otimes \mathbf{x}_n^\top \end{bmatrix} \mathbf{f} = \mathbf{A}\mathbf{f} = \mathbf{0},$$

Using Singular Value Decomposition we can get

$$\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^\top$$

and $\mathbf{f}$ is the last column of $\mathbf{V}$. Then, reshape $\mathbf{f}$ to a 3×3 matrix.
Since the rank of $\mathbf{F}$ is 2, we should map the result to the closet rank 2 matrix.

$$\hat{\mathbf{F}} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^\top = \mathbf{U}diag(\sigma_1, \sigma_2, 0)\mathbf{V}^\top$$

Finally denormalize $\hat{\mathbf{F}}$ and divide it by its norm.

Result

$$\mathbf{F}_{DLT} = \begin{bmatrix} 2.1808477322764 \times 10^{-8} & 4.01626269895722 \times 10^{-6} & -0.00153201201977501 \\ -2.86980652524888 \times 10^{-6} & 9.96423811864565 \times 10^{-7} & -0.00988842520376557 \\ 0.0013296079950874 & 0.00850530562163636 & 0.999912878144723 \end{bmatrix}.$$

(e) **Nonlinear estimation**
Use $\mathbf{F}_{DLT}$ and the triangulated 3D points as an initial estimate to an iterative estimation method, specifically the sparse Levenberg-Marquardt algorithm, to determine the Maximum Likelihood estimate of the fundamental matrix $\mathbf{F}$ that minimizes the reprojection error. The initial estimate of the 3D points must be determined using two-view optimal triangulation method. Additionally, parameterize the homogeneous 3D scene points that are being adjusted using the parameterization of homogeneous vectors.

Solution
The parameter vector is $(\hat{\omega}_{\mathbf{U}}^\top, \hat{\omega}_{\mathbf{V}}^\top, \hat{s}, \hat{\mathbf{X}_1}^\top, \hat{\mathbf{X}_2}^\top, ..., \hat{\mathbf{X}_n}^\top)$. The first three elements are the parameter of the fundamental matrix. The rest of the parameter vector is the parameterized 3D scene points.

To parameterize $\mathbf{F}$, first do SVD

$$\mathbf{F} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^\top.$$

Check if $det(\mathbf{U}) < 1$, $\mathbf{U} = -\mathbf{U}$, if $det(\mathbf{V}) < 1$, $\mathbf{V} = -\mathbf{V}$. $\omega_\mathbf{U}$ and $\omega_\mathbf{V}$ is the Angle-Axis representation to $\mathbf{U}$ and $\mathbf{V}$. s is the parameterization of homogeneous vector Σ.

3D scene points are determined using two-view optimal triangulation method. For each point in image 1, its projection on image 2 is a 2D line. Theoretically, this line will pass the corresponding point in image 2. However, because of the noise or other measurement error, the line will not exactly pass the point. As a result, we need first find the corrected correspondences $\hat{\mathbf{x}} \leftrightarrow \hat{\mathbf{x}}'$ that minimize the geometric error subject to the epipolar constrain $\hat{\mathbf{x}}^\top \mathbf{F} \hat{\mathbf{x}}' = 0$.

i. Define transformation matrices that take the points to the origin.

$$\mathbf{T} = \begin{bmatrix} w & 0 & -x \\ 0 & w & -y \\ 0 & 0 & 2 \end{bmatrix}, \mathbf{T}' = \begin{bmatrix} w' & 0 & -x' \\ 0 & w' & -y' \\ 0 & 0 & 2 \end{bmatrix}.$$

ii. Replace $\mathbf{F}$ by $\mathbf{T}'^{-\top}\mathbf{F}\mathbf{T}^{-1}$.

iii. Compute the right and left epipoles $\mathbf{e} = (e_1, e_2, e_3)^\top$ and $\mathbf{e}' = (e'_1, e'_2, e'_3)^\top$ such that $\mathbf{e}'\mathbf{F} = \mathbf{0}$ and $\mathbf{F}\mathbf{e} = \mathbf{0}$. Normalize $\mathbf{e}$ such that $e_1^2 + e_2^2 = 1$ and do the same to $\mathbf{e}'$.

iv. Form matrices

$$\mathbf{R} = \begin{bmatrix} e_1 & e_2 & 0 \\ -e_2 & e_1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \mathbf{R}' = \begin{bmatrix} e'_1 & e'_2 & 0 \\ -e'_2 & e'_1 & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$

v. Get $\mathbf{F}_s$ by $\mathbf{R}'\mathbf{F}\mathbf{R}^\top$.

vi. Form the polynomial

$$g(t) = t((at+b)^2 + f'^2(ct+d)^2)^2 - (ad-bc)(1+f^2t^2)^2(at+b)(ct+d) = 0$$

solve for t and get 6 roots, where

$$f = e_3, f' = e'_3, a = f_{s,22}, b = f_{s,23}, c = f_{s,32}, d = f_{s,33}.$$

vii. Evaluate the cost function

$$s(t) = \frac{t^2}{1+f^2t^2} + \frac{(ct+d)^2}{(at+b)^2 + f'^2(ct+d)^2}$$

at each real part of each root t. Select the t_{min} that gives the smallest value of $s(t)$.

viii. Evaluate two lines $\mathbf{l} = (tf, 1, -t)^\top$ and $\mathbf{l}' = (-f'(ct+d), at+b, ct+d)^\top$ at t_{min} and find $\hat{\mathbf{x}}$ and $\hat{\mathbf{x}}'$ as the closest point on these lines to the origin.

ix. Transfer back to the original coordinates

$$\hat{\mathbf{x}} = \mathbf{T}^{-1}\mathbf{R}^\top\hat{\mathbf{x}}_s, \hat{\mathbf{x}}' = \mathbf{T}'^{-1}\mathbf{R}'^\top\hat{\mathbf{x}}'_s$$

Then we use $\hat{\mathbf{x}}$ and $\hat{\mathbf{x}}'$ as input which exactly satisfy the epipolar constraint to determine 3D scene points.

i. Map point $\hat{\mathbf{x}}$ to line $\mathbf{l}'$ under fundamental matrix $\mathbf{F}$, $\mathbf{l}' = \mathbf{F}\hat{\mathbf{x}} = (a', b', c')^\top$.

ii. Determine the line $\mathbf{l}'_\perp = (-b'w', a'w', b'x' - a'y')^\top$ which is orthogonal to $\mathbf{l}$ and passes $\hat{\mathbf{x}}'$.

iii. Back project $\mathbf{l}'_\perp$ to a plane $\pi = \mathbf{P}'^\top \mathbf{l}'_\perp = (a, b, c, d)^\top$. $\mathbf{P}' = [\mathbf{m}|\mathbf{e}']$ where $\mathbf{m} = \mathbf{U}\mathbf{Z}diag(\mathbf{d}')\mathbf{V}^\top$, $\mathbf{e}' = -\mathbf{u}_3$,

$$\mathbf{Z} = \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \mathbf{d}' = \begin{bmatrix} \sigma_1 \\ \sigma_2 \\ \frac{\sigma_1+\sigma_2}{2} \end{bmatrix}.$$

iv. Back project point $\hat{\mathbf{x}}$ to 3D line defined by two 3D points

$$\mathbf{P}\mathbf{C} = \mathbf{0}, \mathbf{X} = \mathbf{P}^+\hat{\mathbf{x}}.$$

v. The intersection of 3D line and plane is

$$\mathbf{X}_\pi = \begin{bmatrix} X_2(bY_1 + cZ_1 + dT_1) & - & X_1(bY_2 + cZ_2 + dT_2) \\ Y_2(aX_1 + cZ_1 + dT_1) & - & Y_1(aX_2 + cZ_2 + dT_2) \\ Z_2(aX_1 + bY_1 + dT_1) & - & Z_1(aX_2 + bY_2 + dT_2) \\ T_2(aX_1 + bY_1 + cZ_1) & - & T_1(aX_2 + bY_2 + cZ_2) \end{bmatrix}.$$

Since $\mathbf{P} = [\mathbf{I}|\mathbf{0}]$ and $\mathbf{C} = (0, 0, 0, 1)^\top$, $\mathbf{X}_\pi = (dx, dy, dw, -(ax + by + cw))^\top$.

Now we have all parameters. The measurement vector is $(\hat{\tilde{\mathbf{x}}}_1^\top, \hat{\tilde{\mathbf{x}}}_2^\top, ..., \hat{\tilde{\mathbf{x}}}_n^\top, \hat{\tilde{\mathbf{x}}}_1'^\top, \hat{\tilde{\mathbf{x}}}_2'^\top, ..., \hat{\tilde{\mathbf{x}}}_n'^\top)^\top$. The cost is $\epsilon^\top \Sigma_{\tilde{\mathbf{x}}}^{-1} \epsilon + \epsilon' \Sigma_{\tilde{\mathbf{x}}'}^{-1} \epsilon''$. We assume $\Sigma_{\tilde{\mathbf{x}_i}}$ is an identity matrix.

For Jacobian matrix, here it is very sparse, so there is no need to calculate the full matrix. We only need to calculate three terms:

$$\mathbf{A}'_i = \frac{\partial \hat{\tilde{\mathbf{x}}}'_i}{\partial(\hat{\mathbf{w}}_\mathbf{U}^\top, \hat{\mathbf{w}}_\mathbf{V}^\top, \hat{s})}, \mathbf{B}'_i = \frac{\partial \hat{\tilde{\mathbf{x}}}_i}{\partial \hat{\mathbf{X}}_i}, \mathbf{B}''_i = \frac{\partial \hat{\tilde{\mathbf{x}}}'_i}{\partial \hat{\mathbf{X}}_i}.$$

For normal equations matrix, $\mathbf{J}^\top \Sigma_\mathbf{x}^{-1} \mathbf{J}$, we only need to calculate three terms:

$$\mathbf{U}' = \sum_i \mathbf{A}_i'^\top \Sigma_{\tilde{\mathbf{x}}_i}^{-1} \mathbf{A}'_i,$$

$$\mathbf{V}_i = \mathbf{B}_i^\top \Sigma_{\tilde{\mathbf{x}}_i}^{-1} \mathbf{B}_i + \mathbf{B}_i'^\top \Sigma_{\tilde{\mathbf{x}}'_i}^{-1} \mathbf{B}'_i,$$

$$\mathbf{W}'_i = \mathbf{A}_i'^\top \Sigma_{\tilde{\mathbf{x}}'_i}^{-1} \mathbf{B}'_i.$$

The normal equations vector is $(\epsilon_{\mathbf{a}'}^\top, \epsilon_{\mathbf{b}_1}^\top, \epsilon_{\mathbf{b}_2}^\top, ..., \epsilon_{\mathbf{b}_n})^\top$ where

$$\epsilon_{\mathbf{a}'} = \sum_i \mathbf{A}_i'^\top \Sigma_{\tilde{\mathbf{x}}'_i}^{-1} \epsilon'_i,$$

$$\epsilon_{\mathbf{b}_i} = \mathbf{B}_i^\top \Sigma_{\tilde{\mathbf{x}}'_i}^{-1} \epsilon_i + \mathbf{B}_i'^\top \Sigma_{\tilde{\mathbf{x}}'_i}^{-1} \epsilon'_i.$$

The augmented normal equations are

$$\mathbf{S}' = \mathbf{U}' + \lambda \mathbf{I} - \sum_i \mathbf{W}'_i (\mathbf{V}_i + \lambda \mathbf{I})^\top \mathbf{W}_i'^\top,$$

$$\mathbf{e}' = \epsilon_{\mathbf{a}'} - \sum_i \mathbf{W}'_i (\mathbf{V}_i + \lambda \mathbf{I})^{-1} \epsilon_{\mathbf{b}_i}$$

So we can get $\delta_{\mathbf{a}'}$ by solving $\mathbf{S}'\delta_{\mathbf{a}'} = \mathbf{e}'$ and then $\delta_{\mathbf{b}_i} = (\mathbf{V}_i + \lambda\mathbf{I})^{-1}(\epsilon_{\mathbf{b}_i} - \mathbf{W}_i'^{\top}\delta_{\mathbf{a}'})$. Add these two to the parameter vector to get the new parameter vector. Deparameterize the parameter vector and project the scene points to image 1 and image 2. Calculate the current cost and compare it the with previous cost. If the current cost is less than the previous cost, keep the parameter vector and error vector and set $\lambda = 0.1\lambda$, else set $\lambda = 10\lambda$. Iteration until the difference of cost between two iterations is less than 0.0001.

Result
The costs of each iteration are as follow:

Iteration	Cost
0	87.4687
1	82.3936
2	82.0508
3	82.0508

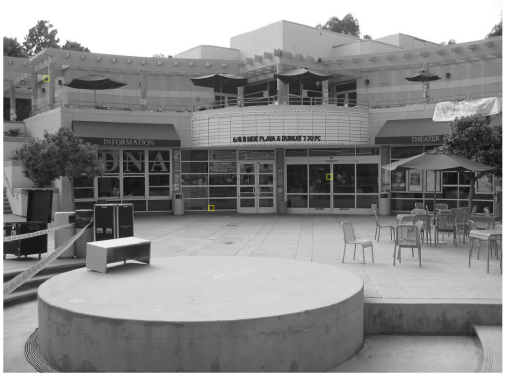
The $\mathbf{F}_{LM}$ is

$$\mathbf{F}_{LM} = \begin{bmatrix} -2.11500599391966 \times 10^{-8} & -4.05842511471236 \times 10^{-6} & 0.00153254741224547 \\ 2.91245066595171 \times 10^{-6} & -9.88134279228048 \times 10^{-7} & 0.00995646117565414 \\ -0.00133567364832102 & -0.0085811676441079 & -0.999911545933502 \end{bmatrix}.$$

(f) **Point to line mapping**
Qualitatively determine the accuracy of $\mathbf{F}_{LM}$ by mapping points in image 1 to epipolar lines in image 2. Choose three distinct points $\mathbf{x}_{1,2,3}$ distributed in image 1 that are not in the set of inlier correspondences and map them to epipolar lines $\mathbf{l}'_{1,2,3} = \mathbf{F}_{LM}\mathbf{x}_{1,2,3}$ in the second image under the fundamental matrix $\mathbf{F}_{LM}$.

Solution
To quantitatively evaluate the accuracy of $\mathbf{F}_{LM}$, we can calculate the distance between the matched points in image 2 and the epipolar line projected by points in image 1. To qualitatively determine the accuracy, we can see whether the epipolar lines pass the real corresponding points in image 2.

Result
As we can see from figure 5(b), the epipolar lines do not pass the circled points since they are outliers. The distances are 24.2748, 16.4754 and 16.7165 respectively. If we look at the real correspondences in image 2, which we can find manually, the epipolar lines do pass these points or very close to these points. As a result, the fundamental matrix is accurate for inliers but does not work for outliers.



(a)

(b)

Figure 5: (a) IMG_5030. (b) IMG_5031.

Appendix

Listing 1: Part(a)

```
clear;clc;
I0 = imread('../data/IMG_5030.JPG');
I1 = imread('../data/IMG_5031.JPG');
%% Corner Detection
fprintf('Detecting corners...')
win_detect = 9;
win_spr = 9;
eigen_th = 5.57;
[r0,c0,x_f0,y_f0] = CornerCoordinate(I0,win_detect,win_spr,eigen_th);
[r1,c1,x_f1,y_f1] = CornerCoordinate(I1,win_detect,win_spr,eigen_th);
save('../data/c.mat','x_f0','y_f0','x_f1','y_f1');
fprintf('Done')
%% Plot
figure
imshow(I0);
% saveas(gcf,'Input0.png');
figure
imshow(I1);
% saveas(gcf,'Input1.png');
figure
imshow(I0);hold on
plot(y_f0,x_f0,'bs','MarkerSize',win_detect);hold off
% saveas(gcf,'CornerDetection0.png');
```

```
figure
imshow(I1);hold on
plot(y_f1,x_f1,'bs','MarkerSize',win_detect);hold off
% saveas(gcf,'CornerDetection1.png');
%% Print
disp(['Features in the first image: ',num2str(numel(x_f0))]);
disp(['Features in the second image: ',num2str(numel(x_f1))]);
```

Listing 2: Function

```
function [r,c,x_f1,x_f2] = CornerCoordinate(im,win1,win2,threshold)
im = rgb2gray(im);
A = zeros(size(im));
k = [-1 8 0 -8 1]' / 12;
imx = double(imfilter(im,k','symmetric'));
imy = double(imfilter(im,k ,'symmetric'));
%% Gradient matrix
for i = (win1+1)/2 : size(im,1) - (win1-1)/2
    for j = (win1+1)/2 : size(im,2) - (win1-1)/2
        im_win_x = imx(i - (win1-1)/2:i + (win1-1)/2,...
                       j - (win1-1)/2:j + (win1-1)/2);
        im_win_y = imy(i - (win1-1)/2:i + (win1-1)/2,...
                       j - (win1-1)/2:j + (win1-1)/2);

        N = zeros(2,2);
        N(1,1) = sum(sum(im_win_x.^2));
        N(1,2) = sum(sum(im_win_x .* im_win_y));
        N(2,1) = N(1,2);
        N(2,2) = sum(sum(im_win_y.^2));
        N = N/win1^2;
        lambda = (trace(N) - sqrt(trace(N)^2 - 4*det(N)))/2;
        if lambda > threshold
            A(i,j) = lambda;
        end
    end
end
%% Non-maximum suppression
B = ordfilt2(A,win1^2,ones(win1,win1));
C = (B == A & B ~= 0);
[r,c] = find(C);
x_f1 = zeros(size(r));
x_f2 = zeros(size(r));
%% find corner
for k = 1:numel(r)
    T = zeros(2,2);
    y = zeros(2,1);
    im_win_x = imx(r(k) - (win2-1)/2:r(k) + (win2-1)/2,...
```

```
                    c(k) - (win2-1)/2:c(k) + (win2-1)/2);
    im_win_y = imy(r(k) - (win2-1)/2:r(k) + (win2-1)/2,...
                    c(k) - (win2-1)/2:c(k) + (win2-1)/2);

    xx = [r(k) - (win2-1)/2 : r(k) + (win2-1)/2]' .* ones(win2);
    yy = [c(k) - (win2-1)/2 : c(k) + (win2-1)/2] .* ones(win2);

    y(1) = sum(sum( xx.*im_win_x.^2 + yy.*im_win_x.*im_win_y ));
    y(2) = sum(sum( xx.*im_win_x.*im_win_y + yy.*im_win_y.^2 ));
    T(1,1) = sum(sum(im_win_x.^2));
    T(1,2) = sum(sum(im_win_x .* im_win_y));
    T(2,1) = T(1,2);
    T(2,2) = sum(sum(im_win_y.^2));
    x = T\y;

    x_f1(k) = x(1);
    x_f2(k) = x(2);
end
end
```

Listing 3: Part(b)

```
close all; clear; clc;
load('../data/c.mat');
I0 = imread('../data/IMG_5030.JPG');
I1 = imread('../data/IMG_5031.JPG');
%% Feature Matching
fprintf('Matching features...')
win_match = 11;
simi_th = 0.6;
dist_th = 0.7;
[X0,Y0,X1,Y1] = FeatureMatch(I0,I1,x_f0,y_f0,x_f1,y_f1,win_match,simi_th,dist_th);
x1_inhomo = [Y0';X0'];
x2_inhomo = [Y1';X1'];
save('../data/x.mat','x1_inhomo','x2_inhomo','win_match');
fprintf('Done\n')
disp(['Matched featrues: ',num2str(numel(X0))]);
%% Plot
figure
imshow(insertShape(I0,'Line',[Y0 X0 Y1 X1],'Color','blue'));hold on
plot(Y0,X0,'bs','MarkerSize',win_match);hold off
% saveas(gcf,'FeatureMatch0.png');
figure
imshow(insertShape(I1,'Line',[Y1 X1 Y0 X0],'Color','blue'));hold on
plot(Y1,X1,'bs','MarkerSize',win_match);hold off
% saveas(gcf,'FeatureMatch1.png');
```

Listing 4: Function

```
function [X0,Y0,X1,Y1] = FeatureMatch(im0,im1,x0,y0,x1,y1,win,simi_th,dist_th)
im0 = double(rgb2gray(im0));
im1 = double(rgb2gray(im1));
half_win = (win-1)/2;
%%
xx0 = x0+half_win;
xx1 = x1+half_win;
yy0 = y0+half_win;
yy1 = y1+half_win;
im0 = padarray(im0,[half_win,half_win],'symmetric');
im1 = padarray(im1,[half_win,half_win],'symmetric');
%% Interpolation
xc = zeros(numel(xx0),numel(xx1));
crd = zeros(numel(xx0),numel(xx1));
im0_win_intep = zeros(win,win,numel(xx0));
im1_win_intep = zeros(win,win,numel(xx1));
for i = 1:numel(xx0)
    XX0 = fix(xx0(i)) - half_win:ceil(xx0(i)) + half_win;
    YY0 = fix(yy0(i)) - half_win:ceil(yy0(i)) + half_win;
    im0_win = im0(XX0,YY0);
    [XX0,YY0] = meshgrid(XX0,YY0);
    [Xq,Yq] = meshgrid(xx0(i)-half_win:xx0(i)+half_win,...
        yy0(i) - half_win:yy0(i) + half_win);
    im0_win_intep(:,:,i) = interp2(XX0,YY0,im0_win,Xq,Yq,'linear');
end

for j = 1:numel(xx1)
    XX1 = fix(xx1(j)) - half_win:ceil(xx1(j)) + half_win;
    YY1 = fix(yy1(j)) - half_win:ceil(yy1(j)) + half_win;
    im1_win = im1(XX1,YY1);
    [XX1,YY1] = meshgrid(XX1,YY1);
    [Xq,Yq] = meshgrid(xx1(j)-half_win:xx1(j)+half_win,...
        yy1(j) - half_win:yy1(j) + half_win);
    im1_win_intep(:,:,j) = interp2(XX1,YY1,im1_win,Xq,Yq,'linear');
end
%% Xcorrelation matrix
for i = 1:numel(xx0)
    for j = 1:numel(xx1)
        if sqrt( (xx0(i)-xx1(j))^2 + (yy0(i)-yy1(j))^2) > 120
            xc(i,j) = -1;
        else
            xc(i,j) = corr2(im0_win_intep(:,:,i),im1_win_intep(:,:,j));
        end
    end
```

```
end
%% One-to-One Matching
while max(xc(:)) > simi_th
    [r,c] = find(xc == max(xc(:)));
    i = r(1);
    j = c(1);
    mx = xc(i,j);
    xc(i,j) = -1;
    next_mx = max(max(xc(i,:),[],2),max(xc(:,j)));
    if (1-mx) < (1-next_mx)*dist_th
        crd(i,j) = 1;
    end
    xc(i,:) = -1;
    xc(:,j) = -1;
end
[i,j] = find(crd);
X0 = x0(i);
Y0 = y0(i);
X1 = x1(j);
Y1 = y1(j);
end
```

Listing 5: Part(c)

```
close all; clear; clc;
load('../data/x.mat');
I0 = imread('../data/IMG_5030.JPG');
I1 = imread('../data/IMG_5031.JPG');
x1 = padarray(x1_inhomo,[1 0],1,'post');
x2 = padarray(x2_inhomo,[1 0],1,'post');
n = size(x1,2);
%%
syms alpha
syms a1 a2 a3 a4 a5 a6 a7 a8 a9
syms b1 b2 b3 b4 b5 b6 b7 b8 b9
tmp_F1 = [a1,a4,a7
          a2,a5,a8
          a3,a6,a9];
tmp_F2 = [b1,b4,b7
          b2,b5,b8
          b3,b6,b9];
t = det(alpha*tmp_F1+tmp_F2);
f = matlabFunction(flip(coeffs(t,alpha)));
%% mSAC
consensus_min_cost = inf;
max_trials = inf;
trials = 0;
```

```
threshold = 0;
tolerance = chi2inv(0.95,1) * 1;
p = 0.99;
s = 7;
rng(4)
while trials < max_trials && consensus_min_cost > threshold
    %% Select a random sample
    t = randperm(n,s)';
    x1_rand =  x1(:,t);
    x2_rand =  x2(:,t);
    % x1
    mu_x1 = mean(x1_rand,2);
    sigma_x1 = sum(var(x1_rand,0,2));
    s_x1 = sqrt(2/sigma_x1);
    T1 = [s_x1       0    -mu_x1(1)*s_x1
        0         s_x1    -mu_x1(2)*s_x1
        0          0          1        ];
    x1_norm = T1*x1_rand;
    % x2
    mu_x2 = mean(x2_rand,2);
    sigma_x2 = sum(var(x2_rand,0,2));
    s_x2 = sqrt(2/sigma_x2);
    T2 = [s_x2       0    -mu_x2(1)*s_x2
        0         s_x2    -mu_x2(2)*s_x2
        0          0          1        ];
    x2_norm = T2*x2_rand;
    %% Calculate model
    A = zeros(s,9);
    for i = 1:s
        A(i,:) = kron(x2_norm(:,i)', x1_norm(:,i)');
    end
    [~,~,V] = svd(A);
    F1 = reshape(V(:,end),   3,3)';
    F2 = reshape(V(:,end-1),3,3)';
    alpha = roots(f(F1(1),F1(2),F1(3),F1(4),F1(5),F1(6),F1(7),F1(8),F1(9),...
                    F2(1),F2(2),F2(3),F2(4),F2(5),F2(6),F2(7),F2(8),F2(9)));
    for i = 1:3
        if isreal(alpha(i))
            F_norm = alpha(i)*F1 + F2;
            break
        end
    end
    F = T2'*F_norm*T1;
    %% Error for each point
    error = SampsonError(x1,x2,F);
    %% Calculate cost
```

```
70      cost = sum(error .* (error <= tolerance) + tolerance * (error > tolerance));
71      %% Update maximum trials
72      if cost < consensus_min_cost
73          consensus_min_cost = cost;
74          F_min_cost = F;
75          inliers = error <= tolerance;
76          w = sum(inliers) / n;
77          max_trials = log(1-p) / log(1-w^s);
78      end
79      trials = trials + 1;
80  end
81  %% Save in- & out-liers
82  x1_inlier_inhomo = x1_inhomo(:,inliers);
83  x2_inlier_inhomo = x2_inhomo(:,inliers);
84  save('../data/x_inlier.mat','x1_inlier_inhomo','x2_inlier_inhomo');
85  x1_inhomo(:,inliers) = [];
86  x2_inhomo(:,inliers) = [];
87  x1_outlier_inhomo = x1_inhomo;
88  x2_outlier_inhomo = x2_inhomo;
89  save('../data/x_outlier.mat','x1_outlier_inhomo','x2_outlier_inhomo');
90  %% Plot
91  figure
92  imshow(insertShape(I0,'Line',[x1_inlier_inhomo',x2_inlier_inhomo'],...
93      'Color','blue'));hold on
94  plot(x1_inlier_inhomo(1,:)',x1_inlier_inhomo(2,:)',...
95      'bs','MarkerSize',win_match);hold off
96  % saveas(gcf,'FeatureMatch_inlier0.png');
97
98  figure
99  imshow(insertShape(I1,'Line',[x2_inlier_inhomo',x1_inlier_inhomo'],...
100     'Color','blue'));hold on
101 plot(x2_inlier_inhomo(1,:)',x2_inlier_inhomo(2,:)',...
102     'bs','MarkerSize',win_match);hold off
103 % saveas(gcf,'FeatureMatch_inlier1.png');
104 %% Print
105 clc;
106 disp(['The number of inliers is ',num2str(sum(inliers))]);
107 disp(['The value of max_trials is ',num2str(max_trials),]);
108 disp(['The the number of iteration is ',num2str(trials)]);
```

Listing 6: Function

```
1  function error = SampsonError(x1,x2,F)
2  n = size(x1,2);
3  error = zeros(1,n);
4  for i = 1:n
5      error(i) = (x2(:,i)'*F*x1(:,i))^2   /  ...
```

```
                ((x2(:,i)'*F(:,1))^2 + (x2(:,i)'*F(:,2))^2 + ...
                 (F(1,:)*x1(:,i))^2 + (F(2,:)*x1(:,i))^2);
end
end
```

Listing 7: Part(d)

```
clear;clc
%% Load
load('../data/x_inlier.mat');
x1 = padarray(x1_inlier_inhomo,[1 0],1,'post');
x2 = padarray(x2_inlier_inhomo,[1 0],1,'post');
n = size(x1,2);
%% Data Normalization
% x1
mu_x1 = mean(x1,2);
sigma_x1 = sum(var(x1,0,2));
s_x1 = sqrt(2/sigma_x1);
T1 = [s_x1      0    -mu_x1(1)*s_x1
      0       s_x1   -mu_x1(2)*s_x1
      0        0          1       ];
x1_norm = T1*x1;
% x2
mu_x2 = mean(x2,2);
sigma_x2 = sum(var(x2,0,2));
s_x2 = sqrt(2/sigma_x2);
T2 = [s_x2      0    -mu_x2(1)*s_x2
      0       s_x2   -mu_x2(2)*s_x2
      0        0          1       ];
x2_norm = T2*x2;
%% DLT F
A = zeros(n,9);
for i = 1:n
    A(i,:) = kron(x2_norm(:,i)', x1_norm(:,i)');
end
[~,~,V] = svd(A);
F_norm = reshape(V(:,end),3,3)';
%% Rank 2 constraint
[U,D,V] = svd(F_norm);
F_norm = U*diag([D(1,1),D(2,2),0])*V';
F_DLT = T2'*F_norm*T1;
%% Print
F_DLT = F_DLT/norm(F_DLT,'fro');
save('../data/F_DLT.mat','F_DLT');
format longg
disp('F_DLT = ')
disp(F_DLT)
```

Listing 8: Part(e)

```
clear;clc
%% Load
load('../data/x_inlier.mat');
load('../data/F_DLT.mat');
n = size(x1_inlier_inhomo,2);
x1 = padarray(x1_inlier_inhomo,[1 0],1,'post');
x2 = padarray(x2_inlier_inhomo,[1 0],1,'post');
%% Initialize parameter vector
% wu,wv,s
[wu,wv,s] = Fparameterization(F_DLT);
% P
P1 = [eye(3),zeros(3,1)];
P2 = F2P(wu,wv,s);
% X_hat
g = polyg;
X = TwoViewTriangulation(x1,x2,wu,wv,s,g);
X_hat = Xparameterization(X);
%% LM
itr = 0;
disp('LM:')
fprintf('itr\tcost\n')
fprintf('--------------------\n')
% cost
previous_cost = inf;
% tolerance
tolerance = 0.0001;
% step_1
lambda = 0.001;
x1_hat = P1 * Xdeparameterization(X_hat);
x2_hat = P2 * Xdeparameterization(X_hat);
x1_hat_inhomo = x1_hat(1:2,:) ./ x1_hat(3,:);
x2_hat_inhomo = x2_hat(1:2,:) ./ x2_hat(3,:);
epsilon1 = reshape(x1_inlier_inhomo - x1_hat_inhomo,[],1);
epsilon2 = reshape(x2_inlier_inhomo - x2_hat_inhomo,[],1);
% step_2
[f_A,f_B1,f_B2] = jcbFunction;
[A,B1,B2] = jcb(wu,wv,s,X_hat,f_A,f_B1,f_B2);
% initial cost
init_cost = epsilon1'*epsilon1 + epsilon2'*epsilon2;
current_cost = init_cost;
fprintf('%d\t%.4f\n', itr, current_cost)
while tolerance < previous_cost - current_cost || previous_cost < current_cost
    % step_3_4
    [U,V,W] = NormalEquationsMatrix(A,B1,B2);
```

```
45      [epsilon_a,epsilon_b] = NormalEquationsVector(A,B1,B2,epsilon1,epsilon2);
46      [delta_a,delta_b] = AugmentedNormalEquations(U,V,W,epsilon_a,epsilon_b,lambda);
47      % step_5
48      wu0 = wu + delta_a(1:3);
49      wv0 = wv + delta_a(4:6);
50      s0 = s + delta_a(7);
51      X_hat0 =  X_hat + delta_b;
52      % step_6
53      x1_hat0 = P1 * Xdeparameterization(X_hat0);
54      x2_hat0 = F2P(wu0,wv0,s0) * Xdeparameterization(X_hat0);
55      x1_hat_inhomo0 = x1_hat0(1:2,:) ./ x1_hat0(3,:);
56      x2_hat_inhomo0 = x2_hat0(1:2,:) ./ x2_hat0(3,:);
57      epsilon10 = reshape(x1_inlier_inhomo - x1_hat_inhomo0,[],1);
58      epsilon20 = reshape(x2_inlier_inhomo - x2_hat_inhomo0,[],1);
59      % step_7
60      previous_cost = current_cost;
61      current_cost = epsilon10'*epsilon10 + epsilon20'*epsilon20;
62      if current_cost < previous_cost
63          wu = wu0;
64          wv = wv0;
65          s = s0;
66          X_hat = X_hat0;
67          epsilon1 = epsilon10;
68          epsilon2 = epsilon20;
69          lambda = 0.1*lambda;
70          [A,B1,B2] = jcb(wu,wv,s,X_hat,f_A,f_B1,f_B2);
71          if current_cost < init_cost
72              itr = itr + 1;
73              fprintf('%d\t%.4f\n', itr, current_cost)
74          end
75      else
76          lambda = 10*lambda;
77      end
78  end
79  fprintf('--------------------\n\n')
80  %% F_LM
81  F_LM = Fdeparameterization(wu,wv,s);
82  F_LM = F_LM/norm(F_LM,'fro');
83  save('../data/F_LM.mat','F_LM');
84
85  format longg
86  disp('F_LM = ')
87  disp(F_LM)
```

Listing 9: Function

```
1  function g = polyg
```

```
% input(a,b,c,d,f1,f2)
syms t
syms f1 f2 a b c d
f = t*((a*t+b)^2+f2^2*(c*t+d)^2)^2-(a*d-b*c)*(1+f1^2*t^2)^2*(a*t+b)*(c*t+d);
g = matlabFunction(flip(coeffs(f,t)));
end
```

Listing 10: Function

```
function X_PI = TwoViewTriangulation(x1,x2,wu,wv,s,f)
n = size(x1,2);
sigma = Xdeparameterization(s);
D = diag([sigma;0]);
F = expm(skew(wu))*D*expm(skew(wv))';
%% 2D
x1_optimal = zeros(size(x1));
x2_optimal = zeros(size(x2));
for i = 1:n
    %% Fs1
    T1 = [x1(3,i),        0, -x1(1,i)
                0, x1(3,i), -x1(2,i)
                0,        0,  x1(3,i)];
    T2 = [x2(3,i),        0, -x2(1,i)
                0, x2(3,i), -x2(2,i)
                0,        0,  x2(3,i)];
    Fs = (T2' \ F) / T1;
    %% Fs2
    [U,~,V] = svd(Fs);
    e1 = V(:,end);
    e2 = U(:,end);
    e1 = e1/sqrt(e1(1)^2 + e1(2)^2);
    e2 = e2/sqrt(e2(1)^2 + e2(2)^2);
    R1 = [ e1(1), e1(2), 0
          -e1(2), e1(1), 0
               0,     0, 1];
    R2 = [ e2(1), e2(2), 0
          -e2(2), e2(1), 0
               0,     0, 1];
    Fs = R2*Fs*R1';
    %% min t
    a = Fs(2,2);b=Fs(2,3);c=Fs(3,2);d=Fs(3,3);f1=e1(3);f2=e2(3);
    t = real(roots(f(a,b,c,d,f1,f2)));
    ss = t.^2 ./ (1+f1^2*t.^2) + (c*t+d).^2 ./ ((a*t+b).^2 + f2^2 *(c*t+d).^2);
    [~,id] = min(ss);
    t_min = t(id(end));
    %% closest point
    l1 = [t_min*f1,1,-t_min]';
```

```
39      l2 = [-f2*(c*t_min+d),a*t_min+b,c*t_min+d]';
40      xs1 = [-l1(1)*l1(3),-l1(2)*l1(3),l1(1)^2+l1(2)^2]';
41      xs2 = [-l2(1)*l2(3),-l2(2)*l2(3),l2(1)^2+l2(2)^2]';
42      %% original coordinate
43      x1_optimal(:,i) = (T1\R1')*xs1;
44      x2_optimal(:,i) = (T2\R2')*xs2;
45  end
46  %% 3D
47  l2 = F*x1_optimal;
48  l2_orthogonal = [-l2(2,:) .* x2_optimal(3,:)
49                    l2(1,:) .* x2_optimal(3,:)
50                    l2(2,:) .* x2_optimal(1,:) - l2(1,:) .* x2_optimal(2,:)];
51  PI = F2P(wu,wv,s)'*l2_orthogonal;
52  X_PI = [ PI(4,:).*x1_optimal
53          -PI(1,:).*x1_optimal(1,:)-PI(2,:).*x1_optimal(2,:)-PI(3,:).*x1_optimal(3,:)];
54  end
```

Listing 11: Function

```
1  function w = AAparameterization(UV)
2  [~,~,V] = svd(UV-eye(3));
3  v = V(:,end);
4  v_hat = zeros(3,1);
5  v_hat(1) = UV(3,2) - UV(2,3);
6  v_hat(2) = UV(1,3) - UV(3,1);
7  v_hat(3) = UV(2,1) - UV(1,2);
8  sin_theta = v.'*v_hat/2;
9  cos_theta = (trace(UV) - 1)/2;
10 theta = atan2(sin_theta,cos_theta);
11 w = theta * v / norm(v);
12 w = w*(1-2*pi/theta*ceil((theta-pi)/(2*pi)));
13 end
```

Listing 12: Function

```
1  function UV = AAdeparameterization(w)
2  theta = norm(w);
3  wx = skew(w);
4  UV = cos(theta)*eye(3) + sinc(theta/pi)*wx + (1-cos(theta))*(w*w.')/theta^2;
5  end
```

Listing 13: Function

```
1  function v = Xparameterization(v_bar)
2  v_bar = v_bar ./ sqrt(sum(v_bar.^2));
3  a = v_bar(1,:);
4  b = v_bar(2:end,:);
5  v = 2*acos(a) ./ sin(acos(a)) .* b;
```

```
6  v_norm = sqrt(sum(v.^2));
7  v = (1 - 2*pi ./ v_norm .* ceil((v_norm-pi)/2/pi)) .* v;
8  end
```

Listing 14: Function

```
1  function v_bar = Xdeparameterization(v)
2  v_norm = sqrt(sum(v.^2));
3  a = cos(v_norm/2);
4  b = sin(v_norm/2) ./ v_norm .* v;
5  v_bar = [a;b];
6  end
```

Listing 15: Function

```
1  function [wu,wv,s] = Fparameterization(F)
2  [U,Sigma,V] = svd(F);
3  if round(det(U)) == -1
4      U = -U;
5  end
6  if round(det(V)) == -1
7      V = -V;
8  end
9  sigma = [Sigma(1,1);Sigma(2,2)];
10 wu = AAparameterization(U);
11 wv = AAparameterization(V);
12 s = Xparameterization(sigma);
13 end
```

Listing 16: Function

```
1  function F = Fdeparameterization(wu,wv,s)
2  U = AAdeparameterization(wu);
3  V = AAdeparameterization(wv);
4  sigma = Xdeparameterization(s);
5  Sigma = diag([sigma;0]);
6  F = U*Sigma*V';
7  end
```

Listing 17: Function

```
1  function [A,B1,B2] = jcb(wu,wv,s,X_hat,f_A,f_B1,f_B2)
2  n = size(X_hat,2);
3  A = zeros(2,7,n);
4  B1 = zeros(2,3,n);
5  B2 = zeros(2,3,n);
6  for i = 1:n
7      A(:,:,i) = f_A(X_hat(1,i),X_hat(2,i),X_hat(3,i),s,...
```

```
8                       wu(1),wu(2),wu(3),wv(1),wv(2),wv(3));
9       B1(:,:,i) = f_B1(X_hat(1,i),X_hat(2,i),X_hat(3,i));
10      B2(:,:,i) = f_B2(X_hat(1,i),X_hat(2,i),X_hat(3,i),s,...
11                       wu(1),wu(2),wu(3),wv(1),wv(2),wv(3));
12  end
13  end
```

Listing 18: Function

```
1   function [f_A,f_B1,f_B2] = jcbFunction
2   syms wu1 wu2 wu3 wv1 wv2 wv3 s X_hat1 X_hat2 X_hat3
3   %% P1 P2
4   wu = [wu1;wu2;wu3];
5   wv = [wv1;wv2;wv3];
6   P2 = F2P(wu,wv,s);
7   P1 = [eye(3),zeros(3,1)];
8   %% X
9   X_hat = [X_hat1; X_hat2; X_hat3];
10  X = Xdeparameterization(X_hat);
11  %% x1 x2
12  x2 = P2*X;
13  x2_inhomo = x2(1:2) / x2(3);
14  x1 = P1*X;
15  x1_inhomo = x1(1:2) / x1(3);
16  %% jacobian
17  % input(X_hat1,X_hat2,X_hat3,s,wu1,wu2,wu3,wv1,wv2,wv3)
18  f_A = matlabFunction(jacobian(x2_inhomo,[wu.',wv.',s]));
19  % input(X_hat1,X_hat2,X_hat3)
20  f_B1 = matlabFunction(jacobian(x1_inhomo,X_hat));
21  % input(X_hat1,X_hat2,X_hat3,s,wu1,wu2,wu3,wv1,wv2,wv3)
22  f_B2 = matlabFunction(jacobian(x2_inhomo,X_hat));
23  end
```

Listing 19: Function

```
1   function [U,V,W] = NormalEquationsMatrix(A,B1,B2)
2   n = size(A,3);
3   U = zeros(size(A,2),size(A,2));
4   V = zeros(size(B1,2),size(B1,2),n);
5   W = zeros(size(A,2),size(B1,2),n);
6   for i = 1:n
7       U = U + A(:,:,i)' * A(:,:,i);
8       V(:,:,i) = B1(:,:,i)'*B1(:,:,i) + B2(:,:,i)'*B2(:,:,i);
9       W(:,:,i) = A(:,:,i)'*B2(:,:,i);
10  end
11  end
```

Listing 20: Function

```
function [epsilon_a,epsilon_b] = NormalEquationsVector(A,B1,B2,epsilon1,epsilon2)
n = size(A,3);
epsilon_a = zeros(size(A,2),size(epsilon2,2));
epsilon_b = zeros(size(B1,2),n);
for i = 1:n
    epsilon_a = epsilon_a + A(:,:,i)'*epsilon2(2*i-1:2*i);
    epsilon_b(:,i) = B1(:,:,i)'*epsilon1(2*i-1:2*i)+B2(:,:,i)'*epsilon2(2*i-1:2*i);
end
end
```

Listing 21: Function

```
function [delta_a,delta_b]=AugmentedNormalEquations(U,V,W,epsilon_a,epsilon_b,lambda)
n = size(V,3);
delta_b = zeros(size(epsilon_b,1),n);
S = U + lambda*eye(size(U));
e = epsilon_a;
for i = 1:n
    S = S-W(:,:,i)*((V(:,:,i)+lambda*eye(size(V(:,:,i))))\W(:,:,i)');
    e = e-W(:,:,i)*((V(:,:,i)+lambda*eye(size(V(:,:,i))))\epsilon_b(:,i));
end
delta_a = S\e;
for i = 1:n
    delta_b(:,i)=(V(:,:,i)+lambda*eye(size(V(:,:,i))))\...
                 (epsilon_b(:,i)-W(:,:,i)'*delta_a);
end
end
```

Listing 22: Function

```
function a_x = skew(a)
a_x = [    0, -a(3),  a(2)
        a(3),     0, -a(1)
       -a(2),  a(1),     0];
end
```

Listing 23: Function

```
function P = F2P(wu,wv,s)
U = AAdeparameterization(wu);
V = AAdeparameterization(wv);
sigma = Xdeparameterization(s);
Z = [0 -1 0
     1  0 0
     0  0 1];
d = [sigma(1);sigma(2);(sigma(1)+sigma(2))/2];
```

```
m = U*Z*diag(d)*V.';
P = [m,-U(:,3)];
end
```

Listing 24: Part(f)

```
close all; clear; clc;
%% Load Data
load('../data/x_outlier.mat');
load('../data/F_LM.mat');
I0 = imread('../data/IMG_5030.JPG');
I1 = imread('../data/IMG_5031.JPG');
x1 = padarray(x1_outlier_inhomo,[1 0],1,'post');
n = size(x1,2);
%% Mapping
rng(1)
t = randperm(n,3)';
x1_rnd =  x1(:,t);
x2_rnd_inhomo = x2_outlier_inhomo(:,t);
l2 = F_LM*x1_rnd;
%% Plot
figure
imshow(rgb2gray(I0));hold on
plot(x1_outlier_inhomo(1,t)',x1_outlier_inhomo(2,t)','ys','MarkerSize',7);
hold off
% saveas(gcf,'Outlier0.png');
figure
x = (1:size(I0,2))';
y = (-l2(1,:) .* x - l2(3,:)) ./ l2(2,:);
imshow(rgb2gray(I1));hold on
plot(x2_rnd_inhomo(1,:)',x2_rnd_inhomo(2,:)','ys','MarkerSize',7);
plot(x,y,'y','LineWidth',2);hold off
% saveas(gcf,'Outlier1.png');
%% Distance
clc;
distance=abs(l2(1,:).*x2_rnd_inhomo(1,:)+l2(2,:).*x2_rnd_inhomo(2,:)+l2(3,:))...
          ./ sqrt(l2(1,:).^2+l2(2,:).^2);
disp(['The distance between the first point and the line is ',...
    num2str(distance(1))]);
disp(['The distance between the second point and the line is ',...
    num2str(distance(2))]);
disp(['The distance between the third point and the line is ',...
    num2str(distance(3))]);
```