Computer Vision 252B Hw5

Hao-en Sung (wrangle1005@gmail.com)
Department of Computer Science
University of California, San Diego
San Diego, CA 92092

1 Point on line closest to the origin (5 points)

Firstly, I can find the line that perpendicular to l but through origin, i.e. $l_\perp$ by calculating $(0, 0, 1)^\intercal \times (a, b, 1)^\intercal = (b, -a, 0)$, since the normal of l is (a, b). After that, we can again calculate the intersection point of l and $l_\perp$ by cross product as follows.

$$\begin{aligned} l \times l_\perp &= (a, b, c) \times (b, -a, 0) \\ &= (-ac, -bc, a^2 + b^2) \end{aligned}$$

2 Programming: Automatic estimation of the fundamental matrix (150 points)

In this problem, I will fix projection matrix P for first image as a canonical matrix and extract projection matrix P' for second image from calculated fundamental matrix F. To do so, we need to adjust fundamental matrix and 3D scene points jointly to retrieve better results.

2.1 Feature detection (20 points)

This part is covered by the homework 1, where I convolute the original image with a designed kernel to calculate the gradient, detect all potential corners by solving a gradient matrix problem with window size 7, utilize *Non-maximum Suppression* with window size 7 to filter out too closed corner candidates, and filter out corners whose λ is below *lambda_threshold*$= 65$. It is noticeable that, in order to increase the robustness, I average the corner detection values in a window before applying the *lambda_threshold*. At the end, I re-calculate the center of those corners as final results.

There are 1394 features detected in the first image and 1366 features found in the second image. The detected corners for both images are shown in Fig. 1.

2.2 Feature matching (15 points)

This part is also covered by homework 1, where I calculate the correlation coefficient between all pairs of corner window in image 1 and image 2, and iteratively choose the remaining pairs with largest correlation coefficient value whose absolute distance of x and y does not exceed 100. I use *similarity_threshold*$= 0.8$ and *distance_threshold*$= 0.7$ to reject those unqualified matches.

At the end, there are 338 matched feature points found between image 1 and image 2, which are shown in Fig. 2.

2.3 Outlier rejection (20 points)

I first need to do the data normalization then apply mSAC algorithm to do the outlier rejection. In each iteration, I will choose 7 pairs of 2D points from both first image and second image. Firstly, I

30th Conference on Neural Information Processing Systems (NIPS 2016), Barcelona, Spain.

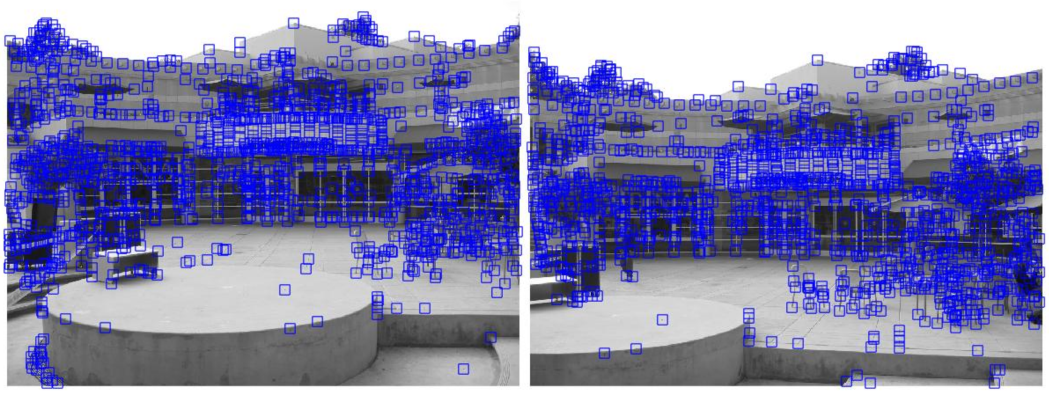

Figure 1: Picture for Detected Corners

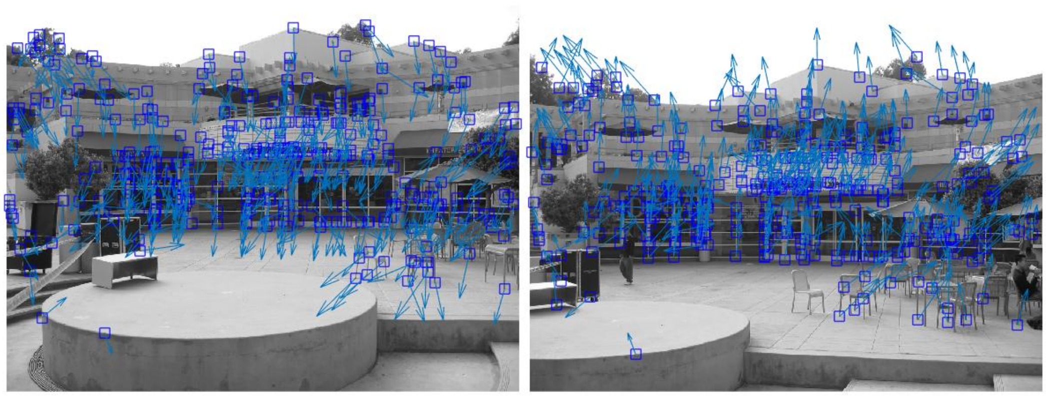

Figure 2: Picture for Matched Corners

need to solve a linear algebra problem

$$A \cdot f = 0,$$

where

$$A = \begin{bmatrix} x_1'^{\intercal} \otimes x_1^{\intercal} \\ x_2'^{\intercal} \otimes x_2^{\intercal} \\ \vdots \\ x_7'^{\intercal} \otimes x_7^{\intercal} \end{bmatrix}.$$

Since fundamental matrix has 9 degrees of freedom and there are only 7 points, the null space of matrix A is 2. In other words, I need to find out two possible solutions f_1 and f_2, which are the last two columns of V retrieved from Singular Value Decomposition (SVD), then add constraints to make the determine of their linear combination, i.e. $\alpha * f_1 + f_2$, be 0. To solve this, I need to use *syms* function to simulate the equation and solve α with functions *collect* and *root*.

After finding out 1 or 3 real solutions, I need to examine all of them and keep the one with smallest Sampson error, which can be calculated for each pair of points x_i and x_i' as follows.

$$\epsilon = x_i'^{\intercal} F x_i$$
$$J = \begin{bmatrix} x_i' F_{1,1} + y_i' F_{2,1} + F_{3,1} & x_i' F_{1,2} + y_i' F_{2,2} + F_{3,2} \\ x_i F_{1,1} + y_i F_{1,2} + F_{1,3} & x_i F_{2,1} + y_i F_{2,2} + F_{2,3} \end{bmatrix}$$
$$\lambda = \frac{-\epsilon}{J^{\intercal} J}$$
$$\delta_i = J^{\intercal} \cdot \lambda$$

At the end, I use $\|\delta_i\|^2$ as the error and $\min(\|\delta_i\|^2, chi2inv(0.95, 7))$ as cost for point i.

The way to dynamically determine the number of maximum iterations are covered in homework 3 report. Here, I use the same settings that $p = 0.99$, $\alpha = 0.95$, and $\sigma^2 = 1$, and my algorithm finds out the inliers in *MAX_ITERATIONS*$= 37.8762081166275$.

As the result of mSAC algorithm, it reduces matched corner pairs from 338 to 248 and achieve Rooted Mean Squared Error (RMSE) 535.9307136808, whose result is shown as Fig. 3.

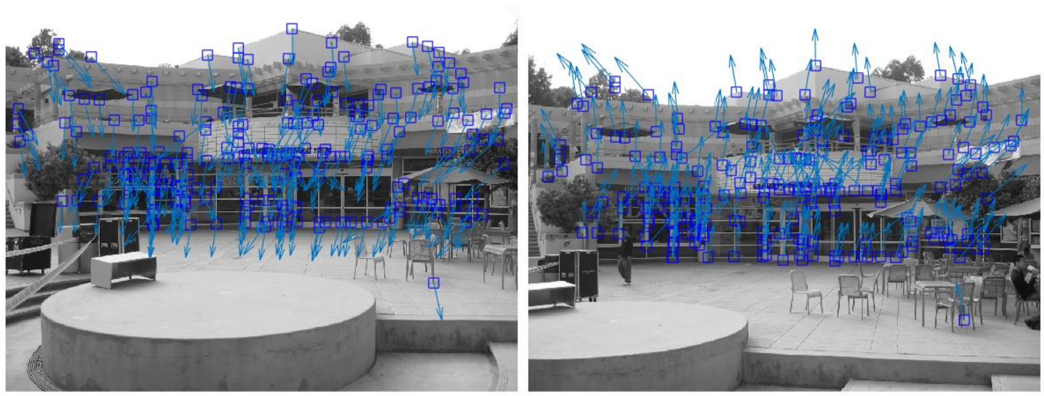

Figure 3: Picture for Robust Matched Corners

2.4 Linear estimation (15 points)

Within Direct Linear Transformation (DTL) algorithm, I firstly normalize points in both image 1 and image 2. After that, I directly solve a linear algebra problem

$$A \cdot f = 0,$$

where

$$A = \begin{bmatrix} x_1'^{\intercal} \otimes x_1^{\intercal} \\ x_2'^{\intercal} \otimes x_2^{\intercal} \\ \vdots \\ x_n'^{\intercal} \otimes x_n^{\intercal} \end{bmatrix}$$
$$f = \text{vec}(F),$$

and return f as my linear estimation of F (or denoted as F_{DLT}).

My normalized F_{DLT} matrix can be expressed as

$$\begin{bmatrix} -4.68499762447309e-09 & 1.79358721726428e-07 & -8.73491219858134e-08 \\ -1.13801863652536e-06 & -1.68387299119616e-07 & 0.0112621730492271 \\ 0.000375412950587891 & -0.0104693705461033 & -0.999881700403768 \end{bmatrix},$$

and it successively reduces the RMSE from 535.9307136808 to 349.8087322929, which is a very significant improvement.

2.5 Nonlinear estimation (70 points)

Here, I still need to do the data normalization before all the other optimization procedure.

In this part, I first use Sampson Correction (details are mentioned in problem 3) to correct 2D points, where corrected point $\hat{x_i}$ can be express as

$$\hat{x_i} = x_i + \delta_x$$
$$\hat{x_i'} = x_i' + \delta_{x'},$$

where

$$\begin{bmatrix} \delta_x \\ \delta_{x'} \end{bmatrix} = \delta_i.$$

After Sampson Correction, I need to do optimal triangulation on $\hat{x_i}$ and $\hat{x_i'}$ for the initialization of 3D scene points. The detailed procedure for each paired points $x = (x, y, w)$ and $x' = (x', y', w')$ can be summarized as follows.

1. Make fundamental matrix to be special fundametal matrix, i.e. $F_s = T'^{-\intercal} F T^{-1}$, where

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

2. Calculate epipoles of F_s, i.e. $e = null(F_s)$ and $e' = null(F_s^\intercal)$
3. Normalize epipoles, such that $e = \frac{e}{\sqrt{e_1^2 + e_2^2}}$ and $e' = \frac{e'}{\sqrt{e_1'^2 + e_2'^2}}$
4. Make fundamental matrix to be special fundametal matrix, i.e. $F_s = R' F_s R^\intercal$, where

$$R = \begin{bmatrix} e_1 & e_2 & 0 \\ -e_2 & e_1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$
$$R' = \begin{bmatrix} e_1' & e_2' & 0 \\ -e_2' & e_1' & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$

5. Solve polynomial $g(t) = 0$ with t, where

$$g(t) = t[(at+b)^2 + f'^2(ct+d)^2]^2 - (ad - bc) \cdot (1 + f^2 t^2)^2 \cdot (at+b) \cdot (ct+d)$$
$$f = e_3$$
$$f' = e_3'$$
$$a = F_{s\ 2,2}$$
$$b = F_{s\ 2,3}$$
$$c = F_{s\ 3,2}$$
$$d = F_{s\ 3,3}$$

6. Since $g(t) = 0$ is a 6-degree polynomial equation for t, there are 6 solutions of t. One needs to pick up one t, whose real part r minimizes the cost function $s(r)$

$$s(r) = \frac{r^2}{1 + f^2 r^2} + \frac{(cr+d)^2}{(ar+b)^2 + f'^2(cr+d)^2}$$

.

7. Calculate the corresponding two epipolar lines l and l' as follows.

$$l = (rf, 1, -r)^\intercal$$
$$l' = (-f'(cr+d), ar+b, cr+d)^\intercal$$

8. Utilize the conclusion in first problem to find out a point x_l on line $l = (a, b, c)^\intercal$ that is closest to point x.

$$x_l = \begin{bmatrix} -ac \\ -bc \\ a^2 + b^2 \end{bmatrix}$$

   Apply same procedure to l' and x' to find out x_l'.

9. Find out epiline of x, i.e. $l' = F \cdot x = (a', b', c')^\intercal$.

10. Find out $l'_\perp$ which perpendicular to l' and through x', i.e. $l'_\perp = (-b'w', a'w', b'x' - a'y')^\intercal$.

11. Backproject $l'_\perp$ to plane π, where $\pi = P'l'_\perp = (a, b, c, d)^\intercal$. P' is the projection matrix for second image and can be calculated as $P' = UZdiag(D')V^\intercal$, where

$$
\begin{aligned}
UDV^\intercal &= F \\
Z &= \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & -1 & 1 \end{bmatrix} \\
D' &= \begin{bmatrix} D_{1,1} & 0 & 0 \\ 0 & D_{2,2} & 0 \\ 0 & 0 & \frac{D_{1,1}+D_{2,2}}{2}. \end{bmatrix}
\end{aligned}
$$

12. Use two 3D points — camera origin C as X_1 and $P^\dagger x$ as X_2 — to indicate a line and calculate the 3D scene points as the intersection of the line and plane π.

$$
X_\pi = \begin{bmatrix} X_2(bY_1 + cZ_1 + dT_1) - X_1(bY_2 + cZ_2 + dT_2) \\ Y_2(aY_1 + cZ_1 + dT_1) - Y_1(aY_2 + cZ_2 + dT_2) \\ Z_2(aY_1 + bZ_1 + dT_1) - Z_1(aY_2 + bZ_2 + dT_2) \\ T_2(aY_1 + bZ_1 + cT_1) - T_1(aY_2 + bZ_2 + dT_2) \end{bmatrix}
$$

If $P = [I|0]$, $P^\dagger = P^\intercal = \begin{bmatrix} I \\ 0 \end{bmatrix}$, and $C = (0, 0, 0, 1)^\intercal$, X_π can be further simplified.

$$
X_\pi = \begin{bmatrix} dX_2 \\ dY_2 \\ dZ_2 \\ -(aX_2 + bY_2 + cZ_2) \end{bmatrix}
$$

Next, I fix projection matrix from scene plane to first image, i.e. P, as identity matrix and iteratively update projection matrix from scene plane to second image, i.e. P', as well as scene points. To do so, I regard parameterized F and parameterized 3D homogeneous scene points X as parameters, 2D inhomogeneous points in both first image and second image as measurements, and solve an *Augumented Normal Equations* to find out the updates for both F and X.

One easiest way to obtain the *Jacobian* matrix J for this problem is to consider J as a $4n \times (3n + 7)$ huge matrix, which is shown as follows.

$$
\begin{bmatrix} 0 & \begin{bmatrix} B_1 & & 0 \\ & \ddots & \\ 0 & & B_n \end{bmatrix} \\ \begin{bmatrix} A'_1 \\ \vdots \\ A'_n \end{bmatrix} & \begin{bmatrix} B'_1 & & 0 \\ & \ddots & \\ 0 & & B'_n \end{bmatrix} \end{bmatrix},
$$

where

$$
\begin{aligned}
A'_i &= \frac{\partial \hat{x'_i}}{\partial(\hat{W_u}^\intercal, \hat{W}_v, \hat{S})} = \frac{\partial \hat{x}_i{}'}{\partial \bar{P'}} \cdot \frac{\partial \bar{P'}}{\partial(\hat{W_u}^\intercal, \hat{W}_v, \hat{S})},\\
B_i &= \frac{\partial \hat{x}_i}{\partial \hat{X}_i} = \frac{\partial \hat{x}_i}{\partial \bar{X}_i} \cdot \frac{\partial \bar{X}_i}{\partial \hat{X}_i},\\
B'_i &= \frac{\partial \hat{x'_i}}{\partial \hat{X}_i} = \frac{\partial \hat{x'_i}}{\partial \bar{X}_i} \cdot \frac{\partial \bar{X}_i}{\partial \hat{X}_i},\\
\hat{W}_u &\text{ is the updated parameterized vector of matrix } U,\\
\hat{W}_v &\text{ is the updated parameterized vector of matrix } V,\\
\hat{S} &\text{ is the updated parameterized scalar of vector } D,\\
\hat{X}_i &\text{ is the updated 3D homogeneous scene points},\\
\bar{X}_i &\text{ is the updated 3D parameterized scene points},\\
\hat{x}_i &\text{ is the estimated 2D inhomogeneous points in first image},\\
\hat{x'_i} &\text{ is the estimated 2D inhomogeneous points in second image}.
\end{aligned}
$$

With *Jacobian* matrix J, I then can calculate the update for parameters δ by solving a *Augmented Normal Equation* as follows.

$$(J^\intercal \Sigma_x^{-1} J + \lambda I) \cdot \delta = J^\intercal \Sigma_x^{-1} \epsilon,$$

where

$$
\Sigma_x = \begin{bmatrix}
\begin{bmatrix} \sqrt{\frac{2}{\mathrm{var}(x_i)}} & & \\ & \ddots & \\ & & \sqrt{\frac{2}{\mathrm{var}(x_i)}} \end{bmatrix} & 0 \\
0 & \begin{bmatrix} \sqrt{\frac{2}{\mathrm{var}(x'_i)}} & & \\ & \ddots & \\ & & \sqrt{\frac{2}{\mathrm{var}(x'_i)}} \end{bmatrix}
\end{bmatrix} \text{ is the covariance matrix}
$$

$$
\epsilon = \begin{bmatrix} x_1 - \hat{x_1} \\ \cdots \\ x_n - \hat{x_n} \\ x'_1 - \hat{x'_1} \\ \cdots \\ x'_n - \hat{x'_n} \end{bmatrix} \text{ is the difference between ground truth points and estimated points,}
$$

λ is a dynamic parameter, which is set as a step size.

After solving δ, I will update parameters F and X and perform the estimation and update for next round until the cost $\epsilon^\intercal \Sigma_x^{-1} \epsilon$ converges to a minimum. Then, I will return my non-linear estimation — projection matrix F (or said F_{LM}).

My normalized F_{LM} matrix can be expressed as

$$
\begin{bmatrix}
-1.09983325876086e-09 & 2.13467451471971e-07 & -5.14723675847964e-05 \\
-1.16432301249744e-06 & -1.54037690917501e-07 & 0.0113872043107472 \\
0.000419969744351145 & -0.0106114511394522 & -0.99987876747998
\end{bmatrix},
$$

however, the RMSE increases slightly from 349.8087322929 to 353.7854297991.

My implementation converges in four rounds and the error log can be shown as follows.

$$\epsilon^\intercal \Sigma_x^{-1} \epsilon = [114.707008995594 \quad 100.20057222354 \quad 100.199148559104 \quad 100.199148559102]$$

.

2.6 Point to line mapping (10 points)

Three points are randomly selected from rejected features in first image. I then use fundamental matrix F to project three points to three corresponding epipoler lines. The image is included as Fig. 4.

Figure 4: Picture for Epipolar Line Mapping

Summary

From mSAC solving for 2D points to 2D points matching problem to linear estimation and non-linear estimation of projection matrix, RMSE reduces dramatically. On the other hand, one can tell that non-linear estimation further minimizes the geometric error when compared with linear estimation.

Appendix

Problem 1

Code Listing 1: Feature Detection

```
function mat = featureDetect(I, lambda_threshold, nw)

%% Parameters
hw = int32(floor(nw/2));
[n, m] = size(I);

%% Convolutional Kernel
k = [-1; 8; 0; -8; 1] / 12;

%% Calculate Gradient
Gx = conv2(double(I), k', 'same');
Gy = conv2(double(I), k, 'same');

%% Precalculate Squared Gradient
Gxx = Gx .* Gx;
Gxy = Gx .* Gy;
Gyy = Gy .* Gy;

%% Corner Detection
em = zeros(n, m);
for r = 1+nw:n-nw
    for c = 1+nw:m-nw
        vxx = sum(sum(Gxx(r-hw:r+hw, c-hw:c+hw)));
        vyy = sum(sum(Gyy(r-hw:r+hw, c-hw:c+hw)));
        vxy = sum(sum(Gxy(r-hw:r+hw, c-hw:c+hw)));
        ATA = [vxx, vxy; vxy, vyy] / (nw * nw);
        val = max(trace(ATA)^2 - 4*det(ATA), 0);
        em(r, c) = (trace(ATA) - sqrt(val)) / 2;
    end
end

%% Non-maximum Suppression
mat = [];
for r = 1+nw:n-nw
    for c = 1+nw:m-nw
        vmax = max(max(em(r-hw:r+hw, c-hw:c+hw)));
        if em(r, c) == vmax && vmax > lambda_threshold
            mat = [mat; [c, r]];
        end
    end
end

%% Find Real Corners
[idx_x, idx_y] = meshgrid(1:m, 1:n);
xGxx = idx_x .* Gxx;
xGxy = idx_x .* Gxy;
yGxy = idx_y .* Gxy;
yGyy = idx_y .* Gyy;

for i = 1:size(mat,1)
    c = mat(i, 1);
    r = mat(i, 2);
    vxx = sum(sum(Gxx(r-hw:r+hw, c-hw:c+hw)));
    vyy = sum(sum(Gyy(r-hw:r+hw, c-hw:c+hw)));
    vxy = sum(sum(Gxy(r-hw:r+hw, c-hw:c+hw)));
    xVxx = sum(sum(xGxx(r-hw:r+hw, c-hw:c+hw)));
    xVxy = sum(sum(xGxy(r-hw:r+hw, c-hw:c+hw)));
    yVxy = sum(sum(yGxy(r-hw:r+hw, c-hw:c+hw)));
    yVyy = sum(sum(yGyy(r-hw:r+hw, c-hw:c+hw)));
```

```
    A = [vxx, vxy; vxy, vyy];
    b = [xVxx + yVxy; xVxy + yVyy];
    mat(i, :) = (A \ b)';
end
```

Problem 2

Code Listing 2: Feature Match

```
function [lx, rx] = featureMatch(preI, nxtI, pref, nxtf, nw)

%% Parameters
hw = floor(nw/2);
[n,m] = size(preI);
similarity_threshold = 0.8;
dist_threshold = 0.7;

preI = double(preI);
nxtI = double(nxtI);

%% Create Correlations
corrw = zeros(size(pref,1), size(nxtf,1));
for i = 1:size(pref,1)
    for j = 1:size(nxtf,1)
        % check proximality
        if abs(pref(i,1) - nxtf(j,1)) > 100 ...
                || abs(pref(i,2) - nxtf(j,2)) > 100 ...
            continue
        end

        [px, py] = ...
            meshgrid(max(pref(i,1)-hw,1):min(pref(i,1)+hw,m), ...
                     max(pref(i,2)-hw,1):min(pref(i,2)+hw,n));
        [nx, ny] = ...
            meshgrid(max(nxtf(j,1)-hw,1):min(nxtf(j,1)+hw,m), ...
                     max(nxtf(j,2)-hw,1):min(nxtf(j,2)+hw,n));

        prew = interp2(preI, px, py);
        nxtw = interp2(nxtI, nx, ny);
        corrw(i,j) = corr2(prew,nxtw);
    end
end

%% Find Largest Element Iteratively
lx = [];
rx = [];
while true
    [maxv,maxi] = max(corrw(:));
    % stop while the maximum one is not large enough
    if maxv < similarity_threshold
        break
    end

    % get the window corrdinate
    [r,c] = ind2sub(size(corrw),maxi);
    corrw(r,c) = -1;

    % get the potential two window corrdinates
    [nrv,~] = max(corrw(r,:));
    [ncv,~] = max(corrw(:,c));

    % stack them into arrays
    if nrv > ncv
        if (1-maxv) < (1-nrv) * dist_threshold
```

```
            lx = [lx; [pref(r,1), pref(r,2)]];
            rx = [rx; [nxtf(c,1), nxtf(c,2)]];
        end
    else
        if (1-maxv) < (1-ncv) * dist_threshold
            lx = [lx; [pref(r,1), pref(r,2)]];
            rx = [rx; [nxtf(c,1), nxtf(c,2)]];
        end
    end

    % reset to -1
    for j = 1:size(nxtf,1)
        corrw(r,j) = -1;
    end
    for i = 1:size(pref,1)
        corrw(i,c) = -1;
    end
end
```

Problem 3

Code Listing 3: mSAC Algorithm

```
% lx, rx: 2D homogenerous points
function [best_F, best_bmap, MAX_TRIALS, MIN_COST] = mSAC(lx, rx)
    assert(size(lx,1) == size(rx,1));
    n = size(lx,1);

    % Data Normalization
    lxm = mean(lx);
    lxv = var(lx);
    lxs = sqrt(2 / (lxv(1)+lxv(2)));
    lT = [lxs, 0, -lxm(1)*lxs; 0, lxs, -lxm(2)*lxs; 0, 0, 1];
    lx = lx * lT';

    rxm = mean(rx);
    rxv = var(rx);
    rxs = sqrt(2 / (rxv(1)+rxv(2)));
    rT = [rxs, 0, -rxm(1)*rxs; 0, rxs, -rxm(2)*rxs; 0, 0, 1];
    rx = rx * rT';

    % Parameters
    MIN_COST = Inf;
    MAX_TRIALS = Inf;
    THRESHOLD = 1;
    TOLERANCE = chi2inv(0.95,1);
    PROBABILITY = 0.99;
    IMAG_EPS = 1e-8;

    % Initialize return values
    best_F = zeros(3,3);
    best_bmap = false(n,1);

    trials = 0;
    s = RandStream('mt19937ar','Seed',514);
    while trials < MAX_TRIALS && MIN_COST > THRESHOLD
        trials = trials + 1;

        % Random 7 numbers
        idx = randperm(s, n);
        idx = idx(1:7);

        % Create matrix x_i' kron x_i
        A = zeros(7,9);
```

```
        for i = 1:7
            A(i, :) = kron(rx(idx(i),:), lx(idx(i),:));
        end

        % Find out f = vec(F)
        [~,~,V] = svd(A);
        f1 = reshape(V(:,end-1), 3, 3)';
        f2 = reshape(V(:,end), 3, 3)';

        syms a;
        F = a * f1 + f2;
        detF = det(F);
        equ = collect(detF, a);
        sol = double(root(equ, a));

        % Enumerate all results
        sbest_cost = Inf;
        for i = 1:size(sol,1)
            if imag(sol(i)) < IMAG_EPS
                a = real(sol(i));
                F = a * f1 + f2;

                % Calculate delta
                delta = calDelta(lx, rx, lT, rT, F);

                % Calculate error
                error = calError(delta);

                % Calculate cost
                cost = calCost(error, TOLERANCE);

                % Preserve the best model
                if cost < sbest_cost
                    sbest_F = F;
                    sbest_bmap = error < TOLERANCE;
                    sbest_n_in = sum(sbest_bmap);
                    sbest_cost = cost;
                end
            end
        end

        if sbest_cost < MIN_COST
            MIN_COST = sbest_cost;
            best_F = sbest_F;
            best_bmap = sbest_bmap;
            best_n_in = sbest_n_in;

            w = best_n_in / n;
            MAX_TRIALS = log(1-PROBABILITY) / log(1-w^7);
        end
    end

    % Data Denormalization
    best_F = rT' * best_F * lT;
end

function delta = calDelta(lx, rx, lT, rT, F)
    n = size(lx,1);
    lx = lx / lT';
    lx = lx ./ lx(:,3);
    rx = rx / rT';
    rx = rx ./ rx(:,3);
    F = rT' * F * lT;

    delta = zeros(n,4);
```

```
    for i = 1:n
        J = [rx(i,1)*F(1,1)+rx(i,2)*F(2,1)+F(3,1), ...
             rx(i,1)*F(1,2)+rx(i,2)*F(2,2)+F(3,2), ...
             lx(i,1)*F(1,1)+lx(i,2)*F(1,2)+F(1,3), ...
             lx(i,1)*F(2,1)+lx(i,2)*F(2,2)+F(2,3)];
        e = rx(i,:) * F * lx(i,:)';
        lambda = -e / (J*J');
        delta(i,:) = J * lambda;
    end
end

function error = calError(delta)
    n = size(delta,1);
    error = zeros(n,1);
    for i = 1:n
        error(i,1) = delta(i,:) * delta(i,:)';
    end
end

function cost = calCost(error, TOLERANCE)
    cost = sum(min(error, TOLERANCE));
end
```

Problem 4

Code Listing 4: Direct Linear Transformation

```
% lx, rx: 2D homogenerous points
function F = DLT_nc(lx, rx)
    assert(size(lx,1) == size(rx,1));
    n = size(lx,1);

    % Data Normalization
    lxm = mean(lx);
    lxv = var(lx);
    lxs = sqrt(2 / (lxv(1)+lxv(2)));
    lT = [lxs, 0, -lxm(1)*lxs; 0, lxs, -lxm(2)*lxs; 0, 0, 1];
    lx = lx * lT';

    rxm = mean(rx);
    rxv = var(rx);
    rxs = sqrt(2 / (rxv(1)+rxv(2)));
    rT = [rxs, 0, -rxm(1)*rxs; 0, rxs, -rxm(2)*rxs; 0, 0, 1];
    rx = rx * rT';

    % Left Null Space of F
    A = zeros(n,9);
    for i = 1:n
        A(i, :) = kron(rx(i,:), lx(i,:));
    end

    % Solve for F
    [~,~,V] = svd(A,'econ');
    f = V(:,end);
    F = reshape(f, 3, 3)';

    % Add constraint to F
    [U,D,V] = svd(F,'econ');
    D(3,3) = 0;
    F = U * D * V';

    % Data Denormalization
    F = rT' * F * lT;
end
```

Problem 5

Code Listing 5: Levenberg–Marquardt Algorithm

```
% lx, rx: 2D homogenerous points
function [F, logs] = LM_nc(lx, rx, F)
    assert(size(lx,1) == size(rx,1));
    n = size(lx,1);

    % Data Normalization
    lxm = mean(lx);
    lxv = var(lx);
    lxs = sqrt(2 / (lxv(1)+lxv(2)));
    lT = [lxs, 0, -lxm(1)*lxs; 0, lxs, -lxm(2)*lxs; 0, 0, 1];
    lx = lx * lT';

    rxm = mean(rx);
    rxv = var(rx);
    rxs = sqrt(2 / (rxv(1)+rxv(2)));
    rT = [rxs, 0, -rxm(1)*rxs; 0, rxs, -rxm(2)*rxs; 0, 0, 1];
    rx = rx * rT';

    F = inv(rT') * F * inv(lT);

    % Parameterize F
    [U,D,V] = svd(F);
    S = [D(1,1), D(2,2)];

    % Constrain U, V
    if det(U) < 0
        U = -U;
    end
    if det(V) < 0
        V = -V;
    end

    % Parameterize Wu, Wv, S
    Wu = angleParameterize(U);
    Wv = angleParameterize(V);
    pS = homoParameterize(S);

    % Angle Normalize Wu, Wv, S
    if norm(Wu) > pi
        Wu = (1 - 2*pi/norm(Wu) * ceil((norm(Wu)-pi)/(2*pi))) * Wu;
    end
    if norm(Wv) > pi
        Wv = (1 - 2*pi/norm(Wv) * ceil((norm(Wv)-pi)/(2*pi))) * Wv;
    end
    if norm(pS) > pi
        pS = (1 - 2*pi/norm(pS) * ceil((norm(pS)-pi)/(2*pi))) * pS;
    end

    % DeParameterize Wu, Wv, S, and pX
    U = angleDeparameterize(Wu);
    V = angleDeparameterize(Wv);
    S = homoDeparameterize(pS)';

    % DeParameterize F
    F = U * diag([S, 0]) * V';

    % Sampson Correction
    delta = calDelta(lx, rx, lT, rT, F);
    s_lx = lx / lT';
    s_lx(:,1:2) = s_lx(:,1:2) + delta(:,1:2);
    s_lx = s_lx * lT';
```

```
s_rx = rx / rT';
s_rx(:,1:2) = s_rx(:,1:2) + delta(:,3:4);
s_rx = s_rx * rT';

if exist('../tmp/optimal.mat', 'file') == 2
    load('../tmp/optimal.mat');
else
    % Optimal Triangulation
    [s_lx, s_rx] = optimalTriangulate(s_lx, s_rx, F);
    save('../tmp/optimal.mat', 's_lx', 's_rx');
end

% 3D Scene Points Initialization
sX = getScenePoints(s_lx, s_rx, U, S, V, F);

% Parameterize sX
pX = zeros(n,3);
for i = 1:n
    pX(i,:) = homoParameterize(sX(i,:));
end

% Angle Normalize pX
for i = 1:n
    if norm(pX(i,:)) > pi
        pX(i,:) = (1 - 2*pi/norm(pX(i,:)) * ...
            ceil((norm(pX(i,:))-pi)/(2*pi))) * pX(i,:);
    end
end

% Deparameterize pX
for i = 1:n
    sX(i,:) = homoDeparameterize(pX(i,:)')';
end

% Covariance Matrix
Z = diag([repmat(lxs^2, 1, 2*n), ...
    repmat(rxs^2, 1, 2*n)]);

% Initialization
lambda = 0.001;
perr = Inf;

% Estimate
[n_lx, n_rx] = estimate(sX, U, S, V);

ex = calEpsilon(lx, rx, n_lx, n_rx);
err = ex'*inv(Z)*ex;

% Error Log
logs = err;

while abs(perr-err) > 0.0001
    % Get Right P
    rP = getRP(U, S, V);

    % Calculate J
    J = zeros(4*n,7+3*n);

    % Fill up J with A_i''
    partial_F = zeros(12,7);

    u = [-S(2)*V(:,2), S(1)*V(:,1), (S(1)+S(2))/2*V(:,3); 0, 0,
                                              -1];
    partial_u = kron(eye(3), u);
    partial_F(:,1:3) = partial_u * angleJacobian(Wu);
```

```
partial_v = zeros(12,9);
partial_v(1:3,:) = kron(eye(3), ...
    [S(1)*U(1,2), -S(2)*U(1,1), (S(1)+S(2))/2*U(1,3)]);
partial_v(5:7,:) = kron(eye(3), ...
    [S(1)*U(2,2), -S(2)*U(2,1), (S(1)+S(2))/2*U(2,3)]);
partial_v(9:11,:) = kron(eye(3), ...
    [S(1)*U(3,2), -S(2)*U(3,1), (S(1)+S(2))/2*U(3,3)]);
partial_F(:,4:6) = partial_v * angleJacobian(Wv);

partial_s = [U(1,2)*V(:,1) + U(1,3)/2*V(:,3), ...
     U(1,3)/2*V(:,3) - U(1,1)*V(:,2); ...
     0, 0;
     U(2,2)*V(:,1) + U(2,3)/2*V(:,3), ...
     U(2,3)/2*V(:,3) - U(2,1)*V(:,2); ...
     0, 0;
     U(3,2)*V(:,1) + U(3,3)/2*V(:,3), ...
     U(3,3)/2*V(:,3) - U(3,1)*V(:,2); ...
     0, 0];
partial_ss = [-0.5 * S(2:end); ...
     SINC(norm(pS)/2)/2 * eye(1) ...
     + 1/(4*norm(pS)) * DSINC(norm(pS)/2) ...
     * (pS'*pS)];
partial_F(:,7) = partial_s * partial_ss;

for i = 1:n
    partial_xp = 1/n_rx(i,3) * ...
        [sX(i,:), zeros(1,4), -n_rx(i,1)/n_rx(i,3)*sX(i,:);
                                                  ...
         zeros(1,4), sX(i,:), -n_rx(i,2)/n_rx(i,3)*sX(i,:)];
    J(2*n+2*i-1:2*n+2*i,1:7) = partial_xp * partial_F;
end

% Fill up J with B_i'
for i = 1:n
    partial_x1x = 1/n_lx(i,3) * ...
        [1, 0, -n_lx(i,1)/n_lx(i,3), 0;
         0, 1, -n_lx(i,2)/n_lx(i,3), 0];
    partial_xx = [-0.5 * sX(i,2:end); ...
          SINC(norm(pX(i,:))/2)/2 * eye(3) ...
          + 1/(4*norm(pX(i,:))) * DSINC(norm(pX(i,:))/2) ...
          * (pX(i,:)'*pX(i,:))];
    J(2*i-1:2*i,7+3*i-2:7+3*i) = partial_x1x * partial_xx;
end

% Fill up J with B_i''
for i = 1:n
    partial_x2x = 1/n_rx(i,3) * ...
        [rP(1,:) - n_rx(i,1)/n_rx(i,3) * rP(3,:);
         rP(2,:) - n_rx(i,2)/n_rx(i,3) * rP(3,:)];
    partial_xx = [-0.5 * sX(i,2:end); ...
          SINC(norm(pX(i,:))/2)/2 * eye(3) ...
          + 1/(4*norm(pX(i,:))) * DSINC(norm(pX(i,:))/2) ...
          * (pX(i,:)'*pX(i,:))];
    J(2*n+2*i-1:2*n+2*i,7+3*i-2:7+3*i) ...
        = partial_x2x * partial_xx;
end

while true
    % Solve delta and update Wu, Wv, pS, and pX
    d = (J'*inv(Z)*J + lambda*eye(7+3*n)) \ (J'*inv(Z)*ex);
    n_Wu = Wu + d(1:3);
    n_Wv = Wv + d(4:6);
    n_pS = pS + d(7);
    n_pX = pX;
```

```
for i = 1:n
    n_pX(i,:) = n_pX(i,:) + d(7+3*i-2:7+3*i)';
end

% Angle Normalize Wu, Wv, S, and pX
if norm(n_Wu) > pi
    n_Wu = (1 - 2*pi/norm(n_Wu) * ...
        ceil((norm(n_Wu)-pi)/(2*pi))) * n_Wu;
end
if norm(n_Wv) > pi
    n_Wv = (1 - 2*pi/norm(n_Wv) * ...
        ceil((norm(n_Wv)-pi)/(2*pi))) * n_Wv;
end
if norm(n_pS) > pi
    n_pS = (1 - 2*pi/norm(n_pS) * ...
        ceil((norm(n_pS)-pi)/(2*pi))) * n_pS;
end
for i = 1:n
    if norm(n_pX(i,:)) > pi
        n_pX(i,:) = (1 - 2*pi/norm(n_pX(i,:)) ...
            * ceil((norm(n_pX(i,:))-pi)/(2*pi))) ...
            * n_pX(i,:);
    end
end

% DeParameterize Wu, Wv, S, and pX
n_U = angleDeparameterize(n_Wu);
n_V = angleDeparameterize(n_Wv);
n_S = homoDeparameterize(n_pS)';
n_sX = zeros(n,4);
for i = 1:n
    n_sX(i,:) = homoDeparameterize(n_pX(i,:)')';
end

% Constrain U, V
if det(n_U) < 0
    n_U = -n_U;
end
if det(n_V) < 0
    n_V = -n_V;
end

% Parameterize Wu, Wv
n_Wu = angleParameterize(n_U);
n_Wv = angleParameterize(n_V);

% DeParameterize F
n_F = n_U * diag([n_S, 0]) * n_V';

% Estimate
[n_lx, n_rx] = estimate(n_sX, n_U, n_S, n_V);

% Calculate Error
nex = calEpsilon(lx, rx, n_lx, n_rx);
nerr = nex'*inv(Z)*nex;

% Check Error
if nerr < err
    U = n_U;
    V = n_V;
    S = n_S;
    F = n_F;
    sX = n_sX;
    Wu = n_Wu;
    Wv = n_Wv;
```

```
                pS = n_pS;
                pX = n_pX;
                lambda = 0.1 * lambda;
                break;
            else
                lambda = 10 * lambda;
            end
        end

        % Update error
        perr = err;
        err = nerr;
        logs = [logs, err];
    end

    % Data Denormalization
    F = rT' * F * lT;
end

function rP = getRP(U, S, V)
    z = [0, -1, 0; 1, 0, 0; 0, 0, 1];
    d = [S(1), S(2), (S(1) + S(2)) / 2];
    m = U * z * diag(d) * V';
    rP = [m, -U(:,3)];
end

function [n_lx, n_rx] = optimalTriangulate(lx, rx, F)
    assert(size(lx,1) == size(rx,1));
    n = size(lx,1);

    n_lx = zeros(size(lx));
    n_rx = zeros(size(rx));
    for i = 1:n
        % Special Fundamental Matrix
        lT = [lx(i,3), 0, -lx(i,1); ...
              0, lx(i,3), -lx(i,2); ...
              0, 0, lx(i,3)];
        rT = [rx(i,3), 0, -rx(i,1); ...
              0, rx(i,3), -rx(i,2); ...
              0, 0, rx(i,3)];
        Fs = inv(rT') * F * inv(lT);

        % epipoles
        le = null(Fs);
        re = null(Fs');

        % scale epipoles
        le = le ./ sqrt(le(1)^2 + le(2)^2);
        re = re ./ sqrt(re(1)^2 + re(2)^2);

        % Special Fundamental Matrix
        lR = [le(1), le(2), 0; ...
              -le(2), le(1), 0; ...
              0, 0, 1];
        rR = [re(1), re(2), 0; ...
              -re(2), re(1), 0; ...
              0, 0, 1];
        Fs = rR * Fs * lR';

        % solve t
        syms t;
        lf = le(3);
        rf = re(3);
        a = Fs(2,2);
        b = Fs(2,3);
```

```
        c = Fs(3,2);
        d = Fs(3,3);
        gt = t * ((a*t+b)^2 + rf^2*(c*t+d)^2)^2 ...
            - (a*d-b*c) * (1+lf^2*t^2)^2 * (a*t+b) * (c*t+d);
        equ = collect(gt, t);
        sol = double(root(equ, t));

        % find t with smallest cost
        mmin = Inf;
        for j = 1:size(sol,1)
            t = real(sol(j));
            st = t^2/(1+lf^2*t^2) + (c*t+d)^2/((a*t+b)^2+rf^2*(c*t+d)
                                                                   ^2);
            if st < mmin
                mmin = st;
                best_t = t;
            end
        end

        ll = [best_t * lf, 1, -best_t];
        rl = [-rf*(c*best_t+d), a*best_t+b, c*best_t+d];

        n_lx(i,:) = (inv(lT) * lR' * ...
            [-ll(1)*ll(3), -ll(2)*ll(3), ll(1)^2+ll(2)^2]')';
        n_rx(i,:) = (inv(rT) * rR' * ...
            [-rl(1)*rl(3), -rl(2)*rl(3), rl(1)^2+rl(2)^2]')';
    end
end

function sX = getScenePoints(lx, rx, U, S, V, F)
    assert(size(lx,1) == size(rx,1));
    n = size(lx,1);

    rP = getRP(U, S, V);
    sX = zeros(n, 4);
    for i = 1:n
        rl = (F * lx(i,:)')';
        nl = [-rl(2)*rx(i,3), rl(1)*rx(i,3),...
              rl(2)*rx(i,1)-rl(1)*rx(i,2)];
        p = nl * rP;

        sX(i,1:3) = p(4) * lx(i,:)';
        sX(i,4) = -p(1:3) * lx(i,:)';
    end
end

function partialW = angleJacobian(W)
    I = eye(3);
    theta = norm(W);
    nW = W/norm(W);
    nWx = skewMatrix(nW);

    partialW = zeros(9,3);
    for i = 1:3
        dW = cos(theta)*nW(i,1)*nWx + sin(theta)*nW(i,1)*nWx*nWx ...
            + sin(theta)/theta*skewMatrix(I(i,:)'-nW(i,1)*nW) ...
            + (1-cos(theta))/theta ...
            * (I(i,:)'*nW' + nW*I(i,:) - 2*nW(i,1)*(nW*nW'));
        partialW(1:9,i) = vector(dW);
    end
end

function [n_lx, n_rx] = estimate(sX, U, S, V)
    rP = getRP(U, S, V);
    n_lx = ([eye(3), zeros(3,1)] * sX')';
```

```
    n_rx = (rP * sX')';
end

% n_lx, n_rx have been normalized by their third columns
function ex = calEpsilon(lx, rx, n_lx, n_rx)
    n_lx = n_lx ./ n_lx(:,3);
    n_rx = n_rx ./ n_rx(:,3);
    ex = [vector(lx(:,1:2)) - vector(n_lx(:,1:2)); ...
          vector(rx(:,1:2)) - vector(n_rx(:,1:2))];
end

function vx = vector(x)
    vx = x';
    vx = vx(:);
end

function p = homoParameterize(v)
    v = v / norm(v);
    a = v(1);
    b = v(2:end);
    p = 2 / SINC(acos(a)) * b;
end

function v = homoDeparameterize(p)
    v = [cos(norm(p)/2), SINC(norm(p)/2)/2 * p']';
end

function ret = SINC(x)
    if x == 0
        ret = 1;
    else
        ret = sin(x) / x;
    end
end

function ret = DSINC(x)
    if x == 0
        ret = 0;
    else
        ret = cos(x) / x - sin(x) / x^2;
    end
end

function W = angleParameterize(M)
    [~, ~, V] = svd(M - eye(3));
    a = V(:,end);
    b = [M(3,2)-M(2,3); M(1,3)-M(3,1); M(2,1)-M(1,2)];

    sin_theta = 0.5 * a' * b;
    cos_theta = 0.5 * (trace(M)-1);
    theta = atan2(sin_theta, cos_theta);

    W = theta * a / norm(a);
end

function M = angleDeparameterize(W)
    theta = norm(W);

    if theta < 1e-7
        assert(false);
    else
        sin_theta = sin(theta);
        cos_theta = cos(theta);
        nW = W / norm(W);
        nWx = skewMatrix(nW);
```

```
        M = cos_theta*eye(3) + sin_theta*nWx ...
            + (1-cos_theta)*(nW*nW');
    end
end

function Wx = skewMatrix(W)
    Wx = [0, -W(3,1), W(2,1); ...
          W(3,1), 0, -W(1,1); ...
          -W(2,1), W(1,1), 0];
end

function delta = calDelta(lx, rx, lT, rT, F)
    n = size(lx,1);
    lx = lx / lT';
    lx = lx ./ lx(:,3);
    rx = rx / rT';
    rx = rx ./ rx(:,3);
    F = rT' * F * lT;

    delta = zeros(n,4);
    for i = 1:n
        J = [rx(i,1)*F(1,1)+rx(i,2)*F(2,1)+F(3,1), ...
             rx(i,1)*F(1,2)+rx(i,2)*F(2,2)+F(3,2), ...
             lx(i,1)*F(1,1)+lx(i,2)*F(1,2)+F(1,3), ...
             lx(i,1)*F(2,1)+lx(i,2)*F(2,2)+F(2,3)];
        e = rx(i,:) * F * lx(i,:)';
        lambda = -e / (J*J');
        delta(i,:) = J * lambda;
    end
end
```

Problem 5

Code Listing 6: Calculate Epipolar Lines label

```
function [pts] = epiline(line, xmin, xmax, ymin, ymax)
    c_pts = [-(line(2)*ymin+line(3)) / line(1), ymin; ... % bottom
             -(line(2)*ymax+line(3)) / line(1), ymax; ... % top
             xmin, -(line(1)*xmin+line(3)) / line(2); ... % left
             xmax, -(line(1)*xmax+line(3)) / line(2)]; ... % right

    pts = [];
    for i = 1:4
        if c_pts(i,1) >= xmin && c_pts(i,1) <= xmax && ...
                c_pts(i,2) >= ymin && c_pts(i,2) <= ymax
            pts = [pts; c_pts(i,:)];
        end
    end
    assert(size(pts,1) == 2);
end
```

Main Function

Code Listing 7: Main Function

```
%% Read Files
preI = imread('../dat/IMG_5030.JPG');
preI = rgb2gray(preI);
nxtI = imread('../dat/IMG_5031.JPG');
nxtI = rgb2gray(nxtI);

%% Parameters
```

```
lambda_threshold = 65;
nw = 7;

%% Problem 1: Extract Features
fprintf('\n\nProblem 1\n');

% Draw Original Figures
res = figure('visible','off');
res.PaperPosition = [0 0 8 3];

subaxis(1, 2, 1, 'sh', 0.01, 'sv', 0, 'padding', 0, 'margin', 0.01);
imshow(preI);

subaxis(1, 2, 2, 'sh', 0.01, 'sv', 0, 'padding', 0, 'margin', 0.01);
imshow(nxtI);

saveas(res, '../res/pc_origin.jpg');

if exist('../tmp/features.mat', 'file') == 2
    load('../tmp/features.mat');
else
    % Extract features
    pref = featureDetect(preI, lambda_threshold, nw);
    nxtf = featureDetect(nxtI, lambda_threshold, nw);
    save('../tmp/features.mat', 'pref', 'nxtf');
end
fprintf('Number of extracted features: %d\n', size(pref, 1));
fprintf('Number of extracted features: %d\n', size(nxtf, 1));

% Draw Feature-Deteced Figures
res = figure('visible','off');
res.PaperPosition = [0 0 8 3];

subaxis(1, 2, 1, 'sh', 0.01, 'sv', 0, 'padding', 0, 'margin', 0.01);
imshow(preI);
hold on
plot(pref(:,1),pref(:,2),'bs', 'MarkerSize', nw);
hold off

subaxis(1, 2, 2, 'sh', 0.01, 'sv', 0, 'padding', 0, 'margin', 0.01);
imshow(nxtI);
hold on
plot(nxtf(:,1),nxtf(:,2),'bs', 'MarkerSize', nw);
hold off

saveas(res, '../res/pc_detect.jpg');

%% Problem 2: Match Features
fprintf('\n\nProblem 2\n');

if exist('../tmp/match_features.mat', 'file') == 2
    load('../tmp/match_features.mat');
else
    % Match Features
    [lx, rx] = featureMatch(preI, nxtI, pref, nxtf, nw);
    save('../tmp/match_features.mat', 'lx', 'rx');
end
dx = rx-lx;
fprintf('Number of matches: %d\n', size(lx, 1));

% Draw Figures
res = figure('visible','off');
res.PaperPosition = [0 0 8 3];

subaxis(1, 2, 1, 'sh', 0.01, 'sv', 0, 'padding', 0, 'margin', 0.01);
```

```
imshow(preI);
hold on
plot(lx(:,1), lx(:,2), 'bs', 'MarkerSize', nw);
quiver(lx(:,1), lx(:,2), dx(:,1), dx(:,2), 0);
hold off

subaxis(1, 2, 2, 'sh', 0.01, 'sv', 0, 'padding', 0, 'margin', 0.01);
imshow(nxtI);
hold on
plot(rx(:,1), rx(:,2), 'bs', 'MarkerSize', nw);
quiver(rx(:,1), rx(:,2), -dx(:,1), -dx(:,2), 0);
hold off

saveas(res, '../res/pc_match.jpg');

%% Problem 3: Outlier Reection
% Homogenize
lx(:,3) = ones(size(lx,1),1);
rx(:,3) = ones(size(rx,1),1);

if exist('../tmp/outlier.mat', 'file') == 2
    load('../tmp/outlier.mat');
else
    % Outlier Rejection
    [F, bmap, MAX_TRIALS, cost] = mSAC(lx, rx);
    save('../tmp/outlier.mat', 'F', 'bmap', 'MAX_TRIALS', 'cost');
end
fprintf('\n\nProblem 3\n');
fprintf('Number of Inliers: %d\n', sum(bmap));
fprintf('Number of MaxTrials: %.10f\n', MAX_TRIALS);
fprintf('Final Cost: %.10f\n', cost);
F = F ./ norm(F,'fro');
RMSE = calRMSE(lx, rx, F);
fprintf('RMSE: %.10f\n', RMSE);

% Reject Outliers
rej_lx = lx(~bmap,:);
lx = lx(bmap,:);
rx = rx(bmap,:);
dx = rx-lx;

% Draw Figures
res = figure('visible','off');
res.PaperPosition = [0 0 8 3];

subaxis(1, 2, 1, 'sh', 0.01, 'sv', 0, 'padding', 0, 'margin', 0.01);
imshow(preI);
hold on
plot(lx(:,1), lx(:,2), 'bs', 'MarkerSize', nw);
quiver(lx(:,1), lx(:,2), dx(:,1), dx(:,2), 0);
hold off

subaxis(1, 2, 2, 'sh', 0.01, 'sv', 0, 'padding', 0, 'margin', 0.01);
imshow(nxtI);
hold on
plot(rx(:,1), rx(:,2), 'bs', 'MarkerSize', nw);
quiver(rx(:,1), rx(:,2), -dx(:,1), -dx(:,2), 0);
hold off

saveas(res, '../res/pc_robust_match.jpg');

%% Problem 4: DLT Algorithm (Linear Estimation)
F = DLT_nc(lx, rx);
F = F ./ norm(F,'fro');
fprintf('\n\nProblem 4\n');
```

```
fprintf('F_DLT:\n'); disp(F);
RMSE = calRMSE(lx, rx, F);
fprintf('RMSE: %.10f\n', RMSE);

%% Problem 5: Levenberg-Marquardt Algorithm (NonLinear Estimation)
[F, logs] = LM_nc(lx, rx, F);
F = F ./ norm(F,'fro');
fprintf('\n\nProblem 5\n');
fprintf('F_LM:\n'); disp(F);
fprintf('Error log:\n'); disp(logs);
RMSE = calRMSE(lx, rx, F);
fprintf('RMSE: %.10f\n', RMSE);

%% Problem 6: Point to Line Mapping
s = RandStream('mt19937ar','Seed',514);

% Random 3 Numbers
idx = randperm(s, size(rej_lx,1));
idx = idx(1:3);

% Select Points
lp = rej_lx(idx,:);

% Find Epilines
rl = (F * lp')';

% Draw Figures
res = figure('visible','off');
res.PaperPosition = [0 0 8 3];

subaxis(1, 2, 1, 'sh', 0.01, 'sv', 0, 'padding', 0, 'margin', 0.01);
imshow(preI);
hold on
plot(lp(:,1), lp(:,2),'bs', 'MarkerSize', nw);
hold off

subaxis(1, 2, 2, 'sh', 0.01, 'sv', 0, 'padding', 0, 'margin', 0.01);
imshow(nxtI);
hold on
for i = 1:3
    pts = epiline(rl(i,:), 1, size(preI,2), 1, size(preI,1));
    plot(pts(:,1), pts(:,2), 'r-');
end
hold off

saveas(res, '../res/pc_mapping.jpg');
```

Utility Function

Code Listing 8: Calculate Rooted Mean Squared Error

```
function RMSE = calRMSE(lx, rx, F)
    res = rx * F * lx';
    RMSE = sqrt(sum(sum(res.^2)));
end
```

Code Listing 9: SubAxis Utility: Main Code

```
function h=subaxis(varargin)
%SUBAXIS Create axes in tiled positions. (just like subplot)
%   Usage:
%      h=subaxis(rows,cols,cellno[,settings])
%      h=subaxis(rows,cols,cellx,celly[,settings])
```

```
%      h=subaxis(rows,cols,cellx,celly,spanx,spany[,settings])
%
% SETTINGS: Spacing,SpacingHoriz,SpacingVert
%           Padding,PaddingRight,PaddingLeft,PaddingTop,PaddingBottom
%           Margin,MarginRight,MarginLeft,MarginTop,MarginBottom
%           Holdaxis
%
%           all units are relative (i.e. from 0 to 1)
%
%           Abbreviations of parameters can be used.. (Eg MR instead
                                 of MarginRight)
%           (holdaxis means that it wont delete any axes below.)
%
%
% Example:
%
%   >> subaxis(2,1,1,'SpacingVert',0,'MR',0);
%   >> imagesc(magic(3))
%   >> subaxis(2,'p',.02);
%   >> imagesc(magic(4))
%
% 2001-2014 / Aslak Grinsted  (Feel free to modify this code.)

f=gcf;



UserDataArgsOK=0;
Args=get(f,'UserData');
if isstruct(Args)
    UserDataArgsOK=isfield(Args,'SpacingHorizontal')&isfield(Args,'
                                       Holdaxis')&isfield(Args,'rows')
                                       &isfield(Args,'cols');
end
OKToStoreArgs=isempty(Args)|UserDataArgsOK;

if isempty(Args)&&(~UserDataArgsOK)
    Args=struct('Holdaxis',0, ...
        'SpacingVertical',0.05,'SpacingHorizontal',0.05, ...
        'PaddingLeft',0,'PaddingRight',0,'PaddingTop',0,'PaddingBottom
                                            ',0, ...
        'MarginLeft',.1,'MarginRight',.1,'MarginTop',.1,'MarginBottom'
                                            ,.1, ...
        'rows',[],'cols',[]);
end
Args=parseArgs(varargin,Args,{'Holdaxis'},{'Spacing' {'sh','sv'}; '
                                   Padding' {'pl','pr','pt','pb'}; '
                                   Margin' {'ml','mr','mt','mb'}});

if (length(Args.NumericArguments)>2)
    Args.rows=Args.NumericArguments{1};
    Args.cols=Args.NumericArguments{2};
%remove these 2 numerical arguments
    Args.NumericArguments={Args.NumericArguments{3:end}};
end

if OKToStoreArgs
    set(f,'UserData',Args);
end


switch length(Args.NumericArguments)
   case 0
       return % no arguments but rows/cols....
   case 1
```

```
        if numel(Args.NumericArguments{1}) > 1 % restore subplot(m,n,[x
                                               y]) behaviour
            [x1 y1] = ind2sub([Args.cols Args.rows],Args.
                                               NumericArguments{1}(1));
                                               % subplot and ind2sub
                                               count differently (
                                               column instead of row
                                               first) --> switch cols/
                                               rows
            [x2 y2] = ind2sub([Args.cols Args.rows],Args.
                                               NumericArguments{1}(end)
                                               );
        else
            x1=mod((Args.NumericArguments{1}-1),Args.cols)+1; x2=x1;
            y1=floor((Args.NumericArguments{1}-1)/Args.cols)+1; y2=y1;
        end
%        x1=mod((Args.NumericArguments{1}-1),Args.cols)+1; x2=x1;
%        y1=floor((Args.NumericArguments{1}-1)/Args.cols)+1; y2=y1;
    case 2
       x1=Args.NumericArguments{1};x2=x1;
       y1=Args.NumericArguments{2};y2=y1;
    case 4
       x1=Args.NumericArguments{1};x2=x1+Args.NumericArguments{3}-1;
       y1=Args.NumericArguments{2};y2=y1+Args.NumericArguments{4}-1;
    otherwise
       error('subaxis argument error')
end


cellwidth=((1-Args.MarginLeft-Args.MarginRight)-(Args.cols-1)*Args.
                                   SpacingHorizontal)/Args.cols;
cellheight=((1-Args.MarginTop-Args.MarginBottom)-(Args.rows-1)*Args.
                                   SpacingVertical)/Args.rows;
xpos1=Args.MarginLeft+Args.PaddingLeft+cellwidth*(x1-1)+Args.
                                   SpacingHorizontal*(x1-1);
xpos2=Args.MarginLeft-Args.PaddingRight+cellwidth*x2+Args.
                                   SpacingHorizontal*(x2-1);
ypos1=Args.MarginTop+Args.PaddingTop+cellheight*(y1-1)+Args.
                                   SpacingVertical*(y1-1);
ypos2=Args.MarginTop-Args.PaddingBottom+cellheight*y2+Args.
                                   SpacingVertical*(y2-1);

if Args.Holdaxis
    h=axes('position',[xpos1 1-ypos2 xpos2-xpos1 ypos2-ypos1]);
else
    h=subplot('position',[xpos1 1-ypos2 xpos2-xpos1 ypos2-ypos1]);
end


set(h,'box','on');
%h=axes('position',[x1 1-y2 x2-x1 y2-y1]);
set(h,'units',get(gcf,'defaultaxesunits'));
set(h,'tag','subaxis');



if (nargout==0), clear h; end;
```

Code Listing 10: SubAxis Utility: Argument Parse Code

```
function ArgStruct=parseArgs(args,ArgStruct,varargin)
% Helper function for parsing varargin.
%
%
% ArgStruct=parseArgs(varargin,ArgStruct[,FlagtypeParams[,Aliases]])
```

```
%
% * ArgStruct is the structure full of named arguments with default
                                   values.
% * Flagtype params is params that don't require a value. (the value
                                   will be set to 1 if it is present)
% * Aliases can be used to map one argument-name to several argstruct
                                   fields
%
%
% example usage:
% --------------
% function parseargtest(varargin)
%
% %define the acceptable named arguments and assign default values
% Args=struct('Holdaxis',0, ...
%        'SpacingVertical',0.05,'SpacingHorizontal',0.05, ...
%        'PaddingLeft',0,'PaddingRight',0,'PaddingTop',0,'
                                   PaddingBottom',0, ...
%        'MarginLeft',.1,'MarginRight',.1,'MarginTop',.1,'MarginBottom
                                   ',.1, ...
%        'rows',[],'cols',[]);
%
% %The capital letters define abrreviations.
% %  Eg. parseargtest('spacingvertical',0) is equivalent to
                                   parseargtest('sv',0)
%
% Args=parseArgs(varargin,Args, ... % fill the arg-struct with values
                                   entered by the user
%           {'Holdaxis'}, ... %this argument has no value (flag-type)
%           {'Spacing' {'sh','sv'}; 'Padding' {'pl','pr','pt','pb'}; '
                                   Margin' {'ml','mr','mt','mb'}});
%
% disp(Args)
%
%
%
%
% Aslak Grinsted 2004

%
                                   -------------------------------------------------------

%   Copyright (C) 2002-2004, Aslak Grinsted
%   This software may be used, copied, or redistributed as long as it
                                   is not
%   sold and this copyright notice is reproduced on each copy made.
                                   This
%   routine is provided as is without any express or implied
                                   warranties
%   whatsoever.

persistent matlabver

if isempty(matlabver)
    matlabver=ver('MATLAB');
    matlabver=str2double(matlabver.Version);
end

Aliases={};
FlagTypeParams='';

if (length(varargin)>0)
    FlagTypeParams=lower(strvcat(varargin{1}));  %#ok
    if length(varargin)>1
        Aliases=varargin{2};
```

```
    end
end


%---------------Get "numeric" arguments
NumArgCount=1;
while (NumArgCount<=size(args,2))&&(~ischar(args{NumArgCount}))
    NumArgCount=NumArgCount+1;
end
NumArgCount=NumArgCount-1;
if (NumArgCount>0)
    ArgStruct.NumericArguments={args{1:NumArgCount}};
else
    ArgStruct.NumericArguments={};
end


%--------------Make an accepted fieldname matrix (case insensitive)
Fnames=fieldnames(ArgStruct);
for i=1:length(Fnames)
    name=lower(Fnames{i,1});
    Fnames{i,2}=name; %col2=lower
    Fnames{i,3}=[name(Fnames{i,1}~=name) ' ']; %col3=abreviation
                                       letters (those that are
                                       uppercase in the ArgStruct) e.g
                                       . SpacingHoriz->sh
    %the space prevents strvcat from removing empty lines
    Fnames{i,4}=isempty(strmatch(Fnames{i,2},FlagTypeParams)); %Does
                                       this parameter have a value?
end
FnamesFull=strvcat(Fnames{:,2}); %#ok
FnamesAbbr=strvcat(Fnames{:,3}); %#ok

if length(Aliases)>0
    for i=1:length(Aliases)
        name=lower(Aliases{i,1});
        FieldIdx=strmatch(name,FnamesAbbr,'exact'); %try abbreviations
                                            (must be exact)
        if isempty(FieldIdx)
            FieldIdx=strmatch(name,FnamesFull); %&??????? exact or not
                                                ?
        end
        Aliases{i,2}=FieldIdx;
        Aliases{i,3}=[name(Aliases{i,1}~=name) ' ']; %the space
                                           prevents strvcat from
                                           removing empty lines
        Aliases{i,1}=name; %dont need the name in uppercase anymore
                                           for aliases
    end
    %Append aliases to the end of FnamesFull and FnamesAbbr
    FnamesFull=strvcat(FnamesFull,strvcat(Aliases{:,1})); %#ok
    FnamesAbbr=strvcat(FnamesAbbr,strvcat(Aliases{:,3})); %#ok
end

%--------------get parameters--------------------
l=NumArgCount+1;
while (l<=length(args))
    a=args{l};
    if ischar(a)
        paramHasValue=1; % assume that the parameter has is of type '
                                           param',value
        a=lower(a);
        FieldIdx=strmatch(a,FnamesAbbr,'exact'); %try abbreviations (
                                           must be exact)
        if isempty(FieldIdx)
```

```
            FieldIdx=strmatch(a,FnamesFull);
        end
        if (length(FieldIdx)>1) %shortest fieldname should win
            [mx,mxi]=max(sum(FnamesFull(FieldIdx,:)==' ',2));%#ok
            FieldIdx=FieldIdx(mxi);
        end
        if FieldIdx>length(Fnames) %then it's an alias type.
            FieldIdx=Aliases{FieldIdx-length(Fnames),2};
        end

        if isempty(FieldIdx)
            error(['Unknown named parameter: ' a])
        end
        for curField=FieldIdx' %if it is an alias it could be more
                                           than one.
            if (Fnames{curField,4})
                if (l+1>length(args))
                    error(['Expected a value for parameter: ' Fnames{
                                                       curField,1}])
                end
                val=args{l+1};
            else %FLAG PARAMETER
                if (l<length(args)) %there might be a explicitly
                                                   specified value for
                                                    the flag
                    val=args{l+1};
                    if isnumeric(val)
                        if (numel(val)==1)
                            val=logical(val);
                        else
                            error(['Invalid value for flag-parameter:
                                                               '
                                                               Fnames{
                                                               curField
                                                               ,1}])
                        end
                    else
                        val=true;
                        paramHasValue=0;
                    end
                else
                    val=true;
                    paramHasValue=0;
                end
            end
            if matlabver>=6
                ArgStruct.(Fnames{curField,1})=val; %try the line
                                                   below if you get an
                                                    error here
            else
                ArgStruct=setfield(ArgStruct,Fnames{curField,1},val);
                                                   %#ok <-works in old
                                                    matlab versions
            end
        end
        l=l+1+paramHasValue; %if a wildcard matches more than one
    else
        error(['Expected a named parameter: ' num2str(a)])
    end
end
```