Zhu, Zhongjian
University of California, San Diego

March 30, 2018

HOMEWORK 3

Problem 1. **Programming: Estimation of the camera pose (rotation and translation of a calibrated camera)**

(a) **Outlier rejection**
The corresponding 3D scene and 2D image points contain both inlier and outlier correspondences. For the inlier correspondences, the scene points have been randomly generated and projected to image points under a camera projection matrix, then noise has been added to the image point coordinates. The camera calibration matrix is given by

$$K = \begin{bmatrix} 1545.0966799187809 & 0 & 639.5 \\ 0 & 1545.0966799187809 & 359.5 \\ 0 & 0 & 1 \end{bmatrix}.$$

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.

Solution
Before running this algorithm, we should first normalize the 2D inhomogeneous points. 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)
        {
                consensus_min_cost = consensus_cost
                consensus_min_cost_model = model
                number of inliers
                w = # of inliers / # of data points
                max_trials = log(1-p) / log(1-w^s)
        }
}
```

The first two step is the initialization. We set some large number to cost and trial times in order to get into the iteration. Besides, we need also to assume the probability that at least one of the random samples does not contain any outliers, p, sample size, s. Those two parameters are used to update the maximum number of trials. For the cost calculation, we need set a tolerance which is determined by the probability that a data point is an inlier, α, and the codimension, m. In the iteration step, first, the threshold here is not important, so we can just set it to 0.

Sample selection step is to randomly select three 3D inhomogeneous points in world coordinate frame and calculate the distances between each two of them,say a, b and c. Then use the corresponding 2D normalized homogeneous points to get the direction vectors of those three 3D points, $\mathbf{j}_1$, $\mathbf{j}_2$ and $\mathbf{j}_3$.

Next step is the calculation of the model which is the camera pose here, $\hat{\mathbf{P}}$. By doing this we need first to use the 3-point algorithm of Finsterwalder to calculate the coordinates of those three points in camera coordinate frame.

First, we know the side lengths of these three points

$$a = \|\mathbf{p}_2 - \mathbf{p}_3\|$$

$$b = \|\mathbf{p}_1 - \mathbf{p}_3\|$$

$$c = \|\mathbf{p}_1 - \mathbf{p}_2\|$$

where $\mathbf{p}_i$ is the 3D point in camera coordinate frame, which is what we need. Also we know the projected 2D points, $\mathbf{q}_i$, so we can get the direction vector of the 3D points, $\mathbf{j}_i$. Hence, we can get the angle between each two of them

$$\cos\alpha = \mathbf{j}_2 \cdot \mathbf{j}_3$$

$$\cos\beta = \mathbf{j}_1 \cdot \mathbf{j}_3$$

$$\cos\gamma = \mathbf{j}_1 \cdot \mathbf{j}_2$$

and represent 3D points as

$$\mathbf{p}_i = s_i \mathbf{j}_i, i = 1, 2, 3.$$

So we only need to calculate s_i.

$$s_2 = u s_1$$

$$s_3 = v s_1$$

$$s_1^2 = \frac{a^2}{u^2 + v^2 - 2uv\cos\alpha} = \frac{b^2}{1 + v^2 - 2v\cos\beta} = \frac{c^2}{1 + u^2 - 2u\cos\gamma}.$$

The solution is summarized by Finsterwalder and Scheufele. The main idea is to find a root of a cubic polynomial and the roots of two quadratic polynomials. The cubic equation for λ is

$$G\lambda^3 + H\lambda^2 + I\lambda + J = 0$$

where

$$G = c^2(c^2\sin^2\beta - b^2\sin^2\gamma)$$

$$H = b^2(b^2 - a^2)\sin^2\gamma + c^2(c^2 + 2a^2)\sin^2\beta + 2b^2c^2(-1 + \cos\alpha\cos\beta\cos\gamma)$$

$$I = b^2(b^2 - c^2)\sin^2\alpha + a^2(a^2 + 2c^2)\sin^2\beta + 2a^2b^2(-1 + \cos\alpha\cos\beta\cos\gamma)$$

$$J = a^2(a^2\sin^2\beta - b^2\sin^2\alpha).$$

The quadratic equations are

$$Au^2 + 2Buv + Cv^2 + 2Du + 2Ev + F = 0$$

and

$$(B^2 - AC)u^2 + 2(BE - CD)u + E^2 - CF = (up + q)^2$$

where

$$A = 1 + \lambda$$
$$B = -\cos\alpha$$
$$C = \frac{b^2 - a^2}{b^2} - \lambda\frac{c^2}{b^2}$$
$$D = -\lambda\cos\gamma$$
$$E = (\frac{a^2}{b^2} + \lambda\frac{c^2}{b^2})\cos\beta$$
$$F = \frac{-a^2}{b^2} + \lambda(\frac{b^2 - c^2}{b^2}).$$

The solutions are

$$u_{large} = \frac{sgn(L)}{K}[|L| + \sqrt{(L^2 - KM)}]$$
$$u_{small} = \frac{M}{Ku_{large}}$$
$$v = um + n$$

where

$$K = b^2 - m^2c^2$$
$$L = c^2(\cos\beta - n)m - b^2\cos\gamma$$
$$M = -c^2n^2 + 2c^2n\cos\beta + b^2 - c^2$$
$$m = [-B \pm p]/C$$
$$n = [-(E \mp q)]/C$$
$$p = \sqrt{B^2 - AC}$$
$$q = sign(BE - CD)\sqrt{E^2 - CF}.$$

This algorithm may have complex roots which we do not need. So here we have to find at least one real solution. If all the solutions are complex, we can re-select three points and run this algorithm again.

Since this algorithm might give us as many as four real solutions, we need to compare them. The standard is to choose the one that has the minimal error. So first we use these estimated points in camera coordinate frame and their corresponding points in world coordinate frame to estimate

$\hat{\mathbf{P}}$. Once we have $\hat{\mathbf{P}}$, we can calculate the mean squared error for all solutions and choose the one that has the minimum MSE. First we calculate

$$\mu_x = \frac{1}{n}\sum_{i=1}^{n}\mathbf{x}_i$$

$$\mu_y = \frac{1}{n}\sum_{i=1}^{n}\mathbf{y}_i$$

$$\sigma_x^2 = \frac{1}{n}\sum_{i=1}^{n}\|\mathbf{x}_i - \mu_{\mathbf{x}}\|^2$$

$$\sigma_y^2 = \frac{1}{n}\sum_{i=1}^{n}\|\mathbf{y}_i - \mu_{\mathbf{y}}\|^2$$

$$\Sigma_{xy} = \frac{1}{n}\sum_{i=1}^{n}(\mathbf{y}_i - \mu_{\mathbf{y}})(\mathbf{x}_i - \mu_{\mathbf{x}}).$$

When $\text{rank}(\Sigma_{xy} \geqslant m-1)$

$$\mathbf{R} = \mathbf{USV}^\top$$

$$\mathbf{t} = \mu_{\mathbf{y}} - c\mathbf{R}\mu_{\mathbf{x}}$$

$$c = \frac{1}{\sigma_x^2}tr(DS)$$

where $UDV^\top$ is a singular value decomposition of Σ_{xy} and

$$\mathbf{S} = \begin{cases}\mathbf{I} & \text{if } \det(\Sigma_{xy}) \geqslant 0\\ diag(1,1,...,-1) & \text{if } \det(\Sigma_{xy}) < 0\end{cases}.$$

Finally, the mean squared error is

$$e^2(\mathbf{R},\mathbf{t},c) = \frac{1}{n}\sum_{i=1}^{n}\|\mathbf{y}_i - (c\mathbf{R}\mathbf{x}_i + \mathbf{t})\|^2.$$

Next step is to calculate the error for each points projected using the camera pose, which is the squared distance. Next we need 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 choose $\alpha = 0.95$ and $m = 2$. 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

Eventually, we will find the inliers. However, 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, the range is from about 20 to 50. 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 sequence which can get 48 inliers. The number of maximum trials is 6.4189, so it runs 7 times to find the consensus set.

(b) **Linear estimation**
Estimate the nomalized camera projection matrix $\hat{\mathbf{P}}_{\mathbf{linear}} = [\mathbf{R}_{\mathbf{linear}}|\mathbf{t}_{\mathbf{linear}}]$ from the resulting set of inlier correspondences using the linear estimation method (based on the EPnP methon).

Solution
First calculate the control points in world coordinate frame

$$\widetilde{\mathbf{X}} = \alpha_1\widetilde{\mathbf{C}}_1 + \alpha_2\widetilde{\mathbf{C}}_2 + \alpha_3\widetilde{\mathbf{C}}_3 + \alpha_4\widetilde{\mathbf{C}}_4$$

where $\alpha_1 + \alpha_2 + \alpha_3 + \alpha_4 = 1$ and $\widetilde{\mathbf{C}}_i$ is the control point in world coordinate frame. Then we can use

$$\begin{bmatrix}\widetilde{\mathbf{C}}_2 - \widetilde{\mathbf{C}}_1 & \widetilde{\mathbf{C}}_3 - \widetilde{\mathbf{C}}_1 & \widetilde{\mathbf{C}}_4 - \widetilde{\mathbf{C}}_1\end{bmatrix}\begin{bmatrix}\alpha_2\\ \alpha_3\\ \alpha_4\end{bmatrix} = \widetilde{\mathbf{X}} - \widetilde{\mathbf{C}}_1$$

to solve for α_i. Next, we use the α_i to calculate the control points in camera coordinate frame by solving

$$\begin{bmatrix}\mathbf{m}_1 & \mathbf{m}_2 & \mathbf{m}_3 & \mathbf{m}_4\end{bmatrix}\begin{bmatrix}\widetilde{\mathbf{C}}_{cam1}\\ \widetilde{\mathbf{C}}_{cam2}\\ \widetilde{\mathbf{C}}_{cam3}\\ \widetilde{\mathbf{C}}_{cam4}\end{bmatrix} = \mathbf{0}$$

where

$$\mathbf{m}_i = \begin{bmatrix}\alpha_i & 0 & -\alpha_i\hat{\tilde{x}}\\ 0 & \alpha_i & -\alpha_i\hat{\tilde{y}}\end{bmatrix}.$$

Then deparameterize 3D points in camera coordinate frame.

$$\widetilde{\mathbf{X}}_{cami} = \alpha_{i1}\widetilde{\mathbf{C}}_{cam1} + \alpha_{i2}\widetilde{\mathbf{C}}_{cam2} + \alpha_{i3}\widetilde{\mathbf{C}}_{cam3} + \alpha_{i4}\widetilde{\mathbf{C}}_{cam4}.$$

Finally scale $\widetilde{\mathbf{X}}_{cami}$ by β where

$$\beta = \begin{cases} -\sqrt{\frac{\sigma^2_{\widetilde{\mathbf{X}}}}{\sigma^2_{\widetilde{\mathbf{X}}_{cam}}}} & \text{if } \widetilde{Z}^{\mu}_{cam} < 0 \\ \\ \sqrt{\frac{\sigma^2_{\widetilde{\mathbf{X}}}}{\sigma^2_{\widetilde{\mathbf{X}}_{cam}}}} & \text{otherwise}\end{cases}.$$

Now we get the 3D points in camera coordinate frame, we use the same method as part (b) to get the camera pose, $\mathbf{R}, \mathbf{t}$.

Result

$$\mathbf{R}_{linear} = \begin{bmatrix} 0.278447371550749 & -0.690718604868692 & 0.667364121124838 \\ 0.661808637312523 & -0.365573521123451 & -0.65449624004416 \\ 0.696043381446171 & 0.623910097323043 & 0.355330552589179 \end{bmatrix}.$$

$$\mathbf{t}_{linear} = \begin{bmatrix} 5.58202213963106 \\ 7.59512640749481 \\ 175.906948079384 \end{bmatrix}.$$

(c) **Nonlinear estimation**
Use $\mathbf{R}_{linear}$ and $\mathbf{t}_{linear}$ as an initial estimate to the Levenberg-Marquardt estimation method to determine the Maximum Likelihood estimate of the camera pose that minimizes the projection error under the normalized camera projection matrix $\hat{\mathbf{P}} = [\mathbf{R}|\mathbf{t}]$.

Solution
The Levenberg-Marquardt algorithm is as follow:

I $\lambda = 0.001$;
$\epsilon = \widetilde{\mathbf{x}} - \hat{\widetilde{\mathbf{x}}}$

II $\mathbf{J} = \frac{\partial \hat{\widetilde{\mathbf{x}}}}{\partial \hat{\mathbf{p}}}$

III $\mathbf{J}^\top \mathbf{\Sigma^{-1}} \mathbf{J} \delta = \mathbf{J}^\top \mathbf{\Sigma^{-1}} \epsilon$

IV $(\mathbf{J}^\top \mathbf{\Sigma^{-1}} \mathbf{J} + \lambda \mathbf{I})\delta = \mathbf{J}^\top \mathbf{\Sigma^{-1}} \epsilon$, solve for δ.

V $\hat{\mathbf{p}}_\mathbf{0} = \hat{\mathbf{p}} + \delta$, candidate parameter vector.

VI $\hat{\mathbf{p}}_\mathbf{0} \mapsto \hat{\widetilde{\mathbf{x}}}_\mathbf{0}$;
$\epsilon_0 = \widetilde{\mathbf{x}} - \hat{\widetilde{\mathbf{x}}}_\mathbf{0}$

VII If $\epsilon_\mathbf{0}^\top \mathbf{\Sigma}_{\widetilde{\mathbf{x}}}^{\mathbf{-1}} \epsilon_\mathbf{0}$ cost less than $\epsilon^\top \mathbf{\Sigma}_{\widetilde{\mathbf{x}}}^{\mathbf{-1}} \epsilon$,
$\hat{\mathbf{p}} = \hat{\mathbf{p}}_\mathbf{0}$, $\epsilon = \epsilon_0$, $\lambda = 0.1\lambda$, go to step II or terminate.
Else,
$\lambda = 10\lambda$, go to step IV

First we need to parameterize $\mathbf{R}$, $\mathbf{R} = e^{[\omega]_\times}$.

$$(\mathbf{R} - \mathbf{I})\mathbf{v} = \mathbf{0}$$

$\mathbf{v}$ is the null space of $\mathbf{R} - \mathbf{I}$. So it can be calculated using SVD.

$$\sin\theta = \frac{\mathbf{v}^\top \mathbf{v}}{2}$$

$$\cos\theta = \frac{Tr(\mathbf{R} - 1)}{2}$$

$$\theta = \tan^{-1}(\frac{\sin\theta}{\cos\theta})$$

. Finally, we get the deparameterized rotation matrix

$$\omega = \theta\frac{\mathbf{v}}{\|\mathbf{v}\|}$$

.

All 2D points used in this problem are normalized, $\hat{\mathbf{x}} = \mathbf{K}^{-1}\mathbf{x}$. Also $\hat{\mathbf{x}} = \widetilde{\mathbf{X}}_{rotate} + \mathbf{t}$, where $\widetilde{\mathbf{X}}_{rotate} = e^{[\omega]_\times}\widetilde{\mathbf{X}}$.

$$e^{[\omega]_\times}\widetilde{\mathbf{X}} = \begin{cases} \widetilde{\mathbf{X}} + \omega \times \widetilde{\mathbf{X}} & \text{for zero or small rotations} \\ \\ \widetilde{\mathbf{X}} + \mathrm{sinc}(\|\omega\|)(\omega \times \widetilde{\mathbf{X}}) + \frac{1-\cos(\|\omega\|)}{\|\omega\|^2}\omega \times (\omega \times \widetilde{\mathbf{X}}) & \text{otherwise} \end{cases}.$$

In this case, we set angles, $\theta = \|\omega\|$, smaller than 5 degrees, $\pi/36$ radians, are small.
The Jacobian matrix here is a 2n×6 matrix, where n is the total number of samples. For every two rows $\mathbf{A}_i = \frac{\partial\hat{\hat{\mathbf{x}}}_i}{\partial\omega^\top, \mathbf{t}^\top}$. Since we have already now the function $f : \mathbf{X} \mapsto \hat{\hat{\mathbf{x}}}$, so we can use the Symbolic Toolbox of MATLAB to get the Jacobian matrix directly. Since here we use normalized points, we still need to calculate covariance propagation. We assume the covariance matrix of $\widetilde{\mathbf{x}}$ is an identity matrix, $\mathbf{I}$. So $\Sigma_{\hat{\hat{\mathbf{x}}}} = \mathbf{J}\mathbf{J}^\top$, where $\mathbf{J}$ is the first two columns and rows of $\mathbf{K}^{-1}$. In step three and four, we get δ and then in step five we update the parameters. Next, deparameterize ω by doing matrix logarithm, $\omega = \ln\mathbf{R}$, and get the camera pose matrix. Project 3D points using camera pose matrix and dehomogenenize the projected points. Calculate the current cost using the normalized inhomogeneous points and the projected inhomogeneous points. The terminate condition is the difference of cost between two iterations is less than 0.00001.

Result

The costs of every step are

Iteration	Cost
0	71.5421
1	71.4556
2	71.4555

The parameters are

$$\omega_{LM} = \begin{bmatrix} 1.33677874302384 \\ -0.0307072673745825 \\ 1.41387299017944 \end{bmatrix}.$$

$$\mathbf{R}_{LM} = \begin{bmatrix} 0.278334558217948 & -0.69081471280433 & 0.667311700987395 \\ 0.661190896671845 & -0.366131638967175 & -0.654808537746334 \\ 0.696675298729545 & 0.623476267006436 & 0.354853311411656 \end{bmatrix}.$$

$$\mathbf{t}_{LM} = \begin{bmatrix} 5.57027683548205 \\ 7.52808246620935 \\ 175.910218890022 \end{bmatrix}.$$

Appendix

```
close all; clear; clc;
%% Load Data
x_img_inhomo = load('../data/hw3_points2D.txt')';
X_wld_inhomo = load('../data/hw3_points3D.txt')';
x_img_homo = padarray(x_img_inhomo,[1,0],1,'post');
X_wld_homo = padarray(X_wld_inhomo,[1,0],1,'post');
K = [1545.0966799187809,0,639.5;0,1545.0966799187809,359.5;0,0,1];
x_img_norm_homo = K\x_img_homo;
x_img_norm_homo = x_img_norm_homo./sign(x_img_norm_homo(3,:)) / norm(x_img_norm_homo);
x_img_norm_inhomo = x_img_norm_homo(1:2,:) ./ x_img_norm_homo(3,:);
n = size(x_img_inhomo,2);
%% mSAC
rng(53)
consensus_min_cost = inf;
max_trials = inf;
trials = 0;
threshold = 0;
tolerance = chi2inv(0.95,2) * 1;
p = 0.99;
s = 3;
k = 1;
while trials < max_trials && consensus_min_cost > threshold
    %% Select a random sample
    i = randperm(n,3)';
    p1_wld = X_wld_inhomo(:,i(1));
    p2_wld = X_wld_inhomo(:,i(2));
    p3_wld = X_wld_inhomo(:,i(3));

    q1 = [x_img_norm_inhomo(:,i(1));1] ;
    q2 = [x_img_norm_inhomo(:,i(2));1] ;
    q3 = [x_img_norm_inhomo(:,i(3));1] ;

    j1 = q1/norm(q1);
    j2 = q2/norm(q2);
    j3 = q3/norm(q3);

    a = norm(p2_wld-p3_wld);
    b = norm(p1_wld-p3_wld);
    c = norm(p1_wld-p2_wld);
    %% Calculate model
    %       Finsterwalder
    [X1_cam,X2_cam,X3_cam]=Finsterwalder(a,b,c,j1,j2,j3);
    %       Projection Matrix
    P_hat = CalRt3P(p1_wld,p2_wld,p3_wld,X1_cam,X2_cam,X3_cam);
```

```
    %           Check if solution exist
    if ~numel(P_hat)
        continue
    end
    trials = trials + 1;
    %% Error for each point
    x_pro_homo = K * P_hat * X_wld_homo;
    x_pro_inhomo = x_pro_homo(1:2,:) ./ x_pro_homo(3,:);
    error = sum((x_img_inhomo - x_pro_inhomo).^2);
    %% Calculate cost
    cost = sum(error .* (error <= tolerance) + tolerance * (error > tolerance));
    %% Update maximum trials
    if cost < consensus_min_cost
        consensus_min_cost = cost;
        P_min_cost = P_hat;
        inliers = error <= tolerance;
        w = sum(inliers) / n;
        max_trials = log(1-p) / log(1-w^s);
    end
    k = k + 1;
end
fprintf('inliers:\t %d\n', sum(inliers))
fprintf('maximum trials:\t %.4f\n\n', max_trials)
%% b
x_img_norm_inlier_inhomo = x_img_norm_inhomo(:,inliers);
X_wld_inlier_inhomo = X_wld_inhomo(:,inliers);
X_wld_inlier_homo = X_wld_homo(:,inliers);
n = size(x_img_norm_inlier_inhomo,2);
%% Control points in world coordinate frame
mu_X_wld = mean(X_wld_inlier_inhomo,2);
Sigma_X_wld = cov(X_wld_inlier_inhomo');
[~,~,V] = svd(Sigma_X_wld);
var_X_wld = trace(Sigma_X_wld);
s = sqrt(var_X_wld / 3);
C1_wld_inhomo = mu_X_wld;
C2_wld_inhomo = s*V(:,1) + mu_X_wld;
C3_wld_inhomo = s*V(:,2) + mu_X_wld;
C4_wld_inhomo = s*V(:,3) + mu_X_wld;
%% Parameterize 3D points
A = [C2_wld_inhomo - C1_wld_inhomo,...
     C3_wld_inhomo - C1_wld_inhomo,...
     C4_wld_inhomo - C1_wld_inhomo];
b = X_wld_inlier_inhomo - C1_wld_inhomo;
X_prm_wrd_homo = [1-sum(A\b);A\b];
%% Control points in camera coordinate frame
m = zeros(2*n,12);
```

```
for i = 1:n
    m(2*i - 1 : 2*i,:) = [X_prm_wrd_homo(1,i) 0 -X_prm_wrd_homo(1,i)*x_img_norm_inlier_inho
                          0 X_prm_wrd_homo(1,i) -X_prm_wrd_homo(1,i)*x_img_norm_inlier_inho
end
[~,~,V] = svd(m);
C1_cam_inhomo = V(1:3,end);
C2_cam_inhomo = V(4:6,end);
C3_cam_inhomo = V(7:9,end);
C4_cam_inhomo = V(10:12,end);
%% Deparameterize 3D points in camera coordinate frame
X_cam_inhomo = [C1_cam_inhomo,C2_cam_inhomo,C3_cam_inhomo,C4_cam_inhomo] * X_prm_wrd_homo;
mu_X_cam = mean(X_cam_inhomo,2);
Sigma_X_cam = cov(X_cam_inhomo');
var_X_cam = trace(Sigma_X_cam);
if mu_X_cam(3) < 0
    beta = -sqrt(var_X_wld/var_X_cam);
else
    beta = sqrt(var_X_wld/var_X_cam);
end
X_cam_inhomo = beta*X_cam_inhomo;
P_lin = CalRtnP(X_wld_inlier_inhomo,X_cam_inhomo);
R_lin = P_lin(:,1:3);
t_lin = P_lin(:,end);
format longg
disp('R_lin = ')
disp(R_lin)
disp('t_lin = ')
disp(t_lin)

%% LM
itr = 0;
disp('LM:')
fprintf('itr\tcost\n')
fprintf('--------------------\n')
%% Jacobian
syms w1 w2 w3 t1 t2 t3 X1 X2 X3
ww = [w1;w2;w3];
XX = [X1;X2;X3];
tt = [t1;t2;t3];
theta = norm(ww);
Xrotate_large = XX + sinc(theta/pi)*cross(ww,XX) + (1-cos(theta))/theta^2*cross(ww,cross(w
x_homo_large = Xrotate_large + tt;
x_inhomo_large = x_homo_large(1:2)/x_homo_large(3);
f_large([w1 w2 w3 t1 t2 t3 X1 X2 X3]) = jacobian(x_inhomo_large,[ww.',tt.']);

Xrotate_small = XX + cross(ww,XX);
```

```
x_homo_small = Xrotate_small + tt;
x_inhomo_small = x_homo_small(1:2)/x_homo_small(3);
f_small([w1 w2 w3 t1 t2 t3 X1 X2 X3]) = jacobian(x_inhomo_small,[ww.',tt.']);
%% Initialization
x_prj_norm_inlier_homo = [R_lin,t_lin]*X_wld_inlier_homo;
x_prj_norm_inlier_inhomo = x_prj_norm_inlier_homo(1:2,:) ./ x_prj_norm_inlier_homo(3,:);

w_lin = parameterization(R_lin);
p_hat = [w_lin;t_lin];

previous_cost = inf;
tolerance = 0.00001;

% step_1
lambda = 0.001;
K_inv = inv(K);
sigma = eye(2*n);
for i = 1:n
    sigma(2*i-1:2*i,2*i-1:2*i) = K_inv(1:2,1:2) * eye(2) * K_inv(1:2,1:2)';
end
epsilon = reshape(x_img_norm_inlier_inhomo - x_prj_norm_inlier_inhomo,[],1);




% format short
% iteration = 0;
init_cost = epsilon'*(sigma\epsilon);
current_cost = init_cost;
fprintf('%d\t%.4f\n', itr, current_cost)

J = jcb(f_large,f_small,X_wld_inlier_inhomo,w_lin,t_lin);
while tolerance < previous_cost - current_cost
    itr = itr + 1;
    % step_3_4
    delta = (J'*(sigma\J)+lambda*eye(6))\(J'*(sigma\epsilon));
    % step_5
    p_hat0 = p_hat + delta;
    w0 = p_hat0(1:3);
    t0 = p_hat0(end-2:end);
    % step_6
    R0 = deparameterization(w0);
    x_prj_norm_inlier_homo = [R0,t0]*X_wld_inlier_homo;
    x0_prj_norm_inlier_inhomo = x_prj_norm_inlier_homo(1:2,:) ./ x_prj_norm_inlier_homo(3,
    epsilon0 = reshape(x_img_norm_inlier_inhomo - x0_prj_norm_inlier_inhomo,[],1);
    % step_7
```

```
    format short
    previous_cost = epsilon'*(sigma\epsilon);
    current_cost = epsilon0'*(sigma\epsilon0);
    fprintf('%d\t%.4f\n', itr, current_cost)
    if current_cost < previous_cost
        p_hat = p_hat0;
        epsilon = epsilon0;
        lambda = 0.1*lambda;
        J = jcb(f_large,f_small,X_wld_inlier_inhomo,w0,t0);
    else
        lambda = 10*lambda;
    end
end
fprintf('--------------------------------\n\n')
w_LM = p_hat(1:3);
R_LM = deparameterization(w_LM);
t_LM = p_hat(end-2:end);
format longg
disp('w_LM = ')
disp(w_LM)
disp('R_LM = ')
disp(R_LM)
disp('t_LM = ')
disp(t_LM)
```

```
function P_hat = CalRt3P(X1_wld,X2_wld,X3_wld,X1_cam,X2_cam,X3_cam)
P_hat = [];
n = 3;
[m,soln] = size(X1_cam);

X = [X1_wld,X2_wld,X3_wld];
mu_x = mean(X,2);
sigma_x = norm(X-mu_x,'fro')^2/n;
err = inf;
for i = 1:soln
    Y = [X1_cam(:,i),X2_cam(:,i),X3_cam(:,i)];
    mu_y = mean(Y,2);
    Sigma_xy = (Y-mu_y)*(X-mu_x)'/n;
    [U,D,V] = svd(Sigma_xy);
    S = eye(m);
    if rank(Sigma_xy) < m-1
        continue
    elseif rank(Sigma_xy) == m-1
        if round(det(U)*det(V)) == -1
            S(end) = -1;
        end
```

```
    else
        if det(Sigma_xy) < 0
            S(end) = -1;
        end
    end
    R = U*S*V';
    c = trace(D*S)/sigma_x;
    t = mu_y - c*R*mu_x;
    if norm(Y - c*R*X - t)^2 < err
        err = norm(Y - c*R*X - t)^2;
        P_hat = [R,t];
    end
end
end
```

```
function P_hat = CalRtnP(X_wld_inhomo,X_cam_inhomo)
[m,n] = size(X_cam_inhomo);

mu_x = mean(X_wld_inhomo,2);
mu_y = mean(X_cam_inhomo,2);
Sigma_xy = (X_cam_inhomo-mu_y)*(X_wld_inhomo-mu_x)'/n;
[U,~,V] = svd(Sigma_xy);
S = eye(m);
if rank(Sigma_xy) >= m-1
    if det(Sigma_xy) < 0
        S(end) = -1;
    end
    R = U*S*V';
    c = 1;
    t = mu_y - c*R*mu_x;
    P_hat = [R,t];
else
    P_hat = [];
end
end
```

```
function [p1,p2,p3]=Finsterwalder(a,b,c,j1,j2,j3)
% q: 2D inhomogeneous normalized points(camero coordinate) 2*1
% p: 3D inhomogeneous points(camero coordinate) 3*1
m = zeros(2,1);
n = zeros(2,1);

p1 = [];
p2 = [];
p3 = [];

```

```
cos_alpha = dot(j2,j3);
cos_beta = dot(j1,j3);
cos_gamma = dot(j1,j2);

G = c^2*(c^2*(1-cos_beta^2) - b^2*(1-cos_gamma^2));
H = b^2*(b^2 - a^2)*(1-cos_gamma^2) + c^2*(c^2 + 2*a^2)*(1-cos_beta^2)...
    + 2*b^2*c^2*(-1 + cos_alpha*cos_beta*cos_gamma);
I = b^2*(b^2 - c^2)*(1-cos_alpha^2) + a^2*(a^2 + 2*c^2)*(1-cos_beta^2)...
    + 2*a^2*b^2*(-1 + cos_alpha*cos_beta*cos_gamma);
J = a^2*(a^2*(1-cos_beta^2) - b^2*(1-cos_alpha^2));

lambda0 = roots([G H I J]);
lambda_real = [];
for i = 1:numel(lambda0)
    if isreal(lambda0(i))
        lambda_real = [lambda_real,lambda0(i)];
    end
end
%%
for j = 1:numel(lambda_real)
    A = 1 + lambda_real(j);
    B = -cos_alpha;
    C = (b^2 - a^2)/b^2 - lambda_real(j)*c^2/b^2;
    D = -lambda_real(j)*cos_gamma;
    E = (a^2/b^2 + lambda_real(j)*c^2/b^2)*cos_beta;
    F = -a^2/b^2 + lambda_real(j)*(b^2-c^2)/b^2;

    p = sqrt(B^2 - A*C);
    q = sign(B*E - C*D)*sqrt(E^2 - C*F);

    m(1) = (-B + p)/C;
    n(1) = -(E - q)/C;
    m(2) = (-B - p)/C;
    n(2) = -(E + q)/C;

    A = b^2 -m.^2*c^2;
    B = c^2*(cos_beta - n) .* m - b^2*cos_gamma;
    C = -c^2*n.^2 + 2*c^2*n*cos_beta + b^2 - c^2;

    u_large = -sign(B) ./ A .* (abs(B) + sqrt(B.^2 - A .* C));
    u_small = C ./ (A .* u_large);

    u = [];
    for i = 1:2
        if isreal(u_large(i))
            u = [u,[u_large(i);u_small(i)]];
```

```
            end
        end
%       u
        if numel(u) ~= 0
            v = u .* m + n;
            u = reshape(u,[],1);
            v = reshape(v,[],1);

            s1 = sqrt(c^2 ./ (1 + u.^2 - 2*u*cos_gamma));
            s2 = u .* s1;
            s3 = v .* s1;

            p1 = s1' .* j1;
            p2 = s2' .* j2;
            p3 = s3' .* j3;
            break
        end
end
```

```
function J = jcb(f_large,f_small,X_wrd_inhomo,w,t)
n = size(X_wrd_inhomo,2);
J = zeros(2*n,6);
theta = norm(w);
if theta < pi/36
    f = f_small;
else
    f = f_large;
end
for i = 1:n
    J(2*i-1 : 2*i,:) = f(w(1),w(2),w(3),t(1),t(2),t(3),X_wrd_inhomo(1,i),X_wrd_inhomo(2,i)
end
end
```

```
function w = parameterization(R)
[~,~,V] = svd(R-eye(3));
v = V(:,end);
v_hat = zeros(3,1);
v_hat(1) = R(3,2) - R(2,3);
v_hat(2) = R(1,3) - R(3,1);
v_hat(3) = R(2,1) - R(1,2);
sin_theta = v'*v_hat/2;
cos_theta = (trace(R) - 1)/2;
theta = atan2(sin_theta,cos_theta);
w = theta * v / norm(v);
w = w*(1-2*pi/theta*ceil((theta-pi)/(2*pi)));
end
```

```
function R = deparameterization(w)
theta = norm(w);
wx = skew(w);
R = cos(theta)*eye(3) + sinc(theta/pi)*wx + (1-cos(theta))/theta^2*w*w';
end
```

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