16-720J: Homework 5
RANSAC and 3D Reconstruction

Wenbo Zhao

December 1, 2015

1 Theory Questions

Question 1.1 (5pts)

Assume one correspondence on the images is p_1, p_2. After normalization $\hat{p}_1 = [0, 0, 1]^T$, $\hat{p}_2 = [0, 0, 1]^T$. We have

$$\hat{p}_1^T F \hat{p}_2 = 0,$$

yields

$$[0\,0\,1] \begin{bmatrix} F_{1,1} & F_{1,2} & F_{1,3} \\ F_{2,1} & F_{2,2} & F_{2,3} \\ F_{3,1} & F_{3,2} & F_{3,3} \end{bmatrix} [0\,0\,1]^T = F_{3,3} = 0$$

Question 1.2 (10 pts)

Fundamental matrix $F = K^{-T} E K'^{-1} = K^{-T} [t]_\times R K'^{-1}$. When two cameras differ only from pure translation with x axis, we have

$$K = K' \;\; R = I$$

(I is identity matrix)

$$t = [1\,0\,0]^T$$

(suppose translation is 1) thus yields

$$F = [t]_\times = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & -1 \\ 0 & 1 & 0 \end{bmatrix}$$

such follows for $p = [x\,y\,1]^T$ and $p' = [x'\,y'\,1]^\tau$

$$p'^T F p = 0 \;\; y = y'$$

which means epipolar lines are parallel to x axis.

Question 1.3 (10 pts)

When viewing the image of an object and its image in the mirror, the correspondences in the two images differ only with inversed depth, *i.e.* $p = [x\,y\,z]$ and $p' = [x\,y\,-z]$. This is equivalent to two cameras viewing the object in the mirror plane, and the two cameras differ from each other with pure translation parallel to depth axis. Such we have proved in **Question 1.2** that in this case F is skew-symmetric ($x^T F x = 0$ for any x).

Question 1.4 (10 pts)

(1)Given p_1 and H, we have

$$p_2 = Hp_1$$

the epipolar line is

$$l = [e_2]_\times p_2 = [e_2]_\times Hp_1$$

(2) NO. Because when pure translation $t = [0\,0\,0]^T$,

$$F = K^{-T}EK'^{-1} = K^{-T}[t]_\times RK'^{-1} = 0,$$

then $l = Fp_2 = 0$, not work.

2 Implementation Questions

Fundamental matrix estimation

The Eight Point Algorithm

Question 2.1 (10 pts)

$F_{8,clean} =$

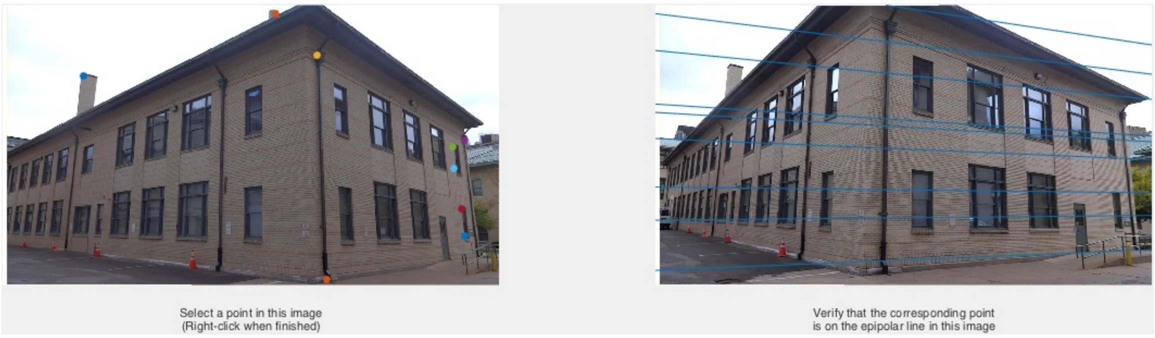
```
   -0.0000    0.0000   -0.0003
   -0.0000   -0.0000    0.0021
    0.0001   -0.0022    0.1811
```

$F_{8,noisy} =$

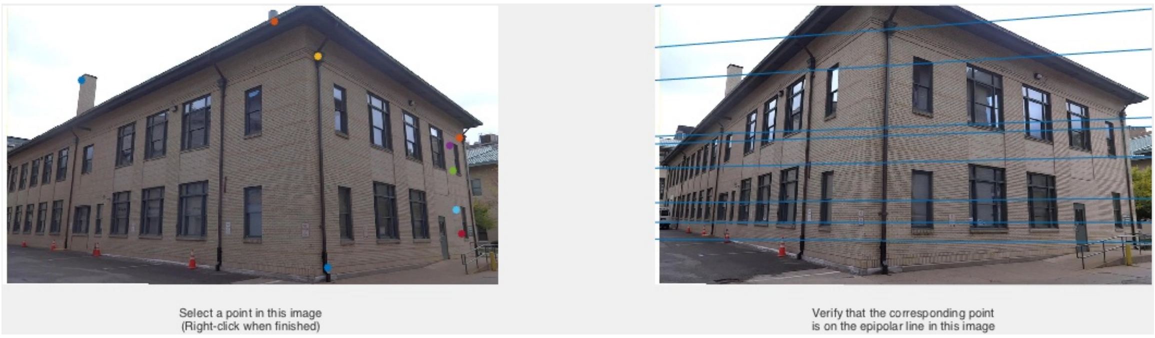
```
   -0.0000   -0.0000    0.0002
   -0.0000    0.0000    0.0017
    0.0001   -0.0019    0.0016
```

$F_{8,clean}$ is more accurate because it is more close to skew-symmetric matrix.

```
function F = eightpoint_norm(pts1, pts2, normalizaton_constant)
% Run eight point algorithm to find the Fundamental matrix between points.
% >> pts1, pts2: points, 2-by-N
% >> normalization_constant: the larger dimension of an image
% << F: fundamental matrix
%
% written by: Wenbo Zhao (wzhao1#andrew.cmu.edu)
% log: (v0.1)-(first draft)-(11-27-2015)
%
% Declaration: I lost all my scripts due to matlab crash! No backup!
% No version control! No history command! It drove me mad!
% I have to write them again! I am upset!
%
% Stay cool... Calm down... A good style is my style... Smile... and
% write
%
% Check dimension
if ¬isequal(size(pts1), size(pts2))
```



clean



noisy

Figure 1: Display epipolar.

```
    error('Dimensions of input points must match!');
end
N = size(pts1, 2);
if size(pts1, 1)≠3
    pts1 = [pts1; ones(1,N)];
    pts2 = [pts2; ones(1,N)];
end
% Normalize: using a different scheme, rendering normalization_constant
% useless
[pt1, T1] = normalize_point(pts1);
[pt2, T2] = normalize_point(pts2);
% Construct A
% A = [xx1 yx1 x1 xy1 yy1 y1 x y 1];
x = pt1(1,:); y = pt1(2,:); x1 = pt2(1,:); y1 = pt2(2,:);
A = [x'.*x1'  y'.*x1'  x1'  x'.*y1'  y'.*y1'  y1'  x'  y'  ones(1,N)'];
% SVD(A)
[¬,¬,V] = svd(A,0);
% Get F
F = reshape(V(:,9), [3, 3]);
[U1, D1, V1] = svd(F);
F = U1 * diag([D1(1,1) D1(2,2) 0]) * V1'; % rank 2
F = T2' * F * T1;
end

function [p, T] = normalize_point(pt)
% Normalize to zero mean and unit variance
% Ref: 1. the attached pdf file in ...
    http://www.researchgate.net/post/Calculating_the_fundamental_matrix_using_the_eight_
%       2. ...
    http://ece631web.groups.et.byu.net/Lectures/ECEn631%2013%20-%208%20Point%20Algorithm
% Make homo
N = size(pt, 2);
```

```
if size(pt,1)≠3
    pt = [pt; ones(1,N)];
end
% Find center
center  = mean(pt(1:2,:), 2);
% Displacement to center
dist = pt(1:2,:) - repmat(center, [1, N]);
d = sqrt(dist(1,:).^2 + dist(2,:).^2);
% Scale
s = sqrt(2)/mean(d);
% Transform matrix
T = [s 0 -s*center(1); 0 s -s*center(2); 0 0 1];
p = T*pt;
end
```

The Seven Point Algorithm

Question 2.2 (15 pts)

$F_{7,clean} =$

$$\begin{matrix} 0.0000 & -0.0000 & 0.0052 \\ 0.0000 & 0.0000 & -0.0473 \\ -0.0023 & 0.0488 & 1.0698 \end{matrix}$$

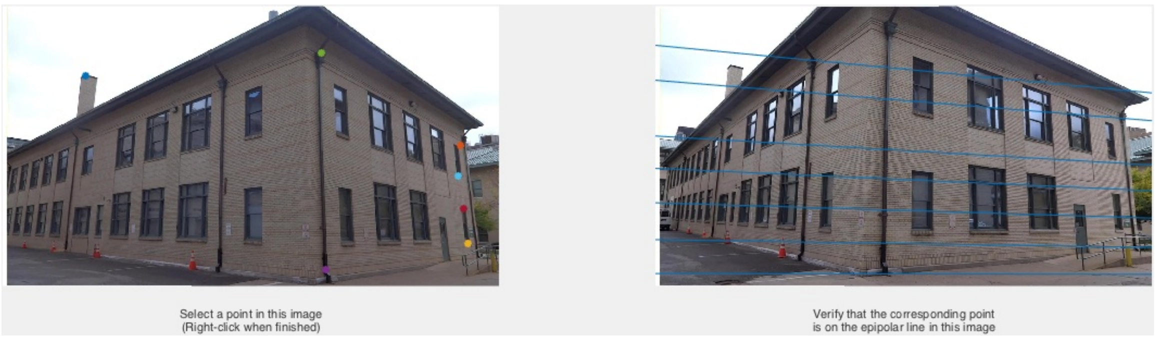


Figure 2: Display epipolar: clean, seven point.

```
function F = sevenpoint_norm(p1, p2, normalizaton_constant)
% Run seven point algorithm to find the Fundamental matrix between points.
% >> pts1, pts2: points, 2-by-N
% >> normalization_constant: the larger dimension of an image
% << F: fundamental matrix
%
% written by: Wenbo Zhao (wzhao1#andrew.cmu.edu)
% log: (v0.1)-(first draft)-(11-27-2015)
%
% Check dimension
if ¬isequal(size(p1), size(p2))
    error('Dimensions of input points must match!');
end
N = size(p1, 2);
if size(p1, 1)≠3
    p1 = [p1; ones(1,N)];
    p2 = [p2; ones(1,N)];
```

```
end
% Normalize

% --- Construct A
% A = [xx1 yx1 x1 xy1 yy1 y1 x y 1];
% ::Better way getting A
[idx idy] = find(ones(size(p1,1)));
A = (p1(idx,:).*p2(idy,:))';
[¬,¬,V] = svd(A,0);

% --- Solve det(aF1+(1-a)F2)=0
% A contains 2-D null space with linear basis combination
% f=af1+(1-a)f2 satisfying Af=0 ==> det(aF1+(1-a)F2)=0
F1 = V(:,end-1); F2 = V(:,end);
[F, a] = solve_det(F1, F2);
% F = T2' * F * T1;
end

function [p, T] = normalize_point(pt)
% Normalize to zero mean and unit variance
% Ref: 1. the attached pdf file in ...
    http://www.researchgate.net/post/Calculating_the_fundamental_matrix_using_the_eight_p
%       2. ...
    http://ece631web.groups.et.byu.net/Lectures/ECEn631%2013%20-%208%20Point%20Algorithm
% Make homo
N = size(pt, 2);
if size(pt,1)≠3
    pt = [pt; ones(1,N)];
end
% Find center
center  = mean(pt(1:2,:), 2);
% Displacement to center
dist = pt(1:2,:) - repmat(center, [1, N]);
d = sqrt(dist(1,:).^2 + dist(2,:).^2);
% Scale
s = sqrt(2)/mean(d);
% Transform matrix
T = [s 0 -s*center(1); 0 s -s*center(2); 0 0 1];
p = T*pt;
end

function [F, a] = solve_det(F1, F2)
F1 = reshape(F1, [3, 3]);
F2 = reshape(F2, [3, 3]);

% Compose det(aF1+(1-a)F2)
c3 =     -det([F2(:,1) F1(:,2) F1(:,3)]) + det([F1(:,1) F2(:,2) F2(:,3)]) ... + ...
         det([F1(:,1) F1(:,2) F1(:,3)]) + det([F2(:,1) F2(:,2) F1(:,3)]) ... + ...
         det([F2(:,1) F1(:,2) F2(:,3)]) - det([F1(:,1) F2(:,2) F1(:,3)]) ... - ...
         det([F1(:,1) F1(:,2) F2(:,3)]) - det([F2(:,1) F2(:,2) F2(:,3)]);

c2 =     det([F1(:,1) F1(:,2) F2(:,3)]) -2*det([F1(:,1) F2(:,2) ... F2(:,3)])- ...
         2*det([F2(:,1) F1(:,2) F2(:,3)]) + det([F2(:,1) F1(:,2) ... F1(:,3)])-...
         2*det([F2(:,1) F2(:,2) F1(:,3)]) + det([F1(:,1) F2(:,2) ... F1(:,3)])+...
```

```
            3*det([F2(:,1) F2(:,2) F2(:,3)]);

c1 =        det([F2(:,1) F2(:,2) F1(:,3)]) + det([F1(:,1) F2(:,2) F2(:,3)]) ... + ...
            det([F2(:,1) F1(:,2) F2(:,3)]) -3*det([F2(:,1) F2(:,2) F2(:,3)]);

c0 =        det([F2(:,1) F2(:,2) F2(:,3)]);
detF = [c3;c2;c1;c0];
a = roots(detF);

% Get F
F{1} = a(1)*F1+(1-a(1))*F2;
F{2} = a(2)*F1+(1-a(2))*F2;
F{3} = a(3)*F1+(1-a(3))*F2;

end
```

Computing F from Noisy Correspondences with RANSAC

Question 2.3 (10 pts)

$F_{7,ransac} =$

```
 0.0000    0.0000   -0.0002
-0.0000   -0.0000    0.0017
-0.0000   -0.0019    0.3012
```

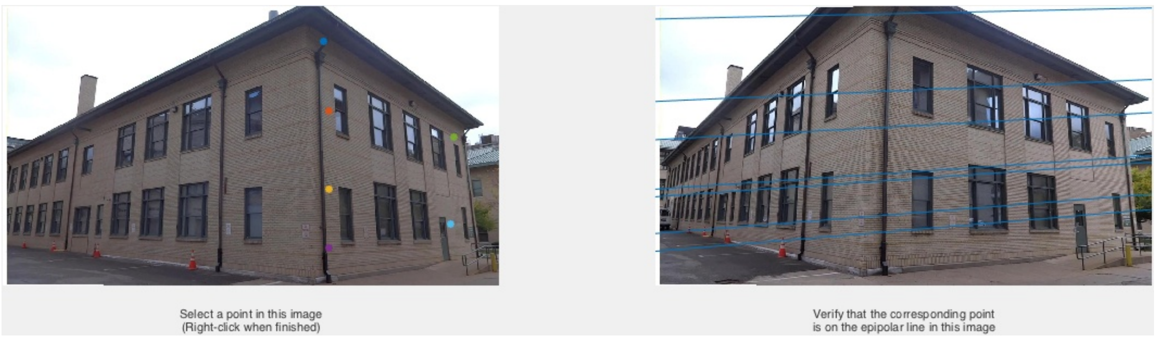


Figure 3: Display epipolar: noisy, ransac.

```
function [F_best, inliers_best] = ransacF(pts1, pts2, ...
    normalization_constant)
% Run RANSAC with seven point algorithm to find the fundamental matrix.
% >> pts1, pts2: points, 2-by-N
% >> normalization_constant: the larger dimension of an image
% << F_best: best fundamental matrix
% << inliers_best: best consensus set: indices of inliers of F, 1-by-a
%
% written by: Wenbo Zhao (wzhao1#andrew.cmu.edu)
% log: (v0.1)-(first draft)-(11-28-2015)
%
% Check inputs and set variables
if ¬isequal(size(pts1), size(pts2))
    error('Dimensions of input points must match!');
end
```

```
num_pts     = size(pts1,2); % # of points
threshold   = 0.0005; % threshold to reject outliers
sample_pts = 7; % # of points selected each time
max_iter    = 5000; % max # of iterations
d           = 1000000000; % distance
inliers     = [];
F_best      = [];
inliers_best = [];
debug       = 0;
while max_iter
% Sample
[sample_pt1, ind] = datasample(pts1', sample_pts, 'Replace', false);
sample_pt2        = pts2(:,ind)';

% Compute F
F = sevenpoint_norm(sample_pt1', sample_pt2', normalization_constant);

% Evaluate distance, add inliers to consensus set, and select best F
[F_new, dd, ¬, inliers_new] = selectBestF(F, pts1, pts2, threshold, ...
    sample_pts);
d_new = abs(sum(dd));
if length(inliers_new) < sample_pts % +1)
    max_iter = max_iter-1;
    continue
else
    if(length(inliers_new) ≥ length(inliers))
        d = abs(d_new);
        inliers = inliers_new;
        F_best = F_new;
        inliers_best = inliers;

        if debug
            fprintf('Consensus set size: %d\n', length(inliers));
            fprintf('Distance of points to epipolar line: %f\n', d);
        end
    end
    % Update count
    max_iter = max_iter-1;
end
% Update F with selected inliers

end
end


function [FF, dd, ninliers, inliers] = selectBestF(F, p1, p2, threshold, ...
    sample_pts)
% Evaluate distance between point x2 and the epipolar line Fx1
% return F with most inliers that satisfy x2'Fx1<threshold

% Check dimension
if ¬isequal(size(p1), size(p2))
    error('Dimensions of input points must match!');
end
N = size(p1, 2);
if size(p1, 1)≠3
    p1 = [p1; ones(1,N)];
    p2 = [p2; ones(1,N)];
end
```

```
flag     = 0; % indicates if enough inliers in consensus set selected
ninliers = 0; % # of inliers
if iscell(F)
    for i = 1:length(F)
        d = distPt2EpiLine(F{i}, p1, p2); % compute distance
        ind   = find(abs(d) < threshold); % find inliers
        count = length(ind);
        if (count ≥ ninliers) % && (count ≥ sample_pts+1) % make sure at ...
           least 7+1 inliers
            ninliers = count;
            inliers  = ind;
            dd = d;
            FF = F{i}; % selected F
        else
            continue
        end
    end


else
    d = distPt2EpiLine(F, p1, p2);
    ind = find(abs(d) < threshold);
    count = length(ind);
    if (count > ninliers) % && (count ≥ sample_pts+1)
        ninliers = count;
        inliers  = ind;
        dd = d;
        FF = F; % selected F
    end
end
end

function d = distPt2EpiLine(F, p1, p2)
% Compute point to epipolar line distance
if ¬isequal(size(p1), size(p2))
    error('Dimensions of input points must match!');
end
N = size(p1, 2);
if size(p1, 1)≠3
    p1 = [p1; ones(1,N)];
    p2 = [p2; ones(1,N)];
end

for j = 1:N
    dist(j) = p2(:,j)'*F*p1(:,j);
end

d = [F*p1; F'*p2];
d = dist.^2 ./ sum(d([1 2 4 5], :).^2);
end
```

Generating Novel Views of Smith Hall

Question 2.4 (30 pts)

$F_{7,ransac} =$

```
   -0.0000   -0.0000    0.0004
```

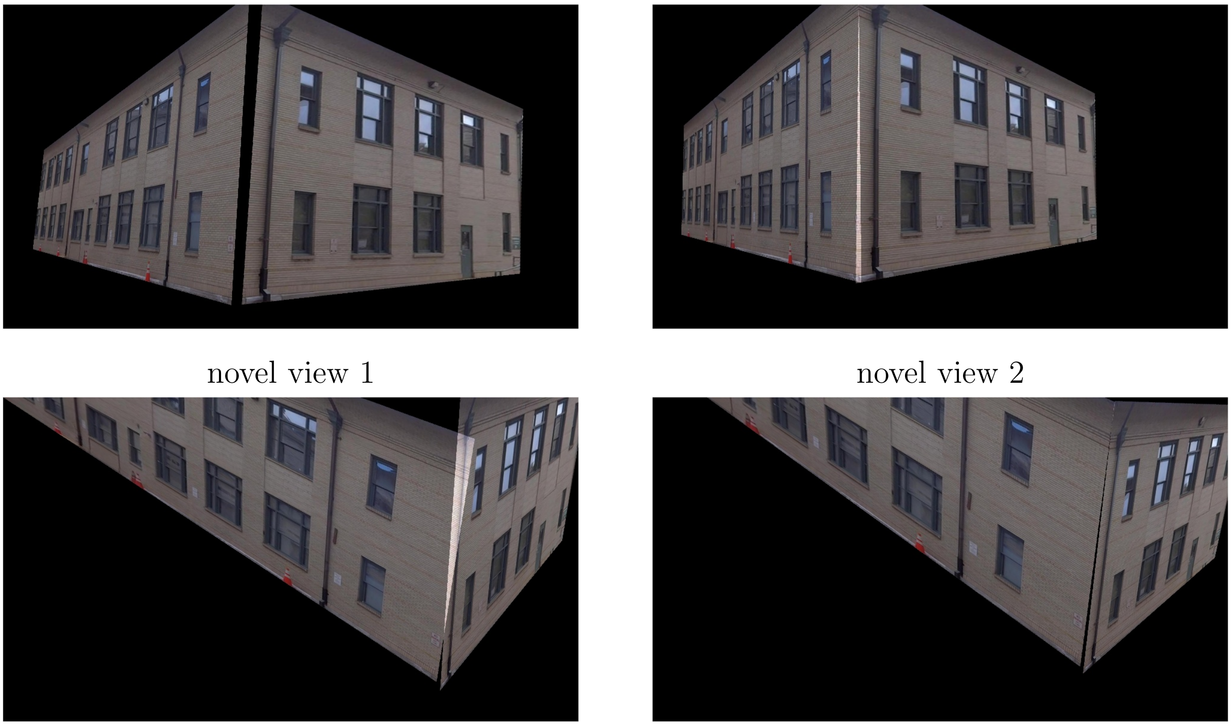
```
   -0.0000    0.0000    0.0065
   -0.0000   -0.0065    0.3715

smith_south_plane =

   -0.3110
   -0.0956
   -0.2955
    0.8982

smith_south_plane =

   -0.3110
   -0.0956
   -0.2955
    0.8982
```



novel view 1 novel view 2

novel view 3 novel view 4

Figure 4: Novel views of Smith Hall

(1) Find fundamental matrix with RANSAC using seven point algorithm.

(2) Generate camera matrix M_1 and M_2, and then compute the spatial point P with triangulation.

(3) Use RANSAC again to fit a plane model and the inliers that belong to this plane, then use the rest points to find another plane.

(4) Draw the novel view.

```
function genNovelView
addpath(genpath('.'));
load('data/K.mat'); %intrinsic parameters K
i1 = imread('data/i1.jpg');
i2 = imread('data/i2.jpg');
load ./data/noisy_correspondences.mat

% Find F
fprintf('Finding fundamental matrix...\n');
normalization_constant = max(size(i1));
[p1, T1] = normalize_point(pts1);
[p2, T2] = normalize_point(pts2);
[F, inliers]  = ransacF(p1, p2, normalization_constant);
F = T2' * F * T1

% Find P
K1 = K; K2 = K;
M2 = camera2(F, K1, K2, pts1, pts2);
M1 = K1*eye(3,4);
P = triangulate(M1, pts1, M2, pts2);

% Find plane with ransac
tic
fprintf('\nFinding plane 1...\n');
[P_para_1, P_best, inliers_best]  = ransacP(P);
P2 = P(:, setdiff([1:1:size(P,2)], inliers_best));
fprintf('\nFinding plane 2...\n');
[P_para_2, ¬, ¬]  = ransacP(P2);
toc

% Plot novel view
smith_south_plane = P_para_1
smith_west_plane  = P_para_2
fprintf('Draw novel view.\n');
frame = drawNovelView(smith_south_plane, smith_west_plane, M2);
figure, imshow(frame)

t = pi/6;
M3 = K1*[1      0       0  1;
        0  cos(t) -sin(t) 2;
        0  sin(t)  cos(t) 1];
frame2 = drawNovelView(smith_south_plane, smith_west_plane, M3);
figure, imshow(frame2)
end

function [p, T] = normalize_point(pt)
% Normalize to zero mean and unit variance
% Ref: 1. the attached pdf file in ...
    http://www.researchgate.net/post/Calculating_the_fundamental_matrix_using_the_eight_p
%      2. ...
    http://ece631web.groups.et.byu.net/Lectures/ECEn631%2013%20-%208%20Point%20Algorithm
% Make homo
N = size(pt, 2);
if size(pt,1)≠3
    pt = [pt; ones(1,N)];
end
% Find center
center  = mean(pt(1:2,:), 2);
% Displacement to center
dist = pt(1:2,:) - repmat(center, [1, N]);
```

```
d = sqrt(dist(1,:).^2 + dist(2,:).^2);
% Scale
s = sqrt(2)/mean(d);
% Transform matrix
T = [s 0 -s*center(1); 0 s -s*center(2); 0 0 1];
p = T*pt;
end
```

```
function [P_para, P_best, inliers_best]  = ransacP(P)
% Run RANSAC to find plane.
% >> P: 3-D points, 3-by-N
% >> P_para: parameters of plane, [a b c d]'
% << P_best: selected 3 points to form the plane
% << inliers_best: best consensus set
%
% written by: Wenbo Zhao (wzhao1#andrew.cmu.edu)
% log: (v0.1)-(first draft)-(11-28-2015)
%

num_pts     = size(P,2); % # of points
threshold   = 0.05; % threshold to reject outliers
sample_pts = 3; % # of points selected each time
max_iter    = 5000; % max # of iterations
d           = 1000000000; % distance
inliers     = [];
P_best      = [];
inliers_best = [];
debug       = 0;
while max_iter
% Sample
[sample_pt, ¬] = datasample(P', sample_pts, 'Replace', false);

% Compute plane
plane = sample_pt';
if numel(plane)≠9
    max_iter = max_iter-1;
    continue
end

% Evaluate distance, add inliers to consensus set, and select best plane
[plane_new, Pn, dd, inliers_new] = selectBestP(plane, P, threshold);
d_new = abs(sum(dd));
if length(inliers_new) < sample_pts % +1)
    max_iter = max_iter-1;
    continue
else
    if(length(inliers_new) ≥ length(inliers))
        d = abs(d_new);
        inliers = inliers_new;
        P_para  = Pn;
        P_best  = plane_new;
        inliers_best = inliers;

        if debug
            fprintf('Consensus set size: %d\n', length(inliers));
            fprintf('Distance of points to plane: %f\n', d);
        end
    end
    % Update count
```

```
    max_iter = max_iter-1;
end
end

end


function [plane_new, Pn, d, inliers] = selectBestP(plane, P, threshold)
% Evaluate distance between points and a plane
% return optimal plane parameters
[Pn, d] = distPt2Plane(plane, P);
inliers= find(abs(d) < threshold);
plane_new = plane;
end

function [Pn, d] = distPt2Plane(plane, P)
% Compute point to plane distance
N = size(P,2); % # of points
m = 1;
switch m
    case 1
        plane = [plane' ones(3,1)];
        if size(plane,2)==3
            plane = [plane; zeros(1,4)];
        end
        [U D V] = svd(plane,0);
        Pn = V(:,4); % normal vector to plane, also the plane parameters
    case 2
        Pn = plane\(-1*ones(3,1));
        Pn = [Pn;1];
end
for i=1:N
    d(i) = P(1,i)*Pn(1) + P(2,i)*Pn(2) + P(3,i)*Pn(3) + Pn(4);
end
end
```