

Project #3 Fundamental Matrix

Name: Yunfeng Wei

CWID: 10394963

E-mail: ywei14@stevens.edu

Steps:

1. Load images and convert it into grayscale as Figure 1.1

```
% Load images
imgLeft = imread('uttower_left.JPG');
imgRight = imread('uttower_right.JPG');
imgLeft = rgb2gray(imgLeft);
imgRight = rgb2gray(imgRight);
```

Figure 1.1

2. Use the custom Harris Corner Detector to find the corners as Figure 2.1. Extract 5 by 5 image patches around every feature point and form a descriptor simply by vectorizing the image pixel values in the raster scan order as Figure 2.2.

```
% Find the corner coordinate
imgLeftDot = CornerDetector(imgLeft);
imgRightDot = CornerDetector(imgRight);
```

Figure 2.1

```
count = CalculateCount(imgLeftDot); % Calculate the total number of the corners
descriptorLeft = zeros(25, count, 'uint8');
positionLeft = zeros(3, count, 'uint16');
current = 1;
for i = 1:size(imgLeftDot,1)
    for j = 1:size(imgLeftDot,2)
        if imgLeftDot(i,j) == 1
            % Extract 5x5 image patches
            % Set the descriptor for the each corner point
            descriptorLeft(:,current) = SetDescriptor(imgLeft, i, j);
            % Store its coordinate
            positionLeft(1,current) = i;
            positionLeft(2,current) = j;
            positionLeft(3,current) = 1;
            current = current + 1;
        end
    end
end

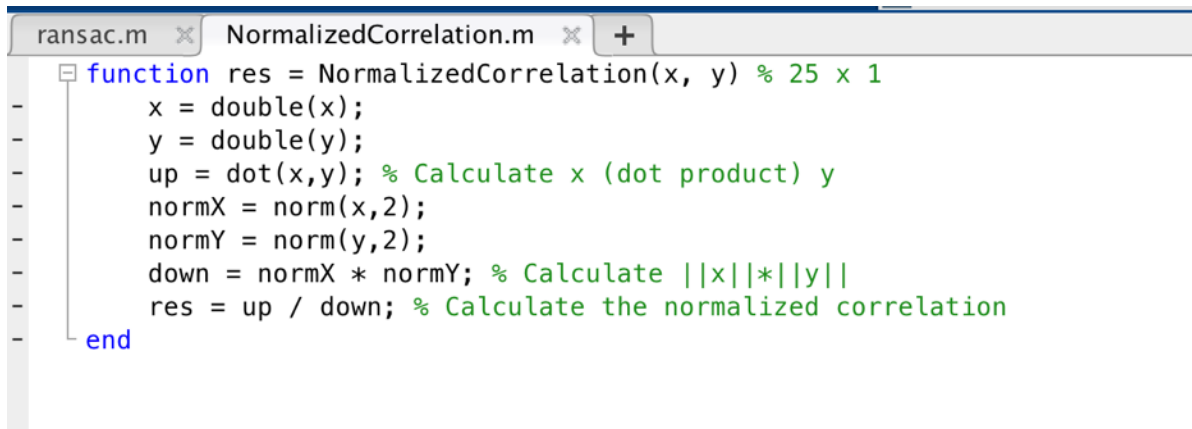
count = CalculateCount(imgRightDot);
descriptorRight = zeros(25, count, 'uint8');
positionRight = zeros(3, count, 'uint16');
current = 1;
for i = 1:size(imgRightDot,1)
    for j = 1:size(imgRightDot,2)
        if imgRightDot(i,j) == 1
            % Extract 5x5 image patches
            % Set the descriptor for the each corner point
            descriptorRight(:,current) = SetDescriptor(imgRight, i, j);
            % Store its coordinate
            positionRight(1,current) = i;
            positionRight(2,current) = j;
            positionRight(3,current) = 1;
            current = current + 1;
        end
    end
end
```

Figure 2.2

3. Compute the normalized correlation between every descriptor in one image and every descriptor in the other image as Figure 3.1. The function `NormalizedCorrelation(x, y)` is used to calculate the normalized correlation as Figure 3.2

```
normCorl = zeros(size(descriptorLeft,2), size(descriptorRight,2));
count = 0;
for i = 1:size(descriptorLeft,2)
    for j = 1:size(descriptorRight,2)
        % Calculate the normalized correlation in the range [-1, 1]
        % between one image and the other image.
        normCorl(i,j) = NormalizedCorrelation(descriptorLeft(:,i), descriptorRight(:,j));
        if normCorl(i,j) > 1
            count = count + 1;
        end
    end
end
```

Figure 3.1



```
function res = NormalizedCorrelation(x, y) % 25 x 1
-   x = double(x);
-   y = double(y);
-   up = dot(x,y); % Calculate x (dot product) y
-   normX = norm(x,2);
-   normY = norm(y,2);
-   down = normX * normY; % Calculate ||x||*||y||
-   res = up / down; % Calculate the normalized correlation
-   end
```

Figure 3.2

4. Select the top descriptor pairs with the largest normalized correlation values. Each point in one image is only paired with one point in another image as Figure 4.1.

```
% Select the top descriptor pairs with the largest normalized correlation
% values
pairs = zeros(size(normCorl,1),1);
for i = 1:size(normCorl,1)
    temp = -1;
    for j = 1:size(normCorl,2)
        if normCorl(i,j) > temp
            temp = normCorl(i,j);
            pairs(i,1) = j;
        end
    end
end
```

Figure 4.1

5. In order to run RANSAC, first initialize the number of samples, loop times, inlier ratio as Figure 5.1.

```

positionLeft = double(positionLeft);
positionRight = double(positionRight);

base = 8; % select eight points
count = 0;
F = zeros(3,3,'double'); % Initialize F
N = 1; % Initialize Number of samples
sample_count = 0; % Initialize current loop times
p = 0.9; % Initialize probability for inlier

```

Figure 5.1

6. Randomly choose 8 points in the best correlation pair, normalize the points so that the mean squared distance between the origin and the data points is 2 pixels. Use eight-point algorithm to compute F and enforce the rank-2 constraint. Finally transform F back to original units as Figure 6.1. Figure 6.2 shows the normalize function for the points, Figure 6.3 shows the eight-point algorithm.

```

while N > sample_count
    % Randomly choose 8 points
    r = randperm(size(normCorl,1)); % 1-807
    r = r(1:base);
    x = zeros(base,3); % 8 * 3
    y = zeros(base,3); % 8 * 3
    for i = 1:base
        x(i,1) = positionLeft(1,r(i));
        x(i,2) = positionLeft(2,r(i));
        x(i,3) = positionLeft(3,r(i));
        temp = pairs(r(i));
        y(i,1) = positionRight(1,temp);
        y(i,2) = positionRight(2,temp);
        y(i,3) = positionRight(3,temp);
    end
    % Normalized X so that the mean squared distance between the
    % origin and the data points is 2 pixels
    [newX, Tx] = Normalize(x);
    [newY, Ty] = Normalize(y);
    finalF = F;
    % Use eight-point algorithm to compute F
    % and enfor the rank-2 constraint
    F = EightPoint(newX,newY);
    % Transfrom F back to original units
    F = Tx' * F * Ty;

```

Figure 6.1

```

function [res, T] = Normalize(x) % Nx3
    x = x';
    % Find the indices
    ind = find(abs(x(3,:)) > eps);

    x(1, ind) = x(1, ind) ./ x(3, ind);
    x(2, ind) = x(2, ind) ./ x(3, ind);
    x(3, ind) = 1;

    % Calculate the mean value of the points pair
    c = mean(x(1:2, ind));
    % Shift the origin to centroid
    temp(1, ind) = x(1, ind) - c(1);
    temp(2, ind) = x(2, ind) - c(2);

    % Calculate the distance of each point
    dist = sqrt(temp(1,ind).^2 + temp(2,ind).^2);
    meanDist = mean(dist(:));

    % Calculate the scale
    scale = 2 / meanDist;

    T = [scale, 0, -scale*c(1); 0, scale, -scale*c(2); 0, 0, 1];
    res = T * x;
    res = res';
end

```

Figure 6.2

```

function res = EightPoint(x, y) % x: 8x3 y: 8x3
    x = double(x);
    y = double(y);

    M = zeros(8,8,'double');
    I = zeros(8,1,'double');
    for i = 1:8
        I(i,1) = 1;
    end
    % Calculate M
    for i = 1:8
        M(i,1) = x(i,1) * y(i,1);
        M(i,2) = x(i,1) * y(i,2);
        M(i,3) = x(i,1);
        M(i,4) = x(i,2) * y(i,1);
        M(i,5) = x(i,2) * y(i,2);
        M(i,6) = x(i,2);
        M(i,7) = y(i,1);
        M(i,8) = y(i,2);
    end

    F = -M \ I;
    finalF = zeros(3,3);
    finalF(1,1) = F(1,1);
    finalF(1,2) = F(2,1);
    finalF(1,3) = F(3,1);
    finalF(2,1) = F(4,1);
    finalF(2,2) = F(5,1);
    finalF(2,3) = F(6,1);
    finalF(3,1) = F(7,1);
    finalF(3,2) = F(8,1);
    finalF(3,3) = 1;

    % Take SVD of finalF and throw out the smallest singular value
    [U,D,V] = svd(finalF,0);
    res = U * diag([D(1,1) D(2,2) 0]) * V';

```

Figure 6.3

7. Calculate the error of the points as the threshold, for each data points in the corner set, calculate the error. If error is less than threshold, set the F as the best Fundamental Matrix for two image as Figure 7.1. Then recalculate the outlier ratio e and the sample number N and loop again until the loop times is greater than N , as Figure 7.2 shows.

```

% Calculate the error as threshold
error = 0;
x = double(x);
y = double(y);
for i = 1:base
    value = (x(i,:) * F * transpose(y(i,:))) ^ 2;
    error = error + value;
end

% For each point pairs, calculate if error is smaller than threshold
% If true, increment count by 1
count = 0;
value = 0;
for i = 1:size(normCorl,1)
    pointX = [positionLeft(1,i), positionLeft(2,i), 1];
    temp = pairs(i);
    pointY = [positionRight(1,temp), positionRight(2,temp), 1];
    pointX = double(pointX);
    pointY = double(pointY);
    value = pointX * F * transpose(pointY);
    value = value ^ 2;
    if (value < error)
        count = count + 1;
    end
end
% If the inliner number is the greatest, set current F
% as the finalF
if (count > finalCount)
    finalCount = count;
    finalF = F;
end
end

```

Figure 7.1

```

end
% Calculate the outlier ratio
e = 1 - count / size(normCorl,1);
% Calculate the samples number
N = log(1-p) / log(1-(1-e)^base);
% Increment current loop times by 1
sample_count = sample_count + 1;

```

Figure 7.2

8. Finally, Figure 8.1 shows the final Fundamental Matrix for the two images.



fundMatrix 				
 3x3 double				
	1	2	3	
1	-3.8302e-05	8.0240e-05	-0.0409	
2	-3.2501e-05	-1.3077e-05	0.0209	
3	0.0340	-0.0310	8.7342	
4				

Figure 8.1