

I Codes in MATLAB

1. ProjectiveReconstruction.m

```

%function ProjectiveReconstruction
%Main function for doing Projective Reconstruction
function ProjectiveReconstruction

close all;
clear all;

[pImg1, pImg2] = loadHomoImgCoor();
[F] = CalcF(pImg1, pImg2);
[P1N, P2] = CalcP(F);
[X] = LinearTriangulation(P1N, P2, pImg1, pImg2);

%Visulization
figure;
scatter3(X(1, :), X(2, :), X(3, :), 10, 'filled');
axis square;
view(32, 75);

end

```

2. EuclideanReconstruction.m

```

%function EuclideanReconstruction
%Main function for doing Euclidean Reconstruction, output both image of Projective and Euclidean Reconstruction
function EuclideanReconstruction

close all;
clear all;

%Projective Reconstruction
[pImg1, pImg2] = loadHomoImgCoor();
[F] = CalcF(pImg1, pImg2);
[P1N, P2] = CalcP(F);
[X] = LinearTriangulation(P1N, P2, pImg1, pImg2);

%3D Homo Calc
[pC1, pC2, Xce] = loadControlPoints();
[Xc] = LinearTriangulation(P1N, P2, pC1, pC2);
[H] = CalcHomo3D(Xc, Xce);
Xe = H * X;
%Normalization
for i = 1:size(Xe, 2)
    Xe(:, i) = Xe(:, i) ./ Xe(end, i);
end

%Visulization
figure;
subplot(1, 2, 2);
scatter3(Xe(1, :), Xe(2, :), Xe(3, :), 10, [1 0 0], 'filled');
axis square;
view(32, 75);
subplot(1, 2, 1);
scatter3(X(1, :), X(2, :), X(3, :), 10, [0 0 1], 'filled');
axis square;
view(32, 75);

end
end

```

3. loadHomoImgCoor.m

```

%function [pImg1, pImg2] = loadHomoImgCoor()
%Load data points in homogeous coordinates from file
function [pImg1, pImg2] = loadHomoImgCoor()

fh = fopen('bh.dat', 'r');
A = fscanf(fh, '%f%f%f%f', [4 inf]);
fclose(fh);

pImg1 = [A(1:2, :); ones(1, size(A,2))];
pImg2 = [A(3:4, :); ones(1, size(A,2))];

end

```

4. CalcF.m

```

%function [F] = CalcF(pImg1, pImg2)
%Calculate the Fundamental Matrix from two corresponding point sets
function [F] = CalcF(pImg1, pImg2)

```

```

%Conditioning
[pCImg1, TImg1] = Conditioning(pImg1);
[pCImg2, TImg2] = Conditioning(pImg2);
%DesignMatrix
[A] = DesignMatrix_F(pCImg1, pCImg2);
%Solve equations with SVD
[U D V] = svd(A);
F = (reshape(V(:,end), 3, 3))';
%Reverse conditioning, get fundamental matrix F
F = TImg2' * F * TImg1;
%Singular Constraint
[U D V] = svd(F);
D(end, end) = 0;
F = U * D * V';

end

```

5. CalcP.m

```

%function [P1N, P2] = CalcP(F)
%Calculate Two Projective Matrices from Fundamental Matrix
%Define first Projective Matrix in normal place([I 0]) and second Projective Matrix derivated from the second
%epipo.
function [P1N, P2] = CalcP(F)

[U, D, V] = svd(F);
e2 = U(:, end);

P1N = [eye(3, 3) zeros(3, 1)];
e2x = [0, -e2(3), e2(2); e2(3), 0, -e2(1); -e2(2), e2(1), 0];
P2 = [e2x*F e2];

end

```

6. LinearTriangulation.m

```

%function [X] = LinearTriangulation(P1, P2, pImg1, pImg2)
%Calculate the Object Points in projective space using linear triangulation
function [X] = LinearTriangulation(P1, P2, pImg1, pImg2)

pointCount = size(pImg1, 2);
X = [];
for i = 1:pointCount
    Ai = [pImg1(1, i).*P1(3,:) - P1(1,:);
          pImg1(2, i).*P1(3,:) - P1(2,:);
          pImg2(1, i).*P2(3,:) - P2(1,:);
          pImg2(2, i).*P2(3,:) - P2(2,:)];
    [U D V] = svd(Ai);
    Xi = V(:,end)./V(end,end);
    X = [X Xi];
end

end

```

7. loadControlPoints.m

```

%function [pImg1, pImg2, pEuclidean] = loadControlPoints()
%Load the control points from file
function [pImg1, pImg2, pEuclidean] = loadControlPoints()

fh = fopen('pp.dat', 'r');
A = fscanf(fh, '%f%f%f%f%f%f', [7 inf]);
fclose(fh);

pImg1 = [A(1:2, :); ones(1, size(A,2))];
pImg2 = [A(3:4, :); ones(1, size(A,2))];
pEuclidean = [A(5:7, :); ones(1, size(A,2))];

end

```

8. CalcHomo3D.m

```

%function [H] = CalcHomo3D(pImg1, pImg2)
%Calculate spatial homography
function [H] = CalcHomo3D(pImg1, pImg2)

%Conditioning
[pCImg1, TImg1] = Conditioning(pImg1);
[pCImg2, TImg2] = Conditioning(pImg2);
%DesignMatrix
[A] = DesignMatrix_Homo3D(pCImg1, pCImg2);
%Solve equations with SVD
[U D V] = svd(A);
R = (reshape(V(:,end), 4, 4))';
%Reverse conditioning, get H

```

```
H = inv(TImg2) * R * TImg1;
H = H ./ H(end, end);

end
```

9. Conditioning.m

```
%function [pOut, T] = Conditioning(pIn)
%Calculate conditioning of pIn
%Output condition matrix T and pOut
function [pOut, T] = Conditioning(pIn)

Tm = eye(size(pIn, 1));
Ts = eye(size(pIn, 1));

for i = 1:(size(pIn, 1) - 1)
    mvalue = -mean(pIn(i, :));
    svalue = 1 ./ mean(abs(pIn(i, :) + mvalue));
    Tm(i, end) = mvalue;
    Ts(i, i) = svalue;
end

T = Ts * Tm;
pOut = T * pIn;

end
```

10. DesignMatrixF.m

```
%function [A] = DesignMatrixF(pSet1, pSet2)
%Establish the design matrix A of F from pSet1 and pSet2
%The Normalized 8-Point Algorithm
function [A] = DesignMatrix_F(pSet1, pSet2)

A = [];
for i = 1:size(pSet1, 2)
    Ai = zeros(1, 9);
    Ai(1) = pSet1(1, i) .* pSet2(1, i);
    Ai(2) = pSet1(2, i) .* pSet2(1, i);
    Ai(3) = pSet2(1, i);
    Ai(4) = pSet1(1, i) .* pSet2(2, i);
    Ai(5) = pSet1(2, i) .* pSet2(2, i);
    Ai(6) = pSet2(2, i);
    Ai(7) = pSet1(1, i);
    Ai(8) = pSet1(2, i);
    Ai(9) = 1;
    A = [A; Ai];
end
[ax, ay] = size(A);
if (ax < ay)
    l = zeros(ay-ax, ay);
    A = [A; l];
end

end
```

11. DesignMatrix_Homo3D.m

```
%function [A] = DesignMatrix_Homo3D(pSet1, pSet2)
%Establish the design matrix A of H from pSet1 and pSet2
function [A] = DesignMatrix_Homo3D(pSet1, pSet2)

A = [];
for i = 1:size(pSet1, 2)
    A1 = [-pSet2(4, i)*pSet1(:, i)' zeros(1, 4) zeros(1, 4) pSet2(1, i)*pSet1(:, i)'];
    A2 = [zeros(1, 4) -pSet2(4, i)*pSet1(:, i)' zeros(1, 4) pSet2(2, i)*pSet1(:, i)'];
    A3 = [zeros(1, 4) zeros(1, 4) -pSet2(4, i)*pSet1(:, i)' pSet2(3, i)*pSet1(:, i)'];
    A = [A; A1; A2; A3];
end
[ax, ay] = size(A);
if (ax < ay)
    l = zeros(ay-ax, ay);
    A = [A; l];
end

end
```

II Example Output

The left image is from Projective Reconstruction.
The right image is from Euclidean Reconstruction.

