CSE 252B - Homework 2

Lingting Ge A53212800

2/7/2017

1 Programming: Estimation of the camera projection matrix

1.1 Linear estimation

Download input data from the course website. The file hw2_points3D.txt contains the coordinates of 50 scene points in 3D (each line of the file gives the $\tilde{\mathrm{X}}_i$, $\tilde{\mathrm{Y}}_i$, and $\tilde{\mathrm{Z}}_i$ inhomogeneous coordinates of a point). The file hw2_points2D.txt contains the coordinates of the 50 corresponding image points in 2D (each line of the file gives the $\tilde{\mathrm{x}}_i$ and $\tilde{\mathrm{y}}_i$ inhomogeneous coordinates of a point). The scene points have been randomly generated and projected to image points under a cam- era projection matrix (i.e., $\mathbf{x}_i = \mathbf{P}\mathbf{X}_i$), then noise has been added to the image point coordinates.

Estimate the camera projection matrix P_{DLT} using the direct linear transformation (DLT) algorithm (with data normalization). You must express $\mathbf{x}_i = \mathbf{P}\mathbf{X}_i$ as $[\mathbf{x}]_i^{\perp}\mathbf{P}\mathbf{X}_i = 0$ (not $\mathbf{x}_i \times \mathbf{P}\mathbf{X}_i = 0$), where $[\mathbf{x}]_i^{\perp}\mathbf{x}_i = 0$, when forming the solution. Include the numerical values of the resulting P_{DLT}, scaled such that $\| P_{DLT} \|_{Fro} = 1$, in your report with sufficient precision such that it can be evaluated (hint: use format shortg in MATLAB prior to displaying your results).

Solution:

My result is as follows:

$$P_{DLT} = \begin{bmatrix} -0.0060446 & 0.0048386 & -0.0088225 & -0.8405 \\ -0.0090945 & 0.0023023 & 0.0061782 & -0.54156 \\ -5.0076e-06 & -4.4768e-06 & -2.5529e-06 & -0.0012515 \end{bmatrix} \tag{1}$$

Here is the general process that I used to solve this problem:

First, read the data, do the data normalization, then we can get normalized data and matrix **T** and **U** for 2D and 3D points respectively. Then, using Householder matrix, we can compute the left null space for each corresponding 2D point. After that, we can get the matrix **A**. Then using this matrix, we can get the vector form of P_{norm}. At last, reshape it, using the formula $\mathrm{P_{DLT}} = \mathrm{inv(T)} * \mathrm{P_{norm}} * \mathrm{U}$, then scale it by the "Fro" norm, we can get the final answer.

1.2 Nonlinear estimation

Use P_{DLT} as an initial estimate to an iterative estimation method, specifically the Levenberg-Marquardt algorithm, to determine the Maximum Likelihood estimate of the camera projection matrix that minimizes the projection error. You must parameterize the camera projection matrix as a parameterization of the

Table 1: cost at each iteration

iteration	cost
0	84.082583222450751
1	82.791273148649665
2	82.790238005509423
3	82.790238005407218
4	82.790238005407673
5	82.790238005407673
6	82.790238005407417
7	82.790238005408284
8	82.790238005407360
9	82.790238005407588
10	82.790238005406891
11	82.790238005407645
12	82.790238005407204
13	82.790238005407858
14	82.790238005407147
15	82.790238005406124
16	82.790238005408071
17	82.790238005406678
18	82.790238005408142
19	82.790238005405968
20	82.790238005407588
21	82.790238005407971
22	82.790238005407161
23	82.790238005405968
24	82.790238005405968
25	82.790238005405968
26	82.790238005405968
27	82.790238005405968
28	82.790238005405968
29	82.790238005405968
30	82.790238005405968

homogeneous vector $\mathbf{p} = vec(\mathrm{P}^{\mathrm{T}})$. It is highly recommended to implement a parameterization of homogeneous vector method where the homogeneous vector is of arbitrary length, as this will be used in following assignments (see section A6.9.2 (page 624) of the textbook, and the corrections and errata).

In your report, show the initial cost (i.e., the cost at iteration 0) and the cost at the end of each successive iteration. Show the numerical values for the final estimate of the camera projection matrix PLM, scaled such that $\| P_{LM} \|_{Fro} = 1$, in your report with sufficient precision such that it can be evaluated.

Solution:

First, the initial cost is 84.082583222450751.

Then, the cost at the end of each successive iteration are show as table 1.

At last, the projection matrix is as follows:

$$P_{LM} = \begin{bmatrix} 0.0060943 & -0.0047265 & 0.0087902 & 0.84364 \\ 0.0090202 & -0.0022929 & -0.0061333 & 0.53666 \\ 4.9909e-06 & 4.4521e-06 & 2.5371e-06 & 0.0012435 \end{bmatrix} \tag{2}$$

For this problem, the process is very clear according to the lecture note. And here is the general process:

First, compute the initial measurement vector, initial cost, and parameterized vector. For initialization, set lambda $\lambda = 0.001$. Then for each iteration, calculate the Jocabian matrix and error, and using Jocabian matrix, lamba, and error to compute the delta for each iteration. Then, plus the vector $\mathbf{P}$ with delta and then calculate the new cost. After that, if the new cost is less than the old cost, then set these parameters to the new ones, and set $\lambda = 0.1\lambda$. However, if the new cost is greater than the old cost, then set $\lambda = 10\lambda$, and don't update other parameters. Keep doing the iteration until convergence.

Besides, there are several points that we should pay attention to: First, we should use normalized data, and $\mathbf{P_{norm}}$, instead of $\mathbf{P_{DLT}}$ as the input of the iterative method. Second, we should pay attention to the correctness of computing Jocabian matrix, since this is the most important part of the whole process.

Appendix: Source code for problem 1

```
% projectionMatrix.m

% Question (a) %
% Linear Estimation %

% DLT algorithm to estimate camera projection matrix

% read the data
point2Dorig = readPoint('hw2_points2D.txt', 2);
point3Dorig = readPoint('hw2_points3D.txt', 3);

% data normalization
[point2D, T] = dataNormalization(point2Dorig, 2);
[point3D, U] = dataNormalization(point3Dorig, 3);

% using DLT compute matrix A
A = DLTAlgorithm(point2D, point3D);

% using svd compute projection matrix P
P_norm = calProjMat(A);
% P_norm = - P_norm;

% scale P with ||P||Fro = 1
P = inv(T) * P_norm * U;
P = P / norm(P, 'fro');
format shortg;
disp(P);

uni = ones(50, 1);
xEst = P * [point3Dorig, uni]';
paramW = xEst(3, :);
xEst = xEst ./ paramW;
% disp(xEst);


% Problem 2%
% Iterative Method %

% use original data instead of normalized
unit = ones(50, 1);
x2D = [point2Dorig, unit]';
x3D = [point3Dorig, unit]';

% data normalization
[x2D_norm, T] = dataNormalization(point2Dorig, 2);
[x3D_norm, U] = dataNormalization(point3Dorig, 3);

x2D = x2D_norm;
x3D = x3D_norm;

% main function
format long;
lambda = 0.001;
measureVector = calMeasureVector(x2D);
% covar = calCovar(x2D, T);
covar = eye(100);
covar = covar * T(1, 1)^2;
% deparamP = P_norm;
deparamP = -[P_norm(1, :), P_norm(2, :), P_norm(3, :)]';
paramP = parameterize(deparamP);
% compute error and cost
proj2D = projection(deparamP, x3D);
error = measureVector - calMeasureVector(proj2D);
```

```
cost = error' * inv(covar) * error;
disp(cost);
% begin iteration
for i = 1 : 30
    % compute Jocabian
    J = calJocabian(x2D, x3D, deparamP, paramP);
    % compute delta
    delta = (J' * inv(covar) * J + lambda * eye(11)) \ (J' * inv(covar) * error);
    % update P
    paramPUpdate = paramP + delta;
    deparamPUpdate = deparameterize(paramPUpdate);
    % compute error and cost
    proj2DUpdate = projection(deparamPUpdate, x3D);
    errorUpdate = measureVector - calMeasureVector(proj2DUpdate);
    costUpdate = errorUpdate' * inv(covar) * errorUpdate;
    disp(costUpdate);
    % make decsion
    if (costUpdate < cost)
        paramP = paramPUpdate;
        deparamP = deparamPUpdate;
        error = errorUpdate;
        cost = costUpdate;
        lambda = 0.1 * lambda;
    else
        lambda = 10.0 * lambda;
    end
end

proMat_norm = reshape(deparamP, [4, 3])';
proMat = inv(T) * proMat_norm * U;
proMat = proMat / norm(proMat, 'fro');
format shortg;
disp(proMat);



% read the data
function point = readPoint(fileName, dim)
    file = fopen(fileName);
    if dim == 2
        point = textscan(file, '%f %f');
    else
        point = textscan(file, '%f %f %f');
    end
    fclose(file);
    point = cell2mat(point);
end

% Data Normalization
function [point, T] = dataNormalization(data, dim)
    % calculate mean and variance
    m = mean(data);
    v = var(data);
    % calculate normalized vector
    if (dim == 2)
        s = sqrt(2.0 / sum(v));
        T = zeros(3, 3);
        T(1, 1) = s;
        T(2, 2) = s;
        T(3, 3) = 1.0;
        T(1, 3) = -1.0 * m(1) * s;
        T(2, 3) = -1.0 * m(2) * s;
    else
        s = sqrt(3.0 / sum(v));
        T = zeros(4, 4);
```

```
        T(1, 1) = s;
        T(2, 2) = s;
        T(3, 3) = s;
        T(4, 4) = 1.0;
        T(1, 4) = -1.0 * m(1) * s;
        T(2, 4) = -1.0 * m(2) * s;
        T(3, 4) = -1.0 * m(3) * s;
    end
    % transfer inhomo to homo data
    unit = ones(50,1);
    point = [data, unit]';
    % normalize the data
    point = T * point;
end

% DLT algorithm
function matA = DLTAlgorithm(point2D, point3D)
    matA = [];
    % using house holder matrix
    % to calculate left null space of x
    for i = 1 : 50
        x = point2D(:, i);
        v = x + sign(x(1)) * norm(x) * [1, 0, 0]';
        Hv = eye(3) - 2.0 * (v * v') / (v' * v);
        leftNull = Hv(2:3, :);
        matA = [matA; kron(leftNull, point3D(:, i)')];
    end
end

% using svd compute projection matrix P
function P = calProjMat(A)
    [U, S, V] = svd(A);
    P = V(:, 12);
    P = reshape(P, [4, 3]);
    P = P';
end



% Question (b)%
% construct measurement vector
function measureVector = calMeasureVector(point)
    measureVector = [];
    for i = 1 : 50
        measureVector = [measureVector, point(1 : 2, i)'];
    end
    measureVector = measureVector';
end

% projection from 3D to 2D
function  res = projection(deparamP, x3D)
    P = reshape(deparamP, [4, 3])';
    res = P * x3D;
    w = res(3, :);
    res = res ./ w;
end

% construct associated covariance
function covar = calCovar(point, T)
    covar = eye(100);
%     for i = 1 : 50
%         covar(2 * i - 1: 2 * i, 2 * i - 1 : 2 * i) = cov(point(1 : 2, i)', point(1 : 2, i)');
%     end
    covar = T(1, 1) * T(1, 1) * covar;
```

```
end

% parameterize
function paramVector = parameterize(P)
    a = P(1);
    b = P(2 : length(P));
    paramVector = (2.0 / (sinc(acos(a)))) * b;
    normP = norm(paramVector);
    if (normP > pi)
        paramVector = (1.0 - 2 * pi / normP * ceil((normP - pi) / 2 * pi)) * paramVector;
        % paramVector = (1.0 - 2 * pi / normP) * paramVector;
    end
end

% deparameterize
function deparamVector = deparameterize(P)
    normP = norm(P);
    deparamVector = [cos(normP / 2.0), ((sinc(normP / 2.0)) / 2.0) * P']';
end

% sinc(x)
function res = sinc(x)
    if x == 0
        res = 1.0;
    else
        res = (sin(x)) / x;
    end
end

% compute Jocabian
% x2D and x3D are homogeneous
function jocabian = calJocabian(x2D, x3D, deparamP, paramP)
    jocabian = [];
    projX2D = projection(deparamP, x3D);
    part2 = jocab2(deparamP, paramP);
    for i = 1: 50
        part1 = jocab1(projX2D(:, i), x3D(:, i), deparamP);
        jocabian = [jocabian; part1 * part2];
    end
end

% compute partial xi partial P bar
function res = jocab1(point2D, point3D, deparamP)
    w = deparamP(9 : 12)' * point3D;
    tmp = zeros(1, 4);
    res = 1 / w * [point3D', tmp, -1.0 * point2D(1) * point3D'; ...
                   tmp, point3D', -1.0 * point2D(2) * point3D'];
end

% compute partial P bar partial P
function res = jocab2(deparamP, paramP)
    normP = norm(paramP);
    res = -0.5 * deparamP(2 : length(deparamP))';
    if (normP == 0)
        res = [res; 0.5 * eye(length(paramP))];
    else
        tmp = 0.5 * (sinc(normP / 2)) * eye(length(paramP)) + 0.25 / normP ...
              * derivSinc(normP / 2) * paramP * paramP';
        res = [res; tmp];
    end
end

% derivative of sinc(x)
function res = derivSinc(x)
    if x == 0
```

```
            res = 0.0;
      else
            res = cos(x) / x - sin(x) / (x * x);
      end
end
```