Autonomous Robotics
Autocalibration Report

Emre Ozan Alkan
{emreozanalkan@gmail.com}
MSCV-5

May 16, 2014

1 Introduction

Camera calibration with known patterns or objects with know euclidean structures are very well known and used techniques. However these methods are not available without priori assumptions to calibrate and get intrinsic parameters. Hence, we need to use autocalibration techniques that doesn't rely on euclidean structures for calibration.

2 Methods

We have been given approximate intrinsic parameter of the camera, fundemental matrixes between the images, projection matrix between the images. We employ 3 different method below to more accurately approximate intrinsic parameters.

2.1 Mendonça and Cipolla

Basic method based on the exploitation of the rigidity constraint. We design cost function, which takes intrinsic parameters and fundemental matrices as parameters, and returns positive value related to difrence between two non-zero singular value of the essential matrix.

800.018875334029	-0.00111165444322615	255.996941864705
0	800.019278207122	256.001480852401
0	0	1

Table 1: Mendonca Cipolla Intrinsic Parameter Approximation

Listing 1: Mendonca and Cipolla Cost

```
function [ C ] = mendoncaCipollaCost( X, Fs )
%MENDONCACIPOLLACOST Summary of this function goes here
%   Detailed explanation goes here

% X(1) = alfa_u
% X(2) = alfa_v
% X(3) = gamma
```

```
% X(4) = u_0
% X(5) = v_0

% Hint: For Mandonca&Cipolla , consider all the weight w_ij = 1

% SVD = and get S, first 2 sigma
% S = [sigma_1    0          0;
%      0          sigma_2    0;
%      0          0          0];
%
% Essential = A' * Fs * A;

A = [X(1)    X(3)    X(4);
     0       X(2)    X(5);
     0       0        1];

E = zeros(3, 3, 10, 10);

C = zeros(1, 10);

for ii = 1 : 10

    for jj = 1 : 10

        if ii == jj
            continue;
        end

        E(:, :, ii, jj) = A' * Fs(:, :, ii, jj) * A;

        [~, S, ~] = svd(E(:, :, ii, jj));

        C(ii) = C(ii) + ((S(1, 1) - S(2, 2)) / S(2, 2));

    end

end

C = sum(C(:));


end
```

Listing 2: Mendonca and Cipolla Test

```
load('data.mat');

X = zeros(5, 1);

X(1) = A(1, 1);
X(2) = A(2, 2);
X(3) = A(1, 2);
X(4) = A(1, 3);
X(5) = A(2, 3);

costFunc = @(X) mendoncaCipollaCost(X, Fs);

[xHat, resnorm] = lsqnonlin(costFunc, X);

% xHat(1) = alfa_u
% xHat(2) = alfa_v
% xHat(3) = gamma
% xHat(4) = u_0
% xHat(5) = v_0
A2 = [xHat(1)     xHat(3)     xHat(4);
      0           xHat(2)     xHat(5);
      0           0           1];
```

2.1.1 Kruppa Equations

Kruppa equations are obtatining intrinsic parameters of the camera using polynomial equations, with a minimum of three displacements. I had hard time to achieve good results without passing the tolerance, and some more parameter values for lsqnonlin function.

800.008699400211	-0.000841395721280208	256.000110817581
0	800.00842085238	256.000185844728
0	0	1

Table 2: Kruppa Equations Intrinsic Parameter Approximation

Listing 3: Kruppa Equations Cost

```
function [ C ] = kruppaCost( X, Fs )
%KRUPPACOST Summary of this function goes here
%   Detailed explanation goes here

% X(1) = alfa_u
% X(2) = alfa_v
% X(3) = gamma
% X(4) = u_0
% X(5) = v_0

A = [X(1) X(3) X(4);
     0    X(2) X(5);
     0    0    1];

w_inv = A * A';

E = zeros(3, 3, 10, 10);

%C = zeros(1, 10);
C = 0;

for ii = 1 : 10

    for jj = 1 : 10

        if ii == jj
            continue;
        end

        %          E(:, :, ii, jj) = A' * Fs(:, :, ii, jj) * A;
        %
        %          %[~, S, ~] = svd(E(:, :, ii, jj));
        %          C(ii) = C(ii) + norm(((Fs(:, :, ii, jj) * w_inv * Fs(:, :, ii, jj)') / norm(Fs(:, :, ii, jj) * w_inv * Fs(:, :, ii, jj)', 'fro')) - ...
        %          ((E(:, :, jj, ii) * w_inv * E(:, :, jj, ii)') / norm(E(:, :, jj, ii) * w_inv * E(:, :, jj, ii)' ,'fro') ), 'fro');
        %          %C(ii) =  C(ii) + ((S(1, 1) - S(2, 2)) / S(2, 2));



        % Abinash's way instead of longer equation
        currFS = Fs(:, :, ii, jj);
        ce = null(currFS');
        F = currFS * w_inv * currFS';
        F = F / norm(F, 'fro');
        currE = [0 -ce(3) ce(2); ce(3) 0 -ce(1); -ce(2) ce(1) 0];
        E = currE * w_inv * currE';
        E = E / norm(E, 'fro');
        tempC = F - E;
        C = C + norm(tempC,'fro');
        % Abinash's way

```

```
        end

end

end
```

Listing 4: Kruppa Equations Test

```
load('data.mat');

X = zeros(5, 1);

X(1) = A(1, 1);
X(2) = A(2, 2);
X(3) = A(1, 2);
X(4) = A(1, 3);
X(5) = A(2, 3);

costFunc = @(Y) kruppaCost(Y, Fs);

options = optimset('Algorithm','levenberg-marquardt','MaxFunEvals',10^50,'TolFun'
    ,10^-100,'TolX',10^-100);
[xHat, resnorm] = lsqnonlin(costFunc, X, [], [], options);

A3 = [xHat(1)   xHat(3)   xHat(4);
      0         xHat(2)   xHat(5);
      0         0         1];
```

2.2 Dual Absolute Quadric

We represent euclidean scene structure formulated in terms of the absolute quadric - the singular dual 3D quadric giving teh Euclidean dot-product between plane normals.

Listing 5: Dual Absolute Quadric

```
load('data.mat');

% A = [800 0 256; 0 800 256; 0 0 1]; % Correct one

w = A * A';
nx = sym('nx', 'real');
ny = sym('ny', 'real');
nz = sym('nz', 'real');
l2 = sym('l2', 'real');

n = [nx; ny; nz]; % Normal of pi inf

Q = [w, (w * n); (n' * w), (n' * w * n)]; % Dual Absolute Quadric

M2 = PPM(:, :, 2);

m2 = M2 * Q * M2';

sol = solve(m2(1, 1) == (l2 * w(1, 1)), ...
            m2(2, 2) == (l2 * w(2, 2)), ...
            m2(3, 3) == (l2 * w(3, 3)), ...
            m2(1, 3) == (l2 * w(1, 3)));

display('sol.nx:');
display(double(sol.nx));
display('sol.ny:');
display(double(sol.ny));
display('sol.nz:');
display(double(sol.nz));
```

```
display('sol.l2:');
display(double(sol.l2));
```

3 References

1. Abinash Pant
2. http://en.wikipedia.org/wiki/Camera_auto-calibration
3. http://homepages.inf.ed.ac.uk/rbf/CVonline/LOCAL_COPIES/FUSIELLO3/node4.html#SECTION00043000000
4. http://homepages.inf.ed.ac.uk/cgi/rbf/CVONLINE/entries.pl?TAG1325
5. http://hal.inria.fr/docs/00/54/83/45/PDF/Triggs-cvpr97.pdf