ECSE 6550 COMPUTER VISION
PROJECT 2

Andrew Cunningham

October 4, 2016

1 Introduction

The process of obtaining a projection matrix from 2D and 3D points is called camera calibration. Camera calibration is an important topic in computer vision because a calibrated camera is required to recover 3D quantitative measures about observed 2D images. For example, an image from a calibrated camera can measure how far an object is from the camera, the height of an object in an image etc. In this project, we explore linear and probabilistic methods for calibrating a camera from both noisy and clean data. A linear least squares approach is used to obtain a projection matrix from clean (well mapped) and noisy 2D/3D points. RANSAC is then used in addition to the linear method to improve the model generated from noisy data

2 Technical Approach

Camera calibration is the procedure by which a projection matrix P is obtained from 3D points and the camera's capture of these 3D points in 2D in the form of an image.

2.1 Projection Matrix

In this project, P is a full perspective projection matrix which is defined as:

$$P = \begin{bmatrix} s_x f & 0 & c_0 \\ 0 & s_y f & r_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} R\ T \end{bmatrix} \tag{1}$$

Where s_x, s_y, f, c_0, r_0 are intrinsic camera parameters and R and T are extrinsic camera parameters. The projection matrix P maps a homogeneous 3D point to a 2D point on an image.

$$\lambda \begin{bmatrix} c \\ r \\ 1 \end{bmatrix} = P \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \tag{2}$$

Camera calibration is applied to find P and not necessarily the specific camera parameters. Due to the fact that P is a 3x4 matrix, we can describe it as follows:

$$P = \begin{bmatrix} P_1 & P_{14} \\ P_2 & P_{24} \\ P_3 & P_{34} \end{bmatrix} \tag{3}$$

2.2 Camera Calibration using Linear Least Squares

The linear method used to obtain a projection matrix P from a set of 3D and 2D points is the 2nd method described in Dr.Ji's lecture notes on camera calibration. To derive a procedure for calibrating the camera, start with the description of the projectio matrix. If we seperate Equation 2 into its seperate equations we get:

$$\lambda c_i = P_1^T \begin{bmatrix} x_i \\ y_i \\ z_i \end{bmatrix} + P_{14} \tag{4}$$

$$\lambda r_i = P_2^T \begin{bmatrix} x_i \\ y_i \\ z_i \end{bmatrix} + P_{24} \tag{5}$$

$$\lambda = P_3^T \begin{bmatrix} x_i \\ y_i \\ z_i \end{bmatrix} + P_{34} \tag{6}$$

Substituting Equation 3 into Equations 1 and 2 to eliminate λ produces

$$\left[P_3^T \begin{bmatrix} x_i \\ y_i \\ z_i \end{bmatrix} + P_{34} \right] c_i = P_1^T \begin{bmatrix} x_i \\ y_i \\ z_i \end{bmatrix} + P_{14} \tag{7}$$

$$\left[P_3^T \begin{bmatrix} x_i \\ y_i \\ z_i \end{bmatrix} + P_{34} \right] r_i = P_2^T \begin{bmatrix} x_i \\ y_i \\ z_i \end{bmatrix} + P_{24} \tag{8}$$

With some rearranging, a system of equations can be formed with P_1,P_{14},P_2,P_{24},P_3,P_{34} as unknowns:

$$\begin{bmatrix} x_i \\ y_i \\ z_i \end{bmatrix}^T P_1 + P_{14} - c_i \begin{bmatrix} x_i \\ y_i \\ z_i \end{bmatrix}^T P_3 - c_i P_{34} = 0 \tag{9}$$

$$\begin{bmatrix} x_i \\ y_i \\ z_i \end{bmatrix}^T P_2 + P_{24} - r_i \begin{bmatrix} x_i \\ y_i \\ z_i \end{bmatrix}^T P_3 - r_i P_{34} = 0 \tag{10}$$

Let $M_i^{3x1} = (x_i, y_i, z_i)$ be the 3D points and $m_i^{2x1} = (c_i, r_i)$ be the corresponding 2D image point. When represented in matrix form with N points:

$$\begin{bmatrix} M_1^T & 1 & \vec{0} & 0 & -c_1 M_1^T & -c_1 \\ \vec{0} & 0 & M_1^T & 1 & -r_1 M_1^T & -r_1 \\ \vdots & & & & & \\ M_N^T & 1 & \vec{0} & 0 & -c_N M_N^T & -c_N \\ \vec{0} & 0 & M_N^T & 1 & -r_N M_N^T & -r_N \end{bmatrix} \begin{bmatrix} P_1 \\ P_{14} \\ P_2 \\ P_{24} \\ P_3 \\ P_{34} \end{bmatrix} = \vec{0} \tag{11}$$

Or more simply,

$$Av = 0 \tag{12}$$

A can be divided into [B,b] where b is the last column of A. With this setup, define V and Y to be:

$$V = P_{34} \begin{bmatrix} Y \\ 1 \end{bmatrix} \tag{13}$$

$$Y = (P_1^T P_{14} P_2^T P_{24} P_3^T)/P_{34} \tag{14}$$

Where $Av = p_{34}(BY + b)$. Now camera calibration can be solved by minimizing $||BY + b||^2$ the solution being:

$$Y = -(B^T B)^{-1} B^T b \tag{15}$$

The solution is found to the scale factor P_{34} which can be recovered using the fact that $|P_3| = 1$

$$P_{34} = \frac{1}{\sqrt{Y^2(9) + Y^2(10) + Y^2(11)}} \tag{16}$$

2.3 RANSAC

The Linear Least Squares method is sensitive to noise in the data used to build the model. Thus, it would be advantageous to be able to select a training set from provided points such that the noise present in this set is minimal. One probabilistic way of doing this is by using RANSAC (Random Sample and Consensus). The RANSAC algorithm follows:

Algorithm 1 RANSAC

1: Select randomly the minimum number of points required to determine the model parameters.
2: Solve for the parameters of the model.
3: Determine how many points from the set of all points fit with a predefined tolerance ϵ.
4: If the fraction of the number of inliers over the total number points in the set exceeds a predefined threshold τ, re-estimate the model parameters using all the identified inliers and terminate.
5: Otherwise, repeat steps 1 through 4 (maximum of N times).

Figure 1: Description of RANSAC taken from [1]

The parameters of RANSAC can be changed to fit the application. In the case of camera calibration, the tolerance of a point can be defined as the distance between the model's prediction of where the point should lie and where it is actually located. Further, the percentage of inliers can be selected to be higher than the predicted percentage of points that are considered noisy in a dataset.

3 Problem Statement and Approach

In this project, a total of four files of data are given. There are two sets of 3D data with corresponding sets of 2D data for two different cameras. Each set contains 72 points. The first set of 3D data cleanly corresponds with the 2D images but the second set introduces noise and mismatches between the 2D and 3D points.

The linear method described in section 2.2 is implemented in Matlab and used to generate projection matrices for both cameras using first clean data then the messy data. Lastly, RANSAC is used on the messy data to try to obtain a better projection matrix.

4 Implementation

The project was implemented in Matlab. The implementation has several parts:

4.1 Code Structure

a. main.m: This file calls the other files to perform three experiments. The first experiment is to obtain projection matrices using the clean data for both the left and right cameras. The second experiment uses the same linear method to obtain projection matrices for the left and right cameras using the messy, mismatched data. The third and final experiment uses RANSAC to select a better set of points from the messy 3D data to improve the model.

b. linearCalibrate1.m: This file contains the implementation of the linear solution for calibrating the camera described in section 2.2

c. ransacFit.m: This file contains the RANSAC implementation for experiment 3. It uses linearCalibrate1.m to create a model for each randomly selected set of data. Parameters for RANSAC were determined experimentally.

d. takeKRandomElements.m: This file will separate k random elements out of the 3D and 2D data. It is a helper function for ransacFit.

4.2 Code

Listing 1: main.m

```
%% Read good data
points3D = importdata('3Dpointnew.txt');
points2Dl = importdata('Left_2Dpoints.txt');
points2Dr = importdata('Right_2Dpoints.txt');

% get projection matrices
leftP = linearCalibrate1(points2Dl, points3D);
rightP = linearCalibrate1(points2Dr, points3D);

% get new points
new2DLPoints = []; new2DRPoints = [];
for i=1:72
point2Dl = (projectPoint(leftP, points3D(i,:)));
point2Dr = (projectPoint(rightP, points3D(i,:)));
new2DLPoints = [new2DLPoints; point2Dl];
new2DRPoints = [new2DRPoints; point2Dr];
end

errorL = new2DLPoints - points2Dl;
errorR = new2DRPoints - points2Dr;
norm(mean(errorL))
norm(mean(errorR))

%% Read bad data
points3D = importdata('bad_3dpts.txt');
points2Dl = importdata('Left_2Dpoints.txt');
points2Dr = importdata('Right_2Dpoints.txt');
```

```
% get projection matrices
leftP = linearCalibrate1(points2Dl, points3D);
rightP = linearCalibrate1(points2Dr, points3D);

% get new points
new2DLPoints = []; new2DRPoints = [];
for i=1:72
point2Dl = (projectPoint(leftP, points3D(i,:)));
point2Dr = (projectPoint(rightP, points3D(i,:)));
new2DLPoints = [new2DLPoints; point2Dl];
new2DRPoints = [new2DRPoints; point2Dr];
end

errorL = new2DLPoints - points2Dl;
errorR = new2DRPoints - points2Dr;
norm(mean(errorL))
norm(mean(errorR))

%% Use RANSAC to try to get better model for bad data
points3D = importdata('bad_3dpts.txt');
points2Dl = importdata('Left_2Dpoints.txt');
points2Dr = importdata('Right_2Dpoints.txt');

% get projection matrices
leftP = ransacFit(points2Dl, points3D, 10, 2, 20, 30);
rightP = ransacFit(points2Dr, points3D, 10, 2, 20, 30);

% get new points
new2DLPoints = []; new2DRPoints = [];
for i=1:72
point2Dl = (projectPoint(leftP, points3D(i,:)));
point2Dr = (projectPoint(rightP, points3D(i,:)));
new2DLPoints = [new2DLPoints; point2Dl];
new2DRPoints = [new2DRPoints; point2Dr];
end

errorL = new2DLPoints - points2Dl;
errorR = new2DRPoints - points2Dr;
norm(mean(errorL))
norm(mean(errorR))
```

Listing 2: linearCalibrate1.m

```
function [ P ] = linearCalibrate1( p2D, p3D )
    % Apply linear solution 2 from lecture notes to solve
    % for projection matrix P
    % form M matrix from points
    B = []; b = [];
    for i=1:size(p2D,1)
    r1 = [p3D(i,1) p3D(i,2) p3D(i,3) 1 0 0 0 0 -p3D(i,1)*p2D(i,1) ...
    -p3D(i,2)*p2D(i,1) -p3D(i,3)*p2D(i,1)];
    r2 = [0 0 0 0 p3D(i,1) p3D(i,2) p3D(i,3) 1 -p3D(i,1)*p2D(i,2) ...
    -p3D(i,2)*p2D(i,2) -p3D(i,3)*p2D(i,2)];
    B = [B;r1;r2];
    b = [b;-p2D(i,1); -p2D(i,2)];
    end

    Y = -inv(B'*B)*B'*b;
    p34 = 1/norm(Y(9)+Y(10)+Y(11));
    V = [p34*Y; p34];
    P = [V(1:4)'; V(5:8)'; V(9:12)'];
end
```

Listing 3: ransacFit.m

```
function [ bestP ] = ransacFit( p2D, p3D, k, ...
    toleranceForInlier, desiredInlierCount, maxIterations )
    bestP = [];
    bestInlierCount = 0;
    for i=1:maxIterations
    inliers = 0;

    % pick set of points to fit
    [rand3D,remaining3D,rand2D,remaining2D]=takeKRandomElements(k, ...
    p2D,p3D);

    % build model with random sets
    P = linearCalibrate1(rand2D,rand3D);

    % find the number of inliers
    for i=1:size(remaining3D,1)
        % get 2D point
        newPoint = projectPoint(P,remaining3D(i,:));
        if norm(newPoint - remaining2D(i,:)) < toleranceForInlier
            inliers = inliers + 1;
        end
    end

    if inliers > bestInlierCount
        bestP = P;
    end

    if inliers > desiredInlierCount
        return
    end
    end

end
```

5 Results and Analysis

The performance of a generated P from a set of data is measured by using the P to create new 2D points and finding the average distance between the P's projection of 3D points to where they should be located (as indicated by the data).

5.1 Experiment 1

The first experiment entailed using the clean 3D data and the two sets of 2D points for the camera to find projection matrices. The linear method performed quite well in this experiment. The average displacement for pixels was below .001 for both the left and right cameras. The projection matrices obtained were:

```
leftP =
```

```
        1.0e+05 *

        -0.0084    0.0075   -0.0006    3.7664
        -0.0002   -0.0006   -0.0108    3.9692
        -0.0000   -0.0000   -0.0000    0.0126


rightP =

        1.0e+05 *

        -0.0055    0.0032    0.0012    2.4126
        -0.0027   -0.0008   -0.0057    3.5838
        -0.0000   -0.0000    0.0000    0.0099

leftErrorAvg =
        2.4811e-04

rightErrorAvg =
        4.0811e-04
```

5.2 Experiment 2

The second experiment used the bad 3D data with the 2D points. The linear method performed very poorly in this experiment. The average error for the left camera was 77 pixels and 96 pixels for the right camera. The projection matrices found were:

```
leftP =

        1.0e+04 *

        -0.0120   -0.0106   -0.0069    3.6535
        -0.0086   -0.0086   -0.0076    3.0964
        -0.0000   -0.0000   -0.0000    0.0124


rightP =

        1.0e+04 *
```

```
        -0.0104   -0.0093   -0.0054    3.0983
        -0.0116   -0.0113   -0.0083    3.8733
        -0.0000   -0.0000   -0.0000    0.0124

leftErrorAvg =
        76.9532

rightErrorAvg =
        95.8879
```

5.3 Experiment 3

The last experiment used RANSAC to better select data from the messy set to build a model. RANSAC helped the performance a lot. The average error became less than one pixel in most runs (performance varied as it is a non-deterministic method.)

```
leftP =

        1.0e+05 *

        -0.0099    0.0100   -0.0006    4.4548
        -0.0003   -0.0004   -0.0130    4.7595
        -0.0000   -0.0000   -0.0000    0.0150


rightP =

        1.0e+05 *

        -0.0042    0.0020    0.0006    1.8273
        -0.0021   -0.0009   -0.0045    2.7087
        -0.0000   -0.0000   -0.0000    0.0075


leftErrorAvg =
        0.7342

rightErrorAvg =
        0.3984
```

6 Conclusion

This project focused on obtaining a projection matrix from a set of matched 2D and 3D points. Linear methods were implemented and found to work well under cases of low noise. RANSAC was implemented to improve the performance of the linear method under noise and mismatching.

References

[1] Konstantinos G Derpanis. Overview of the ransac algorithm. *Image Rochester NY*, 4:2–3, 2010.