16-662 Robot Autonomy Project 2A

William Ku (Andrew ID: wku)

02/18/2015

1

2 Stereo Camera Calibration

For stereo calibration, five image pairs were selected to perform calibration throughout this section: 72, 2796, 2835, 220, and 3400. We picked `rawleft0001.jpg` and `rawright0001.jpg` for the point correspondences. These correspondences were chosen using MATLAB's built-in function `cpselect`. Figure 1 shows the 15 point pairs selected from this method. Instead of eight points, 15 points were used as they gave more accurate results using epipolar line match. See Appendix A for the main code.

2.1 Procedure

To estimate the essential matrix, $\mathbf{E_{est}}$, we employed the normalized eight-point algorithm (see Appendix B) to estimate the fundamental matrix, $\mathbf{F_{est}}$, and obtained the essential matrix using the following relationship:

$$\mathbf{E_{est}} = \mathbf{K_{right}}^T \mathbf{F_{est}} \mathbf{K_{left}},$$

where $\mathbf{K_{right}}$ and $\mathbf{K_{left}}$ are the intrinsic matrices of the right and the left cameras, respectively.

To get the rotation, $\mathbf{R_{est}}$, and the translation, $\mathbf{t_{est}}$ estimates, from the left camera to the right camera, we applied the singular value decomposition (SVD) on the estimated essential

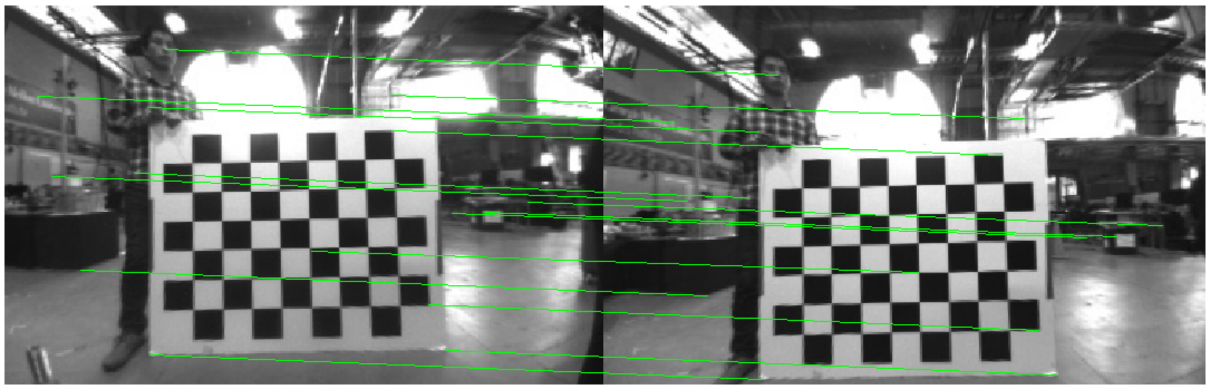


Figure 1: Point Correspondences between `rawleft0001.jpg` and `rawright0001.jpg`.

matrix, $\mathbf{E_{est}} = \mathbf{USV}^T$, and constructed $\mathbf{R_{est}}$ and $\mathbf{t_{est}}$ using the following formula:

$$W = \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix},$$
$$\mathbf{R_{est}} = \mathbf{UWV}^T \quad or \quad \mathbf{UW}^T\mathbf{V}^T,$$
$$[\mathbf{t_{est}}]_x = \mathbf{VWSV}^T \quad or \quad -\mathbf{VWSV}^T,$$

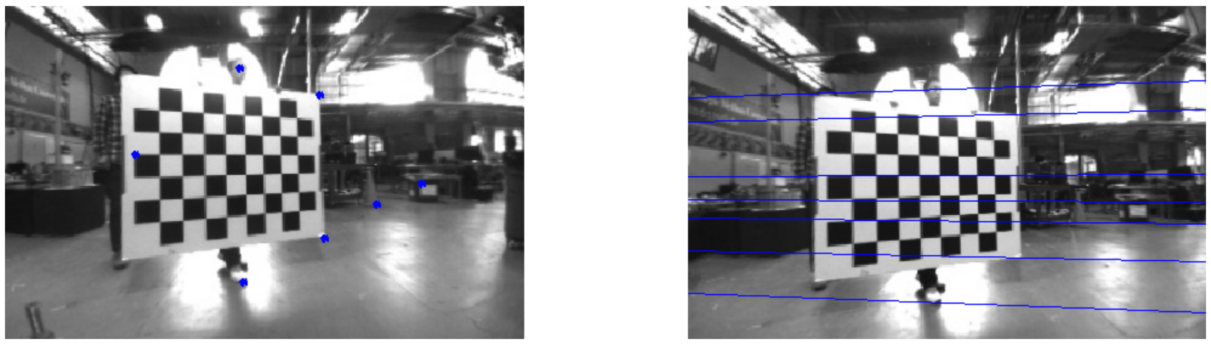
where $[\mathbf{t_{est}}]_x$ is the skew-symmetric matrix related to $\mathbf{t_{est}}$ [1]. The resulting $\mathbf{t_{set}}$ was scaled by the last element for consistency and ease of comparison. The twisted-pair ambiguity outlined in [2] results in four different situations, where only one is physically feasible. This check was done through the triangulation of a point correspondence using the camera matrices. In addition, we enforced the $SO(3)$ criterion, $det(\mathbf{R_{est}}) = 1$, by changing the sign of $\mathbf{E_{est}}$. Changing of the sign is allowed since $\mathbf{E_{est}}$ is a projective element and determined up to scale. For the complete code, refer to Appendix C.

2.2 Evaluation

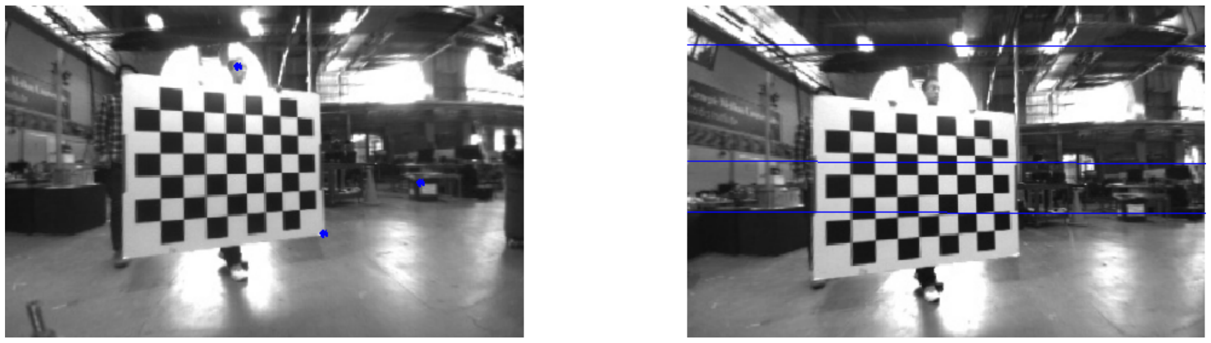
$\mathbf{F_{est}}$ and $\mathbf{F_{gt}}$ were tested against the fundamental matrix relationship, $\mathbf{x_{right}}^T F \mathbf{x_{left}} = 0$, and the mean errors for the 15 points are -0.0545 and 1.7949, respectively.
The epipoles, $\mathbf{e_{est}} = [-2202.9, 91.1, 1]^T$ and $\mathbf{e_{gt}} = [-15271, 151, 1]^T$, of $\mathbf{F_{est}}$ and $\mathbf{F_{gt}}$ were also calculated (see Appendix D).
Using MATLAB's built-in function, `estimateCameraParameters`, the ground truth parameters ($\mathbf{F_{gt}}$, $\mathbf{E_{gt}}$) were generated for stereo calibration. We compared the fundamental matrices, $\mathbf{F_{est}}$ and $\mathbf{F_{gt}}$, by plotting the epipolar lines on the right image given a click point on the left image. Figure 2a shows the clicked points and the corresponding epipolar lines using $\mathbf{F_{est}}$ on images `rawleft1796.jpg` and `rawright1796.jpg`. The epipolar lines estimates are quite robust as the they mostly pass through the interest points. Figure 2b shows the same image pair using $\mathbf{F_{gt}}$. With undetermined reasons, the ground truth epipolar lines underperformed on all of the clicks, with epipolar line estimates offsetting the clicked points towards the top. The epipolar line match was performed on randomly selected images and the results were consistent throughout.

Additionally, $\mathbf{R_{est}}$ and $\mathbf{t_{est}}$ were compared to $\mathbf{R_{gt}}$ and $\mathbf{t_{gt}}$ calculated from $\mathbf{E_{gt}}$ using the same decomposition method in Appendix C:

$$\mathbf{R_{est}} = \begin{bmatrix} 0.9999 & 0.0101 & 0.0043 \\ -0.0101 & 0.9999 & 0.0079 \\ -0.0042 & -0.0079 & 1.0000 \end{bmatrix}, \quad \mathbf{t_{est}} = \begin{bmatrix} -14.2947 \\ -0.1354 \\ 1.0000 \end{bmatrix},$$
$$\mathbf{R_{gt}} = \begin{bmatrix} 0.9999 & -0.0040 & -0.0097 \\ 0.0040 & 1.0000 & -0.0017 \\ 0.0097 & 0.0016 & 1.0000 \end{bmatrix}, \quad \mathbf{t_{gt}} = \begin{bmatrix} -92.6343 \\ 0.1190 \\ 1.0000 \end{bmatrix}.$$



(a) $\mathbf{F}_{\mathbf{est}}$



(b) $\mathbf{F}_{\mathbf{gt}}$

Figure 2: Clicked Points and Epipolar Lines using Different Fundamental Matrices.

It can be shown that both rotation matrices follow a "$\mathbf{I}$ + skew-symmetric matrix" format, expected from the skew-symmetry (except on the diagonal) of general rotation matrices.

2.3 Conclusion

Overall, $\mathbf{F}_{\mathbf{est}}$ and $\mathbf{E}_{\mathbf{est}}$ are good enough for our purpose. Without having the actual $\mathbf{R}$ and $\mathbf{t}$ from the camera set, we did our best efforts to obtain reasonable estimates for the fundamental matrix and the essential matrix. Tests, such as the fundamental matrix relationship, the epipolar line match, and the $SO(3)$ determinant constraint, helped us verify the integrity of our estimates.

References

[1] http://en.wikipedia.org/wiki/Essential_matrix

[2] Richard Hartley and Andrew Zisserman (2003). Multiple View Geometry in computer vision. Cambridge University Press.

Appendix

A Single Camera Calibration

```
clear; clc; close all;

%% Variables
% Calibration images directory
calDir = 'camera_calibration';
% Square size in millimeters
squareSize = 152.3;
% Calibration image indices (max = 3473)
calInd = [72 2796 2535 3220 3400];

%% Calibration
% Convert indices to names
imageNames = arrayfun(@(i) fullfile(calDir, sprintf('rawright%04d.jpg', i)),
imdisp([imageNames], 'Size', [1 5]);

% Estimate the camera parameters
fprintf('Estimating␣camera␣parameters...␣');
params = camcalib(squareSize, imageNames);
fprintf('done!\n');
```

B Camera Calibration Function

```
% Estimates the single/stereo camera calibration parameters.
% @params:
%     - squareSize: the size of checker board squares in millimeters (mm).
%     - imageNames: a cell array of relative-path image file names. Left
%       image files in case of stereo.
%     - (optional) varargin{1}: a cell array of relative-path right image fil
%       names in case of stereo.
%     - params: a struct that contains the camera calibration information.

function params = camcalib(squareSize, imageNames, varargin)
    % Detect images points on the checker board from top left in column major
    if nargin > 2
        [imagePoints, boardSize] = detectCheckerboardPoints(imageNames, varar
    else
        [imagePoints, boardSize] = detectCheckerboardPoints(imageNames);
    end
    % Generate world points on the checker board from top left in column majo
    worldPoints = generateCheckerboardPoints(boardSize, squareSize);

    % Estimate the camera parameters
    params = estimateCameraParameters(imagePoints, worldPoints);
end
```

C Stereo Camera Calibration

```
clear; clc; close all;

%% Variables
% Calibration images directory
imgDir = 'camera_calibration';
matDir = 'mat';
% Square size in millimeters
squareSize = 152.3;
% Calibration image indices (max = 3473)
calInd = [72 2796 2835 3220 3400];
% Correspondence image points
ptsInd = 1;

%% Calibration
% Convert indices to names
imageNamesL = arrayfun(@(i) fullfile(imgDir, sprintf('rawleft%04d.jpg', i)),
imageNamesR = arrayfun(@(i) fullfile(imgDir, sprintf('rawright%04d.jpg', i)),
%imdisp([imageNamesL, imageNamesR], 'Size', [2 5]);

% Estimate the left, right, and stereo (combined) camera parameters
fprintf('Estimating␣camera␣parameters...␣');
load(fullfile(matDir, 'params.mat'));
%paramsS = camcalib(squareSize, imageNamesL, imageNamesR);
params1 = paramsS.CameraParameters1;
params2 = paramsS.CameraParameters2;
fprintf('done!\n');

%% Undistort image
img1d = imread(fullfile(imgDir, sprintf('rawleft%04d.jpg', ptsInd)));
img2d = imread(fullfile(imgDir, sprintf('rawright%04d.jpg', ptsInd)));
img1 = undistortImage(img1d, params1);
img2 = undistortImage(img2d, params2);
```

```
%% Calculate F and E
load(fullfile(matDir, sprintf('pts%04d_15.mat', ptsInd)));
K1 = params1.IntrinsicMatrix'; % Remember to tranpose K since MATLAB returns
K2 = params2.IntrinsicMatrix';
F = eightpoint_norm(pts1, pts2, max(size(img1)));
E = K2' * F * K1;
E = E / E(3,3);

%% Calculate R and t
[R, t] = getRt(-E, K1, K2, pts1(:,1), pts2(:,1));

% Verify x'^T*F*x=0
errF = cell2mat(cellfun(@(p1, p2) [p2; 1]' * F * [p1; 1], num2cell(pts1, 1),

%% Ground truths
gtF = paramsS.FundamentalMatrix;
gtF = gtF / gtF(3,3);
gtE = paramsS.EssentialMatrix;
gtE = gtE / gtE(3,3);

load(fullfile(matDir, 'cameraParameter.mat'));
yuhanF = stereoParams.FundamentalMatrix;
yuhanF = yuhanF / yuhanF(3,3);
yuhanE = stereoParams.EssentialMatrix;
yuhanE = yuhanE / yuhanE(3,3);

[gtR, gtt] = getRt(gtE, K1, K2, pts1(:, 1), pts2(:, 2));
[yuhanR, yuhant] = getRt(yuhanE, K1, K2, pts1(:, 1), pts2(:, 2));
```

D Normalized Eight-Point Algorithm

```
% Estimates the fundamental matrix using least-square solution.
% @params:
%     - pts1: 2xN points from camera 1 image
%     - pts2: 2xN points from camera 2 image
%     - normalization_constant: the larger dimension of a camera image
%     - F: the estimated fundamental matrix

function F = eightpoint_norm(pts1, pts2, normalization_constant)
    % Size of points
    N = size(pts1, 2);
    % Least square matrix
    A = zeros(N, 9); % N-by-9
    % Normalize the input coordinates
    T = [1/normalization_constant, 0, 0; ...
         0, 1/normalization_constant, 0; ...
         0, 0, 1];
    pts1n = T(1:2, 1:2) * pts1;
    pts2n = T(1:2, 1:2) * pts2;

    % Construct the least square matrix
    for i=1:N
        x  = pts1n(1, i); y  = pts1n(2, i);
        xp = pts2n(1, i); yp = pts2n(2, i);
        A(i, :) = [xp*x, xp*y, xp, yp*x, yp*y, yp, x, y, 1];
    end

    % Linear solution
    [~, ~, V] = svd(A);
    % Get the singular vector corresponding to the smallest singular value
    f = V(:, end);

    % Construct the fundamental matrix to 3x3
    F = reshape(f, 3, 3)';

    % Enforce singularity constraint
    [U, S, V] = svd(F);
    S(3, 3) = 0;
    F = U * S * V';
    for i=1:size(pts1, 2)
      [pts2n(:,i); 1]'*F*[pts1n(:,i); 1];
    end

    % Denormalize
    F = T' * F * T;
    F = F / F(end);
end
```

E Rotation and Translation Estimation

```
function [R, t] = getRt(E, K1, K2, p1, p2)
    [U, S, V] = svd(E);
   W = [0 -1 0; 1 0 0; 0 0 1];

    % If det(R)==-1, we inverse sign of E to make det(R)==1
    if (det(U*W*V') < 0 && det(U*W'*V') < 0)
        [U, S, V] = svd(-E);
    end

%       u3 = U*[0 0 1]';
%       Rs = {U*W*V', U*W'*V'};
%       ts = {u3, -u3};
    Rs = {U*W*V', U*W'*V'};
    tx = V*W*S*V';
    t = [tx(3,2) tx(1,3) tx(2,1)]'; % Vee operation to recover t from skew-sy
    ts = {t, -t};

    % Find the correct R and t (only one pair will have Z>0)
    for iR=1:length(Rs)
        R = Rs{iR};
        for it=1:length(ts)
            t = ts{it};
            M1 = K1 * [eye(3,3) [0 0 1]'];
            M2 = K2 * [R t];
            P = triangulate(M1, p1, M2, p2);
            if (det(R) > 0 && t(1) < 0 && P(3) > 0)
                t = t / t(3);
                return;
            end
        end
    end
    fprintf('Error: no R and t results...\n');
end
```

F Epipole Calculation

```
% Calculates the epipole related to the fundamental matrix.
% @params:
%       - F: the fundmental matrix
%       - e: the calculated epipole

function e = epipole(F)
    [~, ~, V] = svd(F);
    v3 = V(:,end);
    e = v3 / v3(3);
end
```