ECSE 6650: Project #3

Due on Sunday, April 17, 2016

Prof. Qiang Ji 12:30 - 1:50 PM

Sabbir Rashid & Andrew Cunningham

April 14, 2016

Contents

Introduction

The purpose of this project is to become more familiar with rectification and 3D reconstruction by implementing a procedure using Matlab. We use two calibration images taken from two different viewpoints in order to determine the intrinsic and extrinsic parameters of the cameras. Furthermore, we use two face images taken from two different view points to perform rectification, 3D reconstruction, and correspondance mapping. Our tasks for this project are as follows:

1. Compute the intrinsic parameters W of the cameras respectively for each view using the calibration data attached and the camera calibration procedure you developed for last project. Compare the W for each view? Are they the same? If not, why?

2. Derive the relative orientation and translation R and T of the two cameras based on the absolute orientation computed for each camera from task 1. Construct the fundamental matrix F using W_l and W_r from task 1 and R and T.

3. Given the matched 2D calibration points and their coordinates, use the 8-point method in the lecture notes (more at 8 Point Wiki) to setup a system of linear equations to compute the fundamental matrix F. Try to impose the rank constraint on F either during optimization (using the constraint $F_3 = \alpha F_2 + \beta F_3$) or through post-proposing via SVD analysis. Compare the computed fundamental matrix with the one computed in task 2. If different, explain why. **(graduate students only)**

4. Given the fundamental matrix computed in tasks 2 or 3, produce the rectified left and right facial images using the rectification procedure introduced in the lecture notes. Write down the rectification process, display the rectified images, and verify the results with example epipolar lines before and after the rectification.

5. Reconstruct the 3D facial points from the matched corresponding image points using the full 3D reconstruction method in the lecture notes.

6. Use the correlation technique (SSD) to identify the corresponding points for two rectified facial images (including those facial points) and display the disparity map, i.e., the image distance between the matched image points. Perform a median filtering on the disparity map to remove holes and incorrect values. **(graduate students only)**

Theoretical Discussion

Camera Calibration using Linear Method 2

The purpose of Camera Calibration is to determine the intrinsic camera parameters, which include (c_0, r_0), focal length f, scale factors s_x, s_y, skew parameter $s = \cot\alpha$ (where α is the skew angle), and lens distortion coefficients, such as k_1 for radial distortion. In general, the instrinsic camera matrix is of the form

$$W = \begin{bmatrix} fs_x & \cot\alpha & c_0 \\ 0 & \frac{fs_y}{\sin\alpha} & r_0 \\ 0 & 0 & 1 \end{bmatrix} .$$

To simplify our problem, we are assuming negligible lens distortion and zero pixel skew, $s = 0$. Having the skew parameter equal to 0 corresponds to a skew angle, $\alpha = 90°$. In this case, the intrinsic camera matrix simplifies to

$$W = \begin{bmatrix} fs_x & 0 & c_0 \\ 0 & fs_y & r_0 \\ 0 & 0 & 1 \end{bmatrix} .$$

Furthermore, we are unable to seperate the focal length for the scaling factors. Therefore, we are concerned with the following four intrinsic parameters, fs_x, fs_y, c_0, and r_0. The methods discussed in this report are used to find the Projection Matrix, $P = WM$, where

$$M = \begin{bmatrix} R & T \end{bmatrix} .$$

Therefore, in addition to the intrinsic camera parameters, we also compute the extrinsic parameters of rotation $R = [r_1; r_2; r_3]$ and translation $T = [t_x; t_y; t_z]$. In this project, to implement camera calibration, we begin by using the Linear Solution 2 discussed in class. For this solution, we are given 2D image points,

$$m_i = \begin{bmatrix} c_i \\ r_i \end{bmatrix} ,\ i = 1 ... N ,$$

as well as corresponding 3D points

$$M_i = \begin{bmatrix} x_i \\ y_i \\ z_i \end{bmatrix} ,\ i = 1 ... N .$$

Representing the Projection Matrix as

$$P = \begin{bmatrix} p_1^t & p_{14} \\ p_2^t & p_{24} \\ p_3^t & p_{34} \end{bmatrix} ,$$

we can write two equations for each pair of 2D/3D points,

$$M_i^t p_1 + p_{14} - c_i M_i^t p_3 - c_i p_{34} = 0 \quad (1)$$
$$M_i^t p_2 + p_{24} - r_i M_i^t p_3 - r_i p_{34} = 0 . \quad (2)$$

Therefore, for N points, we have a system of linear equations of the form $AV = 0$, where

$$A = \begin{bmatrix} M_1^t & 1 & \vec{0} & 0 & -c_1 M_1^t & -c_1 \\ \vec{0} & 0 & M_1^t & 1 & -r_1 M_1^t & -r_1 \\ & & & ... & & \\ M_N^t & 1 & \vec{0} & 0 & -c_N M_N^t & -c_N \\ \vec{0} & 0 & M_N^t & 1 & -r_N M_N^t & -r_N \end{bmatrix} ,$$

is a $2N \times 12$ matrix depending on the 2D/3D points, and

$$V = \begin{bmatrix} p_1^t \\ p_{14} \\ p_2^t \\ p_{24} \\ p_3^t \\ p_{34} \end{bmatrix} ,$$

is the Null Vector of A, which correspond to the Projection Matrix components. For Linear Solution 2, we define

$$A = \begin{bmatrix} B & b \end{bmatrix} ,$$

where B consists of the first 11 columns of A and b is the last column. By setting up the equation

$$AV = p_{34}(BY + b) , \tag{3}$$

we find the solution

$$Y = -(B^t B)^{-1} B^t b , \tag{4}$$
$$V = \begin{bmatrix} p_{34} Y \\ p_{34} \end{bmatrix} . \tag{5}$$

Thus, we are able to solve for the components of the Projection Matrix. We are then able to use the following equations to solve for the intrinsic and extrinsic parameters,

$$r_3 = p_3 , \tag{6}$$
$$t_z = p_{34} , \tag{7}$$
$$c_0 = p_1^t p_3 , \tag{8}$$
$$r_0 = p_2^t p_3 , \tag{9}$$
$$fs_x = ||p_1 \times p_3|| , \tag{10}$$
$$fs_y = ||p_2 \times p_3|| , \tag{11}$$
$$t_x = (p_{14} - c_0 t_z)/(fs_x) , \tag{12}$$
$$t_y = (p_{24} - r_0 t_z)/(fs_y) , \tag{13}$$
$$r_1 = (p_1 - c_0 r_3)/(fs_x) , \tag{14}$$
$$r_2 = (p_2 - r_0 r_3)/(fs_y) . \tag{15}$$

Epipolar Geometry

The central idea of Epipolar Geometry is that all epipolar lines go through the epipole. Therefore, the coordinates of a point P in the left and right images, P_l and P_r, respectively, are related as follows,

$$P_l = RP_r + T , \tag{16}$$
$$P_r = R^t(P_l - T) . \tag{17}$$

We define a skew symmetric matrix S in terms of the translation components, $T = [\ T_x\ T_y\ T_z\ ]^t$,

$$S = \begin{bmatrix} 0 & -T_z & T_y \\ T_z & 0 & -T_x \\ -T_y & T_x & 0 \end{bmatrix} .$$

The Essential Matrix, $E = S^t R$, establishes a link between the epipolar constraint and the relative orientation of the left and right coordinate systems. The coplanar constraint on P_l, P_r, and T lead to the equation,

$$P_l^t E P_r = 0 \; . \tag{18}$$

In terms of the Essential Matrix and the Instrinsic Matrices, the Fundamental Matrix is

$$F = W_l^{-t} E W_r^{-1} \; . \tag{19}$$

The Fundamental Matrix encodes both the extrinsic and intrinsic parameters. Given a point U_l on the left image, the epipolar line on the right image is given by $F^t U_l$. Given a point U_r on the right image, the epipolar line on the left image is given by FU_r. Since U_r and U_l are typically nonzero, it follows that the left epipole e_l is given by the equation

$$U_r^t F^t e_l = F^t e_l = 0 \; , \tag{20}$$

which correspons to the null vector of F^t. Similarly, the right epipole e_r is given by the equation

$$U_l^t F e_r = F e_r = 0 \; , \tag{21}$$

which correspons to the null vector of F.

Rectification

The key idea behind rectification is that, given the extrinsic and intrinsic parameters, we can compute rotation matrixes such that all epipolar lines are parallel to the base line, and conjugate epipolar lines are collinear. The rotation of the left camera, $R_l = [rl_1; rl_2; rl_3]$ can be computed based on the relative translation of the cameras, T, as shown below.

$$rl_1 = \frac{T}{||T||} = \begin{bmatrix} t_x & t_y & t_z \end{bmatrix} \; , \tag{22}$$

$$rl_2 = \frac{1}{\sqrt{t_x^2 + t_y^2}} \begin{bmatrix} t_y & -t_x & 0 \end{bmatrix} \; , \tag{23}$$

$$rl_3 = rl_1 \times rl_2 \; . \tag{24}$$

The rotation of the right camera can be found by multiplying the left rotation with the relative rotation of the two coordinate systems,

$$R_r = R_l R \; . \tag{25}$$

For the left camera, we can determine the rectified image points, (c_l', r_l'), as follows,

$$\lambda \begin{pmatrix} c_l' \\ r_l' \\ 1 \end{pmatrix} = W R_l W_l^{-1} \begin{pmatrix} c_l \\ r_l \\ 1 \end{pmatrix} \; , \tag{26}$$

where λ is a scale factor and W represents the average of the instrinsic matrices,

$$W = \frac{1}{2}(W_l + W_r) \; . \tag{27}$$

Similarly, for the right camera,

$$\lambda \begin{pmatrix} c_r' \\ r_r' \\ 1 \end{pmatrix} = W R_r W_r^{-1} \begin{pmatrix} c_r \\ r_r \\ 1 \end{pmatrix} \; . \tag{28}$$

In order to avoid holes in the rectified image, a backward mapping approach can be used instead,

$$\lambda \begin{pmatrix} c_l \\ r_l \\ 1 \end{pmatrix} = W R_l^t W_l^{-1} \begin{pmatrix} c_l' \\ r_l' \\ 1 \end{pmatrix} \quad , \tag{29}$$

$$\lambda \begin{pmatrix} c_r \\ r_r \\ 1 \end{pmatrix} = W R_r^t W_r^{-1} \begin{pmatrix} c_r' \\ r_r' \\ 1 \end{pmatrix} \quad . \tag{30}$$

3D Reconstruction

3D reconstruction is the process of solving for the 3D coordinates of points from images. In this project, we do 3D reconstruction from two images obtained in a passive stereo system. We perform 3D reconstruction for a passive stereo system if R (rotation between the two cameras), T (translation between the two cameras), W_L (intrinsic parameters of the left camera) and W_R (intrinsic parameters of the right camera) are known. Lastly, if we assume that the object frame aligns with the right camera frame, we can set up the following equations:

$$\lambda_r \begin{pmatrix} c_r \\ r_r \\ 1 \end{pmatrix} = W_r \begin{pmatrix} x \\ y \\ z \end{pmatrix} \quad . \tag{31}$$

$$\lambda_l \begin{pmatrix} c_l \\ r_l \\ 1 \end{pmatrix} = W_l [R \; T] \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} \quad . \tag{32}$$

Using these equations, it is possible to set a least-squares linear system to solve for the 5 unknowns (x,y,z,λ_l,λ_r):

$$Ax = b \tag{33}$$

$$A = \begin{pmatrix} -W_r(1,1) & -W_r(1,2) & -W_r(1,3) & cr & 0 \\ -W_r(2,1) & -W_r(2,2) & -W_r(2,3) & rr & 0 \\ -W_r(3,1) & -W_r(3,2) & -W_r(3,3) & 1 & 0 \\ -P(1,1) & -P(1,2) & -P(1,3) & 0 & cl \\ -P(2,1) & -P(2,2) & -P(2,3) & 0 & rl \\ -P(3,1) & -P(3,2) & -P(3,3) & 0 & 1 \end{pmatrix}, \; b = \begin{pmatrix} 0 \\ 0 \\ 0 \\ -P(1,4) \\ -P(2,4) \\ -P(3,4) \end{pmatrix}, \; x = \begin{pmatrix} X \\ Y \\ Z \\ \lambda_r \\ \lambda_l \end{pmatrix} \tag{34}$$

The solution of which is simply:

$$x = (A^T A)^{-1} A^T b \tag{35}$$

It is worth highlighting the fact that this the 3D coordinates found will be represented in the right camera frame using this approach.

8 Point Algorithm

An algorithm that can be used to estimate the fundamental matrix associated with a stereo camera system is the 8 point algorithm. The fundamental equation relates two matched pixels in image space by:

$$U_l^t F U_r = 0 \tag{36}$$

Where F is the fundamental matrix amd U_l and U_r are points in the left and right images respectively. Given N matched pairs of U_l and U_r and the fundamental equation, a linear equation with 9 unknowns in F can be formed:

$$A^{Nx9} f^{9x1} = 0 \tag{37}$$

$$A = \begin{pmatrix} c_{l1}c_{r1} & c_{l1}r_{r1} & c_{l1} & r_{l1}c_{r1} & r_{l1}r_{r1} & r_{l1} & c_{r1} & r_{r1} & 1 \\ \vdots & & & & & & & & \\ c_{lN}c_{rN} & c_{lN}r_{rN} & c_{lN} & r_{lN}c_{rN} & r_{lN}r_{rN} & r_{lN} & c_{rN} & r_{rN} & 1 \end{pmatrix} \tag{38}$$

Where $f = (f_{11}f_{12}...f_{33})$. If a minimum of 8 points are given, F can be solved for up to a scale factor. If exactly 8 points are given and the $rank(A) = 8$ then f is simply the null vector of A.

Sum of Square Differences

The sum of square differences is a distance metric that can be applied to aid in matching pixels in a stereoscopic camera system. In the context of pixle matching, the sum of squared differences is applied to a two dimensional grid of values:

$$\sum_{(i,j)\in W} (I_1(i,j) - I_2(x+i, y+j))^2 \tag{39}$$

Where W is the grid, I_1 is the image segment to find, I_2 is the image candidate and x and y is the pixel around which the grid in I_2 is centered. The sum of square differences can be applied to match pixels in two images by finding an x and y in the second image that minimizes the sum of squared distances

Results

Intrinsic Parameters

We begin by clearing the workspace and reading in the Camera Calibration Data points. The coordinates and the number of points are stored into matrix variables, as shown in the listing below.

Listing 1: readData.m (Part 2)

```
clc;
clear;
close all;
%% Read Camera Calibration Data Points
[left2Dpoints,~]=importdata('project3_data/calibration/pts_2D_left.txt');
[right2Dpoints,~]=importdata('project3_data/calibration/pts_2D_right.txt');
[points3D,~]=importdata('project3_data/calibration/pts_3D.txt');
%Remove first columns (which correspond to indexes)
left2Dpoints = left2Dpoints(:,2:3);
right2Dpoints = right2Dpoints(:,2:3);
points3D = points3D(:,2:4);
% Number of points
N = length(points3D); % 56 data points
```

We then compute the intrinsic parameters using Linear Method 2 for Camera Calibration, as shown in the listing below.

Listing 2: computeIntrinsicParams.m

```
%% Using Linear Method 2 to solve for the Camera Parameters for each pair of 2D and 3D points
readData;
% Left 2D
m_i = left2Dpoints';
M_i = points3D';
[ fs_x_l , fs_y_l , c_0_l , r_0_l , R_l, T_l] = LinMethod2(m_i, M_i, N);
disp('LinMethod2: Left Camera Parameters');
disp('fs_x = '); disp( fs_x_l );
disp('fs_y = '); disp( fs_y_l );
disp('c_0 = '); disp( c_0_l );
disp('r_0 = '); disp( r_0_l );
disp('R = '); disp(R_l);
disp('T = '); disp(T_l);

% right 2D,
m_i = right2Dpoints';
M_i = points3D';
[ fs_x_r , fs_y_r , c_0_r , r_0_r , R_r, T_r] = LinMethod2(m_i, M_i, N);
disp('LinMethod2: Right Camera Parameters');
disp('fs_x = '); disp( fs_x_r );
disp('fs_y = '); disp( fs_y_r );
disp('c_0 = '); disp( c_0_r );
disp('r_0 = '); disp( r_0_r );
disp('R = '); disp(R_r);
disp('T = '); disp(T_r);

%% Calculate Instrinsic Matrices
% Assuming negligible skew and lens distortion
% Left Instrinsic Matrix
W_l = [fs_x_l 0 c_0_l ;0 fs_y_l r_0_l ;0 0 1];
% Right Intrinsic Matrix
W_r = [fs_x_r 0 c_0_r ;0 fs_y_r r_0_r ;0 0 1];
disp('The Instrinsic Matrices');
disp('W_l = '); disp(W_l); % display left Intrinsic Matrix
disp('W_r = '); disp(W_r); % display right Intrinsic Matrix
```

The results of the above script are shown below.

```
LinMethod2: Left Camera Parameters
fs_x =
   1.6344e+03

fs_y =
   1.5999e+03

c_0 =
   1.0558e+03

r_0 =
  753.2809

R =
   -0.5133    0.8581    0.0132
    0.1364    0.0619    0.9887
    0.8476    0.5093   -0.1489

T =
    9.8720
  -29.0183
  122.5762

LinMethod2: Right Camera Parameters
fs_x =
   1.6559e+03

fs_y =
   1.6370e+03

c_0 =
   1.0076e+03

r_0 =
  719.1598

R =
   -0.8565    0.5160    0.0122
    0.1288    0.1884    0.9736
    0.5001    0.8355   -0.2278

T =
   20.8890
  -16.5085
  122.3363

The Instrinsic Matrices
W_l =
   1.0e+03 *

    1.6344         0    1.0558
         0    1.5999    0.7533
         0         0    0.0010

W_r =
   1.0e+03 *

    1.6559         0    1.0076
         0    1.6370    0.7192
         0         0    0.0010
```

We find that the left and right intrinsic matrices are very similar, allowing the conjecture that the pictures were taken using the same camera. However, the matrices are not exactly the same. This is likely due to the linear approximation used in the calibration method.

Fundamental Matrix

Next we compute the relative rotation and translation, which are used with the intrinsic matrices to compute the Fundamental Matrix, as shown in the listing below.

Listing 3: computeF.m

```
computeIntrinsicParams;
%% Compute Fundamental Matrix from Given Parameters
% Rotation
R = R_l*transpose(R_r); %% works
% Translation
T = -(T_l - R*T_r); %% works
disp('The Relative Rotation and Translation');
disp('R = '); disp(R); % display  left  Intrinsic  Matrix
disp('T = '); disp(T); % display right  Intrinsic  Matrix
%% Calculate Fundamental Matrix
% Translational Skew Matrix
S = [0 -T(3) T(2); T(3) 0 -T(1); -T(2) T(1) 0];
% Essential Matrix
E = transpose(S)*R;
% Fundamental Matrix
F = transpose(inv(W_l))*E*inv(W_r);
disp('The Fundamental Matrix');
disp('F = '); disp(F); % display Fundamental Matrix
```

The results of the above script are shown below.

```
The Relative Rotation and Translation
R =
    0.8826    0.1084    0.4573
   -0.0728    0.9919   -0.1053
   -0.4650    0.0602    0.8833


T =
   62.7152
   -1.7566
  -25.2268


The Fundamental Matrix
F =
    0.0000   -0.0000    0.0089
   -0.0000    0.0000    0.0427
    0.0034   -0.0302  -10.1567
```

8 Point Algorithm

The 8 point algorithm is used to estimate the fundamental matrix. Our implementation uses 8 points

Listing 4: eightPointAlgorithm.m

```
function [ F ] = eightPointAlgorithm( leftPoints, rightPoints )
    % Calculate E using the eight point algorithm on the input points
    % ASSUME at least 8 points are included as arguments

    % form A
    A = [];
    for i=1:5:40
        A = [A formYbar(leftPoints(i,:), rightPoints(i,:))];
    end
    A = A';

    U = null(A);
    F = [U(1,end) U(2,end) U(3,end);
        U(4,end) U(5,end) U(6,end);
        U(7,end) U(8,end) U(9,end)];

    [U,S,V] = svd(F);
    % Enforce essential matrix contstraints
    S(:,3) = [0;0;0];
    F = U*S*V;

end

function [ ybar ] = formYbar( yL, yR )
%Helper function to eight point algorithm
ybar = [yL(1) * yR(1);
    yL(1) * yR(2);
    yL(1);
    yL(2) * yR(1);
    yL(2) * yR(2);
    yL(2);
    yR(1);
    yR(2);
    1];
end
```

The 8 algorithm returned a matrix that represents the fundamental matrix to a scale factor. The fundamental matrix can be scaled to compare it to the fundamental matrix calculated in the previous parts. The scaled fundamental matrix we got from the 8 point algorithm was:

```
F_8Point =


    0.0000    0.0008    0.0084
    0.0000    0.0060    0.0612
   -0.0055   -0.9410  -10.1567


The Fundamental Matrix
F =
    0.0000   -0.0000    0.0089
```

```
  -0.0000    0.0000    0.0427
   0.0034   -0.0302  -10.1567
```

The fundamental matrix found using the 8 point algorithm is not the same as the one calculated using the linear calibration methods on both cameras because the linear solution in the 8 point algorithm is not accurate/robust when noise is introduced. It is possible that there is noise in the data set used to generate the fundamental matrix and that the 8 points selected were outliers or were too close together. The RANSAC algorithm could be implemented on top of the 8 point algorithm to find a set of 8 points that produces a better (and likely closer matching) fundamental matrix.

Rectification

A backward mapping approach is used to perform rectification, as shown in the listing below.

Listing 5: rectifyImages.m (Part 1)

```
%% Rectify Images
computeF;
%% Compute Rectification Rotation of Left Camera row by row
Trect = T/norm(T);
% 1st row
rl1 = Trect';
% 2nd row
rl2 = [Trect(2) -Trect(1) 0]/sqrt(Trect(1)^2+Trect(2)^2);
%rl2 = [-Trect(2) Trect(1) 0]/sqrt(Trect(1)^2+Trect(2)^2);
% 3rd row
rl3 = cross(rl1,rl2);
% Combining rows to create Rect Rot Matrix
R_rect_l = [rl1; rl2; rl3];
% Garbage Collection
clear rl1 rl2 rl3;
R_rect_r = R_rect_l*R;
W = (1/2)*(W_l+W_r);
Xl = W*R_rect_l*inv(W_l);
Xr = W*R_rect_r*inv(W_r);
Xl_inv = inv(Xl);
Xr_inv = inv(Xr);
%% Images
% left image
%left_pattern = imread('project3_data/calibration/left_pattern.jpg');
left_pattern = imread('project3_data/faceimage/left_face.jpg');
left_pattern = rgb2gray(left_pattern);
r_max_l = length(left_pattern(:,1)); % number of rows in the original image
c_max_l = length(left_pattern(1,:)); % number of columns in the original image
% right image
%right_pattern = imread('project3_data/calibration/right_pattern.jpg');
right_pattern = imread('project3_data/faceimage/right_face.jpg');
right_pattern = rgb2gray(right_pattern);
r_max_r = length(right_pattern(:,1)); % number of rows in the original image
c_max_r = length(right_pattern(1,:)); % number of columns in the original image
```

Listing 6: rectifyImages.m (Part 2)

```
%% Rectification Left
newL = zeros(r_max_l,c_max_l);
for i=1:r_max_l
    for j = 1:c_max_l
        cr1temp = Xl_inv*[j i 1]';
        cr1temp = round(cr1temp/cr1temp(3));
        if ((cr1temp(2)>0) && (cr1temp(1)>0) && (cr1temp(2)<r_max_l) && (cr1temp(1)<c_max_l))
            newL(i,j) = left_pattern(cr1temp(2),cr1temp(1));
        end
    end
end
newL=uint8(newL);
imshow(newL);
%% Rectification Right
newR = zeros(r_max_r,c_max_r);
for i=1:r_max_r
    for j = 1:c_max_r
        cr1temp = Xr_inv*[j i 1]';
        cr1temp = round(cr1temp/cr1temp(3));
        if ((cr1temp(2)>0) && (cr1temp(1)>0) && (cr1temp(2)<r_max_r) && (cr1temp(1)<c_max_r))
            newR(i,j) = right_pattern(cr1temp(2),cr1temp(1));
        end
    end
end
figure;
newR=uint8(newR);
imshow(newR);
```

Epipolar Lines were then drawn on the original and rectified images using the following script.

Listing 7: plotEpipolarLines.m (Part 1)

```
rectifyImages;
% Epipolar Lines
% Rectified Fundamental Matrix
F_rect = inv(W)'*inv(R_rect_l)'*E*inv(R_rect_r)*inv(W);

% Original Image
% Original Left Image Points
U_l1 = [1193;386;1]; % Box Corner
U_l2 = [972;1231;1]; % Shirt Triangle
U_l3 = [711;789;1]; % Mouth Edge

% Original Right Image Points
U_r1 = [1282;503;1]; % Box Corner
U_r2 = [1769;1444;1]; % Shirt Triangle
U_r3 = [1447;937;1]; % Mouth Edge

% Corresponding Epipolar Lines in Left Image
line1 = F*U_r1;
line2 = F*U_r2;
line3 = F*U_r3;

x = 1:0.1:c_max_r;
y1 = (-line1(3) - line1(1)*x)/line1(2);
y2 = (-line2(3) - line2(1)*x)/line2(2);
y3 = (-line3(3) - line3(1)*x)/line3(2);

% Plot lines on Left Image
figure;
subplot(2,2,1);
imshow(left_pattern);
hold on;
plot(x,y1,'r');
plot(x,y2,'b');
plot(x,y3,'g');
title('Original Left Image');
%% Corresponding Epipolar Lines in Right Image
line1 = transpose(F)*U_l1;
line2 = transpose(F)*U_l2;
line3 = transpose(F)*U_l3;

x = 1:0.1:c_max_r;
y1 = (-line1(3) - line1(1)*x)/line1(2);
y2 = (-line2(3) - line2(1)*x)/line2(2);
y3 = (-line3(3) - line3(1)*x)/line3(2);
```

Listing 8: plotEpipolarLines.m (Part 2)

```
% Plot lines on Right Image
subplot(2,2,2)
imshow(right_pattern);
hold on;
plot(x,y1,'r');
plot(x,y2,'b');
plot(x,y3,'g');
title('Original Right Image');
%% Rectified images
% Rectified Left Image Points
U_l1 = [1535;354;1]; % Box Corner
U_l2 = [1817;1271;1]; % Shirt Triangle
U_l3 = [1986;781;1]; % Mouth Edge

% Rectified Right Image Points
U_r1 = [606;354;1]; % Box Corner
U_r2 = [67;1271;1]; % Shirt Triangle
U_r3 = [268;781;1]; % Mouth Edge

% Corresponding Epipolar Lines in Left Image
line1 = F_rect*U_r1;
line2 = F_rect*U_r2;
line3 = F_rect*U_r3;

x = 1:0.1:c_max_r;
y1 = (-line1(3) - line1(1)*x)/line1(2);
y2 = (-line2(3) - line2(1)*x)/line2(2);
y3 = (-line3(3) - line3(1)*x)/line3(2);

% Plot lines on Left Image
subplot(2,2,3);
imshow(newL);
hold on;
plot(x,y1,'r');
plot(x,y2,'b');
plot(x,y3,'g');
title('Rectified Left Image');
```

Listing 9: plotEpipolarLines.m (Part 3)

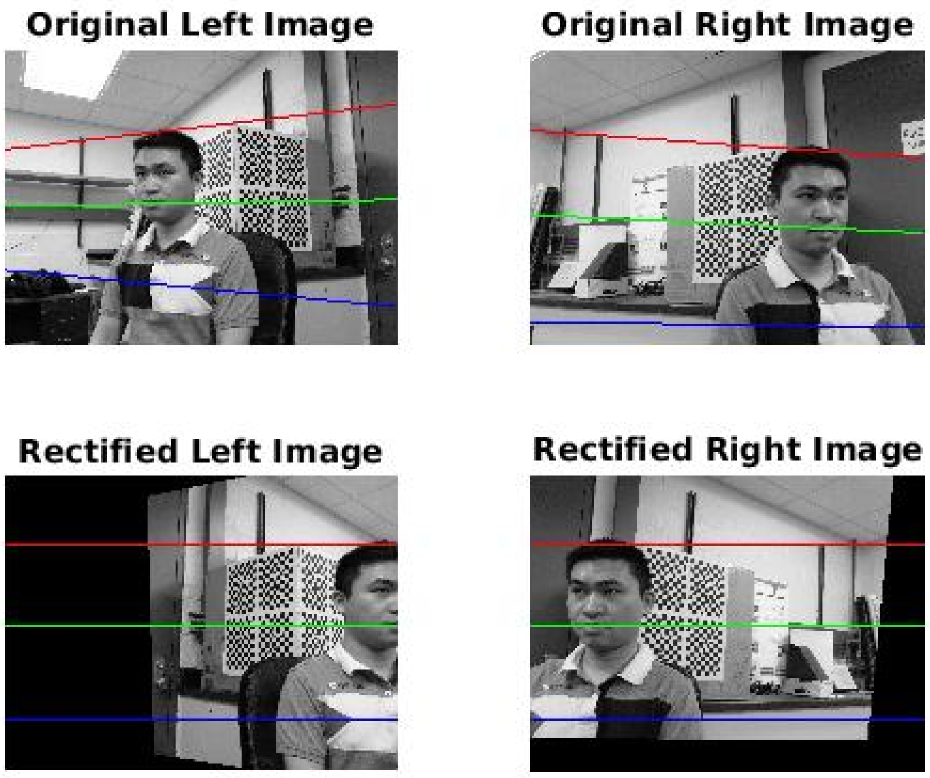
```
%% Corresponding Epipolar Lines in Right Image
line1 = transpose(F_rect)*U_l1;
line2 = transpose(F_rect)*U_l2;
line3 = transpose(F_rect)*U_l3;

x = 1:0.1:c_max_r;
y1 = (-line1(3) - line1(1)*x)/line1(2);
y2 = (-line2(3) - line2(1)*x)/line2(2);
y3 = (-line3(3) - line3(1)*x)/line3(2);

% Plot lines on Right Image
subplot(2,2,4);
imshow(newR);
hold on;
plot(x,y1,'r');
plot(x,y2,'b');
plot(x,y3,'g');
title('Rectified Right Image');
```

The resulting images are shown in the figure below.

Figure 1: Epipolar Lines



3D Facial Reconstruction

Given a set of matched 2D points in the left and right camera frames and the parameters of the two cameras, it is possible to perform 3D reconstruction to find the 3D coordinates of the points that generated the 2D points. Applying the approach developed in the theory section:

Listing 10: reconstruct3D.m

```
function [ X,Y,Z ] = reconstruction3D( Wl,Wr,R,T,cl,rl,cr,rr )
        % P is perspective projection for left camera
    P = Wl*[R T];
    A = [-Wr(1,:) cr 0;
        -Wr(2,:) rr 0;
        -Wr(3,:) 1 0;
        -P(1,1:3) 0 cl;
        -P(2,1:3) 0 rl;
        -P(3,1:3) 0 1];
    B = [0;0;0;P(:,4)];

    result=pinv(A)*B;
    X = result(1);
    Y = result(2);
    Z = result(3);
end
```

We call reconstruct3D on the set of points specified in 'faceimage/pts_left.txt' and 'faceimage/pts_right.txt' to perform 3D reconstruction of the markers:

Listing 11: Applying 3D reconstruction

```
%% PART 5
% read pixels from text file
p2DrMarker = importdata('faceimage/pts_left.txt'); p2DrMarker=p2DrMarker(:,2:3);
p2DlMarker = importdata('faceimage/pts_right.txt'); p2DlMarker=p2DlMarker(:,2:3);

% reconstruct pixels
reconstructed3D = [];
for i=1:size(p2DrMarker,1)
    [x,y,z]=reconstruction3D(Wl,Wr,R,T,p2DrMarker(i,1),p2DrMarker(i,2),p2DlMarker(i,1),p2DlMarker(i
        ,2));
    temp = [x;y;z]; newPoint = Rr'*(-temp-Tr);
    reconstructed3D = [reconstructed3D; newPoint'];
end

% plot results
scatter3(reconstructed3D(:,1),reconstructed3D(:,2),reconstructed3D(:,3))
```

The result of the 3D reconstruction is output in the form of a scatter plot.

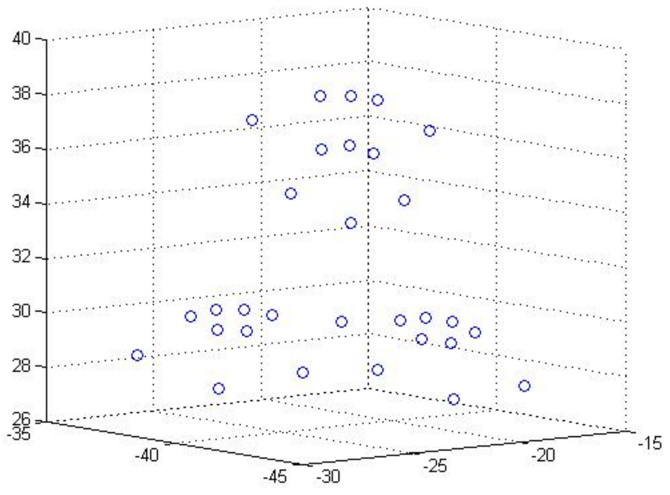


Figure 2: Reconstructed 3D points

Using these found 3D points, we can calculate some features related to the structure of the face. We took the eye points and the mouth points to calculate eye and mouth width. We found that the mouth width to be 5.44 inches and the right eye's width to be 2.4828 inches and the left eye's width to be 2.3809 inches.

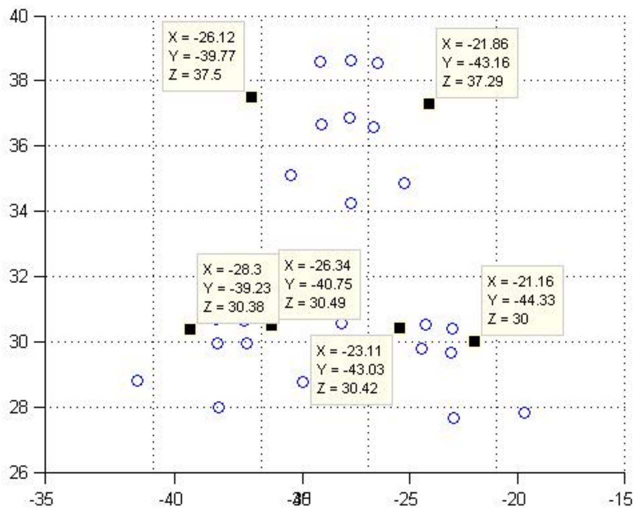


Figure 3: Reconstructed 3D points

SSD Correlation

The last part of this project required us to use SSD to establish correspondence between the left and right images for 3D reconstruction. The SSD we implemented follows the theoretical description:

Listing 12: ssdCorrelate.m

```
function [ rightOriginalPixels ] = ssdCorrelate( rectImageL, rectImageR, leftPixelsIn, Xl, Xr )
%ssdCorrelate use ssd to find specified pixels in one rectified image in
%the other image
% boxs to search will be 3x3
%  return original pixels in both camera frames

% go through each leftPixel
 rightOriginalPixels =[];
bestBox = [];
 offset =1;
for k=1:size( leftPixelsIn )
    c0 = round(leftPixelsIn(k,1));
    r0 = round(leftPixelsIn(k,2));
    boxToSearchFor = rectImageL(r0-offset:r0+offset,c0-offset:c0+offset);

    % scan along the 5 rows surrounding the row of the leftPixel
    minSSD = 1000000;
    bestCandidate = [-1; -1];
    for i=-2:1:2 % search along 5 different rows
        rowToSearch = r0 + i;
        for j=100:600% scan along columns containing the face for row
            currentBox = rectImageR(rowToSearch-offset:rowToSearch+offset,j-offset:j+offset);
            ssd = sumsqrd(boxToSearchFor,currentBox);

            if ssd < minSSD
                minSSD = ssd;
                bestCandidate = [j, rowToSearch];
            end
        end
    end
    minSSD;
    bestCandidate;
    rectRightPixel = (inv(Xr)*[bestCandidate 1]')';
     rightOriginalPixels = [rightOriginalPixels; rectRightPixel(1:2)/rectRightPixel(3)];

end
end
```

ssdCorrelate was called in main.m to convert the points listed in 'faceimage/pts_left.txt' into the rectified image to create a list of pixels to find in the right rectified image. Furthermore, the original rectified image was shifted by 300 pixels to ensure that the entire face was visible. Once a list of points in the rectified left image was obtained, the ssdCorrelate function was called to try to match those points with points in the right image. Once matching points had been found in the right rectified image, the points were converted back into the unrectified image. The code was running pretty slowly for this set of 30 points, so 3D reconstruction was performed on these points to produce a depth cloud of points. The performance of the SSD was evaluated by finding the distance between the found unrectified facial marker points in the right image and the facial marker point positions provided in 'faceimage/pts_right.txt'. The algorithm performed decently with an average error of 81 pixels. This error is high for the purposes of reconstruction but the views of the camera captured a few points outside of the face which resulted in mismatching. The code used to call ssdCorrelate is below:

Listing 13: ssdCorrelate called in main

```
%% PART 6
p2DrMarker = importdata('faceimage/pts_right.txt');
p2DrMarker=p2DrMarker(:,2:3);
p2DlMarker = importdata('faceimage/pts_left.txt');
p2DlMarker=p2DlMarker(:,2:3);

p2DlMarkerRect = [];
% Get rectified coordinates of feature points
for i=1:size(p2DlMarker,1)
    unScaledRectPoint = (Xl*[p2DlMarker(i,:) 1]')';
    scaledPoint = unScaledRectPoint(1:2)/unScaledRectPoint(3);
    shiftedScaledPoint = [scaledPoint(1)+(300/unScaledRectPoint(3)) ...
        scaledPoint(2)];
    p2DlMarkerRect=[p2DlMarkerRect; shiftedScaledPoint];
end
ssdGuess=ssdCorrelate(newL, newR, p2DlMarkerRect, Xl, Xr);
norm(mean(ssdGuess - p2DrMarker))

reconstructed3D = [];
for i=1:size(ssdGuess,1)
    [x,y,z]=reconstruction3D(Wl,Wr,R,T,p2DrMarker(i,1),p2DrMarker(i,2), ...
        ssdGuess(i,1),ssdGuess(i,2));
    temp = [x;y;z]; newPoint = Rr'*(-temp-Tr);
    reconstructed3D = [reconstructed3D; newPoint'];
end
figure
scatter3(reconstructed3D(:,1),reconstructed3D(:,2),reconstructed3D(:,3))
```

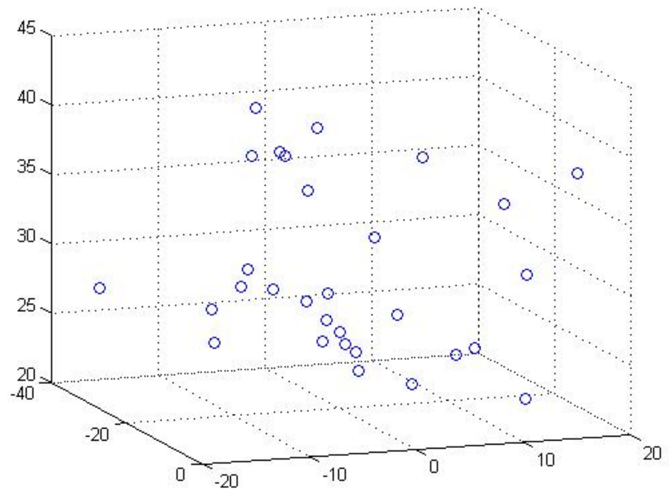


Figure 4: Reconstructed 3D points using SSD

Conclusion

In conclusion, this project helped us become more familiar with rectification and 3D reconstruction. We used a linear calibration method to compute the instrinsic camera parameters, and derived the relative translation and orientation between the two cameras. We then used the intrinsic and extrinsic parameters to calculate the Fundamental matrix. Using a null vector approach, we also calculated the Fundamental Matrix using the 8 point algorithm. We then applied rectification the two images, where a backward mapping approach was used. Using the Fundamental Matrix, epipolar lines were created on both the original and the rectified images. Finally 3D Facial Reconstruction and SSD Correlation was implemented.