NANYANG TECHNOLOGICAL UNIVERSITY

3D Digital Image Correlation Implementation with OpenCV

Presented by:

Qi Yiru U1220682A

3D Digital Image Correlation Implementation with OpenCV

Submitted in Partial Fulfilment of the Requirements

for the Degree of Bachelor of Engineering (Computer Science)

of the Nanyang Technological University

by Qi Yiru

School of Computer Science and Engineering

2015 – 2016

Abstract

The Digital Image Correlation (DIC) technique has been increasingly popular through the last decade since its first introduction in 1970s. It has seen high level of accuracy and feasibility of DIC techniques in measuring material deformation, displacement and optical motion, so it is especially widely applied in the field of experimental mechanics of materials. The techniques of 2D DIC measure the material deformation by using correlation functions to match the reference and target sub-image, and then applying shape functions to determine transformation parameters. By selecting suitable correlation and shape functions to fit in specific requirements in different experiment situations, the empirical results can be impressing. However, the 2D DIC techniques are only capable of measuring displacement within the plane perpendicular to the camera's optical axis, and the results are subject to noises generated by the object surface. In addition, if the 3D movement of the object is to be detected, the 2D DIC techniques cannot be applied in this case. Instead, the 3D DIC techniques which take in the application of stereo vision concept should be exploited.

In this report, the implementation of 3D DIC techniques using OpenCV library is introduced in detail according to the three stages of the process: camera calibration, stereo matching, and 3D reconstruction. The implementation process is explained by OpenCV function usage, and the output of the program is displayed, followed by the possible limitations and future work of the project. It is revealed in the report that 3D DIC implementation is feasible by using C++ with OpenCV support, and the results are satisfying visually to current implementation stage, but these measurements are only applicable for object surface. Further implementation would need hardware which is capable of detecting internal structure of the materials in order to measure the deformation in terms of voxels.

Acknowledgements

I hereby express my sincere gratitude to everyone who always supports me during the process of Final Year Project.

A special thanks to my FYP supervisor, Ass. Prof. Qian Kemao, who kept providing me guidance and encouragement through weekly meetings. His erudite suggestions and readiness to help have motivated me along the way.

Secondly, I am grateful to my schoolmate Wang Siqi, who helped me verify the program results and discussed with me on theory concepts. I learned a lot in this process.

I would also like to appreciate PhD senior Agarwal Nimisha for her help on arranging labs and cameras to assist my work, and also for her concern on my project.

Finally I would like to thank my friends and families, who renew my vigour when I met challenges.

List of Figures

Figure 1: 2D DIC Diagram XII
Figure 2: Set-up for 2D DIC XIV
Figure 3: Stereo Geometry [10] XV
Figure 4: Epipolar Geometry [11] XVI
Figure 5: Physical Pinhole Camera Model [12] XVI
Figure 6: Geometry of Pinhole Camera Model XVII
Figure 7: Bouguet’s Method for Rectification XXIV
Figure 8: Find and Draw Corners XXV
Figure 9: Running Calibration XXVI
Figure 10: Returning Re-projection Error XXVI
Figure 11: Running Rectification XXVII
Figure 12: Returning Disparity Map for Each Image Pair XXVII
Figure 13: Left Camera Matrix XXVIII
Figure 14: Distortion Coefficients of Left Camera XXVIII
Figure 15: Right Camera Matrix XXVIII
Figure 16: Distortion Coefficients of Right Camera XXVIII
Figure 17: Left Rotation Matrix XXIX
Figure 18: Left Projection Matrix XXIX
Figure 19: Right Rotation Matrix XXIX
Figure 20: Right Projection Matrix XXX
Figure 21: Re-projection Matrix XXX
Figure 22: Computing and Displaying Disparity Map XXXII
Figure 23: Point Cloud View in Graphing Calculator 3D XXXIII
Figure 24: Reconstructed 3D Model View XXXIV
Figure 25: Reconstructed 3D Model Side View XXXIV
Figure 26: Reconstructed 3D Model Back View XXXV

List of Equations

Equation 1: 2D DIC Shape Function XIII
Equation 2: Standardized Covariance Cross Correlation Function XIII
Equation 3: Normalized Sum-Square Difference Correlation Function XIII
Equation 4: f_m and g_m in Equation 2 and 3 XIV
Equation 5: Relationship of Depth and Disparity at each pixel (x, y) [10] XV
Equation 6: Basic Expression of 3D Re-projection Based on Pinhole Camera Model XVII
Equation 7: Camera Model Used in OpenCV XX
Equation 8: Camera Matrix A XX
Equation 9: Newly Defined Matrix B Encapsulating Intrinsic Parameters XXI
Equation 10: Retrieving Intrinsic Parameters from B XXI
Equation 11: Handling Radial Distortions in OpenCV XXI
Equation 12: Handling Tangential Distortion in OpenCV XXI
Equation 13: Distortion Coefficients to Be Considered in OpenCV XXI
Equation 14: Definition of Matrix H XXII
Equation 15: Decomposing H into Columns h_1, h_2, h_3 XXII
Equation 16: Calculation of Rotation and Translation Matrices XXII
Equation 17: Back-Projection Error XXIII
Equation 18: Relationship among R_1, R_2 and T_1, T_2 XXIII
Equation 19: Decomposing of T by Rows: T_0, T_1, T_2 XXIII
Equation 20: Essential Matrix XXIII
Equation 21: Fundamental Matrix XXIII
Equation 22: Projection Matrices P1 and P2 XXIV
Equation 23: Definition of Reconstruction Matrix Q XXIV
Equation 24: Expression of Matrix Q XXV
Equation 25: Functions to Calculate Transformation Maps in OpenCV XXXI
Equation 26: Mapping Each Point Using Transformation Maps XXXI
Equation 27: Usage of Matrix Q XXXIII
Equation 28 XXXVIII
Equation 29 XXXVIII
Equation 30 XXXVIII
Equation 31 XXXVIII
Equation 32 XXXVIII
Equation 33 XXXIX
Equation 34 XXXIX
Equation 35 XXXIX
Equation 36 XXXIX
Equation 37: Essential Matrix XXXIX
Equation 38 XL
Equation 39: Fundamental Matrix XL

List of Abbreviations

DIC	Digital Image Correlation
DVC	Digital Volume Correlation
OpenCV	Open Source Computer Vision
PCL	Point Cloud Library
SAD	Sum of Absolute Difference
SSD	Sum of Squared Difference
SSE2	Streaming SIMD Extensions
TBB	Threading Building Blocks

Contents

Abstract III
Acknowledgements IV
List of Figures V
List of Equations VI
List of Abbreviations VII
Introduction X
  Project Motivation and Objective X
  Project Organization X
  Project Schedule XI
Literature and Theory Review XII
  Digital Image Correlation (DIC) XII
    2D DIC XII
    3D DIC XIV
  Epipolar Geometry XV
  Pinhole Camera Model XVI
Implementation Process XVII
  Libraries XVIII
    OpenCV XVIII
    PCL XVIII
  Camera Calibration XIX
    Function Descriptions XIX
    Testing and Results XXV
  Stereo Correspondence XXX
    Function Descriptions XXX
    Testing and Results XXXII
  3D Reconstruction XXXIII
    Function Descriptions XXXIII
Conclusion and Future Work XXXV
  Discussions XXXV
    Re-projection Error XXXV
    Disparity Map XXXV
  Limitations XXXVI

Algorithms Available for Better Disparity Map XXXVI
Detection Restricted to Object Surface XXXVI
Follow-up Work XXXVI
Actual Experiments XXXVI
Subset Deformation Measurement XXXVI
Project Integration XXXVI
User Interface Implementation XXXVII
Appendix XXXVIII
A – Solution for Intrinsic Constraints from Matrix B XXXVIII
B – Essential and Fundamental Matrices XXXIX
Bibliography XLI

Introduction

Project Motivation and Objective

With the increasing popularity of applications based on techniques of Digital Image Correlation (DIC), a number of further researches have been conducted to evaluate and improve the methods in terms of accuracy and speed performance [1]. Implementations of 2D DIC applications are also widely available and have already generated satisfying results [2]. Meanwhile, the great potential of 3D DIC has also been disclosed during the research process [3]. It is necessary to implement the 3D DIC methods and test for its performance for the purpose of overcoming the limitations of 2D DIC techniques [2].

The objective of this project is to exploit existing models and tools to implement a solution proving the feasibility of 3D DIC techniques using the concept of stereo vision.

Project Organization

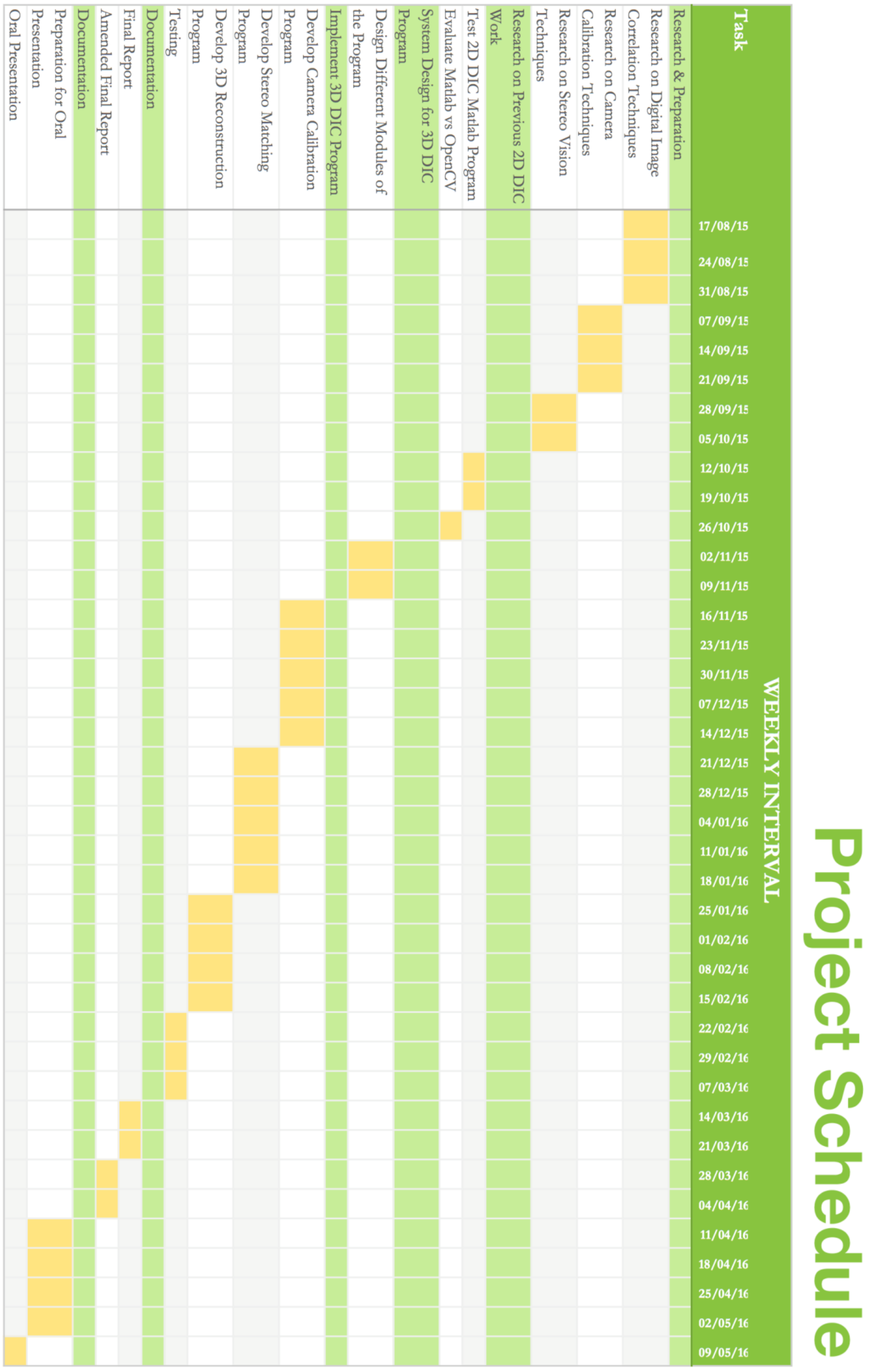
This report begins with the introduction of basic background information of both 2D and 3D DIC and hence leading to the project motivation.

Then the review of key theories referred in the implementation process is provided in the Literature Review section.

In the following section, the implementation stages with detailed functions and algorithms used in each stage are explained thoroughly.

Last but not least, the results and limitations of the implementation are discussed, and the possible future work is proposed specifically.

Project Schedule



Project Schedule

WEEKLY INTERVAL

Task	17/08/15	24/08/15	31/08/15	07/09/15	14/09/15	21/09/15	28/09/15	05/10/15	12/10/15	19/10/15	26/10/15	02/11/15	09/11/15	16/11/15	23/11/15	30/11/15	07/12/15	14/12/15	21/12/15	28/12/15	04/01/16	11/01/16	18/01/16	25/01/16	01/02/16	08/02/16	15/02/16	22/02/16	29/02/16	07/03/16	14/03/16	21/03/16	28/03/16	04/04/16	11/04/16	18/04/16	25/04/16	02/05/16	09/05/16
Research & Preparation																																							
Research on Digital Image Correlation Techniques	🟨	🟨	🟨																																				
Research on Camera Calibration Techniques				🟨	🟨	🟨																																	
Research on Stereo Vision Techniques							🟨	🟨																															
Research on Previous 2D DIC Work																																							
Test 2D DIC Matlab Program									🟨	🟨																													
Evaluate Matlab vs OpenCV											🟨																												
System Design for 3D DIC Program																																							
Design Different Modules of the Program												🟨	🟨																										
Implement 3D DIC Program																																							
Develop Camera Calibration Program														🟨	🟨	🟨	🟨	🟨																					
Develop Stereo Matching Program																			🟨	🟨	🟨	🟨	🟨																
Develop 3D Reconstruction Program																								🟨	🟨	🟨	🟨												
Testing																												🟨	🟨	🟨									
Documentation																																							
Final Report																															🟨	🟨							
Amended Final Report																																	🟨	🟨					
Documentation																																							
Preparation for Oral Presentation																																			🟨	🟨	🟨	🟨	
Oral Presentation																																							🟨

Literature and Theory Review

Digital Image Correlation (DIC)

2D DIC

Applying correlation method to find object displacement from digital images is very useful when the object to be measured is too far or too difficult to be measured directly. For example, the application of correlation in frequency domain to measure spatial distance captured by satellite photography was introduced in 1970 by Anuta [4]. Later, research regarding DIC's application in the field of mechanics was led by Professor Sutton from University of South Carolina in 1980s [5]. The DIC for 2D images only matches the differences within the image plane, so it is usually used to measure the displacement of an object on the same plane without considering the position and lens distortion of the camera, except for typical off-axis images [6]. DIC is used to track the change of a specified subset in one or more digital images. For example, if 2D DIC is used to detect the displaced speckle on a piece of material after a forced deformation, the target and reference coordinate system is depicted in the diagram below.

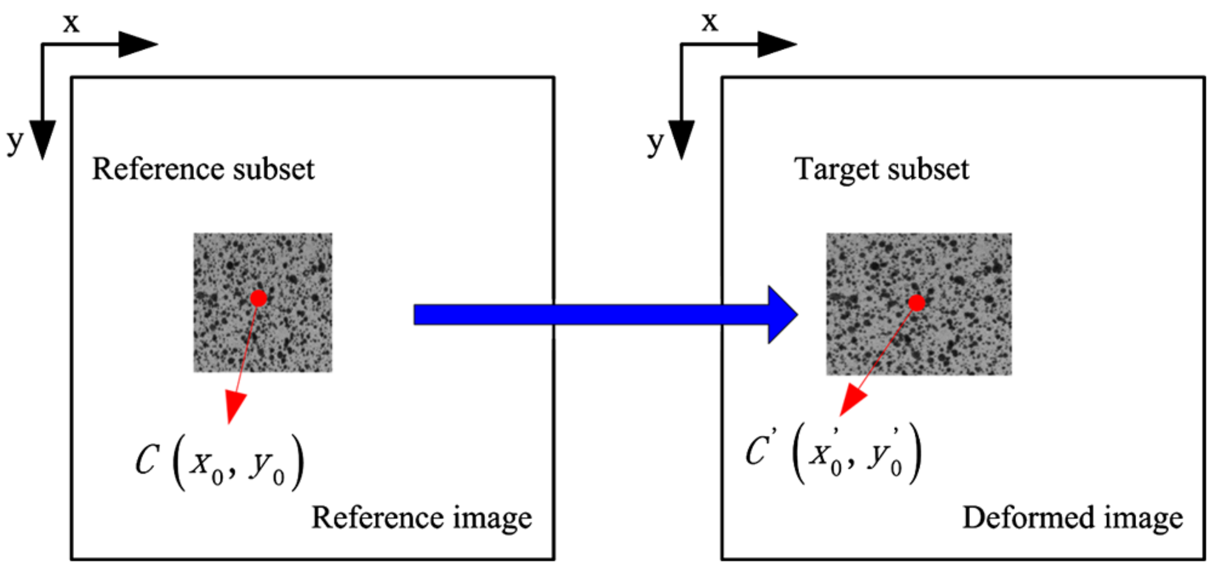


Figure 1: 2D DIC Diagram

In order to measure the deformation of subsets in the image, some shape functions between the reference and target subset coordinates should be defined. As noticeable, besides translation, the displacement usually should also support rotation, shear, and stretch. Since in this case, the deformation is relatively small, and the displacement occurs in a plane perpendicular to the camera optical axis, the shape function can be expressed as a 2D affine function shown in the next page. In the expressions, Δx and Δy are the translation from centres of sub-images. The unknown parameters *u*, *v* are the displacement of centres of sub-images, together with *δu/δx*, *δu/δy*, *δv/δx*, *δv/δy*, which are the displacement gradients, are usually represented by a vector P [7].

Equation 1: 2D DIC Shape Function

$$x^\star = x + u + \frac{\partial u}{\partial x}\Delta x + \frac{\partial u}{\partial y}\Delta y$$
$$y^\star = y + v + \frac{\partial v}{\partial x}\Delta x + \frac{\partial v}{\partial y}\Delta y$$

To match the sub-images and determine the parameter vector P, a correlation function has to be defined for the purpose of maximising the correlation between pixel intensity of corresponding sub-images. Although calculating the correlation in frequency domain might be faster [4], the calculation in spatial domain can achieve more accurate deformation measurement [8], so only the latter is considered in this project. There are different correlation functions available, and they are more or less equivalent to each other especially when the intensity within each matching pixel does not change after deformation although the position of the pixel has changed [7]. The two types of correlation functions are shown below.

Equation 2: Standardized Covariance Cross Correlation Function

$$C_{f,g}(\vec{p}) = \frac{\sum_{x=-M}^{M}\sum_{y=-M}^{M}[f(x,y)-f_m]\times[g(x',y')-g_m]}{\sqrt{\sum_{x=-M}^{M}\sum_{y=-M}^{M}[f(x,y)-f_{\mathrm{m}}]^2}\sqrt{\sum_{x=-M}^{M}\sum_{y=-M}^{M}[g(x',y')-g_{\mathrm{m}}]^2}}$$

Equation 3: Normalized Sum-Square Difference Correlation Function

$$C_{f,g}(\vec{p}) = \sum_{x=-M}^{M}\sum_{y=-M}^{M}\left[\frac{f(x,y)-f_m}{\sqrt{\sum_{x=-M}^{M}\sum_{y=-M}^{M}[f(x,y)-f_{\mathrm{m}}]^2}} - \frac{g(x',y')-g_m}{\sqrt{\sum_{x=-M}^{M}\sum_{y=-M}^{M}[g(x',y')-g_{\mathrm{m}}]^2}}\right]^2$$

Equation 4: f_m and g_m in Equation 2 and 3

$$f_m = \frac{1}{(2M+1)^2} \sum_{x=-M}^{M} \sum_{y=-M}^{M} [f(x,y)]^2 \qquad g_m = \frac{1}{(2M+1)^2} \sum_{x=-M}^{M} \sum_{y=-M}^{M} [g(x',y')]^2$$

The setup for 2D DIC is quite simple, and only one camera is needed. The setup diagram is given below.

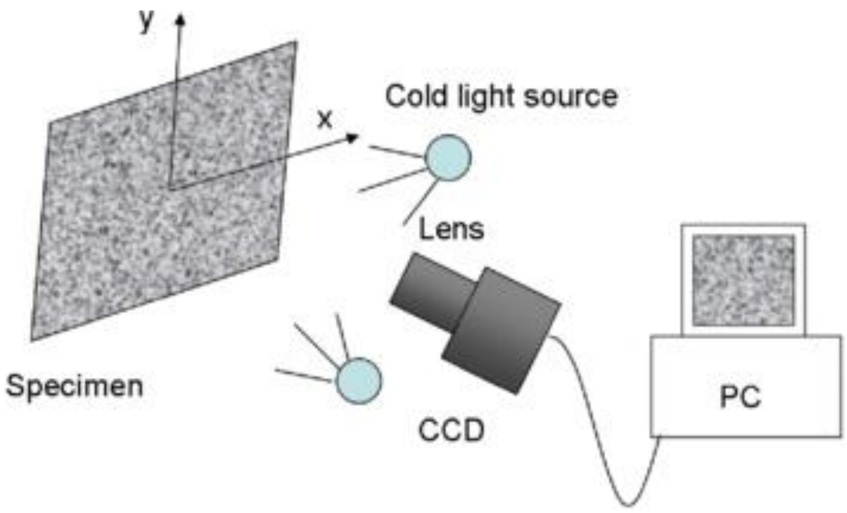


Figure 2: Set-up for 2D DIC

3D DIC

The DIC method described above is based on the assumptions that the material surface should be a plane, and the deformation should also happen within this plane, which must be perpendicular with the optical axis of the camera [2]. In order to break the restrictions especially to detect the 3D displacement of the object, 3D DIC is introduced. The concept of matching reference and target sub-images by applying correlation functions is the same, while the difference and the main task is to obtain depth information of the object. This can be achieved by using the concept of stereo vision, which is explained in the next section.

There is also a new concept invented after 3D DIC, which is Digital Volume Correlation (DVC). While 3D DIC is only able to capture the information from the surface of the object, DVC attempts to reveal the interior deformation within a material volume. This is usually done by taking images consecutively from different depths that could represent a volume. For example, using the technology of tomography, the theory of DVC has been widely applied in the field relevant to biomedicine [9].

Epipolar Geometry

Epipolar geometry elaborates the concept of stereo vision. It is the simulation of human eyes that from a pair of images captured by two cameras, the shift of an object from left to right image is obtained. From this shift information, which is called disparity, the distance from the object to the baseline of the cameras can be interpreted. This process is depicted by the diagram and equation below. In the equation, Z(*x*, *y*) is the depth at pixel (*x*, *y*), and *d*(*x*, *y*) is the disparity. As noticeable, Z is reversely proportional to *d*. This relation can be used to retrieve the 3D coordinates of the object points.

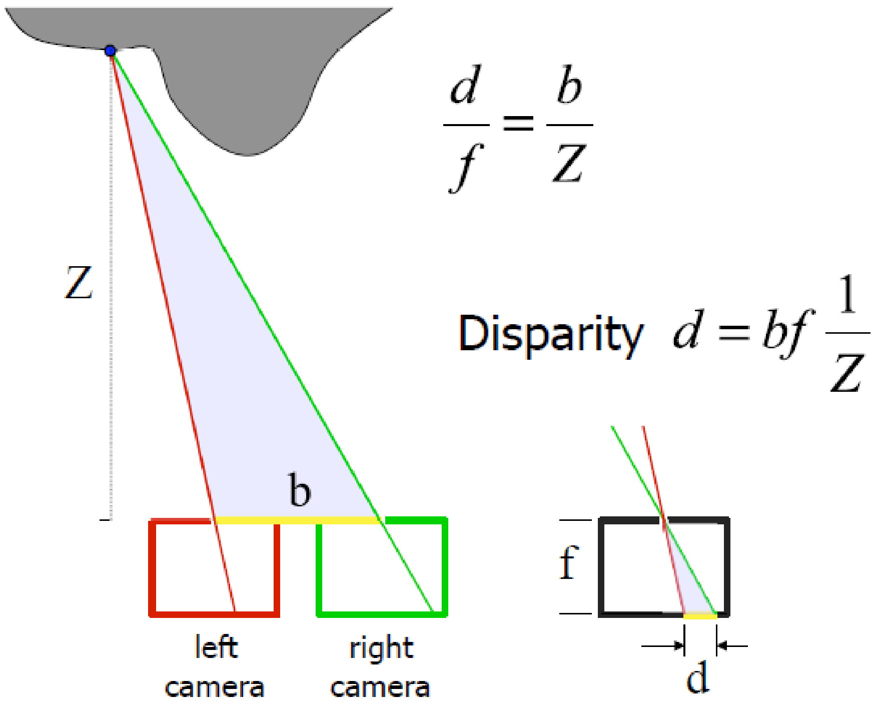


Figure 3: Stereo Geometry [10]

Equation 5: Relationship of Depth and Disparity at each pixel (*x*, *y*) [10]

$$Z(x,y) = \frac{fb}{d(x,y)}$$

If the stereo vision model is examined geometrically, the relationship between the two cameras and also the object is shown in the following diagram. In the diagram, *P* is the object to be detected, and its projections on both image planes are p_1 and p_2 respectively. O_1 and O_2 are the optical centres of cameras, so O_1O_2 is the baseline. Since every plane crossing through the baseline is an epipolar plane, PO_1O_2 is such a plane. This plane also intersects with both image planes, and the intersecting lines $p_1e_1$and p_2e_2 are the epipolar lines. As illustrated in the diagram, the epipolar constraint is defined as: the possible matches of p_1 on the right image are constrained to lie along the epipolar line on the image. This principle is used in the implementation to verify the stereo matching results by checking if the matched points found lie on the epipolar line of the image.

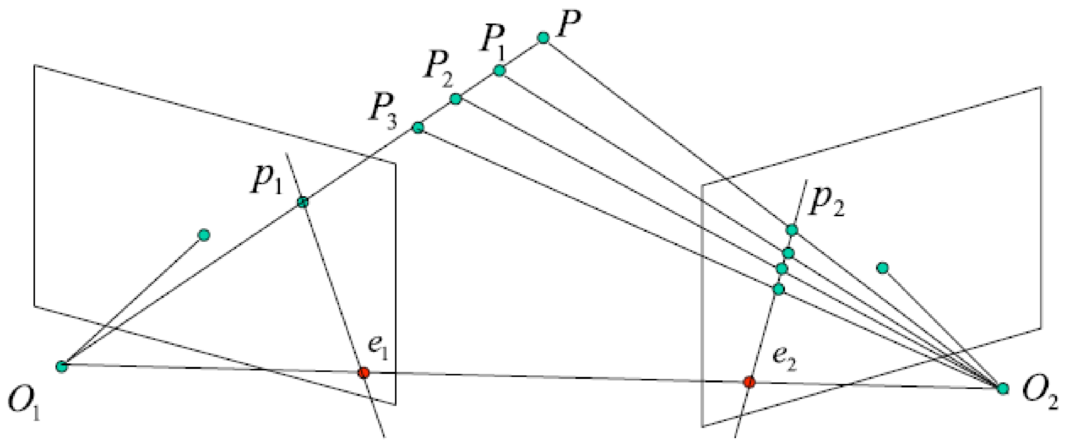


Figure 4: Epipolar Geometry [11]

Pinhole Camera Model

Pinhole cameras are the simple cameras without lens, and the aperture is a small pinhole that allows light going through and projecting an inversed object on the image plane. The physical model and its geometric interpretation are shown in the following figures.

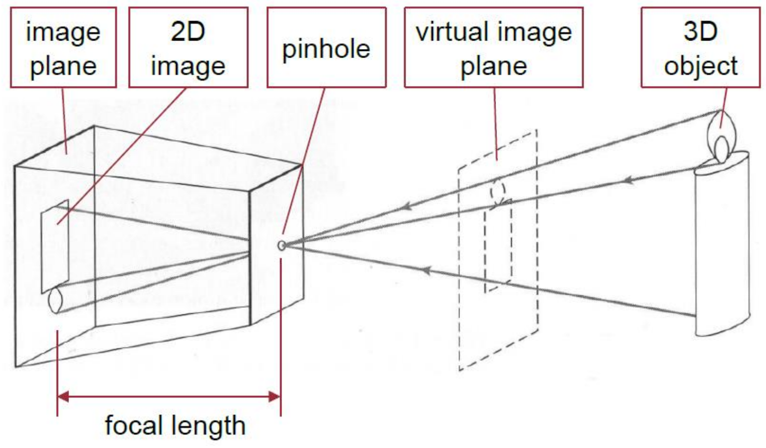


Figure 5: Physical Pinhole Camera Model [12]

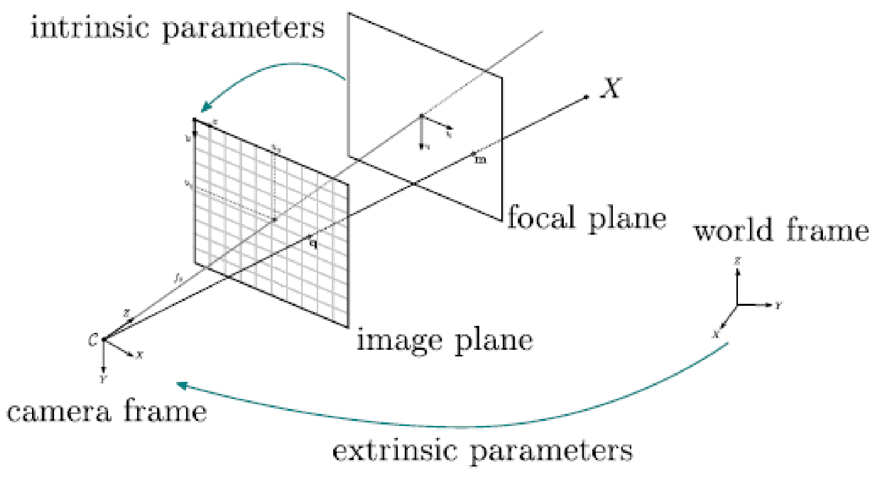


Figure 6: Geometry of Pinhole Camera Model

Although pinhole camera model does not take into account various factors that might affect the estimations of real cameras used, it is still widely adopted in the field of computer vision to re-project 3D scene from 2D image [13]. The mathematical model used is shown below, which is applied by the implementation to be introduced in the next section. In the expression, x_w, y_w, z_w are the world coordinates, and u, v are the image coordinates. A is the camera matrix with intrinsic parameters, and s is a scalar, while **R** and t are the transformation matrices encapsulating extrinsic parameters.

Equation 6: Basic Expression of 3D Re-projection Based on Pinhole Camera Model

$$s\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = A\begin{bmatrix} \mathbf{R} & t \end{bmatrix}\begin{bmatrix} x_w \\ y_w \\ z_w \\ 1 \end{bmatrix}$$

Implementation Process

Initially, there was existing MATLAB code for part of the project, which was the function of two-camera calibration using chessboard. I tried the program on the computer and it worked fine. However there are some problems if I continue on the project with MATLAB. Firstly, besides two cameras, the program needs a projector to mark the detected corners, which made the initial testing work inflexible and time-consuming. Secondly, MATLAB uses interpreted language, which makes the program run slowly. In addition, it is possible that this project would be moved to other platforms for further utilization, which might not be supported by MATLAB due to its poor portability. Thirdly, I did not have much experience with MATLAB, so it was also extra time spent to understand the original code and to build additional functions to it.

During the early research on the implementation method of DIC, I found that OpenCV also provides full range of functions to realize the whole process [14]. OpenCV with C++ also solved the problems of slowness and portability. More importantly, it is open source, so the community base is quite strong for providing sufficient information during implementation process. I also had experience in coding C++ project, so I decided to redo the whole project in C++ with OpenCV library instead of continuing the work on the existing MATLAB program.

The whole project mainly consists of three sub projects, namely calibration, stereo matching, and 3D reconstruction. Firstly, the two cameras need to be calibrated, so their intrinsic and extrinsic parameters are obtained in order to build the matrix for calculating depth information of the object captured by the cameras. Then the calibrated cameras can begin to take pictures of the target object. The task now is to match the points in left image to the points in right image, so the 2D-space coordinates can be calculated, and next the 3D world coordinates can also be computed by the matrix obtained during calibration. The last step is to gather the coordinates of all the detected points and display the point cloud.

Libraries

OpenCV

OpenCV stands for Open Source Computer Vision. It is licensed for both academic and commercial use, and it supports multiple languages and platforms. It also supports multicore processing, which enhances efficiency of the applications. It has a function package for camera calibration and 3D reconstruction, which is the main source of the program in the implementation process.

PCL

PCL stands for Point Cloud Library, which consists of powerful algorithms for processing 3D point clouds. This includes noise filtering, object recognition etc for large scale point clouds, which improves the performance of the program [15]. It also supports the additional colour channel, which enhances the visualisation of the 3D model reconstructed.

Camera Calibration

For the first module, camera calibration, since the original MATLAB project was using chessboard, and there are some existing printed chessboards, I decided to continue using chessboard to do the calibration for my C++ project. OpenCV source library provides the sample code for stereo calibration, and its major functions are sufficient for the purpose of this project, so the source code was mainly modified from the sample code. I also adopted the modifications done by the blogger Martin Peris in order to save the computed matrices. Additional changes were added for the purpose of testing the program and getting better results.

Function Descriptions

FindChessboardCorners()

This function takes in the image and number of corners, and it will return an array storing the coordinates of all the corners. It first normalizes the input image using histogram equalization, so the contrast of black and white is stretched. Then the image is dilated using structure element of rectangular, so the distinct corners are sufficiently separated for detection. After that, it finds and draws out all the contours, which is defined in OpenCV the boundaries of a shape with same intensity. In this case, the contours are the boundaries of each grid on the chessboard. The function filters out contours that are too small or irregular, so only the triangles with acceptable size are found and drawn. Based on these contours, it then identifies the possible corners and ranks them according to correct order.

After finding the estimated corners, another function called *cvFindCornerSubPix()* is then applied to the result in order to get more precise location for the corners. This function basically iterates within each search window to restrict the centre of corners until certain accuracy threshold is reached.

StereoCalibrate()

This is the key function for the step of camera calibration. The function takes in left and right images, and also the coordinate vector of corners found in the previous *FindChessboardCorners()* function. It calculates and returns the intrinsic and extrinsic parameters as matrices according to the pinhole camera model explained in literature review. The formula used is shown in the next page.

Equation 7: Camera Model Used in OpenCV

$$s\, m' = A[R|t]M'$$

$$s\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

The function first calculates the matrix A, which is the matrix of intrinsic parameters, for both cameras. Then the extrinsic parameters are estimated based on the intrinsic parameters. This process is done by a built-in function called *CalibrateCamera()*. The algorithm was implemented based on the solution proposed by Zhang's research [16]. Zhang declares a new matrix B described in the following figure, which is denoted as the image of an absolute conic. The estimation of B is based on the matrix constructed from coordinates of corresponding 2D and 3D points[1]. Therefore, the intrinsic parameters can be calculated according to the equations in figure. Note that γ in the matrix A, which is the skewness of image axes of u and v, in the model used by OpenCV is always zero. This is a safe assumption if recent digital cameras are exploited, and sometimes even more accurate can be obtained with this assumption [7]. The algorithm requires at least six set of points to be input from each camera. In my program, it takes twenty pairs of images from cameras to be calibrated. The principle points c_x and c_y are basically the centre points of the images, which are easy to get. Then the matrix system stacked by all the points are then passed to *cvSolve()* to get the focal length.

Equation 8: Camera Matrix A

$$\mathbf{A} = \begin{bmatrix} \alpha & \gamma & u_0 \\ 0 & \beta & v_0 \\ 0 & 0 & 1 \end{bmatrix}$$

[1] Details are described in Appendix A

Equation 9: Newly Defined Matrix B Encapsulating Intrinsic Parameters

$$\mathbf{B} = \mathbf{A}^{-T}\mathbf{A}^{-1} \equiv \begin{bmatrix} B_{11} & B_{12} & B_{13} \\ B_{12} & B_{22} & B_{23} \\ B_{13} & B_{23} & B_{33} \end{bmatrix}$$

$$= \begin{bmatrix} \frac{1}{\alpha^2} & -\frac{\gamma}{\alpha^2\beta} & \frac{v_0\gamma - u_0\beta}{\alpha^2\beta} \\ -\frac{\gamma}{\alpha^2\beta} & \frac{\gamma^2}{\alpha^2\beta^2} + \frac{1}{\beta^2} & -\frac{\gamma(v_0\gamma - u_0\beta)}{\alpha^2\beta^2} - \frac{v_0}{\beta^2} \\ \frac{v_0\gamma - u_0\beta}{\alpha^2\beta} & -\frac{\gamma(v_0\gamma - u_0\beta)}{\alpha^2\beta^2} - \frac{v_0}{\beta^2} & \frac{(v_0\gamma - u_0\beta)^2}{\alpha^2\beta^2} + \frac{v_0^2}{\beta^2} + 1 \end{bmatrix}$$

Equation 10: Retrieving Intrinsic Parameters from B

$$\begin{aligned} v_0 &= (B_{12}B_{13} - B_{11}B_{23})/(B_{11}B_{22} - B_{12}^2) \\ \lambda &= B_{33} - [B_{13}^2 + v_0(B_{12}B_{13} - B_{11}B_{23})]/B_{11} \\ \alpha &= \sqrt{\lambda/B_{11}} \\ \beta &= \sqrt{\lambda B_{11}/(B_{11}B_{22} - B_{12}^2)} \\ \gamma &= -B_{12}\alpha^2\beta/\lambda \\ u_0 &= \gamma v_0/\beta - B_{13}\alpha^2/\lambda\ . \end{aligned}$$

The function also considers distortions, including radial and tangential distortions. The representation of the distortion coefficient in OpenCV is shown in the figures below. The parameters are solved using Levenberg-Marquardt Algorithm in OpenCV based on Zhang's method [16].

Equation 11: Handling Radial Distortions in OpenCV

$$\begin{aligned} x_{\text{corrected}} &= x(1 + k_1 r^2 + k_2 r^4 + k_3 r^6) \\ y_{\text{corrected}} &= y(1 + k_1 r^2 + k_2 r^4 + k_3 r^6) \end{aligned}$$

Equation 12: Handling Tangential Distortion in OpenCV

$$\begin{aligned} x_{\text{corrected}} &= x + [2p_1 xy + p_2(r^2 + 2x^2)] \\ y_{\text{corrected}} &= y + [p_1(r^2 + 2y^2) + 2p_2 xy] \end{aligned}$$

Equation 13: Distortion Coefficients to Be Considered in OpenCV

$$\text{Distortion}_{\text{coefficients}} = (k_1 \quad k_2 \quad p_1 \quad p_2 \quad k_3)$$

Once the camera matrices and distortion coefficients are computed, the rotation matrix R and translation matrix t can be calculated based on intrinsic parameters. Since we have homography H [17], the rotation and translation matrices can be calculated by the equations shown in the figures.

Equation 14: Definition of Matrix H

$$s\tilde{\mathrm{m}} = \mathbf{H}\tilde{\mathrm{M}} \qquad \text{with} \quad \mathbf{H} = \mathbf{A}\begin{bmatrix}\mathbf{r}_1 & \mathbf{r}_2 & \mathbf{t}\end{bmatrix}$$

Equation 15: Decomposing H into Columns h_1, h_2, h_3

$$\mathbf{H} = \begin{bmatrix}\mathbf{h}_1 & \mathbf{h}_2 & \mathbf{h}_3\end{bmatrix}$$

Equation 16: Calculation of Rotation and Translation Matrices

$$\begin{aligned}\mathbf{r}_1 &= \lambda\mathbf{A}^{-1}\mathbf{h}_1\\ \mathbf{r}_2 &= \lambda\mathbf{A}^{-1}\mathbf{h}_2\\ \mathbf{r}_3 &= \mathbf{r}_1 \times \mathbf{r}_2\\ \mathbf{t} &= \lambda\mathbf{A}^{-1}\mathbf{h}_3 \qquad \lambda = 1/\|\mathbf{A}^{-1}\mathbf{h}_1\| = 1/\|\mathbf{A}^{-1}\mathbf{h}_2\|\end{aligned}$$

Matrix H is found by the function *cvFindHomography()*, which takes in all the point pairs and estimate the perspective transformation from the world coordinate system to the image coordinate system. The return point of the estimation is when the back-projection error is minimised. The expression of back-projection error is given in the next page. After getting all the extrinsic parameters, the Levenberg-Marquardt algorithm is applied in order to further reduce re-projection error. In OpenCV, this is done by calling the *solver()* in *CvLevMarq()* class. This algorithm uses an iterative method to find local minimum for function constructed by squares of non-linear expressions. It is used repetitively to optimise both intrinsic and extrinsic parameters. The re-projection error returned by the program is calculated by taking the standard deviation of coordinates of re-projected and original images. This is achieved by firstly getting coordinates of re-projected image from calling *cvProjectPoints()*, and then the coordinate differences between each re-projected point and original point are calculated and saved to a matrix. After that, the re-projection error is obtained by using *cvNorm()* with *CV_L2* flag set to get the square root of sum of difference squares. Lastly, the error is divided by twice of the total number of points, because the errors for both x, y axis are calculated separately.

Equation 17: Back-Projection Error

$$\sum_i \left(x'_i - \frac{h_{11}x_i + h_{12}y_i + h_{13}}{h_{31}x_i + h_{32}y_i + h_{33}} \right)^2 + \left(y'_i - \frac{h_{21}x_i + h_{22}y_i + h_{23}}{h_{31}x_i + h_{32}y_i + h_{33}} \right)^2$$

The function encapsulates the parameters in the essential and fundamental matrices E and F[2]. The equations that OpenCV uses for calculation of the matrices are given below. It refers to R_1, T_1, and R_2, T_2 as the rotation and translation matrices of the world coordinate object respecting to left and right cameras respectively.

Equation 18: Relationship among R_1, R_2 and T_1, T_2

$$R_2 = R * R_1 T_2 = R * T_1 + T,$$

Equation 19: Decomposing of T by Rows: T_0, T_1, T_2

$$T = [T_0, T_1, T_2]^T$$

Equation 20: Essential Matrix

$$E = \begin{bmatrix} 0 & -T_2 & T_1 \\ T_2 & 0 & -T_0 \\ -T_1 & T_0 & 0 \end{bmatrix} * R$$

Equation 21: Fundamental Matrix

$$F = cameraMatrix2^{-T} E cameraMatrix1^{-1}$$

After the function returns the matrices, the program is possible to verify the calibration result by comparing the epipolar line calculated from the calibrated parameters with the original points on the line. This is done by calling function *cvComputeCorrespondEpilines()*, and the average error is given by the absolute difference between calculated and original points on the line, divided by the total number of points from all the chessboards found. The average error is then printed on the program screen.

StereoRectify()

This function calculates the rectification transformation matrices for both cameras, so the image planes of the two cameras are virtually the same plane. By this means, the epipolar lines will be parallel, which makes the later correspondence step easier. OpenCV can perform both horizontal and vertical rectification, but here I only used the horizontal one, because vertical correspondence is not supported. The algorithm used is based on Bouguet's method

[2] Essential and fundamental matrices are explained in Appendix B

[18], and it returns the rotation matrices R1, R2, and projection matrices P1, P2, for left and right cameras respectively. It takes the average rotation of R, which is the rotation matrix from left camera to right camera, and splits R to both cameras. Then it rotates image planes so they are at the same orientation according to the baseline. The diagram depicting the process of Bouguet's methods as well as the expressions for P1 and P2 are given below.

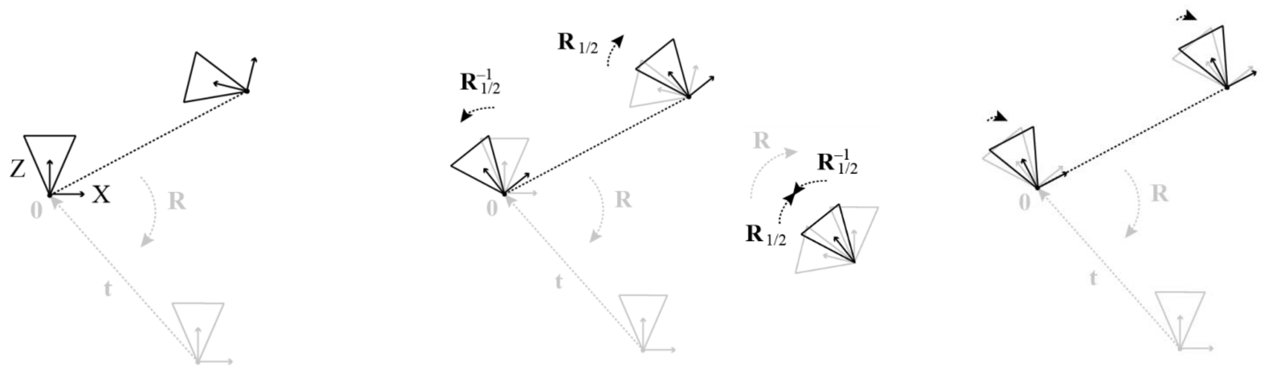


Figure 7: Bouguet's Method for Rectification

Equation 22: Projection Matrices P1 and P2

$$P1 = \begin{bmatrix} f & 0 & cx_1 & 0 \\ 0 & f & cy & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

$$P2 = \begin{bmatrix} f & 0 & cx_2 & T_x * f \\ 0 & f & cy & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

In addition, the function also calculates and returns the matrix Q, which is used for reconstruction 3D coordinates for image points. The usage and expression of Q in the reconstruction process is shown below. In the expression of matrix Q, c_x and c_y are the principle points of left image, and c_x' is for the right image, so the disparity is expressed as (c_x - c_x'). T_x is the length of baseline, which is also the translation from left to right image.

Equation 23: Definition of Reconstruction Matrix Q

$$[X\ Y\ Z\ W]^T = Q * [x\ y\ \mathtt{disparity}(x,y)\ 1]^T$$
$$\mathtt{\_3dImage}(x,y) = (X/W,\ Y/W,\ Z/W)$$

Equation 24: Expression of Matrix Q

$$Q = \begin{bmatrix} 1 & 0 & 0 & -c_x \\ 0 & 1 & 0 & -c_y \\ 0 & 0 & 0 & f \\ 0 & 0 & -1/T_x & (c_x - c'_x)/T_x \end{bmatrix}$$

Testing and Results

Since there were some difficulties in accessing to the limited number of cameras, I decided to use images taken by the blogger Martin Peris to test the program first. The process and results would be similar for the images taken by the cameras in the lab, if only the images are stored in the correct workspace of the project. And the program supports most image formats, so it would be convenient to switch to instant pictures, which is described in the future work section. Here I would like to show you the running process of the program using the downloaded pairs of images and also the calibration results.

The program starts by finding and drawing all the corners detected for each image.

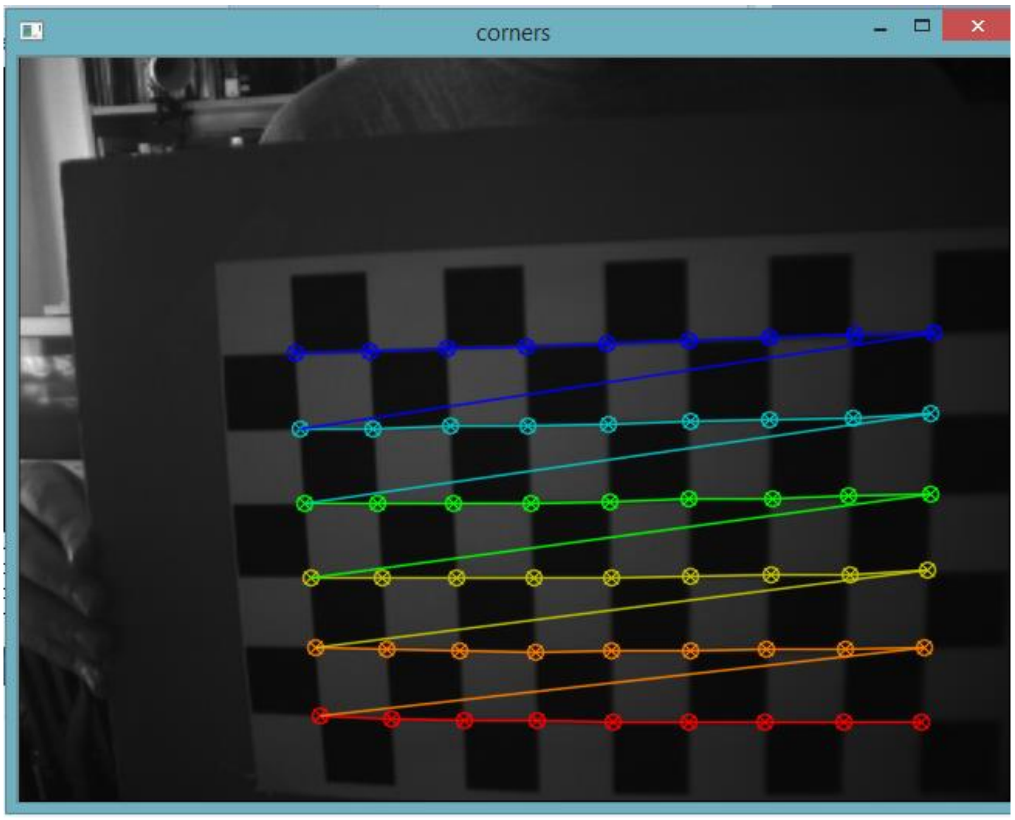
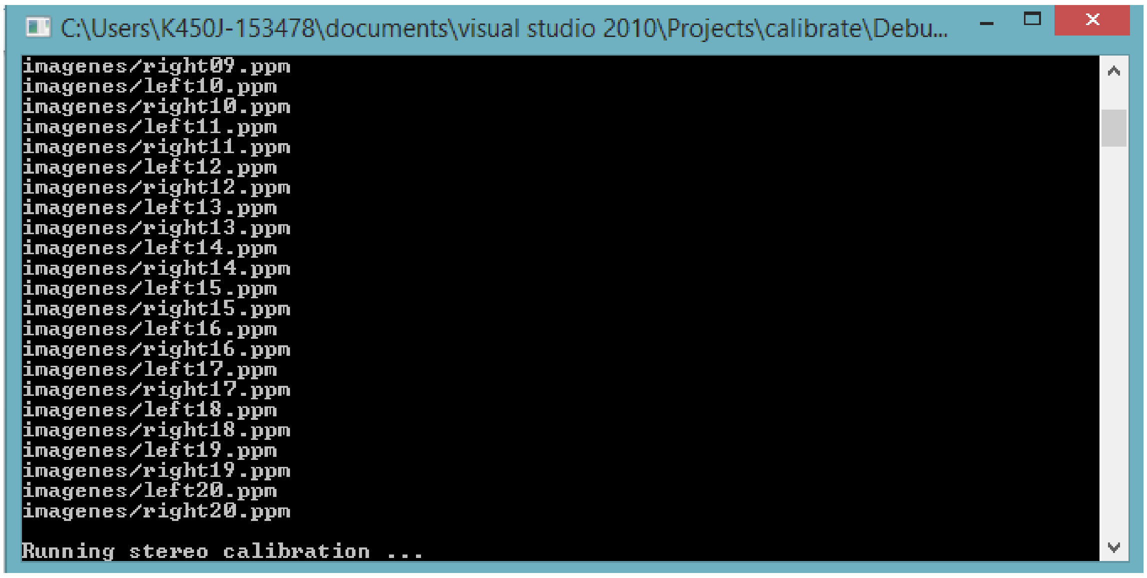


Figure 8: Find and Draw Corners

Then it starts doing the calibration, which takes about 30 seconds.

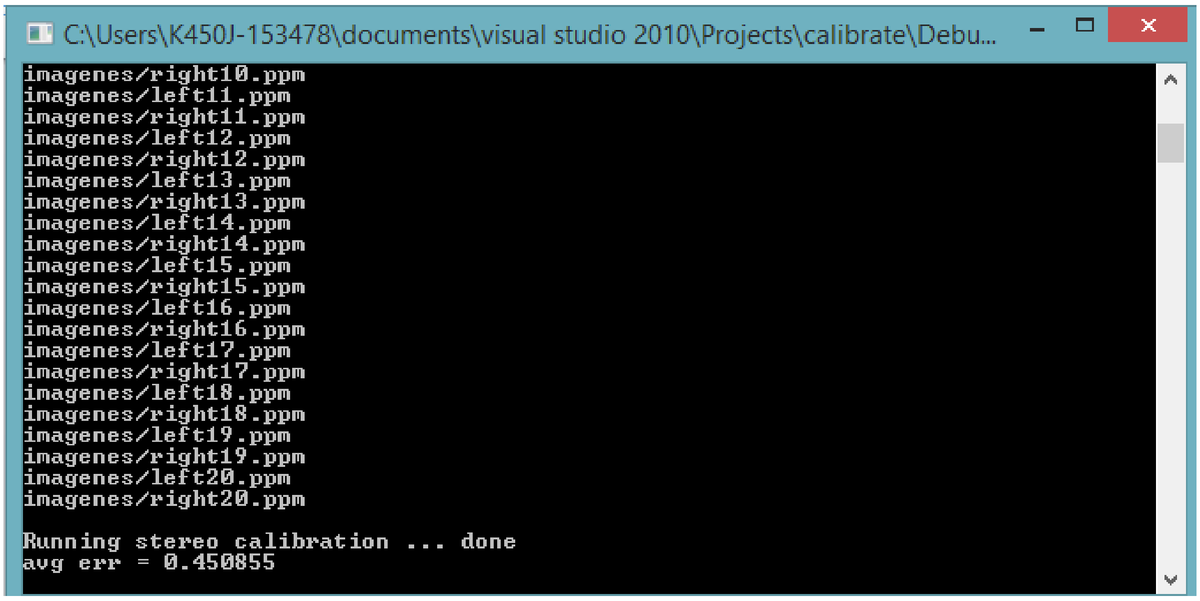

C:\Users\K450J-153478\documents\visual studio 2010\Projects\calibrate\Debu...

```
imagenes/right09.ppm
imagenes/left10.ppm
imagenes/right10.ppm
imagenes/left11.ppm
imagenes/right11.ppm
imagenes/left12.ppm
imagenes/right12.ppm
imagenes/left13.ppm
imagenes/right13.ppm
imagenes/left14.ppm
imagenes/right14.ppm
imagenes/left15.ppm
imagenes/right15.ppm
imagenes/left16.ppm
imagenes/right16.ppm
imagenes/left17.ppm
imagenes/right17.ppm
imagenes/left18.ppm
imagenes/right18.ppm
imagenes/left19.ppm
imagenes/right19.ppm
imagenes/left20.ppm
imagenes/right20.ppm

Running stereo calibration ...
```

Figure 9: Running Calibration

The calibration returns average error by comparing coordinates of matched points with points on the epipolar line.


C:\Users\K450J-153478\documents\visual studio 2010\Projects\calibrate\Debu...

```
imagenes/right10.ppm
imagenes/left11.ppm
imagenes/right11.ppm
imagenes/left12.ppm
imagenes/right12.ppm
imagenes/left13.ppm
imagenes/right13.ppm
imagenes/left14.ppm
imagenes/right14.ppm
imagenes/left15.ppm
imagenes/right15.ppm
imagenes/left16.ppm
imagenes/right16.ppm
imagenes/left17.ppm
imagenes/right17.ppm
imagenes/left18.ppm
imagenes/right18.ppm
imagenes/left19.ppm
imagenes/right19.ppm
imagenes/left20.ppm
imagenes/right20.ppm

Running stereo calibration ... done
avg err = 0.450855
```

Figure 10: Returning Re-projection Error

Then for each pair of images, the program displays the rectified image and also the disparity map.

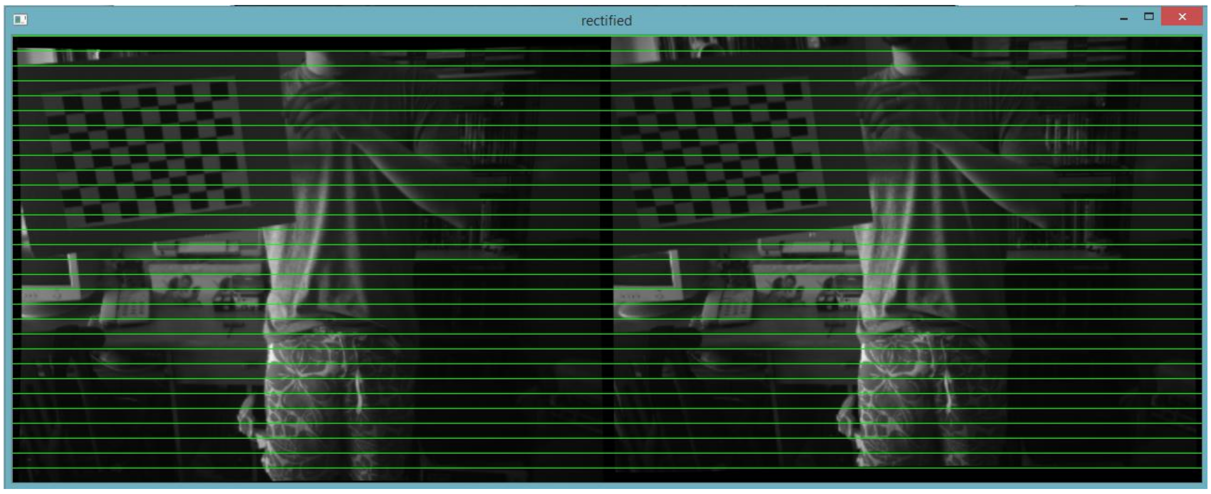


Figure 11: Running Rectification

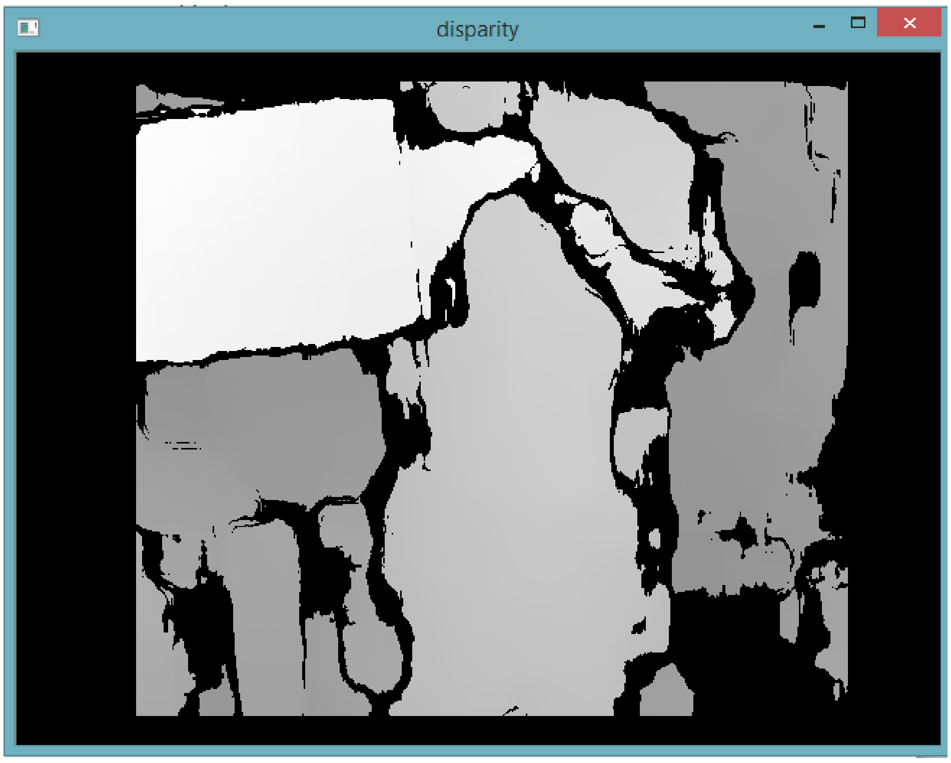
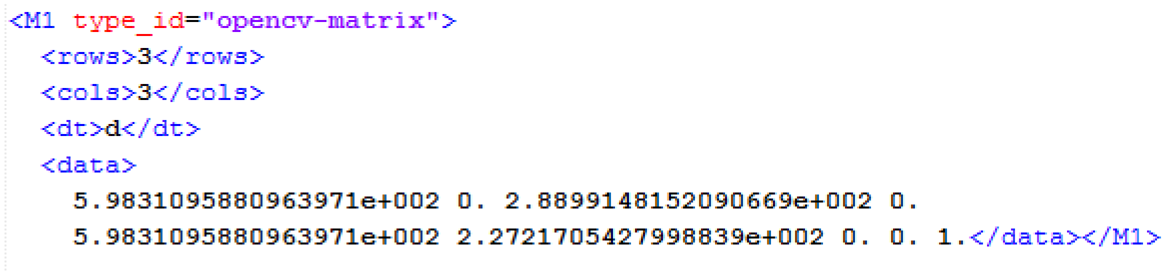


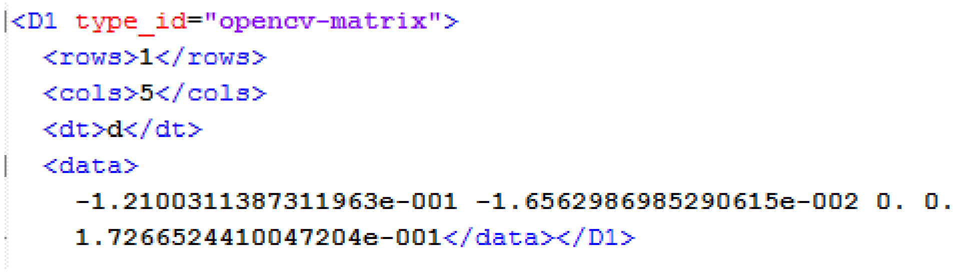
Figure 12: Returning Disparity Map for Each Image Pair

The matrices are saved as xml files, which are then to be read by stereo matching project.



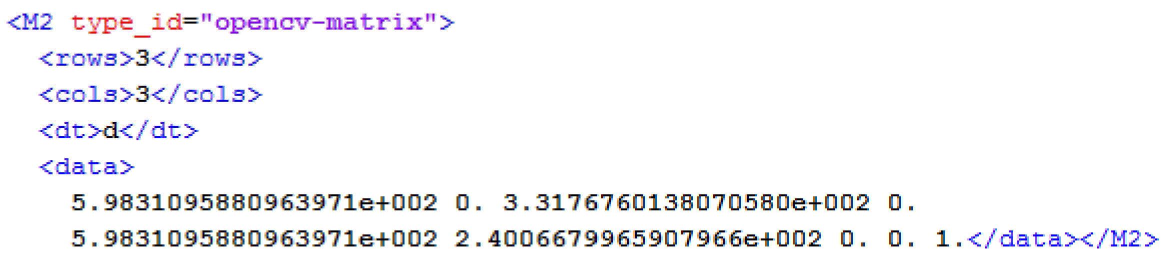
```
<M1 type_id="opencv-matrix">
  <rows>3</rows>
  <cols>3</cols>
  <dt>d</dt>
  <data>
    5.9831095880963971e+002 0. 2.8899148152090669e+002 0.
    5.9831095880963971e+002 2.2721705427998839e+002 0. 0. 1.</data></M1>
```

Figure 13: Left Camera Matrix



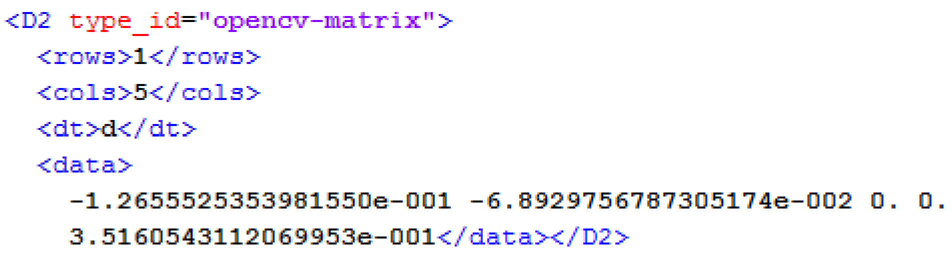
```
<D1 type_id="opencv-matrix">
  <rows>1</rows>
  <cols>5</cols>
  <dt>d</dt>
  <data>
    -1.2100311387311963e-001 -1.6562986985290615e-002 0. 0.
    1.7266524410047204e-001</data></D1>
```

Figure 14: Distortion Coefficients of Left Camera



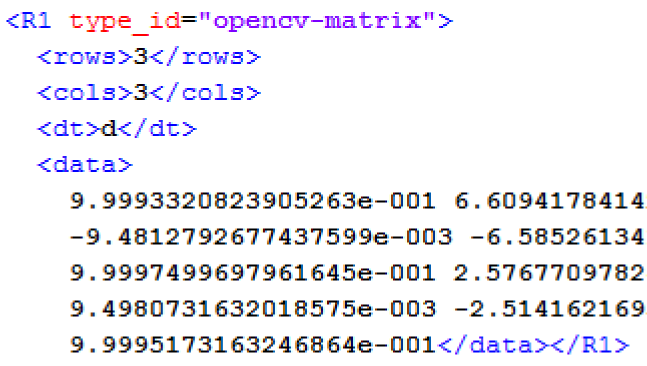
```
<M2 type_id="opencv-matrix">
  <rows>3</rows>
  <cols>3</cols>
  <dt>d</dt>
  <data>
    5.9831095880963971e+002 0. 3.3176760138070580e+002 0.
    5.9831095880963971e+002 2.4006679965907966e+002 0. 0. 1.</data></M2>
```

Figure 15: Right Camera Matrix



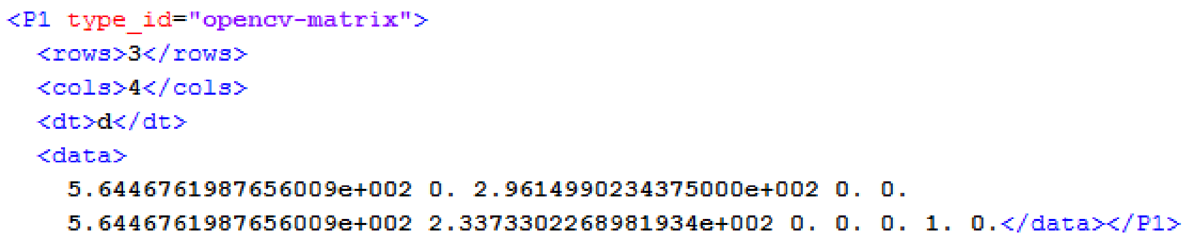
```
<D2 type_id="opencv-matrix">
  <rows>1</rows>
  <cols>5</cols>
  <dt>d</dt>
  <data>
    -1.2655525353981550e-001 -6.8929756787305174e-002 0. 0.
    3.5160543112069953e-001</data></D2>
```

Figure 16: Distortion Coefficients of Right Camera



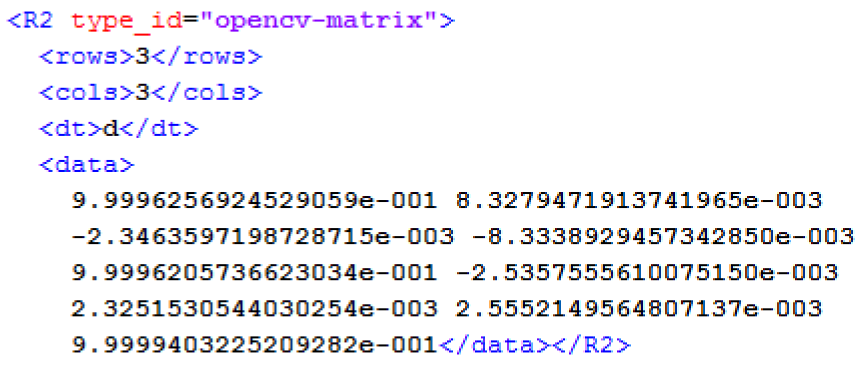
```
<R1 type_id="opencv-matrix">
  <rows>3</rows>
  <cols>3</cols>
  <dt>d</dt>
  <data>
    9.9993320823905263e-001 6.6094178414268893e-003
    -9.4812792677437599e-003 -6.5852613419654432e-003
    9.9997499697961645e-001 2.5767709782461589e-003
    9.4980731632018575e-003 -2.5141621693407093e-003
    9.9995173163246864e-001</data></R1>
```

Figure 17: Left Rotation Matrix



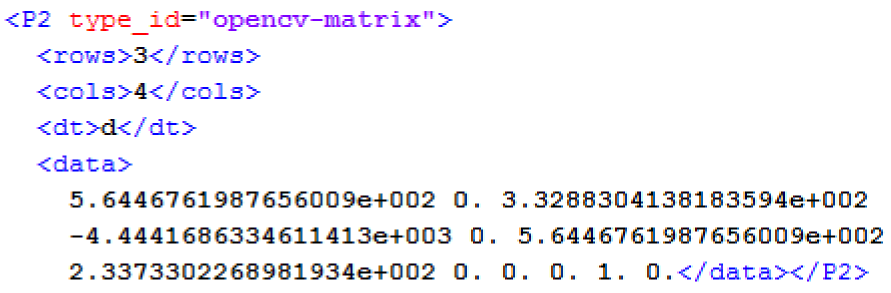
```
<P1 type_id="opencv-matrix">
  <rows>3</rows>
  <cols>4</cols>
  <dt>d</dt>
  <data>
    5.6446761987656009e+002 0. 2.9614990234375000e+002 0. 0.
    5.6446761987656009e+002 2.3373302268981934e+002 0. 0. 0. 1. 0.</data></P1>
```

Figure 18: Left Projection Matrix



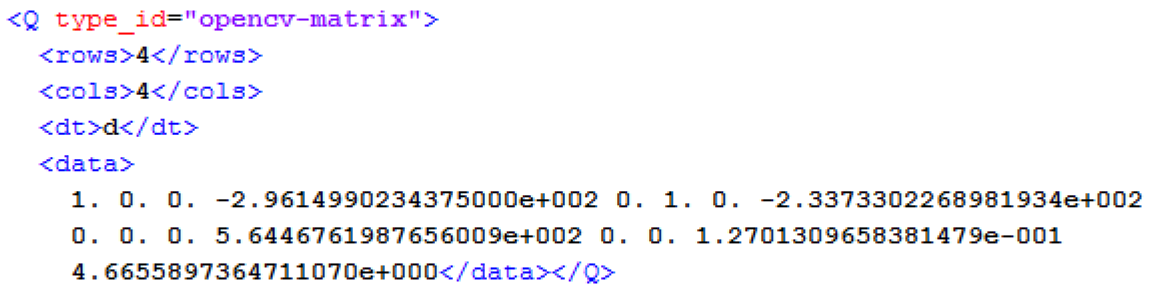
```
<R2 type_id="opencv-matrix">
  <rows>3</rows>
  <cols>3</cols>
  <dt>d</dt>
  <data>
    9.9996256924529059e-001 8.3279471913741965e-003
    -2.3463597198728715e-003 -8.3338929457342850e-003
    9.9996205736623034e-001 -2.5357555610075150e-003
    2.3251530544030254e-003 2.5552149564807137e-003
    9.9999403225209282e-001</data></R2>
```

Figure 19: Right Rotation Matrix



```
<P2 type_id="opencv-matrix">
  <rows>3</rows>
  <cols>4</cols>
  <dt>d</dt>
  <data>
    5.6446761987656009e+002 0. 3.3288304138183594e+002
    -4.4441686334611413e+003 0. 5.6446761987656009e+002
    2.3373302268981934e+002 0. 0. 0. 1. 0.</data></P2>
```

Figure 20: Right Projection Matrix



```
<Q type_id="opencv-matrix">
  <rows>4</rows>
  <cols>4</cols>
  <dt>d</dt>
  <data>
    1. 0. 0. -2.9614990234375000e+002 0. 1. 0. -2.3373302268981934e+002
    0. 0. 0. 5.6446761987656009e+002 0. 0. 1.2701309658381479e-001
    4.6655897364711070e+000</data></Q>
```

Figure 21: Re-projection Matrix

Stereo Correspondence

Function Descriptions

InitUndistortRectifyMap()

This function computes the transformation map for the image, as if the image is taken by a new camera which is rectified and without distortion. The camera intrinsic matrix is actually P1 and P2 calculated by *StereoRectify()*, and the function also takes in distortion and rotation matrices. The transformation maps computed are to be used by the function *remap()* to map the points from one image to another, which is described in next paragraph. In this function, the transformation for each pixel (u, v) from the rectified image to the original raw image is calculated. The formula OpenCV used is given in the next page. The maps for x and y values are stored in separate maps, and this function needs to be called twice after calibration, each for a camera.

Equation 25: Functions to Calculate Transformation Maps in OpenCV

$$
\begin{aligned}
&x \leftarrow (u - c'_x)/f'_x \\
&y \leftarrow (v - c'_y)/f'_y \\
&[X\,Y\,W]^T \leftarrow R^{-1} * [x\,y\,1]^T \\
&x' \leftarrow X/W \\
&y' \leftarrow Y/W \\
&x" \leftarrow x'(1 + k_1 r^2 + k_2 r^4 + k_3 r^6) + 2p_1 x' y' + p_2(r^2 + 2x'^2) \\
&y" \leftarrow y'(1 + k_1 r^2 + k_2 r^4 + k_3 r^6) + p_1(r^2 + 2y'^2) + 2p_2 x' y' \\
&map_x(u,v) \leftarrow x" f_x + c_x \\
&map_y(u,v) \leftarrow y" f_y + c_y
\end{aligned}
$$

Remap()

This function uses the mapping criteria calculated by *initUndistortRectifyMap()* for both x and y axes to generate the mapped images from undistort and rectified image to the original image plane. An interpolation technique is used to map non-corresponding pixels. For a faster performance, I used fixed-point representation of maps instead of floating-points. The concept of this function is given below.

Equation 26: Mapping Each Point Using Transformation Maps

$$dst(x,y) = src(map_x(x,y), map_y(x,y))$$

StereoBM

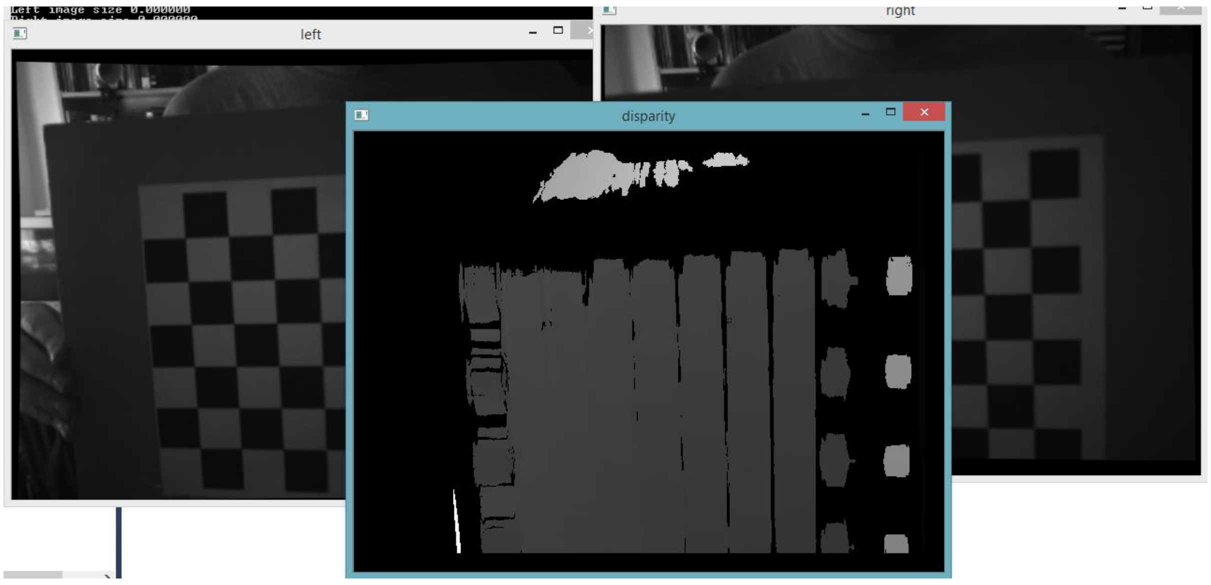
This is a class to perform stereo matching using block matching algorithm. The major function wrapped in the class is *FindStereoCorrespondenceBM()*, which takes in the remapped left and right images and computes the disparity map. In the class constructor, the search range of disparity should be pre-set, that is for each pixel the algorithm will search for the most suitable disparity from default 0 to the number pre-set. Another search criterion that needs to be input to the algorithm is the window size for search, which determines the block size to be compared by the algorithm. Since the block centre is at current pixel, the block size should be set as an odd number. In my program, number of disparities is set as the width of image divided by 8 plus 15 and bitwise and (&) with 16, in order to make it divisible by 16 to enable internal efficiency[3]. And the window size is set to 9. Other numbers are also assigned according to sample from OpenCV and suggestions from other sources [19].

[3] In the functions of *StereoBM*, a lot of Streaming SIMD Extensions (SSE2) instructions are used. Input divisible by 16 is in good alignment with memory allocation in this case [1].

There is also another matching algorithm called semi-global block matching, that is *StereoSGBM*. Since this algorithm is usually slower and gives more noisy maps, I still stuck to *StereoBM*. However, the basic concept to calculate disparity map is the same: For each pixel, find the disparity (horizontal shift) that minimizes the sum of absolute differences (SAD) of pixel intensities within the matching window of left and right images. The functions are parallelized with TBB library to split image into sub-segments instead of accessing individual pixels, and search within multiple sub-segments can be conducted simultaneously to boost the performance.

Testing and Results

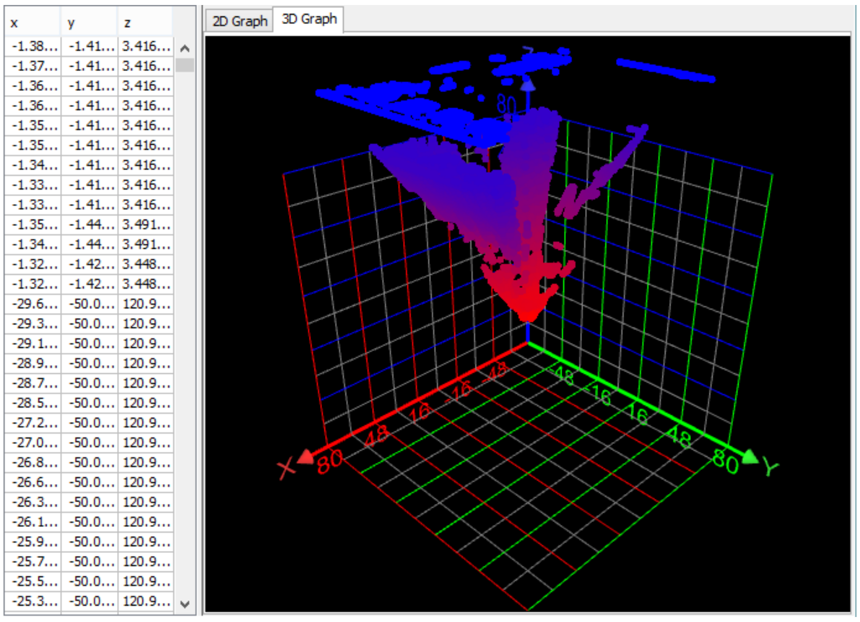
The program first reads in the images and the intrinsic and extrinsic matrices given in calibration project, and it computes and displays the disparity map. The input image pair of this project can be any images taken by the calibrated cameras. Since the calibrated cameras are inaccessible in the last project, here I randomly picked an image pair from the calibration process. The disparity map here is different from the one displayed in camera calibration project, because different argument values were used. Real experiments should be conducted to test the calibration result and decide on the optimised argument values for the functions.



```
Left image size 0.000000
```

Figure 22: Computing and Displaying Disparity Map

It then stores the disparity map and the point cloud. The point cloud generated can be viewed by handy 3D viewers, like Graphing Calculator 3D.



x	y	z
-1.38...	-1.41...	3.416...
-1.37...	-1.41...	3.416...
-1.36...	-1.41...	3.416...
-1.36...	-1.41...	3.416...
-1.35...	-1.41...	3.416...
-1.35...	-1.41...	3.416...
-1.34...	-1.41...	3.416...
-1.33...	-1.41...	3.416...
-1.33...	-1.41...	3.416...
-1.35...	-1.44...	3.491...
-1.34...	-1.44...	3.491...
-1.32...	-1.42...	3.448...
-1.32...	-1.42...	3.448...
-29.6...	-50.0...	120.9...
-29.3...	-50.0...	120.9...
-29.1...	-50.0...	120.9...
-28.9...	-50.0...	120.9...
-28.7...	-50.0...	120.9...
-28.5...	-50.0...	120.9...
-27.2...	-50.0...	120.9...
-27.0...	-50.0...	120.9...
-26.8...	-50.0...	120.9...
-26.6...	-50.0...	120.9...
-26.3...	-50.0...	120.9...
-26.1...	-50.0...	120.9...
-25.9...	-50.0...	120.9...
-25.7...	-50.0...	120.9...
-25.5...	-50.0...	120.9...
-25.3...	-50.0...	120.9...

Figure 23: Point Cloud View in Graphing Calculator 3D

3D Reconstruction

Function Descriptions

ReprojectImageTo3D()

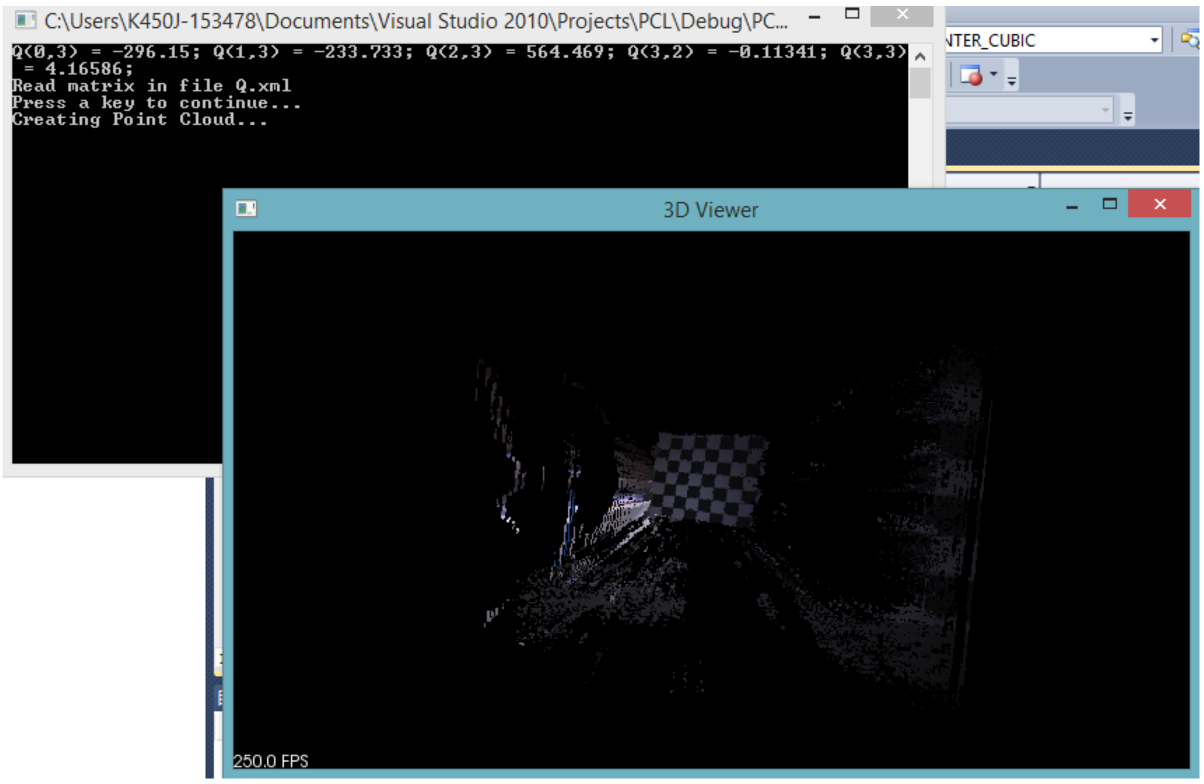
With the disparity map computed in last step, and the re-projection matrix Q during camera calibration, this function is able to calculate and return a 3-channel image with world coordinates for each point. The expression describing the re-projection process is again given below.

Equation 27: Usage of Matrix Q

$$[X\ Y\ Z\ W]^T = \mathtt{Q} * [x\ y\ \mathtt{disparity}(x,y)\ 1]^T$$
$$\mathtt{\_3dImage}(x,y) = (X/W,\ Y/W,\ Z/W)$$

PointCloud

To generate clearer reconstructed 3D model, I decided to exploit Point Cloud Library (PCL). With this additional library, the program is able to generate 3D model that consists of even more points with colour information captured from original image.



```
C:\Users\K450J-153478\Documents\Visual Studio 2010\Projects\PCL\Debug\PC...
Q(0,3) = -296.15; Q(1,3) = -233.733; Q(2,3) = 564.469; Q(3,2) = -0.11341; Q(3,3)
 = 4.16586;
Read matrix in file Q.xml
Press a key to continue...
Creating Point Cloud...
```

Figure 24: Reconstructed 3D Model View

Rotate the 3D model, multiple depth layers within the point cloud can be clearly seen.

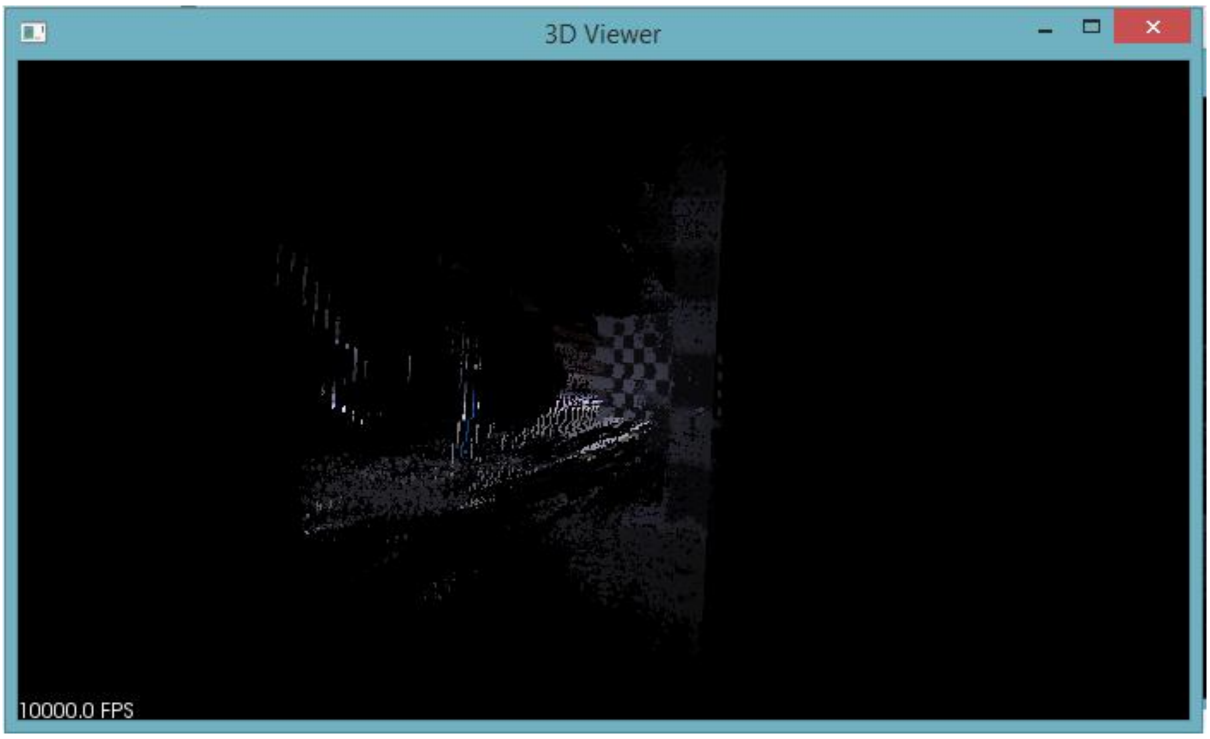


Figure 25: Reconstructed 3D Model Side View

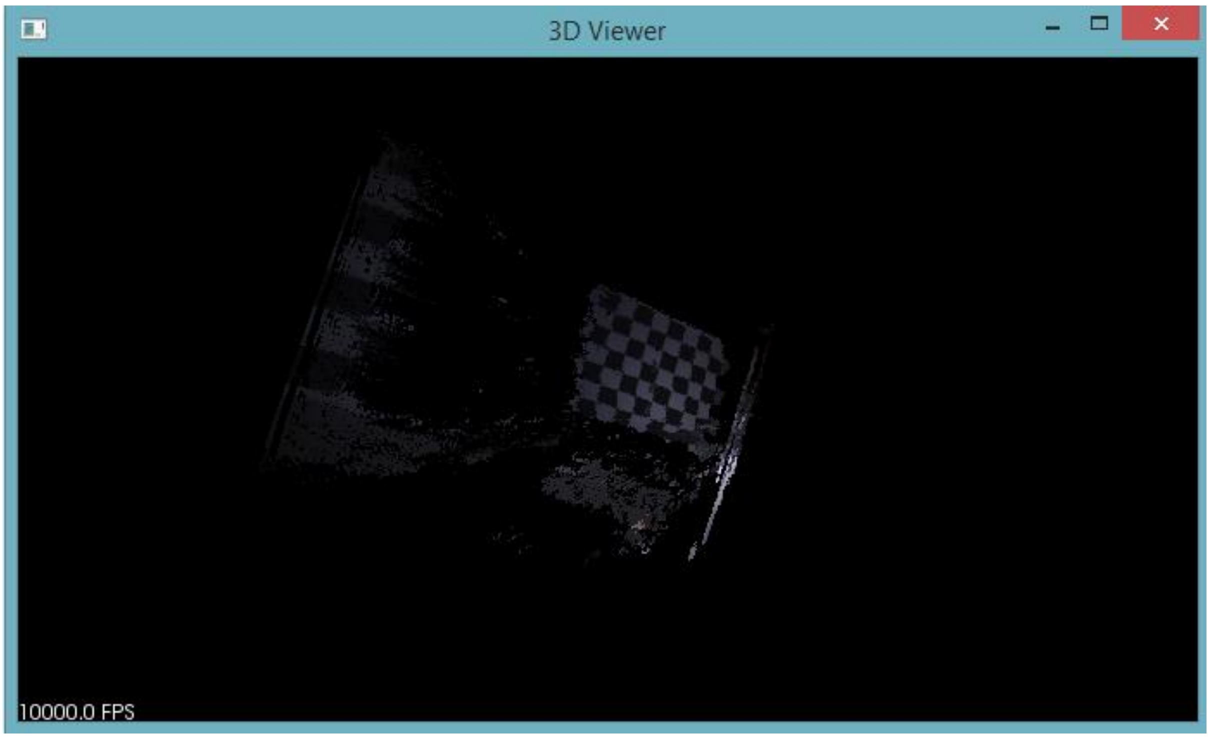


Figure 26: Reconstructed 3D Model Back View

Conclusion and Future Work

Discussions

Re-projection Error

The re-projection error is smaller when fewer sets of images are used to do the calibration. This only means that the results might be very accurate for those particular sets of images and points, while they might not be suitable globally. Images from various angles, position, and bigger set of parameters might help get better results. More images might also add in more noises, which result in unreliable results.

Disparity Map

When conducting stereo matching, OpenCV provides users with several variables to modify the process of searching and computing the disparity map. However, these variables are not well documented, so there is some potential for the map quality enhancement, if more experiments and researches are conducted to further understand these variables.

Limitations

Algorithms Available for Better Disparity Map

The disparity map generated was not very accurate, though various set of function arguments have been tried. It was found that OpenCV does not use best algorithms for generating disparity maps, while such algorithms certainly exist [1].

Detection Restricted to Object Surface

As discussed in the literature review, the 3D model reconstructed still only captures the information of object surface. For detecting 3D material deformation with interior structure change, additional hardware is needed to capture such changes [9].

Follow-up Work

Actual Experiments

I have visited the lab where there were cameras, and I have watched the setup process of them. Although at that time I was testing the MATLAB program for calibration, the setup and the process of taking pictures are the same for my C++ program. Also, the chessboard pictures are also printed and available to use, so it would be fast and easy to take the pictures by myself and to test the program with my own input images.

This is necessary for result verification, because I have no clue to know the real world distance between objects if I am using the images from online. I have to measure the real world coordinates of the object and compare with the reconstructed 3D model in order to verify the accuracy of the program. On the other hand, some problems might emerge when the environment varies, so it is necessary to test the program using images taken in different conditions.

Subset Deformation Measurement

Since now the 3D coordinates of the object at a given time spot can be obtained by the implemented program, the average 3D deformation through time is possible to be calculated if the target sub-image is specified. This can be done by implementing an additional step to match the original and deformed images using correlation functions introduced in the literature review. These functions are available in OpenCV as well [18].

Project Integration

Since currently the 3 projects are separate, it is inconvenient to run them one by one in order to finish the whole process. Therefore, it is necessary to integrate the project into one project to reduce manual work in the long run.

User Interface Implementation

Currently, all the instructions and executions of the program are through command line, which is not user-friendly. In addition, if arguments of the functions are to be changed, the modification has to be made in the program code itself, which makes it very inconvenient and inflexible. A graphic user interface (GUI) should be implemented to make the program easier to use.

Appendix

A – Solution for Intrinsic Constraints from Matrix B

If consider Equation 14 and the denotation of matrix X in Equation 15, we have the following equation, where λ is any scaler:

Equation 28

$$\begin{bmatrix}\mathbf{h}_1 & \mathbf{h}_2 & \mathbf{h}_3\end{bmatrix} = \lambda \mathbf{A} \begin{bmatrix}\mathbf{r}_1 & \mathbf{r}_2 & \mathbf{t}\end{bmatrix}$$

Since r_1 and r_2 are orthonormal, we have the following intrinsic constraints:

Equation 29

$$\mathbf{h}_1^T \mathbf{A}^{-T} \mathbf{A}^{-1} \mathbf{h}_2 = 0$$
$$\mathbf{h}_1^T \mathbf{A}^{-T} \mathbf{A}^{-1} \mathbf{h}_1 = \mathbf{h}_2^T \mathbf{A}^{-T} \mathbf{A}^{-1} \mathbf{h}_2$$

Since B in Equation 9 is symmetric, it can be defined as a 6D vector.

Equation 30

$$\mathbf{b} = [B_{11}, B_{12}, B_{22}, B_{13}, B_{23}, B_{33}]^T$$

If h_i is the *i*th column vector of matrix H in Equation 14, h_i can be expressed as below.

Equation 31

$$\mathbf{h}_i = [h_{i1}, h_{i2}, h_{i3}]^T$$

Now we have the following equation:

Equation 32

$$\mathbf{h}_i^T \mathbf{B} \mathbf{h}_j = \mathbf{v}_{ij}^T \mathbf{b}$$
$$\mathbf{v}_{ij} = [h_{i1}h_{j1}, h_{i1}h_{j2} + h_{i2}h_{j1}, h_{i2}h_{j2},$$
$$h_{i3}h_{j1} + h_{i1}h_{j3}, h_{i3}h_{j2} + h_{i2}h_{j3}, h_{i3}h_{j3}]^T$$

Then the intrinsic constraints in Equation 29 can be rewritten as:

Equation 33

$$\begin{bmatrix} \mathbf{v}_{12}^T \\ (\mathbf{v}_{11} - \mathbf{v}_{22})^T \end{bmatrix} \mathbf{b} = \mathbf{0}$$

If consider the matrices stacked by many images taken, the constraints become the equation below, to which the solution is finding eigenvector of $V^T V$ with respect to the smallest eigenvalue [16]:

Equation 34

$$\mathbf{V}\mathbf{b} = \mathbf{0}$$

B – Essential and Fundamental Matrices

If consider m_1 and m_2 are the normalized coordinates of the image planes after camera calibration, and *R*, **t** are the transformation matrices between these two planes, we have:

Equation 35

$$\bar{\mathbf{m}}_2^T (\mathbf{t} \times R\bar{\mathbf{m}}_1) = 0.$$

If we then define the following, where **y** is any vector:

Equation 36

$$[\mathbf{t}]_x \mathbf{y} = \mathbf{t} \times \mathbf{y}$$

Then equation 35 can be rewritten as a linear equation, where *E* is called essential matrix:

Equation 37: Essential Matrix

$$\bar{\mathbf{m}}_2^T ([\mathbf{t}]_x R\bar{\mathbf{m}}_1) = \bar{\mathbf{m}}_2^T E\bar{\mathbf{m}}_1 = 0.$$

$$E = [\mathbf{t}]_x R$$

In the case that the cameras are not calibrated, the camera matrices A1 and A2 are needed to transform camera coordinates m_1, m_2 to image coordinates m_1, m_2, so we have:

$$\begin{aligned} \mathbf{m}_1 &= A_1\bar{\mathbf{m}}_1 \\ \mathbf{m}_2 &= A_2\bar{\mathbf{m}}_2 \end{aligned}$$

Then the following functions can be derived:

Equation 38

$$(A_2^{-1}\mathbf{m}_2)^T(\mathbf{t} \times RA_1^{-1}\mathbf{m}_1) = 0$$

$$\mathbf{m}_2^T A_2^{-T}(\mathbf{t} \times RA_1^{-1}\mathbf{m}_1) = 0$$

$$\mathbf{m}_2^T F\mathbf{m}_1 = 0$$

Therefore, the Fundamental matrix is defined as following:

Equation 39: Fundamental Matrix

$$F = A_2{}^{-T} E A_1{}^{-1}$$

As noticeable, both Essential and Fundamental matrices describe the geometric relationship between the two camera coordinates. Essential matrix deals with calibrated cameras, while Fundamental matrix deals with uncalibrated cameras [20].

Bibliography

[1] "Digital image correlation – Universal tools versus custom solutions," *Maintenance Problems,* pp. 19-28, 2010.

[2] P. Bing, "Research on Digital Image Correlation with Its Application in Experimental Machanics," Beijing, 2007.

[3] Jianyong Huang, Xiaochang Pan, Shanshan Li, Xiaoling Peng, Chunyang Xiong, and Jing Fang, "A Digital Volume Correlation Technique for 3-D Deformation Measurements of Soft Gels," *International Journal of Applied Mechanics,* vol. 3, no. 2, pp. 335-354, 2011.

[4] P. E. Anuta, "Spatial registration of multispectral and multitemporal digital imagery using fast Fourier transform techniques," *IEEE Trans. Geosci. Electron,* Vols. GE-8, pp. 353-368, 1970.

[5] M.A. Sutton, J.-J. Orteu, H. W. Schreier, Image Correlation for Shape, Motion and Deformation Measurements, 2009.

[6] Helm JD, Deanner JR, "Off-axis two-dimensional digital image correlation," in *SEM X International Congress & Exposition on Experimental and Applied,* Costa Mesa, 2004.

[7] C. SU, L. ANAND, "A new digital image correlation algorithm for whole field displacement measurement," Massachusetts Institute of Technology, Cambridge, MA 02139, USA, 2003.

[8] Sutton MA， McNeill SR., Helm JD, Chao YJ, Advances in Two-Dimensional and Three-Dimensional Computer Vision, Springer Verlag, 2000.

[9] Bay BK, Smith TS, Fyhrie DP, Saad M, "Digital volume correlation: Three-dimensional strain mapping using X-ray Tomography," *Exp Mech,* vol. 39, no. 3, pp. 217-226, 1999.

[10] M. J. Black, "Introduction of Computer Vision," November 2009. [Online].

[11] S. Mattoccia, "Stereo Vision: Algorithms and Applications," 1 2015. [Online].

[12] S. Birchfield, "An Introduction to Projective Geometry (for computer vision)," 1998. [Online].

[13] D. A. Forsyth, J. Ponce., Computer vision: A modern approach, Pearson Education, 2003.

[14] OpenCV Dev Team, "calib3d. Camera Calibration and 3D Reconstruction," 2016. [Online]. Available: http://docs.opencv.org/2.4/modules/calib3d/doc/calib3d.html.

[15] Radu Bogdan Rusu, Steve Cousins, "3D is here: Point Cloud Library (PCL)," in *IEEE International Conference on Robotics and Automation (ICRA),* Shanghai, 2011.

[16] Z. Zhengyou, "A Flexible New Technique for Camera Calibration," *IEEE Transactions on Pattern Analysis and Machine Intelligence,* pp. 1330-1334, 2000.

[17] C. Loop, Z. Zhang, “Computing Rectifying Homographies for Stereo Vision,” CVPR, 1999.

[18] Gary Bradski, Adrian Kaehler, Learning OpenCV, O'Reilly Media, 2008.

[19] Sang Hwa Lee, Siddharth Sharma, “Real-time disparity estimation algorithm for stereo camera systems,” *IEEE TRANSACTIONS ON CONSUMER ELECTRONICS*, vol. 57, no. 3, pp. 1018-1026, 2011.

[20] S. Birchfield, “Essential and fundamental matrices,” Stanford University, 23 4 1998. [Online]. Available: http://robotics.stanford.edu/.

[21] R. Szeliski, Computer vision: Algorithms and Applications, Springer, 2010.

[22] Y. Ma et al., An Invitation to 3-D Vision, New York: Springer Verlag, 2004.