

**Correlation Model for Images taken from cameras
with overlapping views for Distributed Source
Coding**

A Design Project Report

**Presented to the Engineering Division of the Graduate School
Of Cornell University
in Partial Fulfillment of the Requirements for the Degree of
Master of Engineering (Electrical)**

by

Zheshen Zhuang

Project Advisor: Professor Anna Scaglione

Degree Date: Aug 2006

Abstract

Master of Electrical Engineering Program

Cornell University

Design Project Report

Project Title: Correlation Model for Images taken from Cameras with Overlapping Views for Distributed Source Coding

Author: Zheshen Zhuang

Abstract:

This goal of this project is to derive the correlation between images taken from cameras with overlapping views. This work is useful as the framework for the development of a novel distributed source coding (DSC) scheme for images taken in distributed camera sensor networks. A key obstacle to DSC is that the image correlation is dependent not only the intrinsic parameters, positions and orientation of the cameras, but also on the 3D scene which is often unknown. In the case of parallel cameras, a simple definition of scene depth can be defined and a simple relationship exists between image correlation and depth. Thus, depth models can be developed with various constraints and source coding schemes developed based on these models. A key contribution of this project is in identifying and verifying a rectification step so that images taken from cameras in general orientations can be rectified such that the relationship derived from the simple setup holds.

Project Approved by

Project Advisor: _____

Date: _____

Executive Summary

The eventual goal of this project is to find an optimal distributed source coding scheme for images taken multiple cameras with overlapping views of a common scene.

Slepian and Wolf (1973) shows that it is theoretically possible to separately encode two sources and achieve the same rate as a joint encoder *provided the correlation between the two sources are known*. However, the correlation between images depends on the structure of the 3D scene that it is capturing, and at present, a source coding scheme for such images remain elusive

My chief contributions in this semester-long design project are in relating the image correlation to the cameras' intrinsic parameters, positions and orientation and scene structure. For cameras in general position and orientations, no definition of scene depth exist which lends itself easily for analysis. It is found, however that by rectifying the images, a simple definition of depth and a simple relationship between depth and image correlation exist for the images. Thus this project justify the approach of defining scene models based on the definition of depth described, develop source coding schemes that would work for rectified images taken from general positions or orientations

The following are some of the question addressed in this report

- What do we know about one image from the other images simply from the relative orientation and positions of these cameras?
- Can realistic scene models be derived that would lend itself to the development of successful source coding schemes for images?
- What can be done to simplify these models?

Section I, II, III describes what we know about single camera system and two-camera system and systems with three or more cameras respectively. Section IV discusses how we can estimate the parameters for two camera systems. Section V describes the rectification algorithm and Section VI proposes a definition of scene depth and relates *scene depth* to image correlation for rectified image pairs. Section VII introduces scenes models in the literature based on the definition for scene depth and Section VIII follows with the image correlation model for the simplest of these models.

An experiment is also conducted to demonstrate the estimation techniques in IV, the rectification algorithm in VI, and verify the depth-disparity relationship of VII. The experimental set up and results are reported in Section IX. Section X concludes the report.

I. Single Camera System: Image Formation and Camera Projection Matrix

The simplest and most commonly used camera model is the pinhole camera, as shown in Figure 1. The principal axis is defined as the line passing through the camera centre, C and the centre of the image, p . The image formation is equivalent to a projection of the scene onto the image plane (at focal length f away from C) along lines of projections converge onto the camera centre.

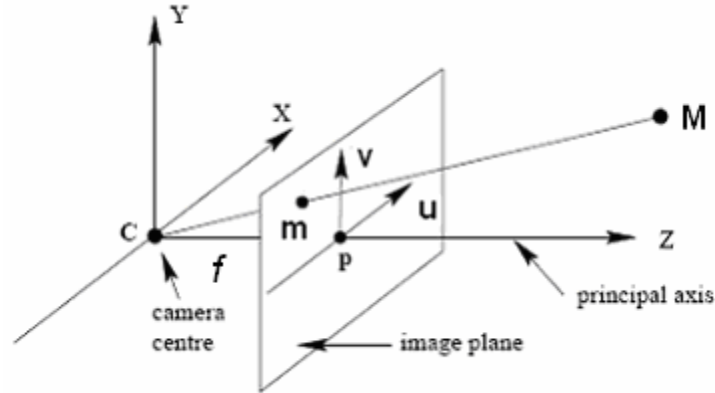


Figure 1. Image formation in a projective camera

Observe that in figure 1, the world frame is centered at C with the z -axis aligned with the principal axis, and the image origin is centered and axes right-handed.

Under the above coordinate frames, the 3-D scene point, M and the corresponding image point, m is related as follows

$$M = \begin{pmatrix} X \\ Y \\ Z \end{pmatrix} \mapsto m = \begin{pmatrix} f \frac{X}{Z} \\ f \frac{Y}{Z} \end{pmatrix}, \text{ where } M \in R^3, m \in R^2 \quad (1)$$

Due to the above nonlinear relationship, it is easier to work with *homogeneous coordinates*.

The equivalent m and M in homogeneous coordinates, denoted by \tilde{m} and \tilde{M} respectively, are given as

$$m = \begin{pmatrix} u \\ v \end{pmatrix} \in R^2 \leftrightarrow \tilde{m} = \lambda \begin{pmatrix} u \\ v \\ 1 \end{pmatrix} \in P^2 \quad \lambda \text{ --any non-zero scalar} \quad (2)$$

$$M = \begin{pmatrix} x \\ y \\ z \end{pmatrix} \in R^3 \leftrightarrow \tilde{M} = s \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} \in P^3, \quad s \text{--any non-zero scalar} \quad (3)$$

Note: In homogeneous coordinates, \tilde{M} with different values of λ is equivalent to the same point.

With homogeneous coordinates, we can simplify (1) as

$$\tilde{m} = \begin{pmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \tilde{M} \quad (4)$$

Note that ‘=’ signs mean projectively equivalent.

Image feature are often available only in its pixel positions where the convention is to have origin is in the top-left corner with the y-coordinates increasing in the downward direction (left-hand axes) as opposed to (u,v) frame in figure 1.

If \tilde{m} is in pixel coordinates, (4) becomes

$$\tilde{m} = K \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \tilde{M}, \quad K = \begin{pmatrix} f\alpha & \gamma & u_0 \\ 0 & f\beta & v_0 \\ 0 & 0 & 1 \end{pmatrix} \quad (5)$$

where

α, β are the normalizing factors from (u,v) coordinates to pixel coordinates.
(Note: $\alpha = \beta$ if the pixels are square.)

u_0, v_0 are the pixel coordinates of the intersection of the principal axis with the image plane.

γ is the skew factor between the two image coordinate axes. (Note: $\gamma = 0$ if the pixel is square/rectangular)

K is known as the camera matrix and capture all the intrinsic camera parameters

(5) can be further generalized for any arbitrary world reference frame by a 3x3 rotation matrix, R, and 3x1 translation vector as follows.

$$\tilde{m} = P\tilde{M}, \quad \text{where } P = K \begin{pmatrix} R & t \end{pmatrix} \quad (6)$$

The 3×4 matrix P is known as the *camera projection matrix* and captures both the intrinsic and *extrinsic parameters* (R, t) of the cameras

Note: P can be uniquely decomposed to $K(R t)$ by QR factorization.

By taking cross product with \tilde{m} on both sides, (6) can also be expressed as

$$\tilde{m} \times P\tilde{M} = 0 \quad (7)$$

Since all the lines of projection passes through the camera centre, C ,

$$\begin{aligned} P\tilde{C} &= 0 \\ R\tilde{C} + t &= 0 \\ t &= -R\tilde{C} \end{aligned} \quad (8)$$

Thus the camera projection matrix can also be written as

$$P = KR(I_{3 \times 3} - C) \quad (9)$$

II. Two Cameras System: Fundamental Matrix

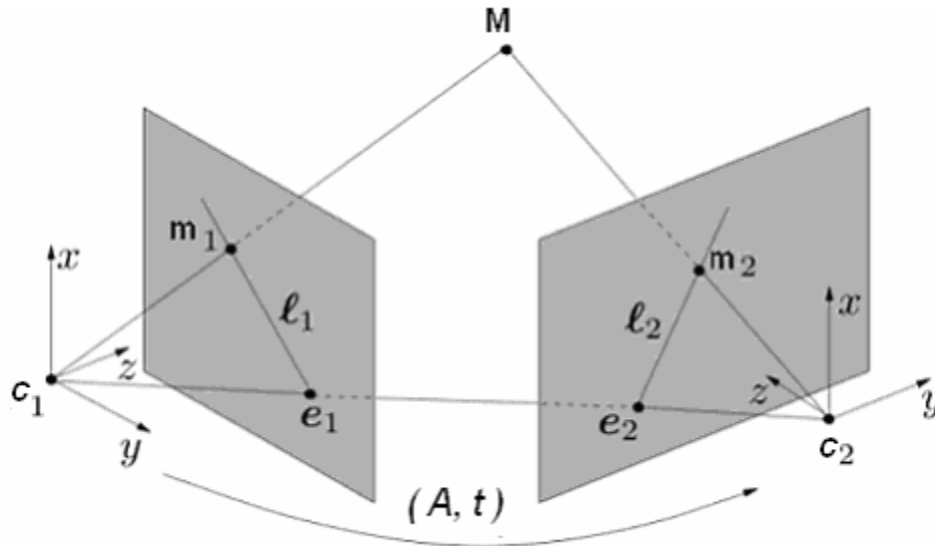


Figure 2. Epipolar geometry

This section discusses what we know about correlation of two views *without knowledge of scene content*. In figure 2, we see that the 3-D point M that projects onto m_1 in image 1 can reside anywhere along the optical ray/line of projection of m_1 . Thus the corresponding point in image 2, m_2 , can lie anywhere on line l_2 . Similarly, without knowledge of the scene, a point, m_2 , in image 2 correspond to a line, l_1 , in image 1,

l_1 and l_2 are known as the epipolar lines of m_1 and m_2 respectively. All the epipolar lines end at a common point called the epipole (e_1, e_2), which is the projection of the conjugate camera centre.

Thus, *without prior scene knowledge*, given full knowledge of one image, the intrinsic parameters of both cameras and their relative position and orientation, *the ambiguity in the corresponding points in the 2nd image is reduced to along the epipolar lines.*

The rest of this section shows that a linear mapping exists between a point and its corresponding epipolar line in homogeneous coordinates i.e.

$$\begin{aligned}\tilde{m}_1 &\mapsto l_2 : l_2 = F\tilde{m}_1 \\ \tilde{m}_2 &\mapsto l_1 : l_1 = \tilde{m}_2^T F\end{aligned}$$

where F is a 3×3 matrix known as the *fundamental matrix*. The relationship between F and the camera project matrices of the two cameras will also be derived.

* * *

If we take the first camera reference frame to be the world reference frame, the camera projection matrices of the two cameras can be expressed as follows:

$$P_1 = K_1(I_{3 \times 3} \ 0) \quad P_2 = K_2(R \ t) = (Q_2 \ q_2) \quad (10)$$

$$s_1 \cdot \tilde{m}_1 = P_1 \tilde{M} = K_1 \tilde{M} \quad s_2 \cdot \tilde{m}_2 = P_2 \tilde{M} = K_2 R \tilde{M} + t_2 \quad (11)$$

Combining the two above equations, we get

$$s_2 \cdot \tilde{m}_2 = s_1 K_2 R K_1^{-1} \tilde{m}_1 + K_2 t \quad (12)$$

To remove the scalar ambiguity s_1 and s_2 we first cross product (12) with $K_2 t$ on both sides.

We can define the skew symmetric matrix of any 3×1 vector q such that

$$q \times y \Leftrightarrow [q]_x \cdot y \quad \text{where } q = \begin{pmatrix} a \\ b \\ c \end{pmatrix}, \quad [q]_x = \begin{pmatrix} 0 & -c & b \\ c & 0 & -a \\ -b & a & 0 \end{pmatrix} \quad (13)$$

The cross product operation is equivalent to pre-multiplying (12) by $[K_2 t]_x$

$$s_2 \cdot [K_2 t]_x \tilde{m}_2 = s_1 [K_2 t]_x K_2 R K_1^{-1} \tilde{m}_1 \quad (14)$$

Pre-multiply (14) both sides by m_2^T gives 0 on the left side.

$$0 = s_1 \tilde{m}_2^T [K_2 t]_x K_2 R K_1^{-1} \tilde{m}_1 \quad (15)$$

We define the *fundamental matrix* F as follows

$$F = [K_2 t]_x K_2 R K_1^{-1} \text{ such that } \tilde{m}_2^T F \tilde{m}_1 = 0 \quad (16)$$

As the epipoles are the projections of the conjugate camera centres

$$\begin{aligned} \tilde{e}_1 &= P_1 \tilde{C}_2 = P_1 \begin{pmatrix} -R^{-1}t \\ 1 \end{pmatrix} = K_1 R^{-1}t \\ \tilde{e}_2 &= P_2 \tilde{C}_1 = P_2 \begin{pmatrix} 0_{1 \times 3} \\ 1 \end{pmatrix} = K_2 t \end{aligned} \quad (17)$$

(16) can also be expressed as

$$F = [\tilde{e}_2]_x K_2 R K_1^{-1} = K_2^{-T} [t]_x R K_1^{-1} = K_2^{-T} R [R^T t]_x K_1^{-1} = K_2^{-T} R K_1^T [\tilde{e}_1]_x \quad (18)$$

Note that a 2-D line with equation $ax + by + c = 0$ can be expressed in homogeneous coordinates as

$$l = \begin{pmatrix} a \\ b \\ c \end{pmatrix} \text{ such that } l^T \tilde{x} = 0 \text{ for any point } x \text{ on the line} \quad (19)$$

The equation of a line in this form can be found as simply the cross product of two points, x_1 and x_2 residing on the line

$$l = \tilde{x}_1 \times \tilde{x}_2 \quad (20)$$

F maps image points onto their corresponding epipolar lines as follows

$$\begin{aligned} F \tilde{m}_1 &= [\tilde{e}_2]_x K_2 R K_1^{-1} \tilde{m}_1 = [\tilde{e}_2]_x \tilde{m}_2 = \tilde{e}_2 \times \tilde{m}_2 = l_2 \\ \tilde{m}_2^T F &= \tilde{m}_2^T K_2^{-T} R K_1^T [\tilde{e}_1]_x = \tilde{m}_1^T [\tilde{e}_1]_x = \tilde{m}_1 \times \tilde{e}_1 = l_1 \end{aligned} \quad (21)$$

The epipoles (e_1, e_2) are the right and left null vectors of F respectively as

$$\begin{aligned} F \tilde{e}_1 &= K_2^{-T} R K_1^T [\tilde{e}_1]_x \tilde{e}_1 = 0 \\ \tilde{e}_2^T F &= \tilde{e}_2^T [\tilde{e}_2]_x K_2 R K_1^{-1} = 0 \end{aligned} \quad (22)$$

III Brief Discussion of Systems with Three or more Cameras.

A $3 \times 3 \times 3$ matrix, called the *trifocal tensor* for the three cameras system exist which is analogous to the fundamental matrix for the two cameras system. It “encapsulates all the (projective) geometric relations between three views that are independent of scene structure.” (Hartley & Zisserman, 2000)

The trifocal tensor captures the following relationships between lines and points in the three images taken from such a system

Given the image points in two of the images corresponding to a feature point in the scene, the trifocal tensor gives the corresponding image point in the 3rd image.

$$\tau : \tilde{m}_i, \tilde{m}_j \mapsto \tilde{m}_k, \quad i \neq j \neq k, \quad i, j, k \in \{1, 2, 3\} \quad (23)$$

Given a line in two of the images corresponding to a straight edge in the scene, the trifocal tensor gives the corresponding line in the 3rd image.

$$\tau : l_i, l_j \mapsto l_k, \quad i \neq j \neq k, \quad i, j, k \in \{1, 2, 3\} \quad (24)$$

Given a image point in one image and a line where the corresponding point resides in the 2nd image, the trifocal tensor gives the corresponding point in the 3rd image.

$$\tau : x_i, l_j \mapsto x_k, \quad i \neq j \neq k, \quad i, j, k \in \{1, 2, 3\} \quad (25)$$

It is also possible to consider quadifocal tensors and define the geometrical relationships for four cameras system and so on. The geometrical relationships and tensors for these systems can be similarly derived from the camera projective matrices discussed in Section I. (Hartley & Zisserman, 2000)

The author would like to highlight, however, that in the absence of occlusion, neglecting resampling error and assuming pixels corresponding to the point in the scene have the same pixel value, *we can theoretically perfectly reconstruct the regions in the third image visible to both of the first two images* using the trifocal tensor.

Thus the greatest challenge and gain lies in the lossy distributed source coding of images from the two camera system given information of only one image. The rest of this report shall thus focus on simplifying the image correlation for a two camera system through rectification.

IV. Estimating the Camera Projection Matrix P_1 , P_2 and Scene Reconstruction

In this section we shall find the least-square linear estimate of P_1 , P_2 and F . We will also describe a linear method for estimating the 3D coordinates of the scene point from a pair of image points $(\tilde{m}_{1i}, \tilde{m}_{2i})$ and estimate of P_1 and P_2

Recall that the least-squares solution to a homogeneous system of linear equations

$$Ax = 0 \quad \text{where } A \in R^{m \times n} \quad m \geq n \quad (26)$$

seeks to minimize $\|Ax\|$ subject to $\|x\| = 1$ and is

$$\hat{x} = V(\text{last column}) \quad \text{where } A = UDV^T \quad (\text{SVD of } A) \quad (27)$$

Observe that P_i is a 3×4 matrix. Thus P has 11 degrees of freedom as it is known only up to a scale ambiguity.

Each correspondence pair $\tilde{M}_i \leftrightarrow \tilde{m}_i$, where $\tilde{m}_i = P\tilde{M}_i$ can be expressed as a homogeneous system of linear equations:

$$\underbrace{\begin{pmatrix} 0^T & -w_i\tilde{M}_i^T & w_i\tilde{M}_i^T \\ w_i\tilde{M}_i^T & 0^T & -x_i\tilde{M}_i^T \\ -y_i\tilde{M}_i^T & x_i\tilde{M}_i^T & 0^T \end{pmatrix}}_{3 \times 12} \begin{pmatrix} P^1 \\ P^2 \\ P^3 \end{pmatrix} = 0 \quad (28)$$

where

$$(P^j)^T \text{ are the } i\text{-th row of } P \text{ and } \tilde{m}_i = \begin{pmatrix} x_i \\ y_i \\ w_i \end{pmatrix} \quad (29)$$

However, as the three equations in (16) are linearly dependent, we shall stack up only the first two equations from each correspondence pair to form a $2n \times 12$ constraint matrix A . The projection matrix, P , can be computed by the linear method outlined in (27)

As each correspondence pair contributes 2 constraints, we *need at least 6 control points* whose world coordinates are known to compute the camera projection matrix P .

Having computed P_1 and P_2 , the fundamental matrix, F , can be computed using equation (18)

The least square method described above minimizes algebraic error and not necessarily geometric error. Since \tilde{m} and \tilde{M} are in homogeneous coordinates with a scale ambiguity

(e.g. \tilde{M} and $\lambda\tilde{M}$ corresponds to the same point for any scalar λ), *these coordinates needs to be normalized before applying the linear estimation* so as to obtain a stable solution.

The $\tilde{m}_i (\in P^2)$ are normalized (scaled and shifted) in such that their centroid is at the origin and their mean norm is $\sqrt{2}$

The $\tilde{M}_i (\in P^3)$ are normalized (scaled and shifted) in such that their centroid is at the origin and their mean norm is $\sqrt{3}$.

A more accurate estimate, \hat{P} can be made if one seeks to minimize the geometric error

$$(\hat{x}_i, \hat{P}) = \arg \min \sum_i d(\hat{x}_i, \hat{P}X_i) \quad (30)$$

This method requires that we estimate not only the parameters of P but also the true image points from noisy measurements. However, this method is not linear but iterative and computationally intensive. In the experiment setup described in Section VIII, the linear method is found to be sufficiently accurate. A more detailed discussion and comparison of estimation methods for P is found in Hartley & Zisserman (2000).

While 3D scene reconstruction is not critical to the purposes of this project, the accuracy of the reconstructed scene helps to verify the concepts and equations described in this project. A linear method for triangulating the 3-D position of a scene point from its image pair $(\tilde{m}_1, \tilde{m}_2)$ in two images and the corresponding camera projection matrix estimates P_1 and P_2 is described below

Recall in (6) that

$$\tilde{m} \times P\tilde{M} = 0$$

(6) can be rewritten as

$$\begin{aligned} y_i(p_i^3 X) - w_i(p_i^2 X) &= 0 \\ x_i(p_i^3 X) - w_i(p_i^1 X) &= 0 \\ x_i(p_i^2 X) - y_i(p_i^1 X) &= 0 \end{aligned} \quad (31)$$

where

$$\tilde{m}_i = \begin{pmatrix} x_i \\ y_i \\ w_i \end{pmatrix}, \quad P_i = \begin{pmatrix} p_i^1 \\ p_i^2 \\ p_i^3 \end{pmatrix}$$

Since the equations in (31) are linearly dependent, only the first two equations are used for each P_i

Thus we again have an equation of the form

$$AX = 0 \quad (32)$$

where

$$A = \begin{pmatrix} y_1 p_1^3 - w_1 p_1^2 \\ x_1 p_1^3 - w_1 p_1^1 \\ y_2 p_2^3 - w_2 p_2^2 \\ x_2 p_2^3 - w_2 p_2^1 \end{pmatrix}$$

and X can be solved using (27)

V. Planar Rectification

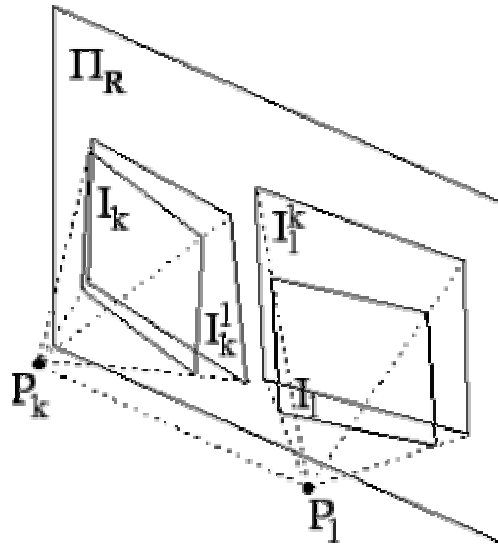


Figure 4. Planar Rectification

Rectification is the process of re-projecting a pair of images taken from general, unconstrained positions such that the output images are equivalent to images taken from a parallel camera setup i.e. the direction of their axes line up. We shall see in the next section how a parallel camera setup aids analysis.

Rectification is relatively straightforward when the camera projection matrices are known. The idea is to apply a pair of *rectification matrices* to the re-project the image pair such that the cameras are “effectively” rotated so their principal axes become parallel. Once these invertible rectification matrices are found, they can be used to rectify any images from the same two cameras provided the cameras position and intrinsic parameter do not changed. The rectified image can be transmitted if desired, and the inverse of the matrices applied to its coordinates to recover the original image. Note that this process is not lossless due to resampling error.

The rest of this section deals with how to find the rectification matrices, from the camera projection matrices. The algorithm below is modified from that proposed by Fusiello, A., Trucco, E., and Verri, A (n.d.)

* * *

Suppose we start off with the following perspective projection matrices:

$$P_{O1} = K_1 R_{O1} (I - c_1) \quad P_{O2} = K_2 R_{O2} (I - c_2) \quad (33)$$

Note: The 1st, 2nd and 3rd row vectors of each rotation matrix correspond to direction of the x-axis, y-axis and z-axis of the camera respectively with respect to the world frame.

We would like the new PPMs of our rectified images to have same rotation matrix, R_{new} i.e. their axes line up.

$$P_{N1} = K_1 R_N (I - c_1) \quad P_{N2} = K_2 R_N (I - c_2) \quad (34)$$

Let $r_{i,X}$ be the row vectors of R_X i.e.

$$R_{O1} = \begin{pmatrix} r_{O1}^{1T} \\ r_{O1}^{2T} \\ r_{O1}^{3T} \end{pmatrix} \quad R_{O2} = \begin{pmatrix} r_{O2}^{1T} \\ r_{O2}^{2T} \\ r_{O2}^{3T} \end{pmatrix} \quad R_N = \begin{pmatrix} r_N^{1T} \\ r_N^{2T} \\ r_N^{3T} \end{pmatrix} \quad (35)$$

The new X axis has to be parallel to the baseline. This is the condition for the epipolar lines to line up horizontally.

$$r_N^1 = \frac{(c_1 - c_2)}{\|c_1 - c_2\|} \quad (36)$$

The rectification matrices ought to minimize the amount of rotation between the old and new axes to minimize re-sampling errors and rectified image size. Thus the new Z axis is chosen as the bilinear vector of the components of the old Z axes orthogonal to the base line

$$r_{3,new} = \frac{\hat{k}_1 + \hat{k}_2}{\sqrt{2}} \quad (37)$$

where

$$\hat{k}_1 = \frac{r_{O1}^3 - r_{O1}^3 \cdot r_N^1}{\|r_{O1}^3 - r_{O1}^3 \cdot r_N^1\|} \quad \hat{k}_2 = \frac{r_{O2}^3 - r_{O2}^3 \cdot r_N^1}{\|r_{O2}^3 - r_{O2}^3 \cdot r_N^1\|}$$

The new Y axis is simply given by

$$r_N^2 = r_N^3 \times r_N^1 \quad (38)$$

Having computed the desired R_N , the rectification matrices T_1 and T_2 should be such that

$$P_{N1} = T_1 P_{O1} \quad P_{N2} = T_2 P_{O2} \quad (39)$$

It should be obvious from (33) and (34) that

$$T_1 = K_1 R_N (K_1 R_{O1})^{-1} \quad T_2 = K_2 R_N (K_2 R_{O2})^{-1} \quad (40)$$

Image rectification is impossible when the epipole is resides within or near the image as the rectified image will be very large. Such degenerate cases occur when there is large forward direction between the two cameras or when the angle between principal axes of the two cameras is large.

VI. Relationship between Depth and Disparity for Rectified Images

The previous section discussed how images taken from a pair of cameras in general positions and orientation can be rectified such that their setup is equivalent to the special case of a parallel camera setup.

In this special case, the epipolar lines run parallel and horizontally along both images i.e. pixels corresponding to the same points in scene has the same y-coordinates in both images. *Thus ambiguity in matching image points is reduced to their x-coordinates.* We shall define this difference as the *disparity*. While rectification is critical in simplifying correspondence mapping in the field of computer vision, it also *allows for a simple definition of scene depth.*

Under the parallel camera setup, we can define a plane containing both camera centre C_1 and C_2 and normal to the principal axes of both cameras. *The scene depth of a 3D point, M , can be defined as the perpendicular distance of that point from that plane.* (Faugeras, 2004)

For the rest of this section, we shall derive a simple relationship between disparity and scene depth which allows us to relate our scene model in Section VII to image correlation model in Section VIII.

* * *

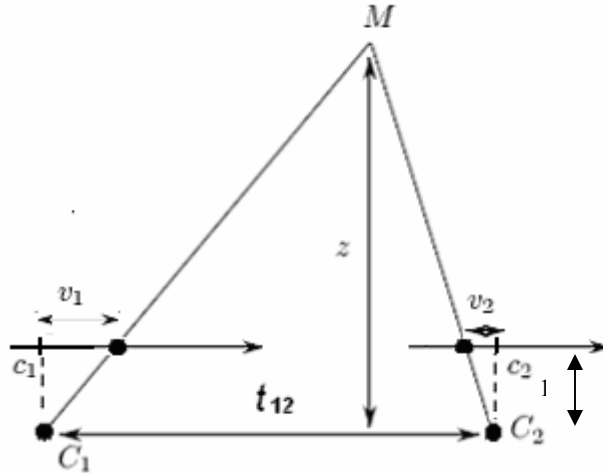


Figure 3. Relationship between depth and disparity

Consider Figure 3. The relationship between disparity and depth of a 3-D point M , measured from the plane is given as

$$d = v_2 - v_1 = \frac{\|c_1 - c_2\|}{z} \quad (41)$$

Note however that v_1 and v_2 are not in pixel coordinates.

v_1 and v_2 are related to the image point \tilde{m}_1 and \tilde{m}_2 as follows

$$v_1 = \frac{p_1(1)}{p_1(3)} \quad v_2 = \frac{p_2(1)}{p_2(3)} \quad (42)$$

where

$$p_1 = K_1^{-1} \tilde{m}_1 \quad p_2 = K_2^{-1} \tilde{m}_2$$

VII. Scene Models

Now that we have established the relationship between disparity and depth for rectified images, we can model the scene as a function of depth defined in (41). Since the ambiguity occurs only in the x-coordinates, it is natural to start with a depth model which is a function only in x .

Belhumeur (1996) proposes three depth models for 3-D scenes in order of increasing sophistication and realism. The first two models are described here as their complexities are more tractable.

Consider v_1 in the range between $[-a, a]$

Model 1 (Single object with smooth surface)

$$Z(x) = z_0 + \underbrace{\alpha \int_{-a}^x w(x) dx}_{\text{Brownian motion}} \quad (43)$$

where

$$w(x) \sim N(0, \delta(x))$$

Model 2 (Discontinuities due to occlusion and object boundaries modeled)

$$Z(x) = \theta + \alpha \int_{-a}^x w(x) dx + \sum_{j=1}^{N(x)} \phi_j$$

where $\theta \sim N(0, \sigma^2)$

$$w(x) \sim N(0, \delta(x)) \quad (44)$$

$$N(x) \sim P_o(\lambda)$$

$$\phi_j \sim U(-a, a), \quad i.i.d.$$

θ, W, ϕ_j are independent

VIII. Image Correlation Model

In this section, the correlation between the pair of rectified images taken from the two-camera system is first derived. To simplify analysis, one shall consider only the correlation between pixels lying on matching epipolar lines in these images. Since we know that these epipolar lines have the same y-coordinates, this is essentially a 1-D problem.

Pixels corresponding to the same 3-D point often differ in pixel value as a result of quantization error, imperfect optics, noise in the imaging system, illumination, and specular reflection. We shall model these differences as an additive, white Gaussian noise. Let I_1 and I_2 be a row of pixels with the same y-coordinates.

$$I_1(x_1) = I_2(x_2) + w$$

$$I_1(x_1) = I_2(x_1 + d(x_1)) + w$$

$$I_1(x_1) = \delta(u - x_1 - d(x_1))I_2(u) + w$$

$$\tilde{I}_1 = M\tilde{I}_2 + \tilde{w} \quad \text{where} \quad M_{x,u} = \delta(u - x - d(x)) \quad (45)$$

1st and 2nd Order Statistics of M

$$\begin{aligned}
 E\{M_{x,u}\} &= E\{\delta(u-x-d_x)\} \\
 &= \int \delta(u-x-d_x) f_{d(x)}(d_x) d(d_x) \\
 &= f_{d(x)}(u-x)
 \end{aligned} \tag{46}$$

$$\begin{aligned}
 R_{u,x,u',x'} &= E\{M_{x,u}M_{x',u'}\} \\
 &= \iint \delta(u-x-d_x)\delta(u'-x'-d_{x'}) f_{d(x)}(d_x, d_{x'}) dd_x dd_{x'} \\
 &= f_{d(x),d(x')}(u-x, u'-x')
 \end{aligned} \tag{47}$$

The depth model $Z(x)$ given in VII allows us to find the 1st and 2nd-order distribution of the disparity, $f_d(d_x)$ and $f_d(d_x, d_{x'})$ in (46), (47)

For the 1st model where the depth function is modeled as a Brownian motion.

1st order pdf of $Z(x)$

$$f_{Z(x)} = \frac{1}{\sqrt{2\pi\alpha(x+a)}} e^{-\frac{(z(x)-z_0)^2}{2\alpha(x+a)}}, \quad x \geq -a \tag{48}$$

2nd order pdf of $Z(x)$

$$f_{Z(x)Z(x')} = \frac{1}{2\pi\alpha\sqrt{(x+a)(x'-x)}} \exp\left\{-\frac{1}{2}\left[\frac{(z(x)-z_0)^2}{2\alpha(x+a)} + \frac{(z(x')-z(x))^2}{2\alpha(x'-x)}\right]\right\}, \quad x' > x \tag{49}$$

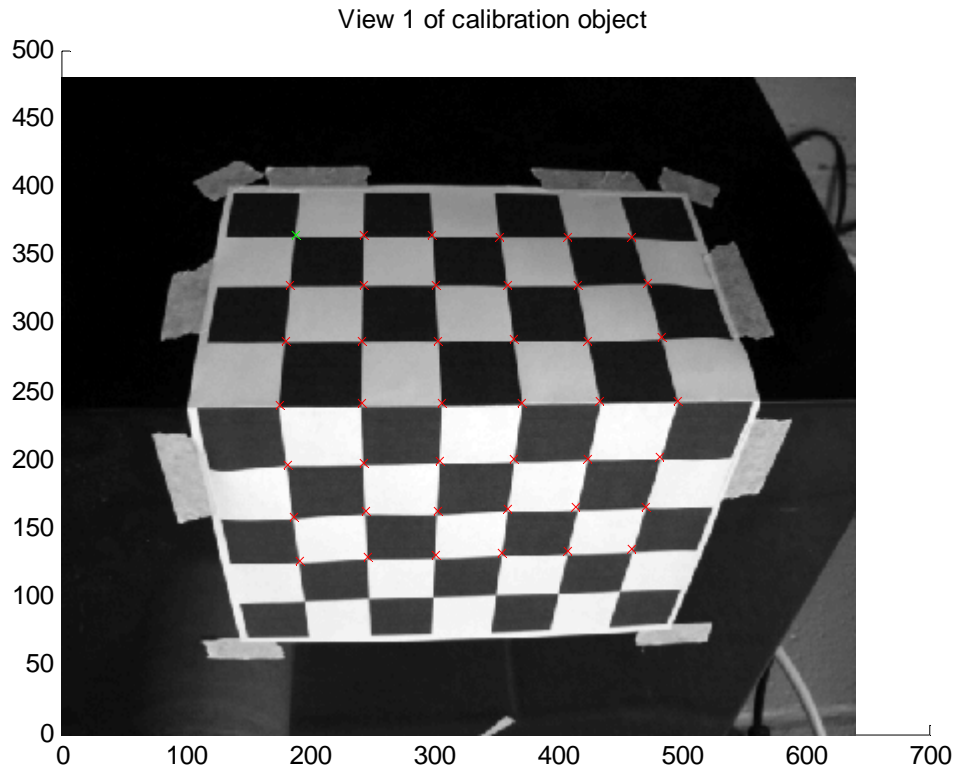
The 1st and 2nd order pdf of $d(x)$ can be calculated (48) and (49) using (41)

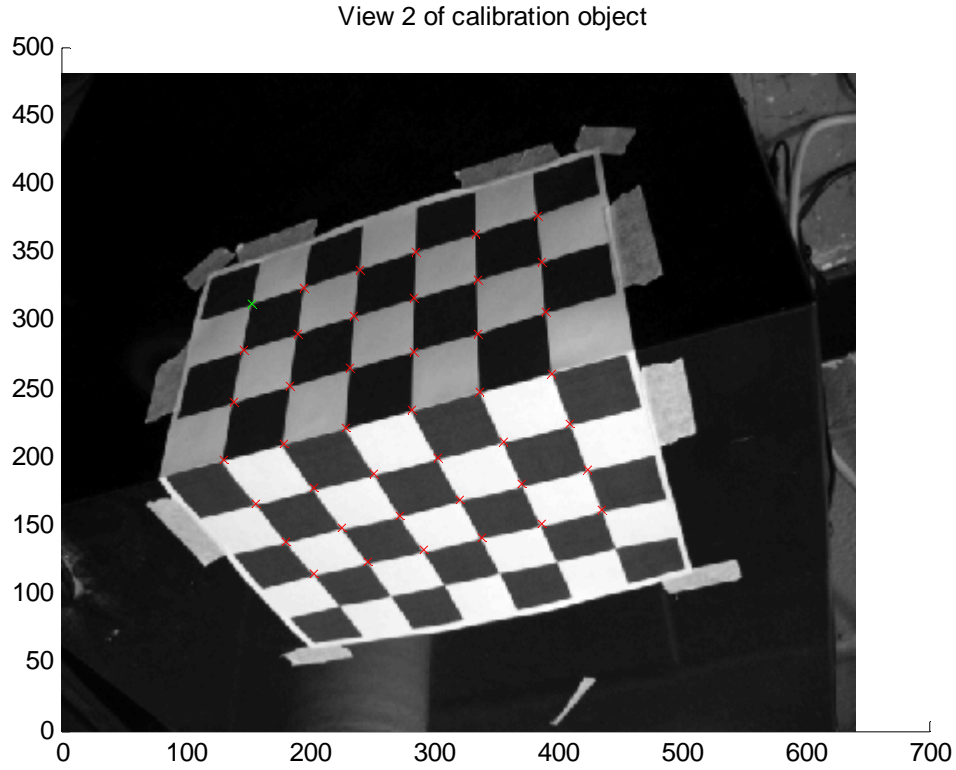
IX. Experimental Setup and Results

This section describes the experimental setup and results that verify the earlier sections.

The following are two images of a calibration object taken from two different positions using two separate Sony Cyber-shot DSC-P200 cameras. The images are actually vertically inverted, so that axes are right-handed.

The calibration object has square grids of length 30mm on two perpendicular planes. The crosses are 41 image points whose pixel coordinates are measured manually for each image. The green cross is taken the origin of the world frame when computing the camera projection matrices and the square length is taken to be unit length. For example, the four corners of the object has world coordinates $(0, 0, 0)$, $(5, 0, 0)$, $(5, 3, 5)$ and $(0, 3, 5)$





The computed camera projection matrices (using PPM.m) are

$$P_1 = \begin{pmatrix} 37.6082 & -8.9505 & 10.4165 & 123.4976 \\ 1.5031 & -35.4779 & -15.8123 & 238.4794 \\ 0.0044 & -0.0345 & 0.0380 & 0.6535 \end{pmatrix} \quad P_2 = \begin{pmatrix} 26.8255 & -9.5235 & 21.3649 & 105.1648 \\ 4.5372 & -32.1917 & -12.6570 & 213.1201 \\ -0.0116 & -0.0309 & 0.0325 & 0.6822 \end{pmatrix}$$

QR factorization of P_1 and P_2 as in (4) gives the following

$$K_1 = \begin{pmatrix} -704.4777 & 1.6578 & 328.0531 \\ 0 & 716.3520 & 236.8172 \\ 0 & 0 & 1.0000 \end{pmatrix} \quad K_2 = \begin{pmatrix} -700.0429 & -6.3903 & 316.6758 \\ 0 & 711.5969 & 246.7082 \\ 0 & 0 & 1.0000 \end{pmatrix}$$

$$R_1 = \begin{pmatrix} -0.9962 & -0.0668 & 0.0551 \\ 0.0124 & -0.7400 & -0.6724 \\ 0.0857 & -0.6692 & 0.7381 \end{pmatrix} \quad R_2 = \begin{pmatrix} -0.9421 & -0.0010 & -0.3353 \\ 0.2241 & -0.7456 & -0.6276 \\ -0.2494 & -0.6664 & 0.7026 \end{pmatrix}$$

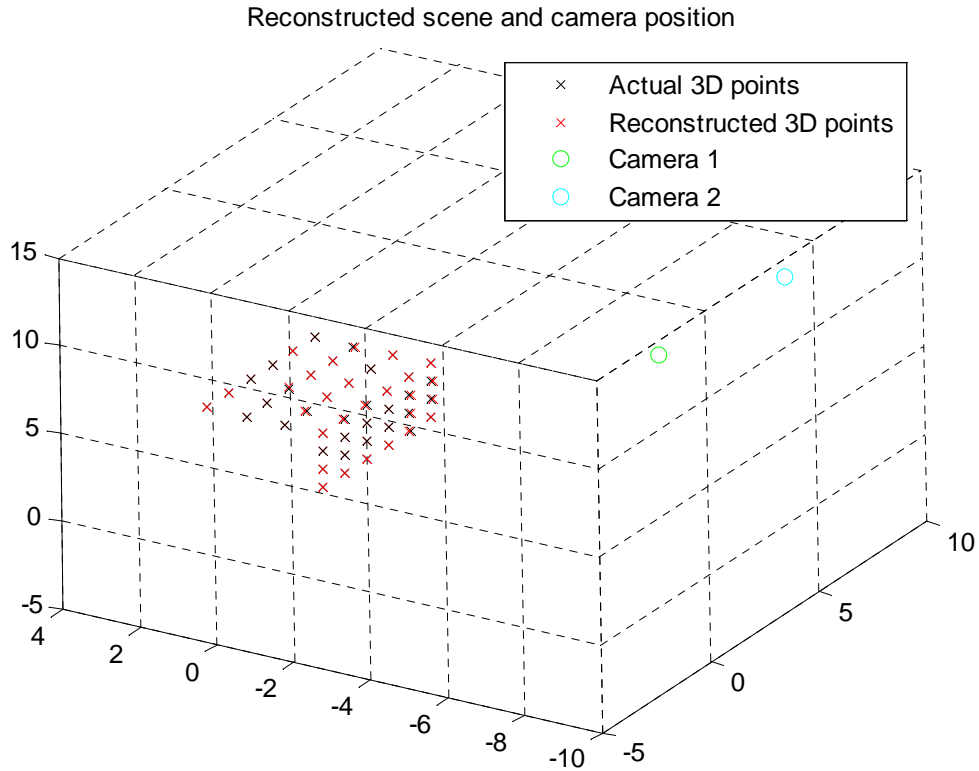
$$t_1 = \begin{pmatrix} 2.5098 \\ 2.2681 \\ 12.6852 \end{pmatrix} \quad c_1 = \begin{pmatrix} 1.3851 \\ 10.3355 \\ -7.9760 \end{pmatrix} \quad t_2 = \begin{pmatrix} 3.4065 \\ 1.3598 \\ 14.7271 \end{pmatrix} \quad c_2 = \begin{pmatrix} 6.5772 \\ 10.8313 \\ -8.3524 \end{pmatrix}$$

F is computed from P_1 and P_2 using (13)

$$F = \begin{pmatrix} -0.0000 & -0.0000 & -0.0048 \\ 0.0000 & -0.0000 & 0.0559 \\ -0.0008 & -0.0423 & 0.9393 \end{pmatrix} \text{ given } |F| = 1$$

We compute $\frac{1}{42} \sum_1^{42} |\tilde{m}_{2,i}^T F \tilde{m}_{1,i}| = 0.0178$ verifying (10) and (13)

The calibration object is reconstructed with the camera positions in the figure below.



Observe that the reconstructed points matches closely with the actual points and the MSE is computed below

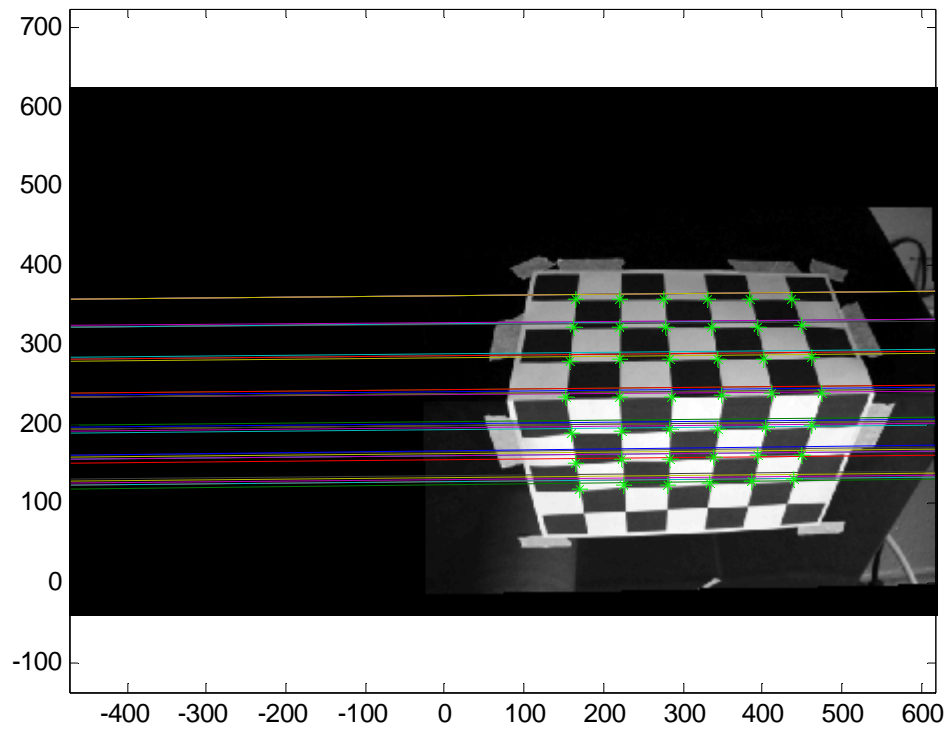
$$MSE = \frac{1}{42} \sum_1^{42} |X_i - \widehat{X}_i| = 0.1027 \text{ units}^2$$

The rectification matrices are found to be

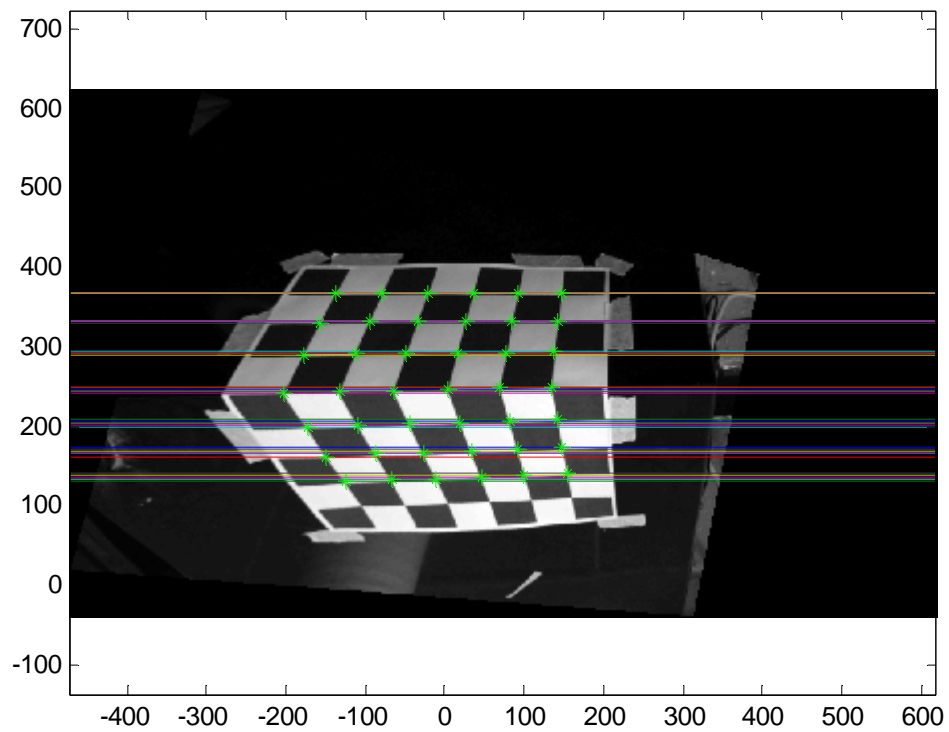
$$T_1 = \begin{pmatrix} 19.6850 & -0.1152 & -497.2827 \\ 0.3881 & 19.4553 & -248.1654 \\ 0.0009 & 0.0002 & 19.0669 \end{pmatrix} \quad T_2 = \begin{pmatrix} 23.1820 & 4.8629 & -7628.8000 \\ -1.8763 & 21.6110 & 339.4400 \\ 0.0109 & 0.00218 & 16.1240 \end{pmatrix}$$

The rectified images are shown below. The epipolar lines in both rectified images are horizontal or nearly so and matches.

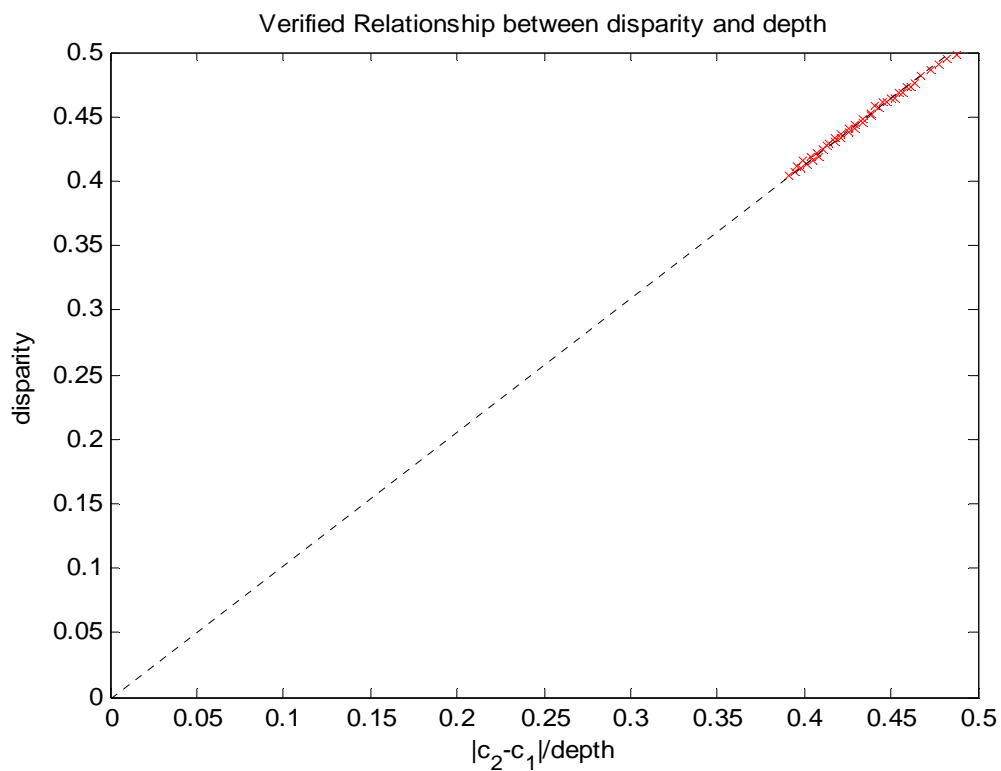
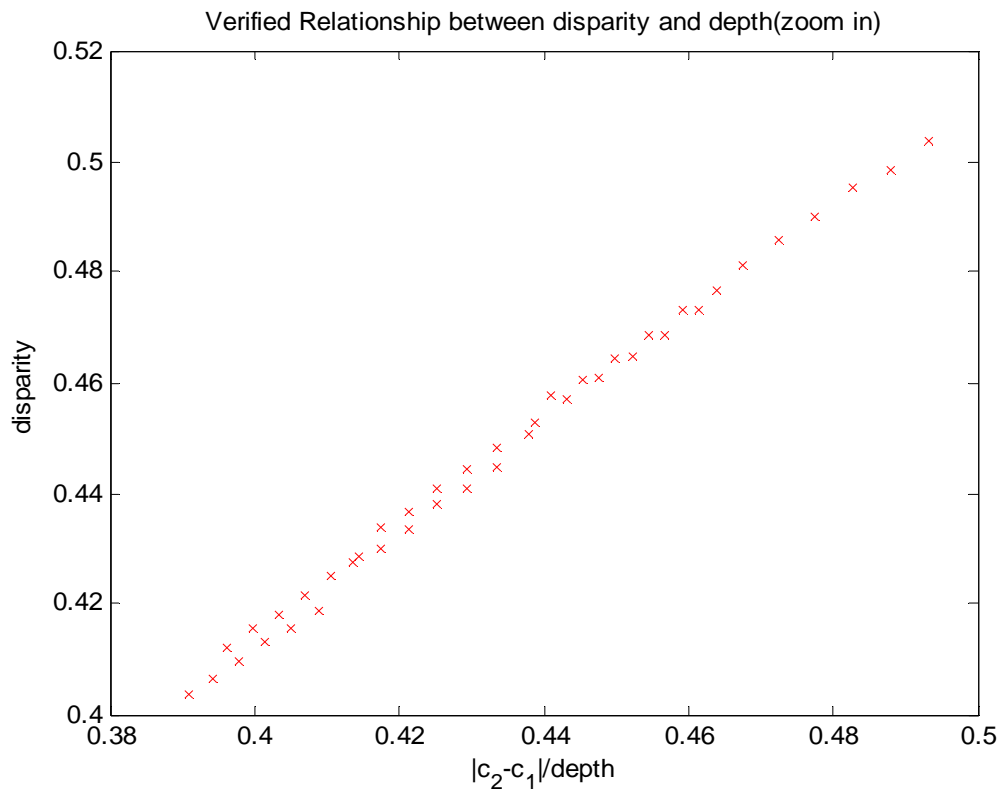
First rectified image



Second rectified image



The disparity and depth of the grid points in the rectified images are computed and plotted below, verifying (42)



X. Conclusion

This project has successfully demonstrated the steps needed to rectify the images from two cameras in general positions such that the following correlation model between the image pairs is as follows:

$$\tilde{I}_1 = M\tilde{I}_2 + \tilde{w} \quad \text{where} \quad M_{x,u} = \delta(u - x - d(x))$$

The 1st and 2nd order statistics of the entries of M may be calculated from scene depth models as demonstrated in Section VII based on the following relationship between disparity and scene depth

$$d(x) = \frac{\|c_1 - c_2\|}{Z(x)}$$

The above relationship, verified for images rectified using the algorithm described in Section IV do not necessarily hold true for rectification methods in general.

The author hopes that the above correlation model would lend itself to the development of novel image source coding schemes

References

- Belhumeur, P. (1996). A Bayesian Approach to Binocular Steropsis. *International Journal of Computer Vision*, 19(3), 237-260.
- Faugeras, O., & Luong, Q.T. (2004). *The geometry of multiple images : The laws that govern the formation of multiple images of a scene and some of their applications*. Boston, MA: The MIT Press.
- Fusiello, A., Trucco, E., & Verri, A. (n.d.). *Rectification with Unconstrained Stereo Geometry*. Heriot-Watt University, Edinburgh, Department of Computing and Electrical Engineering.
- Hartley, R., & Zisserman, A. (2000). *Multiple view geometry in computer vision*. UK: Cambridge University Press.
- Ma, Y., Soatto, S., Kozecka, J., & Sastry, S. S. (2004). *An invitation to 3-D vision*. New York: Springer.
- Slepian, D., & Wolf, J. K. (1973, July). Noiseless Coding of Correlated Information Sources. *IEEE Trans on Information Theory*, 19, 471-480.

APPENDIX. Program Code in MATLAB

The codes for the following m-files are appended here:

1. main.m
2. reconstruct3D.m
3. rectify_im.m
4. bilinear_interpolate.m
5. normalise2dpts.m
6. normalise3dpts.m
7. inhomogeneous.m

main.m

```
% Main program code for M.Eng Project
%
% This program demonstrate the following
% 1. Estimation of Camera Projection matrices P1 and P2
% 2. Recovery of Fundamental matrix from P1 and P2
% 3. Reconstruction of the 3D grid points from P1, P2 and F
% 4. Rectification of the image pair
% 5. Verification of the disparity-depth relationship
%
% Copyright Zheshen Zhuang
% M.Eng 2005-06, Cornell University

% *****
%Loading and displaying the images with the control points
% *****

function main
clear all;
close all;

im1 = rgb2gray(imread('cobject.JPG'));
im2 = rgb2gray(imread('cobject2.JPG'));

% Manually recorded control points using cpselect.m
load camP1.mat
load camP2.mat

figure(1),
hold on
imagesc(im1);
colormap(gray);
plot( cpts(1,1), cpts(2,1), 'gx');
plot( cpts(1,2:end), cpts(2,2:end), 'rx');
title('View 1 of calibration object');
hold off
```

```

figure(2),
hold on
imagesc(im2);
colormap(gray);
plot( cpts2(1,1), cpts2(2,1), 'gx');
plot( cpts2(1,2:end), cpts2(2,2:end), 'rx');
title('View 2 of calibration object');
hold off

% The default pixel coordinates frames has the origin in the top left-
% handed corner with the y-axis pointing downwards. Negating the y-values
% makes the axes right-handed so that P will give positive depth
cpts(:,2) = -cpts(:,2);
cpts2(:,2) = -cpts2(:,2);

% Linear least-square estimation of the camera projection matrices.
% Details on the projection matrices are discussed in Section I and the
% estimation are described in Section III.

P1 =PPM(cpts,X);
P2 =PPM(cpts2,X);

% QR decomposition of the camera projection matrices (Section I eqn(6))
[K1,R1,t1] = decomposeCamP(P1);
[K2,R2,t2] = decomposeCamP(P2);

% Finding the optical centres (Section I eqn(8))
c1 = -inv(P1(1:3,1:3))*P1(:,4);
c2 = -inv(P2(1:3,1:3))*P2(:,4);

% Finding the fundamental matrix (Section II eqn(19))
H = inv( [P1 ; [ 0 0 0 2] ]);
P1p = P1*H;
P2p = P2*H;
F = skew(P2p(:,4))*P2p(1:3,1:3);
F = F ./ norm(F,1);

% Finding the mean residue error | cpts2'*F*cpts | of F
Fresidue = mean(abs(diag(cpts2'*F*cpts)))

% Triangulate the Maximum Likelihood 3D positions of the correspondence pairs
Xest = real(reconstruct3D(cpts,cpts2,F,P1,P2));

% Calculating the MSE between the actual 3-D points and the backprojected
% points
MSE = mean(norm(Xest - X ,2))

% Displaying the true and reconstructed grid points on the calibration
% objects with the recovered position of the camera centres.

figure(3)
hold on
plot3(X(3,:),X(1,:),X(2,:), 'kx');

```

```

plot3(Xest(3,:),Xest(1,:),Xest(2,:), 'rx');
plot3(c1(3), c1(1), c1(2), 'go');
plot3(c2(3), c2(1), c2(2), 'co');
grid on
legend('True 3D pts','Reconstructed pts', 'camera 1','camera 2');
hold off

%-----
% Computing the rectification matrices needed to rotate the two images so
% their epipolar lines lines up and runs horizontally across both images, as
% in a parallel camera setup. (SEE SECTION IV for a more detail explanation
% of the algorithm
%-----

% Let the new x-axes of the rectified cameras be the parallel to the line
% joining the camera centres. This is a hard requirement for the images to
% line up in the y-axis
xNew = c1 - c2;
xNew = xNew / norm(xNew);

% Extracting the old z-axes vectors
zOld1 = R1(3,1:3);
zOld2 = R2(3,1:3);

% Find the component of z-axes perpendicular to the new x-axis and let the
% new z-axis of the rectified cameras be the bilinear vector between them
k1 = zOld1 - dot(zOld1,xNew).*xNew;
k2 = zOld2 - dot(zOld2,xNew).*xNew;

if(norm(k1)<1e-4)
    k1 = [ 0 0 0]';
else
    k1 = k1 / norm(k1);
end

if(norm(k2)<1e-4)
    k2 = [ 0 0 0]';
else
    k2 = k2 / norm(k2);
end
zNew = k1+k2;
zNew = zNew/norm(zNew);

% The new y-axis is perpendicular to the new x-axis and z-axis
yNew = cross(zNew,xNew);

% Computing the new rotation matrix
Rnew = [xNew' ; yNew' ; zNew'];

% Compute the matrix, T1 & T2 for rectifying camera 1 and 2 respectively
T1 = K1*Rnew*inv(P1(1:3,1:3));
T2 = K2*Rnew*inv(P2(1:3,1:3));

```

```

% Compute the new projection matrix of the rectified cameras
Pnew1 = K1*Rnew*[eye(3) -c1];
Pnew2 = K2*Rnew*[eye(3) -c2];

% *****
%Rectifying the images using the computed rectification matrices T1 and T2
% *****

% Finding the corners of the rectified images and hence their sizes
nx = size(im1,2); ny = size(im1,1);
corners1 = inhomogeneous(T1*[1 1 1; nx 1 1; 1 ny 1; nx ny 1]);

nx = size(im2,2); ny = size(im2,1);
corners2 = inhomogeneous(T2*[1 1 1; nx 1 1; 1 ny 1; nx ny 1]);

corners = [corners1 corners2];
minx = floor(min(corners(1,:))-1);
miny = floor(min(corners(2,:))-1);
maxx = ceil(max(corners(1,:))+1);
maxy = ceil(max(corners(2,:))+1);

b = [minx miny maxx maxy];

% Rectifying the images using T1 and T2 to get the rectified images
newim1 = rectify_im(im1, T1, b);
newim2 = rectify_im(im2, T2, b);

newcpts1 = inhomogeneous(T1*cpts);
newcpts2 = inhomogeneous(T2*cpts2);

% plot the rectified images and the epipolar lines. Observe that the
% epipolar lines matches and are horizontal (or almost)
figure(4),
imagesc(minx:maxx, miny:maxy, newim1), axis xy, axis on, hold on
line([minx; maxx], [newcpts1(2,:); newcpts2(2,:)]);
plot(newcpts1(1,:), newcpts1(2,:), 'g*')
axis equal
title('First rectified image');
colormap gray;

figure(5),
imagesc(minx:maxx, miny:maxy, newim2), axis xy, axis on, hold on
line([minx; maxx], [newcpts2(2,:); newcpts2(2,:)]);
plot(newcpts2(1,:), newcpts2(2,:), 'g*')
axis equal
title('Second rectified image');
colormap gray;

% *****
% Verifying equation(35) disparity = |c2-c1|/depth from Section V
% *****
for k=1:size(X,2)
    truedepth(k) = dot(zNew, X(1:3,k)-c1 );

```

```

end

% Correcting for the camera matrix K1 and K2
newcpts1c = inv(K1)*[newcpts1 ; ones(1,size(X,2))];
newcpts2c = inv(K1)*[newcpts2 ; ones(1,size(X,2))];

% Computing the disparity (x-coordinates)
d = newcpts2c(1,:)-newcpts1c(1,:);

figure(6)
plot(norm(c2-c1,2)./truedepth, d, 'rx');
title('Verified Relationship between disparity and depth(zoom in)')
xlabel('|c_2-c_1|/depth');
ylabel('disparity');

figure(7)
plot(norm(c2-c1,2)./truedepth, d, 'rx');
axis([0,0.5,0,0.5]);
title('Verified Relationship between disparity and depth')
xlabel('|c_2-c_1|/depth');
ylabel('disparity');

% *****END OF MAIN PROGRAM*****

%*****
%          SHORT FUNCTIONS USED IN MAIN PROGRAM
%*****

% QR Factorization of the camera projection matrix P

function [K,R,t] = decomposeCamP(P)
Q = inv(P(1:3,1:3));
[U,B] = qr(Q);
R = inv(U);
t = B*P(1:3,4);
K = inv(B);
K = K./K(3,3);

% Return associated skew matrix of a given 3D vector

function S_hat = skew(S)
S_hat = zeros(3,3);
S_hat(1,2) = - S(3); S_hat(1,3) = S(2); S_hat(2,3) = - S(1);
S_hat(2,1) = S(3); S_hat(3,1) = - S(2); S_hat(3,2) = S(1);

Reconstruct3D.m

% reconstruct3D - reconstructs the ML 3-D positions from n pairs of image
% pts given the camera projection matrices P1 and P2 and corresponding
% fundamental matrix F
%
% Inputs:
%   P1, P2 - 3x4

```

```

% F - 3x3
% p1, p2 - 3xn
%
% Output: X - 4xn
%
% Direct implementation of Algorithm 11.1 "Optimal triangulation method"
% from the book "Multiple View Geometry in Computer Vision" by Richard
% Hartley and Andrew Zisserman, p. 304, 2000
%
% Copyright Zheshen Zhuang
% M.Eng 2005-06, Cornell University
%
function X = reconstruct3D(p1,p2,F,P1,P2);

n = size(p1,2);
for k=1:n
    [x1c, x2c] = triangulate(p1(:,k),p2(:,k),F);
    X(:,k) = triangulate_lin(x1c, x2c, P1, P2);
end

X(1,:) = X(1,:)/ X(4,:);
X(2,:) = X(2,:)/ X(4,:);
X(3,:) = X(3,:)/ X(4,:);
X(4,:) = 1;

%      ***END OF function RECONSTRUCT3D****

function [x1c, x2c] = triangulate(p1,p2,F);

x1 = p1(1)/p1(3);
y1 = p1(2)/p1(3);
x2 = p2(1)/p2(3);
y2 = p2(2)/p2(3);

T1 = [ 1 0 -x1 ; 0 1 -y1; 0 0 1];
T2 = [ 1 0 -x2 ; 0 1 -y2; 0 0 1];

F = inv(T2)*F*inv(T1);
e1 = null(F);
e2 = null(F');

e1 = e1 / sqrt(e1(1)^2 + e1(2)^2);
e2 = e2 / sqrt(e2(1)^2 + e2(2)^2);

R1 = [ e1(1) e1(2) 0; -e1(2) e1(1) 0; 0 0 1];
R2 = [ e2(1) e2(2) 0; -e2(2) e2(1) 0; 0 0 1];

F = R2*F*R1';

a = F(2,2);
b = F(2,3);
c = F(3,2);
d = F(3,3);

```

```

f1 = e1(3);
f2 = e2(3);

g = [ 0 0 0 (a^2+f2^2*c^2) (2*a*b+2*f2^2*c*d) f2^2*(b^2+d^2) 0]'...
      -(a*d-b*c) * [ (f1^4*a*c) (f1^4*(a*d+b*c)) (f1^4*b*d +2*f1^2*a*c) ...
                    (2*f1^2*(a*d+b*c)) (a*c+2*f1^2*b*d) (a*d+b*c) b*d ]';

t = roots(g);
t2 = real(t);
cost = t2.^2./(1+f1^2.*(t2.^2)) +(c.*t2+d).^2./ ...
      ((a*t2 + b).^2 +f2^2*(c*t2+d).^2);

min_t_index = find(cost == min(cost), 1, 'first');
min_t      = t(min_t_index);

l1 = [ min_t*f1 1 -min_t]';
l2 = [ -f2*(c*min_t+d) a*min_t+b c*min_t+d ]';

x1c = [ -l1(1)*l1(3) -l1(2)*l1(3) l1(1)^2+l1(2)^2]';
x2c = [ -l2(1)*l2(3) -l2(2)*l2(3) l2(1)^2+l2(2)^2]';

x1c = inv(T1)*R1'*x1c;
x2c = inv(T2)*R2'*x2c;

x1c = x1c./x1c(3);
x2c = x2c./x2c(3);

%      ***END OF function TRIANGULATE****

function X = triangulate_lin(x1, x2, P1, P2)

% Normalise each set of points so that the origin
% is at centroid and mean distance from origin is sqrt(2).
% normalise2dpts also ensures the scale parameter is 1.
x1 = inhomogeneous(x1);
x2 = inhomogeneous(x2);

% Build the constraint matrix
A = [ x1(1)*P1(3,:)- P1(1,:); ...
      x1(2)*P1(3,:)- P1(2,:); ...
      x2(1)*P2(3,:)- P2(1,:); ...
      x2(2)*P2(3,:)- P2(2,:); ...
      ];

[U,D,V] = svd(A,0);
% Extract X from the column of V corresponding to
% smallest singular value.
X = V(:,end);

Rectify_im.m

%RECTIFY_IM applies the rectification transformation to a given image.
%
% newim = rectify_im(im, H, box) applies the rectification transformation

```

```

% H (a 3-by-3 matrix) to the given image, im. The bounding box
% which determines the size of the output image newim, should be of the
% format: [minx,miny,maxx,maxy]
%
% Copyright Du Huynh
% The University of Western Australia
% School of Computer Science and Software Engineering
%
% This code is modified as used by Zheshen Zhuang for academic purposes.
% The image origin is centred at the top-lefthand corner as default by
% Matlab instead of at the centre
function newim = rectify_im(im, H, b)

minx = b(1);
miny = b(2);
maxx = b(3);
maxy = b(4);

% the two matrices xx and yy returned by meshgrid are of the same dimension
[xx,yy] = meshgrid(minx:maxx, miny:maxy);

% dimensions of the new (rectified) image
new_nrows = size(xx,1); new_ncols = size(xx,2);

x = reshape(xx, 1, new_nrows*new_ncols); clear('xx');
y = reshape(yy, 1, new_nrows*new_ncols); clear('yy');

invH = inv(H);
len = length(x);
% We will encounter the "out of memory" problem if the image is large.
% So, to get around the problem, we do the following operation in
% several steps
mm = 50000;
idx=1:mm;
while (1)
    if idx(1) > len
        break;
    elseif idx(end) > len
        idx = idx(1:len-idx(1)+1);
    end
    newxy(:,idx) = invH*([x(idx); y(idx); ones(1,length(idx))]);

    newxy(1,idx) = newxy(1,idx) ./ newxy(3,idx);
    newxy(2,idx) = newxy(2,idx) ./ newxy(3,idx);
    newxy(3,idx) = 1;
    idx = idx + mm;
end
clear('idx');

% convert the x-y image coordinate system (origin at the image centre) to
% row-column coordinate system (origin at the top-left corner) before
% calling bilinear interpolation
nrows = size(im,1); ncols = size(im,2);

```

```

newrc = [0 1 0;1 0 0; 0 0 1]*newxy;
clear('newxy');

% can't interpolate those points that fall outside the image im. So discard them.
idx = find(newrc(1,:) >= 1 & newrc(1,:) <= nrows & ...
    newrc(2,:) >= 1 & newrc(2,:) <= ncols);
newrc = newrc(1:2,idx);
x = x(idx);
y = y(idx);

val = [];
len = size(newrc,2);
idx = 1:mm;
while (1)
    if idx(1) > len
        break;
    elseif idx(end) > len
        idx = idx(1:len-idx(1)+1);
    end
    val(idx,:) = bilinear_interpolate(im, newrc(:,idx));
    idx = idx + mm;
end

% compose the new image, newim, and the 2 arrays, x and y,
% into 1D array
newim = zeros(new_ncols*new_nrows, size(im,3));
% covert into row-column coordinate system also
newim = zeros(new_nrows*new_ncols,1, size(im,3));
rc = (x-minx)*new_nrows + (y-miny) + 1;
if ~isempty(val)
    newim(rc,:) = val;
end

newim = reshape(uint8(newim), new_nrows, new_ncols, size(im,3));

```

bilinear_interpolate.m

```

%BILINEAR_INTERPOLATE performs bilinear interpolation.
%
% val = bilinear_interpolate(im, rc) returns the bilinearly interpolated
% pixel values at the given list of positions, rc.
%
% Input arguments:
% - im should be a m-by-n-by-3 (colour) image or a m-by-n (grey scale) image.
% - rc should be a 2-by-k matrix where k is the number of positions
% whose values are to be computed via bilinear interpolation. The first
% row of the matrix should contain the row-coordinates and the second row the
% column-coordinates of these positions. It is important that all the
% (row,column)-positions stored in the matrix, rc, are within the boundary
% of the image im.
%
% Output argument:

```

```

% - val is the output k-by-3 (if image im has 3 colour bands) or k-by-1
% (if image im is a grey scale image) matrix containing the interpolated pixel
% values.
%
%Created July 2002.
%Last modified September 2003.
%
%Copyright Du Huynh
%The University of Western Australia
%School of Computer Science and Software Engineering
%*****
function val = bilinear_interpolate(im, rc)

nrows = size(im,1); ncols = size(im,2);
% number of bands (3 for coloured images; 1 for gray scale images)
nobands = size(im,3);

% check that all the entries in matrix rc are within the boundary of image im.
if sum(rc(1,:) < 1 | rc(1,:) > nrows | rc(2,:) < 1 | rc(2,:) > ncols)
    error('bilinear_interpolate: elements of the rc matrix must be within the image boundary');
end

% The four corner points used in the bilinear interpolation:
% c4    c3
% +-----+
% |      |
% | o    | (a point o which is stored in a column vector of rc
% |      | and the two corner points used for its bilinear interpolation)
% +-----+
% c1    c2
% The row-column coordinate system is used. All the corner points c1, c2, c3,
% and c4 are two vectors, whose 1st components contains the row coordinates
% and 2nd components contains the column coordinates.

% note that we should have given a small margin for variable rc
% so that the four corner pixels surrounding rc are within the
% image boundary of im. This condition should be enforced in the
% caller of this function.
c4 = (floor(rc));
c2 = (ceil(rc));
c3 = ([c4(1,:); c2(2,:)]);
c1 = ([c2(1,:); c4(2,:)]);

% d(diffRC_idx) = (rc(diffRC_idx) - c1) ./ (c4 - c1 + eps);
%
% the interpolation procedure above fails for those points in
% rc whose x- or y- component is a whole number (in which case,
% the respective components of these points in the c1 and c4 matrices
% would be the same. the formula for d below would cause a division
% by zero problem.
sameC_idx = find(c2(2,:) == c4(2,:));
sameR_idx = find(c2(1,:) == c4(1,:));

```

```

diffRC_idx = find(c2(1,:) ~= c4(1,:) & c2(2,:) ~= c4(2,:));
% now the formula for d can be safely applied...
d(:,diffRC_idx) = (rc(:,diffRC_idx) - c4(:,diffRC_idx)) ./ ...
    (c2(:,diffRC_idx) - c4(:,diffRC_idx) + eps);
d(2,sameC_idx) = 0;
d(1,sameC_idx) = (rc(1,sameC_idx) - c4(1,sameC_idx)) ./ ...
    (c2(1,sameC_idx) - c4(1,sameC_idx) + eps);
d(1,sameR_idx) = 0;
d(2,sameR_idx) = (rc(2,sameR_idx) - c4(2,sameR_idx)) ./ ...
    (c2(2,sameR_idx) - c4(2,sameR_idx) + eps);

% convert c1, c2, c3, c4 into 1D array for fast retrieval of image
% intensity from im
c1 = (c1(2,:)-1)*nrows + c1(1,:);
c2 = (c2(2,:)-1)*nrows + c2(1,:);
c3 = (c3(2,:)-1)*nrows + c3(1,:);
c4 = (c4(2,:)-1)*nrows + c4(1,:);

im = reshape(im, nrows*ncols, nobands);
c1val = im(c1,:);
c2val = im(c2,:);
c3val = im(c3,:);
c4val = im(c4,:);

for i=1:nobands
    val(:,i) = (1-d(1,:)).*(1-d(2,:)).*double(c4val(:,i)) + ...
        d(1,:).*(1-d(2,:)).*double(c1val(:,i)) + ...
        (1-d(1,:)).*d(2,:).*double(c3val(:,i)) + ...
        d(1,:).*d(2,:).*double(c2val(:,i));
    val(:,i) = uint8(val(:,i));
end

return

normalise2dpts.m

% NORMALISE2DPTS - normalises 2D homogeneous points
%
% Function translates and normalises a set of 2D homogeneous points
% so that their centroid is at the origin and their mean distance from
% the origin is sqrt(2). This process typically improves the
% conditioning of any equations used to solve homographies, fundamental
% matrices etc.
%
% Usage: [newpts, T] = normalise2dpts(pts)
%
% Argument:
% pts - 3xN array of 2D homogeneous coordinates
%
% Returns:
% newpts - 3xN array of transformed 2D homogeneous coordinates. The
% scaling parameter is normalised to 1 unless the point is at
% infinity.

```

```

% T - The 3x3 transformation matrix, newpts = T*pts
%
% If there are some points at infinity the normalisation transform
% is calculated using just the finite points. Being a scaling and
% translating transform this will not affect the points at infinity.

% Peter Kovesi
% School of Computer Science & Software Engineering
% The University of Western Australia
% pk at csse uwa edu au
% http://www.csse.uwa.edu.au/~pk
%
% May 2003 - Original version
% February 2004 - Modified to deal with points at infinity.

function [newpts, T] = normalise2dpts(pts)

    if size(pts,1) ~= 3
        error('pts must be 3xN');
    end

    % Find the indices of the points that are not at infinity
    finiteind = find(abs(pts(3,:)) > eps);

    if length(finiteind) ~= size(pts,2)
        warning('Some points are at infinity');
    end

    % For the finite points ensure homogeneous coords have scale of 1
    pts(1,finiteind) = pts(1,finiteind)./pts(3,finiteind);
    pts(2,finiteind) = pts(2,finiteind)./pts(3,finiteind);
    pts(3,finiteind) = 1;

    c = mean(pts(1:2,finiteind)'); % Centroid of finite points
    newp(1,finiteind) = pts(1,finiteind)-c(1); % Shift origin to centroid.
    newp(2,finiteind) = pts(2,finiteind)-c(2);

    meandist = mean(sqrt(newp(1,finiteind).^2 + newp(2,finiteind).^2));

    scale = sqrt(2)/meandist;

    T = [scale 0 -scale*c(1)
         0 scale -scale*c(2)
         0 0 1 ];

    newpts = T*pts;

```

normalise3dpts.m

```

% NORMALISE3DPTS - normalises 2D homogeneous points
%
% Function translates and normalises a set of 2D homogeneous points
% so that their centroid is at the origin and their mean distance from
% the origin is sqrt(3).
% Usage: [newpts, T] = normalise3dpts(pts)
%
% Argument:
% pts - 4xN array of 3D homogeneous coordinates
%
% Returns:
% newpts - 4xN array of transformed 3D homogeneous coordinates. The
% scaling parameter is normalised to 1 unless the point is at
% infinity.
% T - The 4x4 transformation matrix, newpts = T*pts
%
% If there are some points at infinity the normalisation transform
% is calculated using just the finite points. Being a scaling and
% translating transform this will not affect the points at infinity.
%
% Adapted from NORMALISE2DPTS.m by Peter Kovesi
% School of Computer Science & Software Engineering
% The University of Western Australia
% pk at csse uwa edu au
% http://www.csse.uwa.edu.au/~pk
%
% May 2003 - Original version
% February 2004 - Modified to deal with points at infinity.
function [newpts, T] = normalise3dpts(pts)

    if size(pts,1) ~= 4
        error('pts must be 3xN');
    end

    % Find the indices of the points that are not at infinity
    finiteind = find(abs(pts(4,:)) > eps);

    if length(finiteind) ~= size(pts,2)
        warning('Some points are at infinity');
    end

    % For the finite points ensure homogeneous coords have scale of 1
    pts(1,finiteind) = pts(1,finiteind)./pts(4,finiteind);
    pts(2,finiteind) = pts(2,finiteind)./pts(4,finiteind);
    pts(3,finiteind) = pts(3,finiteind)./pts(4,finiteind);
    pts(4,finiteind) = 1;

    c = mean(pts(1:3,finiteind))'; % Centroid of finite points
    newp(1,finiteind) = pts(1,finiteind)-c(1); % Shift origin to centroid.
    newp(2,finiteind) = pts(2,finiteind)-c(2);
    newp(3,finiteind) = pts(3,finiteind)-c(3);

```

```
meandist = mean(sqrt(newp(1,finiteind).^2 + newp(2,finiteind).^2 ...  
+newp(3,finiteind).^2));
```

```
scale = sqrt(3)/meandist;
```

```
T = [scale 0 0 -scale*c(1)  
0 scale 0 -scale*c(2)  
0 0 scale -scale*c(3)  
0 0 0 1  ];
```

```
newpts = T*pts;
```

inhomogeneous.m

```
% Convert Y in 2D homogeneous coordinates to X in inhomogeneous coordinates  
function X = inhomogeneous(Y)
```

```
X = zeros(2,size(Y,2));  
X(1,:) = Y(1,:)./Y(3,:);  
X(2,:) = Y(2,:)./Y(3,:);
```