Lines and Points in Three Views and the Trifocal Tensor

Richard I. Hartley

G.E. CRD, Schenectady, NY, 12301.

Abstract

This paper discusses the basic role of the trifocal tensor in scene reconstruction from three views. This $3 \times 3 \times 3$ tensor plays a role in the analysis of scenes from three views analogous to the role played by the fundamental matrix in the two-view case. In particular, the trifocal tensor may be computed by a linear algorithm from a set of 13 line correspondences in three views. It is further shown in this paper, that the trifocal tensor is essentially identical to a set of coefficients introduced by Shashua to effect point transfer in the three view case. This observation means that the 13-line algorithm may be extended to allow for the computation of the trifocal tensor given any mixture of sufficiently many line and point correspondences. From the trifocal tensor the camera matrices of the images may be computed, and the scene may be reconstructed. For unrelated uncalibrated cameras, this reconstruction will be unique up to projectivity. Thus, projective reconstruction of a set of lines and points may be carried out linearly from three views.

1 Introduction

This paper gives an effective algorithm for the projective reconstruction of a scene consisting of lines and points in space as seen in three views with uncalibrated cameras. The placement of the cameras with respect to the scene is also determined. This algorithm is unique in the literature in that it gives a unified linear approach that can deal with a mixture of points and lines in the uncalibrated case, though Spetsakis and Aloimonos ([25, 23]) described a method for calibrated cameras. Most previous algorithms have been specific to points ([18, 16, 17]) or lines ([24, 29]), but could not handle both. True, one could always use pairs of matching points to determine line matches, which can then be used in an algorithm for reconstruction from lines. This strategem, however, achieves a unified approach for lines and points at considerable expense, since a pair of point matches contains much more information than a single line match (as will be made clear quantitively in this paper). The restraint of using only points or lines forces one to ignore important information available in most images, particularly of man-made objects, since typically, one can find both distinguished points and lines in matching images.

Points are important in that they give much more information than lines. For instance although one can do projective reconstruction from only two views of a set of points ([18]), for lines at least three views are necessary ([29]), since no information whatever about camera placements may be derived from any number of line-to-line correspondences in fewer than three views. On the other hand, lines have several advantages. Firstly, they can normally be determined more accurately than points, often with an accuracy of better than a tenth of a pixel. Secondly, line matches may be used in cases where occlusions occur. Often end points of lines are not visible in the images. For instance, in Fig 1, the left hand slanting roof edge of the rear house may be matched in the three images, although its end point is occluded behind the front house.

Outline. The trifocal tensor arises as a relationship between corresponding lines in three images. This tensor may be computed linearly from a set of line correspondences in three images, and as shown previously in [8, 11], leads to an algorithm for projective reconstruction from line correspondences in three views. A key observation of this paper is the connection (see equation (7)) between this tensor and Shashua's 3-view trilinearity relationships ([20]). In effect, the trifocal tensor, is the same as the set of trilinearity coefficients introduced by Shashua. In [20] Shashua outlined an algorithm for computation of these coefficients from seven point correspondences in three views. It immediately follows from this key observation that one can amalgamate the line and point algorithms into one. This means that one can non-iteratively carry out projective reconstruction from three views of seven points, or 13 lines or anything in between (so to speak). Projective reconstruction from three views of seven points was not considered by Shashua in ([20]). The previous algorithm published in [11] for retrieving the camera matrices, and hence projective structure, from the trifocal tensor was not very stable numerically. In this paper I present a numerically superior algorithm based on techniques of numerical linear algebra. This new algorithm is evaluated experimentally and gives good results, especially when combined with an iterative algorithm to refine the computed structure.

In order to derive the connection between the trifocal tensor and Shashua's trilinearity relationships, I reexamine Shashua's results and place them in the standard context of projection using camera matrices. My hope is that this rederivation has the merit of throwing further light on the meaning of those relationships.

1.1 The Trifocal Tensor

A basic tool in the analysis of this paper is an entity called, here, the *trifocal tensor*. Since this entity has appeared previously in the literature in different guises, it is appropriate to discuss its history. With hindsight, we may atribute the discovery of the trifocal tensor to Spetsakis and Aloimonos ([24]) and Weng, Huang and Ahuja ([29]), where it was used for scene reconstruction from lines in the case of calibrated cameras. Spetsakis and Aloimonos ([25, 23]) also applied it to reconstruction from point and line matches in the calibrated case. It was later shown by the present author in [8, 11] to be equally applicable to projective scene reconstruction from 13 lines in the uncalibrated case. Those papers form the basis for part of this article. In all of the above papers, the entity referred to here as the trifocal tensor was not considered as a tensor, but rather as a set of three 3×3 matrices. Perhaps the first author to refer to it as a tensor was Vieville ([28]) who continued the work on scene reconstruction from lines.

Meanwhile in independent work, Shashua introduced a set of 27 coefficients for a set of four independent trilinearity conditions relating the coordinates of corresponding points in three views with uncalibrated cameras ([20]). Subsequently ([21]) Shashua gave a linear method for computation of the coefficients from only seven point matches in three views.

A key result of this paper is that the set of coefficients introduced by Shashua ([20])

are exactly the same as the entries of the three 3×3 matrices of ([29, 8]), except for a change of sign[1] and rearrangement of the indices. The importance of this result is that it allows an amalgamation of the linear algorithms for points ([21]) and for lines ([11]). This results in an algorithm of significantly greater applicability and power than either of the line or point algorithms alone.

Whereas the line papers [29, 8, 11] consider three 3×3 matrices, Shashua's paper defines his coefficients as the entries of nine 3-vectors. In fact, essentially, we are dealing with a triply indexed $3 \times 3 \times 3$ array of values, which it is natural to treat as a tensor, as suggested by Vieville. Therefore, in this paper, we refer to this three-dimensional array as a tensor, and make use of tensor notation, in particular the standard summation convention for repeated indices. In recent work ([22]) Shashua has also considered his set of coefficients as a tensor. As for the name, I suggest the words *trifocal tensor* in an attempt to establish a standard terminology. Despite the potentially fundamental role played by this tensor in three-view stereo, I believe that the word *fundamental* is too often used for us to adopt the term *fundamental tensor*. The alternative expression *trilinear tensor* seems redundant, since a tensor is intrinsically a linear object.

Since the appearance of the original conference versions of this paper, and the preparation of the original journal version many important new papers have appeared dealing with the trifocal tensor. Due to the speed with which this area has been developing, it was impossible to take into account all the latest work in preparing a final version of the paper. However, it is appropriate to mention here some of the recent papers. The trifocal tensor has been considered in detail by Triggs ([27, 26]). It is from this work that I have taken the idea of distinguishing between covariant and contravariant indices. Triggs's papers and also the paper of Faugeras and Mourrain ([5]) give enlightening new derivations of the trifocal tensor equations and consider the the trifocal tensor in the context of general linear constraints involving multiple views. For correspondences in four views, quadrilinear constraints and their associated tensor have been described in several papers ([27, 26, 5, 30, 15, 13]). Heyden ([15, 13]) uses the four-view tensor to derive a reconstruction algorithm that works with six or more point correspondences in four views. Heyden ([14, 15, 13]) also introduces the concept of a "reduced" trifocal tensor (and reduced fundamental matrix), which results from the trifocal tensor by expressing image points with respect to a canonic affine basis. Finally Armstrong et al. ([1]) use the trifocal tensor in self calibration – that is, derivation of Euclidean structure from three views.

1.2 Notation and Basics

Vectors, sometimes denoted by bold lower-case letters such as $\mathbf{u}$ are singly-indexed ensembles of real numbers. The vector $\mathbf{u}$ (and in the same way other vectors) will more often be denoted as u^i, where i is an index running over the appropriate range (usually $1, \ldots, 3$ or $1, \ldots, 4$). In a similar manner, a notation such as a^i_j denotes a doubly indexed quantity, namely a matrix. We will also be concerned with triply indexed quantities, such as T_i^{jk}. We adopt the usual summation convention for tensors :

> Any index repeated as subscript and superscript in a term involving vectors, matrices and tensors implies a summation over the range of index values.

[1]In order to avoid the sign discrepancy, Shashua's coefficients will be defined with opposite sign in this paper

Any formula involving indices is intended to hold for any choice of values of the free indices (which means those indices that are not repeated).

The placement of an index as subscript or superscript indicates whether it transforms covariantly or contravariantly with respect to a change of basis. This point is explained in Appendix A. Readers willing to accept on faith, or not worry about the position of the indices need not read this appendix. They will find throughout the paper that repeated indices implying summation do turn up, somewhat magically, in matching contravariant and covariant positions.

The three-dimensional space containing the scene will be considered to be the 3-dimensional projective space $\mathcal{P}^3$ and points in space will be represented by homogeneous 4-vectors $\mathbf{x}$. Similarly, image space will be regarded as the 2-dimensional projective space $\mathcal{P}^2$ and points in an image will be represented by homogeneous 3-vectors $\mathbf{u}$. Homogeneous quantities (vectors, matrices or tensors) that differ by a non-zero scale factor are considered to be equal. In this paper, we use the symbol $\approx$ to indicate equality up to a constant non-zero scale. This is necessary, since we want to keep clear the distinction between quantites that are equal and those that are equal up to a constant factor.

The 3D-to-2D mapping induced by a projective camera may be represented by a 3×4 matrix $M = [m^i_j]$ of rank 3, such that if $\mathbf{x}$ and $\mathbf{u}$ are corresponding object and image points then $\mathbf{u} \approx M\mathbf{x}$, or in tensor notation, $u^i \approx m^i_j x^j$. Such a matrix will be called a camera matrix. One special camera matrix is denoted by $[I \mid 0]$, made up of a 3×3 identity matrix I and a final zero column.

Just as points in image space $\mathcal{P}^2$ are represented by homogeneous vectors so are lines in $\mathcal{P}^2$. Bold greek letters such as $\boldsymbol{\lambda}$ represent lines. The point $\mathbf{u}$ lies on the line $\boldsymbol{\lambda}$ if and only if $\lambda_i u^i = 0$. If $\mathbf{u} = (u^1, u^2, 1)^\top$ then the perpendicular distance from the line to the point is given by

$$d(\boldsymbol{\lambda}, \mathbf{u}) = \frac{\lambda_i u^i}{(\lambda_1^2 + \lambda_2^2)^{1/2}} \tag{1}$$

Normal Form for Camera Matrices. Consider a set of lines and points in space viewed by several cameras. We use the word *feature* to represent either a line or a point. We suppose that the image coordinates of each feature as seen in each image are given, but the actual positions of the features in space are unknown. The task of projective reconstruction is to find a set of camera matrices and 3D-lines and points so that each such 3D feature is indeed mapped to the given image feature in each of the images. If the camera matrices are allowed to be arbitrary, then ([7, 4]) the scene can not be reconstructed more precisely than up to an arbitrary 3D projective transformation.

Consider now a reconstruction from three views, and let the three camera matrices be M, M' and M''. We make the assumption that no two of the cameras are located at the same point in space. Let H be formed by adding one extra row to M to make a non-singular 4×4 matrix. Then since $HH^{-1} = I_{4\times4}$, it follows that $MH^{-1} = [I|0]$. Since M may be transformed to $[I \mid 0]$, by applying transformation H to the reconstruction we may assume without loss of generality that $M = [I \mid 0]$.

To save the reader the trouble of having to count primes, we denote the entries of the camera matrices M' and M'' by a^i_j and b^i_j respectively, instead of by m'^i_j and m''^i_j. Thus, the three camera matrices M, M' and M'' may be written in the form $M = [I \mid 0]$, $M' = [a^i_j]$ and $M'' = [b^i_j]$.

2 Transferring lines

In this section we investigate the relationship between the images of a line as taken by three separate cameras.

Given a general camera matrix and a line in the image, the projection of the line from the camera centre forms a plane in space consisting of all points in space that will map onto the given image line. We need to derive a formula for that plane. The simple answer is as follow.

Proposition 2.1. *The plane in space consisting of all points that are mapped to a line* λ_i *by a camera with matrix* $[m^i_j]$ *is given by* $\pi_j = m^i_j \lambda_i$.

Proof. By definition, a point x^j is on the plane π_j if and only if the projected point $m^i_j x^j$ lies on the line λ_i. This latter condition can be expressed as $\lambda_i m^i_j x^j = 0$. On the other hand, x^j lies on π_j if and only if $\pi_j x^j = 0$. Hence, we see that $\pi_j x^j = 0$ if and only if $\lambda_i m^i_j x^j = 0$, and from this we deduce that $\pi_j = \lambda_i m^i_j$ as required. □

In standard matrix notation, we may write this plane as $\boldsymbol{\pi} = M^\top \boldsymbol{\lambda}$.

Now, given three cameras with matrices $M = [I \mid 0]$, $M' = [a^j_i]$ and $M'' = [b^j_i]$, and three lines λ_j, λ'_j and λ''_j in the three images, we seek a relationship between the coordinates of the three lines. Since the three image lines are derived from a single line in space, it follows that the planes corresponding to the three image lines must meet at this line in space. In particular, the three planes $M^\top \boldsymbol{\lambda}$, $M'^\top \boldsymbol{\lambda}'$ and $M''^\top \boldsymbol{\lambda}''$ meet in a line. Writing the coordinate vectors of these three planes as the rows of a matrix we obtain

$$\begin{bmatrix} \lambda_1 & \lambda_2 & \lambda_3 & 0 \\ a^j_1 \lambda'_j & a^j_2 \lambda'_j & a^j_3 \lambda'_j & a^j_4 \lambda'_j \\ b^k_1 \lambda''_k & b^k_2 \lambda''_k & b^k_3 \lambda''_k & b^k_4 \lambda''_k \end{bmatrix} . \tag{2}$$

In writing this matrix, we have taken particular note of the simple form of the matrix $M = [I \mid 0]$. Since the three planes meet in space, there is a linear dependency between the rows of this matrix (2). In particular, there exist constants α and β such that for $i = 1, \ldots, 3$ we have

$$\lambda_i = \alpha(a^j_i \lambda'_j) + \beta(b^k_i \lambda''_k)$$

Furthermore, because of the zero entry in the top row, $0 = \alpha(a^j_4 \lambda'_j) + \beta(b^k_4 \lambda''_k)$. We deduce that $\alpha \approx (b^k_4 \lambda''_k)$ and $\beta \approx -(a^j_4 \lambda'_j)$. Therefore

$$\begin{aligned} \lambda_i &\approx (b^k_4 \lambda''_k)(a^j_i \lambda'_j) - (a^j_4 \lambda'_j)(b^k_i \lambda''_k) \\ &= \lambda'_j \lambda''_k (a^j_i b^k_4 - a^j_4 b^k_i) \end{aligned}$$

Now, we define a $3 \times 3 \times 3$ tensor T^{jk}_i by the expression

$$T^{jk}_i = a^j_i b^k_4 - a^j_4 b^k_i \ . \tag{3}$$

Then we have the following formula.

$$\lambda_i \approx \lambda'_j \lambda''_k T^{jk}_i \ . \tag{4}$$

The tensor T_i^{jk} is the *trifocal tensor*, which is the basic entity investigated in this paper. Formula (4) has been derived previously in [29, 24] for the special case of calibrated cameras. The above derivation shows how generalization to the case of uncalibrated cameras leads to a simplification of the derivation. It should be remarked that the trifocal tensor defined by (3) treats the first image with camera matrix $[I \mid 0]$ differently from the others. This is shown most clearly in (4) in which the lines in the first image are handled differently. The index i in T_i^{jk} appears in the covariant position, whereas the other two indices are contravariant. There are in fact two other related but distinct trifocal tensors in which indices corresponding to the two other images are covariant. We continue to speak of a single trifocal tensor, however, implicitly assuming that one of the images is singled out in this way.

It is worthwhile noting at this stage that the vectors a_4^j and b_4^k appearing in (3) have a geometric meaning. They are the epipoles in the second and third images corresponding to the centre of projection of the first camera, namely the point $(0,0,0,1)^\top$. If these epipoles are known then the entries of the trifocal tensor are linear expressions in the remaining camera matrix entries. This point will be taken up again in section 5.

Given T_i^{jk} and the coordinates λ'_j and λ''_k of matching lines, expression (3) may be used to compute the line in the other image. The application of this process, know as *line transfer* will not be investigated in this paper, however.

We describe now a first method for determining the trifocal tensor. If at least 13 line matches $\boldsymbol{\lambda} \leftrightarrow \boldsymbol{\lambda}' \leftrightarrow \boldsymbol{\lambda}''$ are known, it is possible to solve for the entries of the tensor T_i^{jk}, since each line match provides two linear equations in the 27 unknown tensor entries. In particular, if the line $\boldsymbol{\lambda}$ is presented by specifying the two points on the line, then each such point $\mathbf{u} = (u^1, u^2, u^3)^\top$ gives rise to an equation

$$u^i \lambda'_j \lambda''_k T_i^{jk} = 0 \tag{5}$$

To normalize these equations, the lines $\boldsymbol{\lambda}'$ and $\boldsymbol{\lambda}''$ should be scaled to be unit vectors. Each end point $\mathbf{u}$ should be scaled so that $u^3 = 1$.

3 Transferring Points.

In this section we will investigate the relationship of the trifocal tensor with point-transfer methods, and in particular with the trilinearity relationships of Shashua.

Suppose that a point $\mathbf{x}$ in space is seen in three images, and that as usual the three cameras are given in the normalized form $M = [I \mid 0]$, $M' = [a_j^i]$ and $M'' = [b_j^i]$.

We suppose that the point $\mathbf{x}$ is seen at positions $\mathbf{u}$, $\mathbf{u}'$ and $\mathbf{u}''$ in the three images, where $\mathbf{u}$ (and similarly $\mathbf{u}'$ and $\mathbf{u}''$) is a 3-vector $\mathbf{u} = (u^1, u^2, u^3)^\top$, the representation of the point in homogeneous coordinates. The coordinates $(u^1/u^3, u^2/u^3)$ are the coordinates actually seen in the image. We wish to find a relationship between the coordinates of the points $\mathbf{u}$, $\mathbf{u}'$ and $\mathbf{u}''$. At any point in the following derivation, we may set u^3, u'^3 or u''^3 to 1 to obtain equations relating to measured image coordinates.

Because of the form of the matrix $M = [I \mid 0]$, it is extremely simple to give a formula for the position of the point in space. In particular, since $[I \mid 0]\mathbf{x} \approx \mathbf{u}$, we may write $\mathbf{x} = \begin{pmatrix} \mathbf{u} \\ t \end{pmatrix}$ for some t, yet to be determined. It may be verified that t is the same as the "relative affine invariant", k, considered by Shashua ([20]). Now, projecting this point into the second image by the usual formula $u'^i \approx a^i_j x^j$ gives

$$u'^i \approx a^i_k u^k + a^i_4 t$$

The notation $\approx$ denotes equality up to an unknown scale factor. We may eliminate this scale factor to obtain equations

$$u'^i(a^j_k u^k + a^j_4 t) = u'^j(a^i_k u^k + a^i_4 t)$$

where each choice of the free indices i and j gives a separate equation. Of the three resulting equations, only two are independent. From each of these equations independently, one may compute the value of t. We obtain three separate estimates for t.

$$t = u^k(u'^i a^j_k - u'^j a^i_k)/(u'^j a^i_4 - u'^i a^j_4) \tag{6}$$

Substituting the value of t from (6) we see that the point $\mathbf{x}$ may be written as

$$\begin{aligned} \mathbf{x} &= \begin{pmatrix} \mathbf{u} \\ u^k(u'^i a^j_k - u'^j a^i_k)/(u'^j a^i_4 - u'^i a^j_4) \end{pmatrix} \\ &\approx \begin{pmatrix} (u'^j a^i_4 - u'^i a^j_4)\mathbf{u} \\ u^k(u'^i a^j_k - u'^j a^i_k) \end{pmatrix} \end{aligned}$$

Now, projecting this point via the third camera, $u''^l \approx b^l_k x^k$ we find that

$$\begin{aligned} u''^l &\approx b^l_k u^k(u'^j a^i_4 - u'^i a^j_4) + b^l_4 u^k(u'^i a^j_k - u'^j a^i_k) \\ &\approx u^k u'^i(a^j_k b^l_4 - a^j_4 b^l_k) - u^k u'^j(a^i_k b^l_4 - a^i_4 b^l_k) \end{aligned}$$

Now, referring to (3), we recognize the tensor coefficients T^{jk}_i in this expression :

$$u''^l \approx u^k(u'^i T^{jl}_k - u'^j T^{il}_k) \tag{7}$$

As before we may eliminate the unknown scale factor implied by the $\approx$ sign to get (after some slight rearranging) the equations

$$u^k(u'^i u''^m T^{jl}_k - u'^j u''^m T^{il}_k - u'^i u''^l T^{jm}_k + u'^j u''^l T^{im}_k) = 0^{ijlm} \quad . \tag{8}$$

These are the trilinearity relationships of Shashua ([20]). In these equations, the indices i, j, l and m are free variables, and there is one equation for each choice of indices with $i \neq j$ and $l \neq m$. Since we get the same relation by interchanging i and l, or l and m, we may assume that $i < j$ and $l < m$. There are therefore nine different equations defined by this expression. However, only two of the three choices of pair (i, j) given independent equations, and the same is true for pairs (l, m). Hence, there are 4 linearly independent equations.

Equation (8) may seem a little complex, however, it is easily remembered if one notices its special form. Consider the first term $u^k u'^i u''^m T^{jl}_k$. One considers indices i and j as being paired, and also indices l and m are paired. Now the other terms are obtained by swapping the paired indices i and j, and similarly swapping l and m, leading to a total of four terms. Each swap of a pair of indices causes a change of sign. Equation (8) may be written compactly by defining a tensor ϵ_{pqr}, which is equal to zero unless p, q and r are all distinct, and otherwise equals $+1$ or -1 according to whether (pqr) is an even or odd permutation of the indices (123). Using this notation, (8) becomes simply

$$u^k(u'^i\epsilon_{ijp})(u''^l\epsilon_{lmq})T_k^{jm} = 0_{pq} \tag{9}$$

where each choice of p and q gives a different equation. In a similar manner, (4) may be written as

$$\epsilon^{ipq}\lambda_i\lambda'_j\lambda''_kT_p^{jk} = 0^q \ . \tag{10}$$

The form (9) of the trilinear point relation, shows a similarity to (5). In particular, consider the meaning of the expression $u'^i\epsilon_{ijp}$ for different values of the free variable p. For $p = 1, 2, 3$ vector $u'^i\epsilon_{ijp}$ is equal to $(0, -u'^3, u'^2)^\top$, $(u^3, 0, -u^1)^\top$ and $(-u^2, u^1, 0)^\top$ respectively. The first two of these vectors represent lines through the point u'^i parallel with the horizontal (first) and vertical (second) coordinate axes. The third vector represents a line through the point u^i and the coordinate origin $(0, 0, 1)^\top$. Thus (9) is equivalent to a set of equations $u^k\lambda'_j\lambda''_mT_k^{jm}$ where the lines λ'_j and λ''_m may be chosen independently to be lines through the respective points u'^i and u''^l parallel to the coordinate axes or through the coordinate origin. Geometrically this is stating that the line in space that projects to (for instance) horizontal lines through points u'^i and u''^l in two of the images will project in the other image to a line through the point u^k. Thus, the trilinear point relations (9) may be considered simply as special cases of the line transfer equations (4) or (5). A similar geometric interpretation of the trilinear point relations has been given in [5].

One choice of the four independent equations from (8) is obtained by setting $j = m = 3$, and letting i and l range freely. As stated previously, we may set u^3, u'^3 and u''^3 to 1 to obtain a relationship between observed image coordinates. Equation (8) then becomes

$$u^k(u'^iu''^lT_k^{33} - u''^lT_k^{i3} - u'^iT_k^{3l} + T_k^{il}) = 0 \ . \tag{11}$$

The four different choices of $i, l = 1, 2$ give four different equations in terms of the observed image coordinates.

Given seven point correspondences, we have 28 equations, which is enough to solve for the tensor entries T_i^{jk}. Shashua states that better results are obtained by including six equations for each point match. The experiments reported in a later section use all nine equations. This seems to be a good idea to improve stability, but it is not clear how much advantage (if any) this gives. The effect on run-time of choosing nine rather than four equations per point is considered in the next section, however.

4 Solving for the Trifocal Tensor

We have seen that the entries of the trifocal tensor, T occur in equations involving both points (11) and lines (5). This has the significant implication that we may amalgamate the line and point algorithms into one algorithm. In particular, each line correspondence $\boldsymbol{\lambda} \leftrightarrow \boldsymbol{\lambda}' \leftrightarrow \boldsymbol{\lambda}''$ gives two linear equations in the entries T_i^{jk}, whereas each point correspondence gives four linear equations. Therefore, provided that $2\#lines + 4\#points \geq 26$ we have sufficiently many matches to solve for the T_i^{jk}. The tensor entries T_i^{jk} are found up to a common scale, and we find a solution such that sum of squares of the entries is one. In the case where there are more than 26 equations, we find a least-squares solution satisfying this constraint.

Normalization. Before setting out to write and solve the equations, it is a very good idea to normalize the data by scaling and translating the points. The algorithm does not do well if all points are of the form $(u^1, u^2, 1)^\top$ in homogeneous coordinates with u^1 and u^2 very much larger than 1. A heuristic that works well is to translate the points in each image so that the centroid of all measured points is at the origin of the image coordinates, and then scaling so that the average distance of a point from the origin is $\sqrt{2}$ units. In this way the average point will be something like $(1, 1, 1)^\top$ in homogeneous coordinates, and each of the homogeneous coordinates will be approximately of equal weight. This transformation improves the condition of the matrix of equations, and leads to a much better solution. Despite the seemingly harmless nature of this transformation, this is an essential step in the algorithm.

This normalization step has also been shown ([9]) to be effective in the two-camera case for determining the fundamental matrix.

Line Equations. The chosen method of expressing line equations is as follows. Each line is assumed to be defined by two points. For the second and third images the lines $\boldsymbol{\lambda}'$ and $\boldsymbol{\lambda}''$ are computed from their end points. The end points defining the lines are subject to the same scaling and translation as the point correspondences, described in the previous paragraph, and are normalized to be unit vectors. Corresponding to each of the two points defining the line $\boldsymbol{\lambda}$ in the first images, an equation $u^i \lambda'_j \lambda''_k T_i^{jk}$ is written. In forming this equation, the point u^i is normalized so that its third coordinate is unity. When the set of equations is solved, such an equation will not be satisfied exactly. Instead there will be an error, denoted by ϵ such that $u^i \lambda'_j \lambda''_k T_i^{jk} = \epsilon$. The sum of squares of these errors ϵ is minimized by solving the set of equations. Note that ϵ is related to the Euclidean distance of the end point u^i from the transferred line $\hat{\lambda}_i = \lambda'_j \lambda''_k T_i^{jk}$. In fact, according to (1) we have $\epsilon = (\hat{\lambda}_1^2 + \hat{\lambda}_2^2)^{1/2} d(\hat{\boldsymbol{\lambda}}, \mathbf{u})$. Ideally, we would like to minimize the geometric distance $d(\hat{\boldsymbol{\lambda}}, \mathbf{u})$ instead. This suggests an iterative weighting scheme. At the first iteration, the weight of the equation is unity. At subsequent iterations, the line equation (5) are weighted by $(\hat{\lambda}_1^2 + \hat{\lambda}_2^2)^{-1/2}$. One proceeds in this way for a small number of iterations or until the value of $(\hat{\lambda}_1^2 + \hat{\lambda}_2^2)^{1/2}$ converges to 1. In this latter case, the value of ϵ represents the residual geometric distance. A similar iterative method has been used in [3] for computation of the fundamental matrix. A preliminary application of this iterative method to the present problem was tried with good results, but more research is needed to determine if it makes a significant difference. In general, ways of weighting the line equations and the point equations to give equal importance to each is a possible subject of further research. The results reported in section 7 do not use an iterative weighting scheme.

Typical feature extractors used in computer vision applications extract lines by finding many points along the line. It is shown in Appendix C that in such cases, the data representing such lines may be reduced to a single pair of points.

Solution of the Equations. The set of equations we construct are of the form, $A\mathbf{t} = \mathbf{0}$, where A is the equation matrix and $\mathbf{t}$ is a vector containing the elements of T_i^{jk} to be found. We are not interested in the solution $\mathbf{t} = 0$, and to avoid ambiguity, we impose the constraint $||\mathbf{t}|| = 1$. Since we do not expect an exact solution, our task is to minimize the quantity $||A\mathbf{t}||$ subject to this constraint. The solution to this problem is easily seen (using Langrange multipliers, for instance) to be the unit eigenvector corresponding to the least eigenvalue of $A^\top A$. Being symmetric and positive definite, $A^\top A$ has only real positive eigenvalues. A good way to find this eigenvector is by using the Singular Value Decomposition ([19]). If one writes $A = UDV^\top$, where U and V are orthogonal, and D is diagonal, then the smallest singular value (diagonal element of D) is the square-root of the smallest eigenvalue of $A^\top A$ and the required eigenvector is the corresponding column of V.

We now briefly consider the time complexity of this algorithm. Computation of the Singular Value Decomposition (SVD) is the most time-intensive part of the algorithm for computing T_i^{jk}, since all other parts take linear of constant time. Approximate numbers of floating-point operations (flops) required to compute the SVD of an $m \times n$ matrix are given in [6]. To find matrices U, V and D, a total of $4m^2n + 8mn^2 + 9n^3$ flops are needed. However, if only the matrices V and D are required, as is the case here, then only $4mn^2 + 8n^3$ flops are required. This is an important distinction, since this latter expression does not contain any term in m^2. Table 1 gives the total number of flops for carrying out the SVD for seven (the minimum number) or 100 point correspondences and either four or nine equations per point.

# points	# equations per point	# equations (m)	# operations not computing U	# operations computing U
7	4	28	239,112	425,115
	9	63	341,172	973,215
100	4	400	1,323,864	19,789,947
	9	900	2,781,864	92,905,947

Table 1: *Comparison of the number of flops required to compute the SVD of a matrix of size $m \times n$, for varying values of m and for $n = 27$.*

It may be seen from the second-last column of the table that if U is not required (as is the case in solving for T_i^{jk}) then the difference between numbers of operations is not very significant whether four or nine equations per point are used. The difference is no more than a factor of 9/4 in the limit, and is substantially less for small problems. If however the matrix U is computed, then the number of flops is increased greatly – by more than 30 times when solving for 100 points. Thus, if speed is an issue, it is very important to use a version of the SVD that does not compute the matrix U. The SVD implementation in [19] is therefore not suitable as stands, though it may be easily modified. The recommended procedure, therefore, is to include all nine equations for each point, so as to get an expected increase in stability. However an implementation of the SVD should be used that does not compute the unneeded matrix U.

An alternative to using the SVD is to find the eigenvector of $A^\top A$ directly using the method of Jacobi ([19]). This method is theoretically inferior, since forming the product $A^\top A$ adversely affects the conditioning of the problem. However, my belief is that this does not make a significant difference, since the limiting factor is not the numerical stability of the algorithm, but the deleterious effect of measurement error.

5 Retrieving the Camera Matrices

Formula (3) gives a formula for the trifocal tensor T_i^{jk} in terms of the camera matrices. It is possible to go the other way and retrieve the camera matrices, M' and M'' from the tensor T_i^{jk}, assuming as ever that the first camera has matrix $M = [I \mid \mathbf{0}]$. This is done in two stages. In the first stage, one finds the vectors a_4^i and b_4^i (that is, the last columns of M' and M''). In the second stage, one finds the remaining entries.

We consider the first step first. For $i, j = 1, \ldots 3$ we denote by $\mathbf{e}_{ij}$ the epipole in the i-th image corresponding to the centre of the j-th camera. If the first camera has matrix $M = [I \mid \mathbf{0}]$, then as remarked previously, the epipoles $\mathbf{e}_{21}$ and $\mathbf{e}_{31}$ are the last columns a_4^i and b_4^i of the two camera matrices $M' = [a_j^i]$ and $M'' = [b_j^i]$ respectively. These two epipoles may easily be computed from the tensor T_i^{jk} according to the following proposition.

Proposition 5.2. *For each $i = 1, \ldots, 3$, the matrix $T_i^{\cdot\cdot}$ is singular. Furthermore, the generators of the three left null-spaces have a common perpendicular, the epipole $\mathbf{e}_{21}$. Similarly epipole e_{31} is the common perpendicular of the right nullspaces of the three matrices $T_i^{\cdot\cdot}$.*

This proposition is proven in Appendix C. Also in Appendix C, a closed form solution is given for determining the camera matrices in terms of the trifocal tensor. This closed form method of determining M' and M'' was evaluated in [11]. By carefully examining the results of that method, it has subsequently been found that using these formulae to determine M' and M'' is unstable in the presence of noise, and hence is **not** recommended for actual computation. Therefore, we consider two other techniques for retrieving the camera matrices.

5.1 The direct method

If a_4^i and b_4^i and T_i^{jk} are all known, the equations (3) form a redundant set of linear equations in the remaining entries of the matrices M' and M''. We may solve these equations using linear least-squares methods ([19, 2]) to find $M' = [a_j^i]$ and $M'' = [b_j^i]$. Unfortunately, this method does not seem to give markedly better results than the closed form solution. It is also **not** recommended. To see why this is so, we examine this method more closely.

The set of equations (3) may be written in the form $\mathbf{t} = H\mathbf{y}$ where $\mathbf{y}$ is the set of camera matrix entries that we are seeking to determine, and $\mathbf{t}$ is a vector consisting of the entries of the tensor T_i^{jk}. In the presence of noise, this set of equations does not have an exact solution, and one finds the least-squares solution instead. This can be done by using the Singular Value Decomposition ([19]) or by solving the normal equations $H^\top H\mathbf{y} = H^\top\mathbf{t}$. The result of this method is to find a vector $\mathbf{y}$ that minimizes the squared error $(\mathbf{t} - \hat{\mathbf{t}})^\top(\mathbf{t} - \hat{\mathbf{t}})$, where $\hat{\mathbf{t}} = H\mathbf{y}$.

The reason that this method gives poor results is that errors in the entries of $\mathbf{t}$ are considered as being independent and of equal importance. A better method is to weight the errors in the entries of $\mathbf{t}$ according to their inverse covariance matrix. More specifically, assuming that the coordinates of the point and line correspondences used to compute T_i^{jk} are independent gaussian random variables with equal variance, the estimated values of the entries of $\mathbf{t}$ have an induced inverse covariance matrix given by $A^\top A$ where $A\mathbf{t} = 0$ is the set of equations used to compute $\mathbf{t}$. One seeks a solution $\mathbf{y}$ minimizing the value of $(\mathbf{t} - \hat{\mathbf{t}})^\top (A^\top A)(\mathbf{t} - \hat{\mathbf{t}})$, where as before $\hat{\mathbf{t}} = H\mathbf{y}$. This can be done by solving the normal equations $H^\top (A^\top A)H\mathbf{y} = H^\top (A^\top A)\mathbf{t}$.

Solving this least-squares problem weighted by $A^\top A$ to find $\mathbf{y}$ (the entries of the camera matrices) could be expected to give substantially better results that the unweighted method. I did not try it, however, since the next method to be described uses the covariance matrix in a slightly different, but theoretically preferable manner.

5.2 The recomputation method

The method just described gave a way of extracting the camera matrices from the trifocal tensor. The method described next goes back to the original image measurements in order to compute the camera matrices. These are the same measurements that we used to compute the trifocal tensor. The difference is that now we assume that the the last columns a_4^i of M' and b_4^i of M'' are known. The camera matrices may now be computed in a single step.

The tensor T_i^{jk} is found by solving a system of equations $A\mathbf{t} = 0$ where $\mathbf{t}$ is a vector comprising the desired elements of T_i^{jk}. Furthermore, as in the direct computation method we may write $\mathbf{t} = H\mathbf{y}$ where $\mathbf{y}$ is the vector of the entries a_j^i and b_j^i that we are seeking, and H is the linear relationship expressed by (3). Putting these two equations together, one can find a solution for $\mathbf{y}$ directly by solving $AH\mathbf{y} = A\mathbf{t} = 0$. More exactly, one minimizes $||AH\mathbf{y}||$ subject to the constraint $||\mathbf{y}|| = 1$. The solution is the eigenvector corresponding to the least eigenvalue of $H^\top A^\top AH$. Writing $\hat{\mathbf{t}} = H\mathbf{y}$ where $\mathbf{y}$ is the solution vector, we see that $\hat{\mathbf{t}}$ minimizes $||A\hat{\mathbf{t}}||$. Hence $\hat{\mathbf{t}}$ is the best possible solution to the equations $A\hat{\mathbf{t}} = 0$ subject to the condition that $\hat{\mathbf{t}} = H\mathbf{y}$ for some $\mathbf{y}$.

Compare this with the original solution for T_i^{jk}, found by solving the equations $A\mathbf{t} = 0$. That original method finds a solution for the trifocal tensor, that in the presence of noise will not correspond exactly to a set of camera matrices. The method described in the last paragraph, however, finds a solution to the equations $A\hat{\mathbf{t}} = 0$ that is exactly realizable as a set of camera matrices. Furthermore, this method finds the optimal solution in terms of minimizing the algebraic distance to the observed data. The requirement of course is that one must know the epipoles a_4^i and b_4^i to be able to apply this method.

There is one small problem, which has the potential to cause an instability in this method as just described. In particular, since T_i^{jk} is unchanged by a projective transformation of the cameras, it may be verified that the values of a_j^i and b_j^i are not uniquely determined by T_i^{jk}. In particular, if a_j^i is replaced by $a_j^i + \alpha_j a_4^i$ for any vector α_j, and similarly b_j^i is replaced by $b_j^i + \alpha_j b_4^i$ for the same α_j, then the value of T_i^{jk} defined by equation (3) is unchanged. This means that the matrix H described above is not of full rank, and consequently the matrix $H^\top A^\top AH$ will have a multidimensional eigenspace corresponding to the eigenvalue 0. This means that the solution found will be unstable. This may not be a significant problem, since any solution found by this method will be a valid solution, giving one solution for the camera matrices and all such solutions correspond to the same adjusted trifocal tensor $\hat{T}_i^{jk}$. Nevertheless, I prefer to constrain the solution by adding constraints on the entries a_j^i and b_j^i so as to ensure a stable solution.

One method of constraining a_j^i is by specifying that $\sum_i a_j^i a_4^i = 0$. This gives three constraints, one for each value of $j = 1, \ldots, 3$, which may be interpreted as meaning that 4-th column of $M' = [a_j^i]$ is orthogonal to all the other columns. One may verify that this condition is achieved by a suitable choice of the vector α_j in the last paragraph (in particular, setting $\alpha_j = -\sum_i a_j^i a_4^i$). The condition $\sum_i a_j^i a_4^i = 0$ gives a set of three linear constraints.

The task of solving for a_j^i and b_j^i is therefore a constrained minimization problem of the form: minimize $||AH\mathbf{y}||$ subject to $||\mathbf{y}|| = 1$ and $C\mathbf{y} = 0$, where C is a matrix of linear constraints. This problem may be solved as follows. Let $C = UDV^\top$ be the Singular Value Decomposition of C, where D is a diagonal matrix with r non-zero diagonal entries. In this case, $r = 3$. We ensure that the non-zero diagonal entries of D precede the zero entries. Writing $\hat{\mathbf{y}} = V^\top \mathbf{y}$, the problem may be written : minimize $||AHV\hat{\mathbf{y}}||$ subject to $||V\hat{\mathbf{y}}|| = 1$ and $UD\hat{\mathbf{y}} = 0$. This last constraint is equivalent to $D\hat{\mathbf{y}} = 0$, and is satisfied only if $\hat{\mathbf{y}}$ has r leading zeros. In this case, we may write $\mathbf{y} = V\hat{\mathbf{y}} = V'\mathbf{y}'$, where $\mathbf{y}'$ is the vector formed from $\hat{\mathbf{y}}$ by omitting the leading r zeros, and V' is formed from V by omitting the first r columns. The minimization problem then becomes : minimize $||AHV'\mathbf{y}'||$ subject to $||V'\mathbf{y}'|| = 1$. Since V is orthogonal, the condition $||V'\mathbf{y}'|| = 1$ is equivalent to $||\mathbf{y}'|| = 1$. Thus, the problem has been reduced to our usual minimization problem for which the solution $\mathbf{y}'$ is the eigenvector corresponding to the minimum eigenvalue of $V'^\top H^\top A^\top AHV'$. Finally, $\mathbf{y}$ is computed as $\mathbf{y} = V'\mathbf{y}'$. We summarize now the algorithm for computing the camera matrices.

Algorithm for computing camera-matrices.

1. Compute the trifocal tensor T_i^{jk} by solving a set of equations $A\mathbf{t} = \mathbf{0}$ where $\mathbf{t}$ is the vector containing the entries of T_i^{jk}. Retain the matrix A.
2. Find a_4^i and b_4^i as the common normal to the left (respectively, right) null spaces of the three matrices $T_1^{\cdot\cdot}$, $T_2^{\cdot\cdot}$ and $T_3^{\cdot\cdot}$.
3. Compute the set of equations $\mathbf{t} = H\mathbf{y}$ from (3), where $\mathbf{y}$ is the vector of still unknown entries of a_j^i and b_j^i.
4. Compute the matrix C such that $C\mathbf{y} = 0$ expresses the three constraints $\sum_{i=1}^3 a_j^i a_4^i = 0$.
5. Compute the Singular Value Decomposition $C = UDV^\top$ where diagonal entries of D are sorted with non-zero ones first. Let V' be the matrix obtained from V by eliminating the first three columns.
6. Find the eigenvector $\mathbf{y}'$ corresponding to the smallest eigenvalue of $V'^\top H^\top A^\top AHV'$ either directly by using Jacobi's method ([19]) or by computing the Singular Value Decomposition of AHV'.
7. The required set of 18 values a_j^i and b_j^i are contained in the vector $\mathbf{y} = V'\mathbf{y}'$.

Since this algorithm is more complicated than either of the two previous algorithms (closed-form solution, and direct linear solution) it is comforting to verify that it performs very much better (in terms of measured errors) than they do. Therefore, the two earlier algorithms should not be used to compute the camera matrices.

6 Algorithm Outline

To tie together all the threads of the reconstruction algorithm, an outline will now be given. As input to this algorithm, we assume as set of point-to-point image correspondences, each one of the form $\mathbf{u} \leftrightarrow \mathbf{u}' \leftrightarrow \mathbf{u}''$, and a set of line correspondences of the form $\boldsymbol{\lambda} \leftrightarrow \boldsymbol{\lambda}' \leftrightarrow \boldsymbol{\lambda}''$, where $\#$ lines $+ 2 * \#$ points ≥ 13. The lines are assumed to be specified by giving the two end points of each line. The steps of the algorithm are as follows.

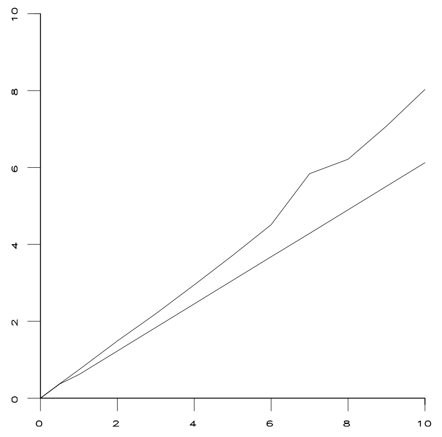
1. **Coordinate scaling and translation.** For each image separately, translate and scale the coordinates of the points such that the centroid of all given coordinates is at the origin, and the average distance of a point from the origin is $\sqrt{2}$.
2. **Computing and normalizing the lines.** Each line $\boldsymbol{\lambda}'$ and $\boldsymbol{\lambda}''$ is computed from its endpoints, and normalized.
3. **Construct the equations.** For each line correspondence, construct a pair of equations of the form (5) in the entries of the tensor T_i^{jk}. Similarly, for each point correspondence, construct nine equations of the form (8) also in the entries of tensor T_i^{jk}.
4. **Computation of the Camera Matrices.** Compute the camera matrices using the algorithm of section 5.2.
5. **Reconstruction.** Given the camera matrices, the points may be reconstructed using the triangulation methods of [12]. The lines may be reconstructed by computing the intersection of the planes in space defined by the image lines.
6. **Unscaling** The effect of the initial scaling and translation of the images may be undone by replacing the image coordinates by their original values, and making a corresponding adjustment to the computed camera matrices.

7 Experimental Evaluation of the Algorithm

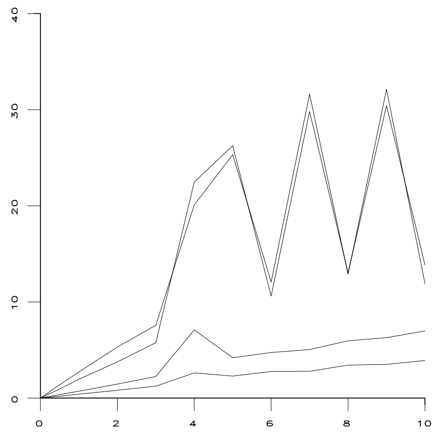
This algorithm has been tested with synthetic and real data. First I describe the tests with synthetic data. A program was written to generate synthetic data, approximating the sort of data that would be taken with a 35mm camera with a standard 50mm lens. The images measured about 600×600 pixels. Between 0 and 10 pixels of noise was added to all three images.

The findings of the experiments will be given in the captions of the following graphs. The error is given in terms of residual 2D error, that is the root-mean-squared (RMS) distance between the original points and the projection of the points obtained by reconstruction. Although this gives only an indirect measurement of the quality of the reconstruction, if the error is of the same order as the injected noise, then this is a good indication that the reconstruction is very good. The graphs all show injected noise on the horizontal axis, and residual error on the vertical axis.

In order to help to judge the quality of the results obtained using the linear algorithm, the outputs of the linear algorithm were used to initialize a full-scale iterative least-squares minimization routine based on Levenberg-Marquardt minimization ([19]). In this iterative estimation step, the camera matrices M' and M'' were allowed to vary as were the 3D locations of the reconstructed points and lines. The error term being minimized was a sum of contributions from all three images. In each image, the error associated with each point was the square of the distance of each measured image point from the reprojected space point. The error associated with each line was the sum of squares of perpendicular distances of the reprojected line from the two points used to define that line. Apart from the possibility of converging to a local minimum, this algorithm should give optimal results, and hence may be used as a yardstick to judge the performance of the linear algorithm. As will be seen in the examples to follow, the linear algorithm performs very favourably compared with the iterative algorithm.



Graph 1 : Residual error for 10 points. *This algorithm shows the results of reconstruction with ten points in three views. Residual error is plotted as a function of injected noise. The upper curve is the linear algorithm, and the lower curve is the iterative refinement. The residual errors are substantially smaller than the injected noise. The algorithm works well with as much as 10 pixels noise in each axial direction.*



Graph 2 : 7 points and 10 lines. *The curves show the results both of the iterative algorithm and those obtained after iterative refinement of the results, using a Levenberg-Marquardt iterative least-squares algorithm. The graphs from the bottom are iterative algorithm line errors, iterative algorithm point errors and then linear algorithm line errors and point errors overlapping. The results after refinement are excellent, with the residual error being substantially less than the injected noise. The linear algorithm does not do quite so well, but the results are acceptable, especially for noise levels less than four pixels.*

7.1 Real Images.

Next, we consider data obtained from a set of three images of a pair of wooden houses, shown in Fig 1. The dimension of the images was 640×480 pixels. The points and edges were extracted automatically and then matched between the images by hand. There were a set of 15 lines and 13 points identifiable in the two images. Since ground truth was not available for the real data, the errors reported below are for the residual error after reprojection of the constructed model into the images. The values reported are the average over all points and lines in three images.

1. For the linear algorithm : 1.05 pixels error for points, and 1.06 pixels for lines.
2. For the iterative refinement : 0.87 pixels for points, and 0.67 pixels for lines.

Thus, at least expressed in terms of residual error, we have achieved a very good reconstruction using the linear-algoritm, which was improved by about 25% using the iterative least-squares method.

In order to find whether these residual errors correspond to realistic levels of noise in the extraction of images features, a further experiment was run in which various levels of noise are injected into the real image data. The coordinates of the image features were first corrected so that the match corresponded precisely with the projection model. This was so that noise could subsequently be injected in a controlled manner. The result of these experiments showed that the actual residual errors found for the real uncorrected data correspond approximately to a a noise level of about 1 pixel in the coordinates of the

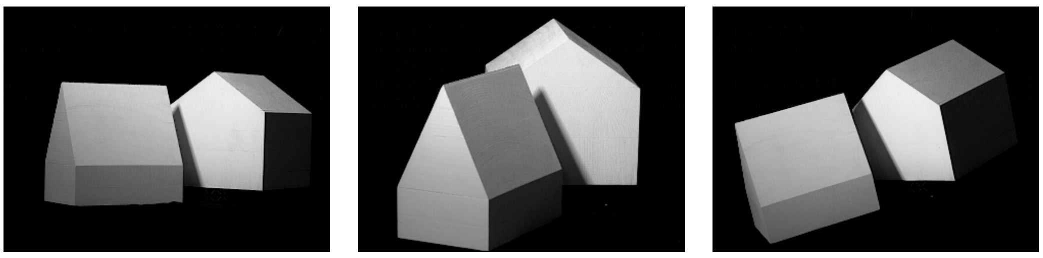
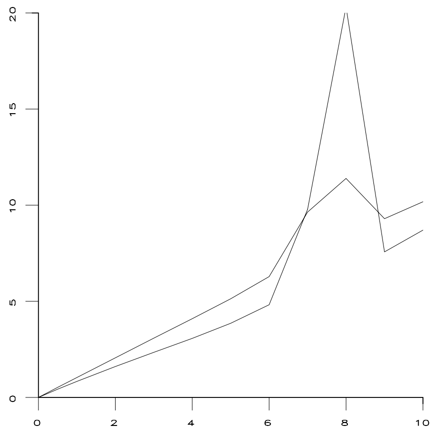
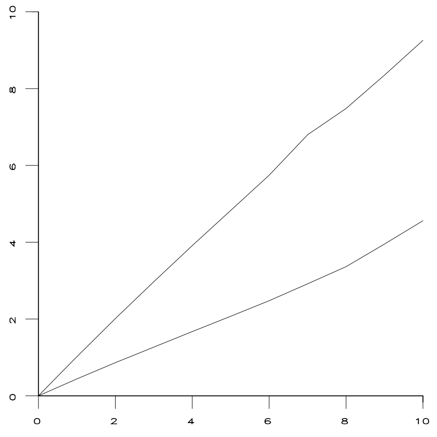


Figure 1: *Three photos of houses*

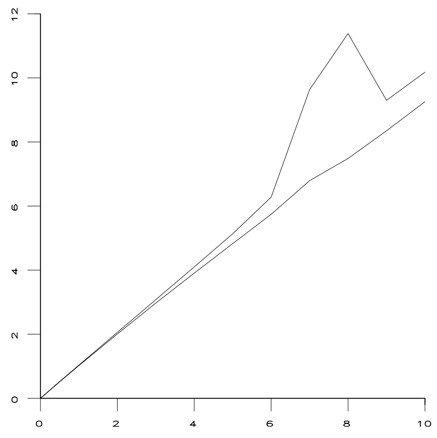
extracted features. This noise level is well below the break-down point of the algorithms, which performed well with up to 8 pixels of noise.



Graph 3 : Synthetic house data - linear. *This shows the residual error resulting from injection of data into the image measurements. The bottom curve shows the line error and the top curve, the point error. For the case of 8 pixels noise, the algorithm evidently hiccuped.*



Graph 4 : Synthetic house data - iterative *This shows the results obtained after refining the linear result using Levenberg-Marquardt. Once more, the bottom curve represents the errors for lines. The results are very good, and the glitch is removed.*



Graph 5 : Synthetic house data - comparison *This compares the point error for the iterative (lower) and linear methods. Except for the case of 8 pixels, where the linear algorithm deviated slightly, there is very little difference. This indicates that the linear algorithm is very nearly as good as the optimum least-squares solution.*

8 Conclusions

The algorithm described in this paper is unique in that no other linear method is known for handling both lines and points in a single non-iterative approach. The results obtained show indicate that this algorithm behaves very well in the presence of realistic amounts of noise. This algorithm has two advantages over algorithms (such as the 8-point algorithm of Longuet-Higgins) that use two views. Firstly, the presence of a third image can be expected to stabilize the solution (as remarked also in [22]), and secondly, the algorithm works for lines as well as points. Lines in images may often be computed with great precision. On the other hand, the algorithm benefits from the presence of points in the input data. The results obtained using this algorithm are much better than those I obtained previously with just lines. This is due to the use of mixed point and line data, and also to the improved algorithm for extracting the camera matrices, reported in section 5.

There are a few points that merit further investigation regarding this algorithm. The method for extracting the epipoles is probably the weak point of the algorithm, in that one must solve three consecutive linear equation sets in a row in order to find the epipoles : compute T_i^{jk}; compute the null-space of each $T_i^{\cdot\cdot}$; compute the normal to the null-spaces. This is probably susceptible to noise degradation. Possibly better methods of finding the epipoles given three views may be found. Shashua ([22]) gives a method for extracting the epipoles that may give better results, thought this has not been determined.

A second point regards the relative weighting of the line and point equations. The method of normalization of the data give some degree of standardization here, but since the two types of equations are derived separately, there is no obvious way to weight them so as to give equal emphasis to each sort of equation. At present, the algorithm seems to favour the lines, since residual line errors typically are smaller. Strategies for weighting the line and points equations are a subject of possible future research.

The trifocal tensor seems to be a basic object in the analysis of the three-view situation. Apart from the use here described in projective scene reconstruction, it has also been used for point transfer, object recognition ([20]) and Euclidean reconstruction ([1]). It is also suitable for line transfer as indicated in this paper. Just as the fundamental matrix neatly encapsulates the geometry of the two-view case, the trifocal tensor serves a similar purpose for three views.

Finally, the perception of T_i^{jk} as a tensor (properly due to Vieville [28] and Shashua) though perhaps only a notational device, eases the formalism involved in the analysis.

Appendix A : Covariant and Contravariant Tensors

Since tensor notation is not commonly used in computer vision, it seems appropriate to give a brief introduction to its use. For more details, the reader is referred to [26, 27]. For simplicity, these concepts will be explained here in the context of low-dimensional projective spaces, rather than in a general context. However, the ideas apply in arbitrary dimensional vector spaces.

Consider a set of basis vectors $\mathbf{e}_i; i = 1, \ldots, 3$ for a 2-dimensional projective space $\mathcal{P}^2$. For reasons to become clear, we will write the indices as subscripts. With respect to this basis, a point in $\mathcal{P}^2$is represented by a set of coordinates u^i, which represents the point $\sum_{i=1}^{3} u^i \mathbf{e}_i$. We write the coordinates with an upper index, as shown. Let $\mathbf{u}$ represent the triple of coordinates, $\mathbf{u} = (u^1, u^2, u^3)^\top$.

Now, consider a change of coordinate axes in which the basis vectors $\mathbf{e}_i$ are replaced by a new basis set $\hat{\mathbf{e}}_j$, where $\hat{\mathbf{e}}_j = \sum_i H_j^i \mathbf{e}_i$, and H is the basis transformation matrix with entries H_j^i. If $\hat{\mathbf{u}} = (\hat{u}^1, \hat{u}^2, \hat{u}^3)^\top$ are the coordinates of the vector with respect to the new basis, then we may verify that $\hat{u} = H^{-1}\mathbf{u}$. Thus, if the basis vectors transform according to H the coordinates of points transform according to the inverse transform H^{-1}.

Next, consider a line in $\mathcal{P}^2$represented by coordinates $\boldsymbol{\lambda}$ with respect to the original basis. With respect to the new basis, it may be verified that the line is represented by a new set of coordinates $\hat{\boldsymbol{\lambda}} = H^\top \boldsymbol{\lambda}$. Thus coordinates of the line transform according to $H^\top$.

As a further example, let P be a matrix representing a mapping between projective (or vector) spaces. If G and H represent basis-transformations in the domain and range spaces, then with respect to the new bases, the mapping is represented by a new matrx $\hat{P} = H^{-1}PG$. Note in these examples, that sometimes the matrix H or $H^\top$ is used in the transformation, and sometimes H^{-1}.

These three examples of coordinate transformations may be written compactly as follows.

$$\hat{u}^i = (H^{-1})^i_j u^j \quad ; \quad \hat{\lambda}_i = H_i^j \lambda_j \quad ; \quad \hat{P}_j^i = (H^{-1})^i_k G^l_j P^k_l \quad ,$$

where we use the tensor summation convention that an index repeated in upper and lower positions in a product represents summation over the range of the index. Note that those indices that are written as superscripts transform according to H^{-1}, whereas those that are written as subscripts transform as H (or G). Note that there is no distinction between indices that are transformed by H, and those that are transformed by $H^\top$ using this tensor notation. In general, tensor indices will transform either by H or H^{-1} – in fact this is the characteristic of a tensor. Those indices that transform according to H are known as *covariant* indices and are written as subscripts. Those indices that transform according to H^{-1} are known as *contravariant* indices, and are written as superscripts.

In this paper we introduced the trifocal tensor T_i^{jk}. Note that this tensor has one covariant and two contravariant indices. This implies a transformation rule :

$$\hat{T}_i^{jk} = F_i^r (G^{-1})_s^j (H^{-1})_t^k T_r^{st} \tag{12}$$

with respect to changes of basis in the three images. This transformation rule is easily deduced from (4). It is worth-while pointing out one possible source of confusion here. The transformation rule (12) shows how the tensor is transformed in terms of *basis* transformations in the three images. Often, we are concerned instead with point coordinate transformations. Thus, if F', G' and H' represent *coordinate* transformations in the images, in the sense that $\hat{u}^j = F_i'^j u^i$, and G' and H' are similarly defined for the other images, then the transformation rule may be written as $\hat{T}_i^{jk} = (F'^{-1})_i^r G_s'^j H_t'^k T_r^{st}$.

Note that for vectors and matrices, such as u^i, λ_i and P_j^i, it is possible to write the transfomation rules using standard linear algebra notation, as was done above. For tensors with three or more indices, this can not conveniently be done. This is an argument for the use of the tensor notation when dealing with the trifocal tensor.

Appendix B : Closed Form Expression for the Camera Matrices.

Formula (3) gives a formula for the tensor T_i^{jk} in terms of the camera matrices. We show in this appendix that conversely, it is possible to express the camera matrices M_i as a closed-form expression in terms of the tensor T_i^{jk}. We will also derive closed-form expressions for fundamental matrices in terms of the trifocal tensor. It will be assumed in this discussion that the rank of each of the matrices $T_i^{\cdot\cdot}$ is at least 2, which will be the case except in certain special camera configurations. There are methods of proceeding in case one or more of the $T_i^{\cdot\cdot}$ has rank one, but we omit any further consideration of these cases. See [29] for the a discussion of methods applying to calibrated cameras. For general camera configurations all $T_i^{\cdot\cdot}$ have rank 2.

In this section, it is convenient to use standard vector notation. Accordingly, we write T_i to mean the matrix $T_i^{\cdot\cdot}$. The camera matrices $M' = [a_j^i]$ and $M'' = [b_j^i]$ will be written as $M' = [A \mid \mathbf{a}_4]$ and $M'' = [B \mid \mathbf{b}_4]$, where $\mathbf{a}_4$ and $\mathbf{b}_4$ are the fourth columns of the respective matrices. For $i = 1, \ldots, 3$, vectors $\mathbf{a}_i$ and $\mathbf{b}_i$ are the i-th columns of the camera matrices. Camera matrix M is assumed as ever to be of the form $M = [I \mid \mathbf{0}]$.

Equation (3) may be written using this notation, as follows.

$$T_i = \mathbf{a}_i \mathbf{b}_4{}^\top - \mathbf{a}_4 \mathbf{b}_i{}^\top \ . \tag{13}$$

Let F_{12} and F_{13} represent the fundamental matrices corresponding to camera pairs (M, M') and (M, M'') respectively. These fundamental matrices have a very simple expression in terms of the camera matrices, as follows ([10]) :

$$F_{12} = \mathbf{a}_4 \times A \qquad ; \qquad F_{13} = \mathbf{b}_4 \times B \tag{14}$$

where notation such as $\mathbf{a}_4 \times A$ means the matrix made up by forming the vector product of $\mathbf{a}_4$ with each of the columns of A separately. We now prove Proposition 5.2.

Proof. Referring to (13), we note that $(\mathbf{a}_4 \times \mathbf{a}_i)^\top T_i = 0$ since $(\mathbf{a}_4 \times \mathbf{a}_i)^\top \mathbf{a}_i = (\mathbf{a}_4 \times \mathbf{a}_i)^\top \mathbf{a}_4 = 0$. It follows that we can compute $\mathbf{a}_4 \times \mathbf{a}_i$ up to an unknown multiplicative factor by finding the null-space of T_i for each $i = 1, \ldots, 3$. However, by (14) we see that $\mathbf{a}_4 \times A = (\mathbf{a}_4 \times \mathbf{a}_1, \mathbf{a}_4 \times \mathbf{a}_2, \mathbf{a}_4 \times \mathbf{a}_3)$ is the fundamental matrix for cameras 0 and 1, and hence has rank 2. It follows that $\mathbf{a}_4$ (or $-\mathbf{a}_4$) may be computed as the unique unit vector normal to all of $\mathbf{a}_4 \times \mathbf{a}_i$ for $i = 1, \ldots, 3$. The vector $\mathbf{b}_4$ is found in a similar manner. □

Once we have computed $\mathbf{a}_4$ and $\mathbf{b}_4$, the computation of the fundamental matrices is easy, according to the following formulae :

$$\begin{aligned} F_{12} &= (\mathbf{a}_4 \times \mathbf{a}_1, \mathbf{a}_4 \times \mathbf{a}_2, \mathbf{a}_4 \times \mathbf{a}_3) = (\mathbf{a}_4 \times T_1\mathbf{b}_4, \mathbf{a}_4 \times T_2\mathbf{b}_4, \mathbf{a}_4 \times T_3\mathbf{b}_4) \\ F_{13} &= (\mathbf{b}_4 \times \mathbf{b}_1, \mathbf{b}_4 \times \mathbf{b}_2, \mathbf{b}_4 \times \mathbf{b}_3) = -(\mathbf{b}_4 \times {T_1}^\top \mathbf{a}_4, \mathbf{b}_4 \times {T_2}^\top \mathbf{a}_4, \mathbf{b}_4 \times {T_3}^\top \mathbf{a}_4) \ . \end{aligned} \tag{15}$$

Next, we derive formulae for the camera matrices M' and M''. To do this, we make use of the assumption also used in section 5.2 that camera matrix M' is normalized such that ${\mathbf{a}_4}^\top \mathbf{a}_i = 0$ for each $i = 1, \ldots, 3$. We also assume that M' and M'' are scaled such that ${\mathbf{a}_4}^\top \mathbf{a}_4 = {\mathbf{b}_4}^\top \mathbf{b}_4 = 1$. This is a valid assumption if we assume that neither or the cameras M' and M'' is located at the same point as the first camera, namely the origin. With these assumption one verifies that ${\mathbf{a}_4}^\top T_i = -{\mathbf{b}_i}^\top$. This means that $M'' = (B \mid \mathbf{b}_4) = (-{T_1}^\top \mathbf{a}_4, -{T_2}^\top \mathbf{a}_4, -{T_3}^\top \mathbf{a}_4 \mid \mathbf{b}_4)$. Furthermore, substituting ${\mathbf{b}_i}^\top = -{\mathbf{a}_4}^\top T_i$ into the formula $T_i = \mathbf{a}_i {\mathbf{b}_4}^\top - \mathbf{a}_4 {\mathbf{b}_i}^\top$ and multiplying by $\mathbf{b}_4$ gives $T_i \mathbf{b}_4 = \mathbf{a}_i + \mathbf{a}_4 {\mathbf{a}_4}^\top T_i \mathbf{b}_4$, from which one derives $\mathbf{a}_i = (I - \mathbf{a}_4 {\mathbf{a}_4}^\top) T_i \mathbf{b}_4$. This gives the following formulae for the camera matrices.

$$\begin{aligned} M' &= (A \mid \mathbf{a}_4) &&= ((I - \mathbf{a}_4 {\mathbf{a}_4}^\top) T_1 \mathbf{b}_4, (I - \mathbf{a}_4 {\mathbf{a}_4}^\top) T_2 \mathbf{b}_4, (I - \mathbf{a}_4 {\mathbf{a}_4}^\top) T_3 \mathbf{b}_4 \mid \mathbf{a}_4) \\ M'' &= (B \mid \mathbf{b}_4) &&= (-{T_1}^\top \mathbf{a}_4, -{T_2}^\top \mathbf{a}_4, -{T_3}^\top \mathbf{a}_4 \mid \mathbf{b}_4) \end{aligned} \tag{16}$$

The correctness of this formula relies on the fact that T_i is of the form $T_i = \mathbf{a}_i {\mathbf{b}_4}^\top - \mathbf{a}_4 {\mathbf{b}_i}^\top$. In other words, if one computes M' and M'' from the T_i using (16) and then recomputes T_i using (3) then one does not retrieve the same values of T_i unless T_i is of the correct form.

Appendix C : Lines specified by several points

In describing the reconstruction algorithm from lines, we have considered the case where lines are specified by their two end-points. Another common way that lines may be specified in an image is as the best line fit to several points. It will be shown now how that case may easily be reduced to the case of a line defined by two end points. Consider a set of points $\mathbf{u}_i$ in an image, normalized to have third component equal to 1. Let $\boldsymbol{\lambda} = (\lambda_1, \lambda_2, \lambda_3)^\top$ be a line, which we suppose is normalized such that $\lambda_1^2 + \lambda_2^2 = 1$. In this case, according to (1) the distance from a point $\mathbf{u}_i$ to the line $\boldsymbol{\lambda}$ is equal to ${\mathbf{u}_i}^\top \boldsymbol{\lambda}$. The squared distance may be written as $d^2 = \boldsymbol{\lambda}^\top \mathbf{u}_i {\mathbf{u}_i}^\top \boldsymbol{\lambda}$, and the sum-of-squares of all distances is

$$\sum_i \boldsymbol{\lambda}^\top \mathbf{u}_i {\mathbf{u}_i}^\top \boldsymbol{\lambda} = \boldsymbol{\lambda}^\top (\sum_i \mathbf{u}_i {\mathbf{u}_i}^\top) \boldsymbol{\lambda} \ .$$

The matrix $E = (\sum_i \mathbf{u}_i {\mathbf{u}_i}^\top)$ is positive-definite and symmetric.

Lemma. *Matrix* $(E - \xi_0 J)$ *is positive semi-definite, where* J *is the matrix* $\mathrm{diag}(1,1,0)$ *and* ξ_0 *is the smallest solution to the equation* $\det(E - \xi J) = 0$.

Proof. We start by computing the vector $\mathbf{x} = (x_1, x_2, x_3)^\top$ that minimizes $\mathbf{x}^\top E\mathbf{x}$ subject to the condition $x_1^2 + x_2^2 = 1$. Using the method of Lagrange multipliers, this comes down to minimizing $\mathbf{x}^\top E\mathbf{x} - \xi(x_1^2 + x_2^2)$, where ξ denotes the Lagrange coefficient. Taking the derivative with respect to $\mathbf{x}$ and setting it to zero, we find that $2E\mathbf{x} - \xi(2x_1, 2x_2, 0)^\top = 0$. This may be written as $(E - \xi J)\mathbf{x} = 0$. It follows that ξ is a root of the equation $\det(E - \xi J) = 0$ and $\mathbf{x}$ is the generator of the null space of $E - \xi J$. Since $\mathbf{x}^\top E\mathbf{x} = \xi\mathbf{x}^\top J\mathbf{x} = \xi(x_1^2 + x_2^2) = \xi$, it follows that to minimize $\mathbf{x}^\top E\mathbf{x}$ one must choose ξ to be the minimum root ξ_0 of the equation $\det(E - \xi J) = 0$. In this case one has $\mathbf{x}_0{}^\top E\mathbf{x}_0 - \xi_0 = 0$ for the minimizing vector $\mathbf{x}_0$. For any other vector $\mathbf{x}$, not necessarily the minimizing vector, one has $\mathbf{x}^\top E\mathbf{x} - \xi_0 \geq 0$. Then, $\mathbf{x}^\top(E - \xi_0 J)\mathbf{x} = \mathbf{x}^\top E\mathbf{x} - \xi_0 \geq 0$, and so $E - \xi_0 J$ is positive semi-definite. □

Since the matrix $E - \xi_0 J$ is symmetric it may be written in the form $E - \xi_0 J = V\mathrm{diag}(r, s, 0)V^\top$ where V is an orthogonal matrix and r and s are positive. It follows that

$$\begin{aligned} E - \xi_0 J &= V\mathrm{diag}(r,0,0)V^\top + V\mathrm{diag}(0,s,0)V^\top \\ &= r\mathbf{v}_1\mathbf{v}_1{}^\top + s\mathbf{v}_2\mathbf{v}_2{}^\top \end{aligned}$$

where $\mathbf{v}_i$ is the i-th column of V. Therefore $E = \xi_0 J + r\mathbf{v}_1\mathbf{v}_1{}^\top + s\mathbf{v}_2\mathbf{v}_2{}^\top$. Then for any line $\boldsymbol{\lambda}$ satisfying $\lambda_1^2 + \lambda_2^2 = 1$ we have

$$\begin{aligned} \sum_i (\mathbf{u}_i{}^\top\boldsymbol{\lambda})^2 &= \boldsymbol{\lambda}^\top E\boldsymbol{\lambda} \\ &= \xi_0 + r(\mathbf{v}_1{}^\top\boldsymbol{\lambda})^2 + s(\mathbf{v}_2{}^\top\boldsymbol{\lambda})^2 \ . \end{aligned}$$

Thus, we have replaced the sum-of-squares of several points by a constant value ξ_0, which is not capable of being minimized, plus the weighted sum-of-squares of the distances to two points $\mathbf{v}_1$ and $\mathbf{v}_2$.

References

[1] Martin Armstrong, Andrew Zisserman, and Richard I. Hartley. Self-calibration from image triplets. In *Computer Vision - ECCV '96, Volume I, LNCS-Series Vol. 1064, Springer-Verlag*, pages 3 – 16, 1996.

[2] K.E. Atkinson. *An Introduction to Numerical Analysis, 2nd Edition.* John Wiley and Sons, New York, 1989.

[3] P. A. Beardsley, A. Zisserman, and D. W. Murray. Sequential update of projective and affine structure from motion. Report OUEL 2012/94, Oxford University, 1994. To appear in IJCV.

[4] O. D. Faugeras. What can be seen in three dimensions with an uncalibrated stereo rig? In *Computer Vision - ECCV '92, LNCS-Series Vol. 588, Springer-Verlag*, pages 563 – 578, 1992.

[5] Olivier Faugeras and Bernard Mourrain. On the geometry and algebra of the point and line correspondences between N images. In *Proc. International Conference on Computer Vision*, pages 951 – 956, 1995.

[6] Gene H. Golub and Charles F. Van Loan. *Matrix Computations, Second edition.* The Johns Hopkins University Press, Baltimore, London, 1989.

[7] R. Hartley, R. Gupta, and T. Chang. Stereo from uncalibrated cameras. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition*, pages 761–764, 1992.

[8] R. I. Hartley. Camera calibration using line correspondences. In *Proc. DARPA Image Understanding Workshop*, pages 361–366, 1993.

[9] R. I. Hartley. In defence of the 8-point algorithm. In *Proc. International Conference on Computer Vision*, pages 1064 – 1070, 1995.

[10] Richard I. Hartley. Projective reconstruction and invariants from multiple images. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 16:1036–1041, October 1994.

[11] Richard I. Hartley. Projective reconstruction from line correspondences. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition*, pages 903–907, 1994.

[12] Richard I. Hartley and Peter Sturm. Triangulation. In *Proc. ARPA Image Understanding Workshop*, pages 957–966, 1994.

[13] Anders Heyden. *Geometry and Algebra of Multiple Projective Transformations.* PhD thesis, Department of Mathematics, Lund University, Sweden, December 1995.

[14] Anders Heyden. Reconstruction from image sequences by means of relative depth. In *Proc. International Conference on Computer Vision*, pages 1058 – 1063, 1995.

[15] Anders Heyden. Reconstruction from multiple images using kinetic depths. Technical Report ISRN LUFTD2/TFMA-95/7003-SE, Department of Mathematics, Lund University, 1995.

[16] B. K. P. Horn. Relative orientation. *International Journal of Computer Vision*, 4:59 – 78, 1990.

[17] B. K. P. Horn. Relative orientation revisited. *Journal of the Optical Society of America, A*, Vol. 8, No. 10:1630 – 1638, 1991.

[18] H.C. Longuet-Higgins. A computer algorithm for reconstructing a scene from two projections. *Nature*, 293:133–135, Sept 1981.

[19] William H. Press, Brian P. Flannery, Saul A. Teukolsky, and William T. Vetterling. *Numerical Recipes in C: The Art of Scientific Computing.* Cambridge University Press, 1988.

[20] Amnon Shashua. Trilinearity in visual recognition by alignment. In *Computer Vision - ECCV '94, Volume I, LNCS-Series Vol. 800, Springer-Verlag*, pages 479–484, 1994.

[21] Amnon Shashua. Algebraic functions for recognition. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 17(8):779–789, August 1995.

[22] Amnon Shashua and Michael Werman. Trilinearity of three perspective views and its associated tensor. In *Proc. International Conference on Computer Vision*, pages 920 – 925, 1995.

[23] Minas E. Spetsakis. A linear algorithm for point and line-based structure from motion. *CVGIP: Image Understanding*, Vol. 56, No. 2:230– 241, September, 1992.

[24] Minas E. Spetsakis and John Aloimonos. Structure from motion using line correspondences. *International Journal of Computer Vision*, 4:3:171–183, 1990.

[25] Minas E. Spetsakis and John Aloimonos. A unified theory of structure from motion. In *DARPA IU Proceedings*, pages 271 – 283, 1990.

[26] Bill Triggs. The geometry of projective reconstruction I: Matching constraints and the joint image. unpublished report, 1995.

[27] Bill Triggs. Matching constraints and the joint image. In *Proc. International Conference on Computer Vision*, pages 338 – 343, 1995.

[28] T. Viéville and Q.T. Luong. Motion of points and lines in the uncalibrated case. Report RR-2054, INRIA, 1993.

[29] J. Weng, T.S. Huang, and N. Ahuja. Motion and structure from line correspondences: Closed-form solution, uniqueness and optimization. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, Vol. 14, No. 3:318–336, March, 1992.

[30] Michael Werman and Amnon Shashua. The study of 3D-from-2D using elimination. In *Proc. International Conference on Computer Vision*, pages 473 – 479, 1995.