Source Location from Single and Multiple Video Streams

Seenu S. Reddi
ReddiSS@AOL.COM
01/12/2004

This document is concerned with algorithms and techniques for videogrammetry, a term coined and derived from photogrammetry. Videogrammetry can be broadly defined as the area in which measurements are made from video streams and is an extension to photogrammetry primarily concerned with measurements from photographs. Photogrammetry is used in aerial and satellite imaging for topographical and geographical measurements and surveying. With the proliferation of video camera technology and availability of inexpensive digital cameras, it is of interest to develop applications in every day life like traffic monitoring and accident investigation with cameras mounted at intersections, surveillance with closed circuit TV, baseball and cricket ball tracking in athletics, and robotic vision. Some techniques presented here are believed to be new as they are based on recent results obtained in range determination [1].

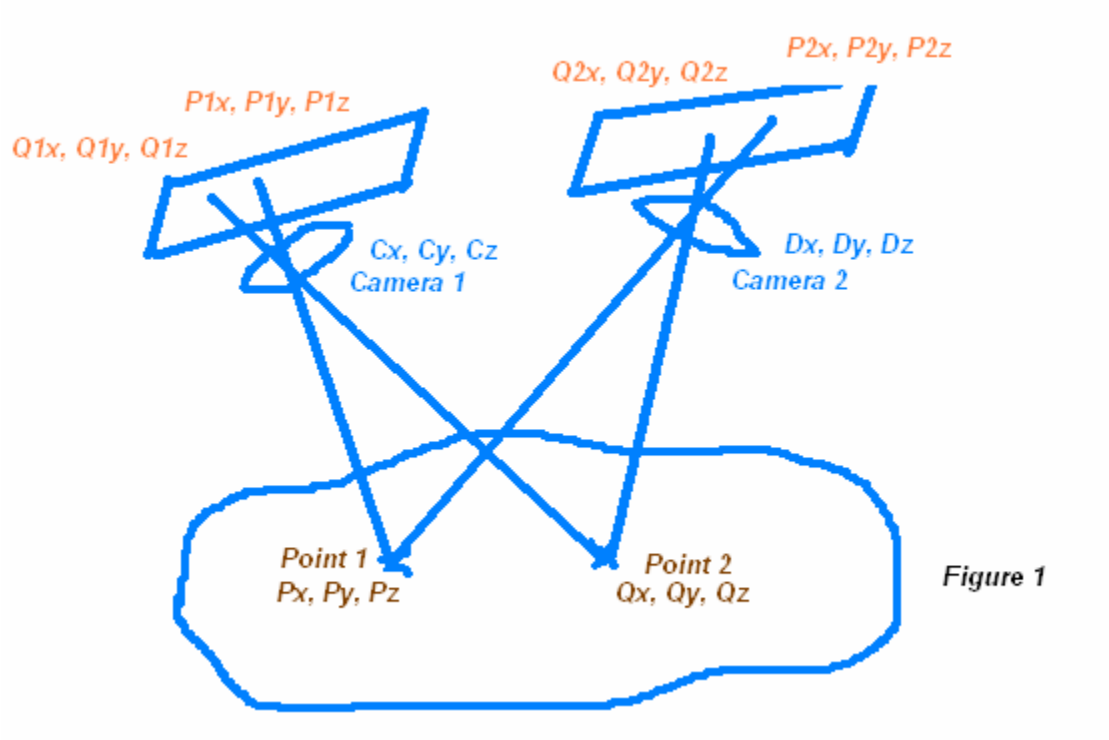

Figure 1

Figure 1 shows a general scenario of two cameras with two points (Px, Py, Pz) and (Qx, Qy, Qz) on the ground. The cameras are at coordinates (Cx, Cy, Cz) and (Dx, Dy, Dz) and the ground points are imaged at (P1x, P1y, P1y) and (Q1x, Q1y, Q1z) for the first point and (P2x, P2y, P2z) and (Q2x, Q2y, Q2z) for the second. (We keep the discussion

to two points and avoid subscripts/superscripts to avoid excessive notational confusion and keep the presentation easier to follow. Once the concepts are established, it is easy for the user to extend the concepts to multiple cameras, views and multiple targets). We first consider the situation where given the coordinates of the cameras and the images of a single point in the focal planes, how one determines the coordinates of the point on the ground. We have the following parametric equations for the lines of interest:

Line passing thru' (P1x, P1y, P1z) and (Cx, Cy, Cz): (1)

$$Px = Cx + r1 * (P1x - Cx),\ Py = Cy + r1 * (P1y - Cy),\ Pz = Cz + r1 * (P1z - Cz)$$

Line passing thru' (P1x, P1y, P1z) and (Dx, Dy, Dz): (2)

$$Px = Dx + r2 * (P2x - Dx),\ Py = Dy + r2 * (P2y - Dy),\ Pz = Dz + r2 * (P2z - Dz)$$

Here the unknowns are Px, Py, Pz, r1 and r2 and as we have six equations, one can solve for the least squared error solution using pseudoinverses.

$$\begin{bmatrix} 1 & 0 & 0 & -(P1x - Cx) & 0 \\ 0 & 1 & 0 & -(P1y - Cy) & 0 \\ 0 & 0 & 1 & -(P1z - Cz) & 0 \\ 1 & 0 & 0 & 0 & -(P2x - Dx) \\ 0 & 1 & 0 & 0 & -(P2y - Dy) \\ 0 & 0 & 1 & 0 & -(P2z - Dz) \end{bmatrix} \begin{bmatrix} Px \\ Py \\ Pz \\ r1 \\ r2 \end{bmatrix} = \begin{bmatrix} Cx \\ Cy \\ Cz \\ Dx \\ Dy \\ Dz \end{bmatrix} \quad (3)$$

Indicating the above matrix equation in a compact notational manner:

$$\mathbf{AP} = \mathbf{C} \quad (4)$$

where **A** is the matrix and **P** and **C** are column vectors. P can be solved in a least squared manner as:

$$\mathbf{P} = (\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T\mathbf{C} \quad (5)$$

from which we can get (Px, Py, Pz) the coordinates of the point. If we have in general n cameras, there will be n + 3 unknowns and 3n equations. Equation (5) will involve finding the (pseudo) inverse of a (n + 3) X (n + 3) matrix.

Note there are practical difficulties associated with the above procedure. The points have to be well marked in the photographs of each camera so that the equations can be set up. To recognize the point, some sort of correlation procedure has to be invoked and there are always errors due to pixel quantization, camera orientation and precision. We introduce the term *target marking* to denote the process of identifying the same target in different snapshots from different cameras and/or at different time fames.

Sometimes target marking is facilitated the nature of target. For instance in scenarios where cricket ball or baseball is tracked, since the shape of the ball is known a priori and the target appearance is usually distinct from its surroundings (ball is colored white or

red) target tracking is easier by detecting the centroid of the circular objects present in the imagery. However in robotic vision where the robot is guided by tracking objects with irregular shapes or traffic monitoring where the vehicle shapes are fuzzy and not well-defined, target marking may be more complex and lead to errors.

Now we consider situation where there are landmarks on the ground whose coordinates are known and the question is whether by examining the target with respect to these landmarks in each snapshot, the target's coordinates can be estimated. There are certain advantages to this method since it eliminates inter-correlation errors that may exist in correlating different snapshots and increases the number of measurements to the target thereby resulting in better target estimates. To be more concrete, consider a single camera situation tracking a target. The only measurement is the position of the target within the snapshot the camera takes. However if there are landmarks say five we will have five measurements to the target and if we can combine these five measurements to estimate target position, there will be better accuracy.

We show now how to combine these measurements to get target coordinates in a single camera situation with multiple known landmarks. Crucial to our development is the following theorem.

Theorem: Consider a three-dimensional system (not necessarily Cartesian) where each point is specified by means of three coordinates (x, y, z). Assume there are four or more reference points (or landmarks) r0, r1, r2, r3, r4, .., rn whose coordinates are known and without loss of generality assume r0 = (0, 0, 0). For target t, Cartesian distances to these points s0, s1, s2, s3, s4, .., sn are known where:

$$si = \sqrt{(x_t - x_{ri})^2 + (y_t - y_{ri})^2 + (z_t - z_{ri})^2}$$

Here (x_t, y_t, z_t) and (x_{ri}, y_{ri}, z_{ri}) are the target and reference coordinates. If the n reference coordinates (x_{ri}, y_{ri}, z_{ri}) are written as a matrix:

$$\mathbf{A} = \begin{bmatrix} x_{r1} & y_{r1} & z_{r1} \\ x_{r2} & y_{r2} & z_{r2} \\ x_{r3} & y_{r3} & z_{r3} \\ . & . & . \\ x_{rn} & y_{rn} & z_{rn} \end{bmatrix}$$

has a rank of three, then the coordinates of target t can be found by forming a quadratic equation and solving it.

The proof is similar to the proof presented in reference 1 which is reproduced in Appendix 1. Note the proof in reference 1 assumes a Cartesian three-dimensional system but it does not really use this fact anywhere for the proof.

Now we proceed to demonstrate how this theorem can be used for target determination from the positions of observed landmarks in a snapshot from a single camera. With

reference to Fig. 1 we can write the following equations for two points (Px, Py, Pz) and (Qx, Qy, Qz):

$$\begin{aligned} P1x &= Cx + rP * (Px - Cx) \\ P1y &= Cy + rP * (Py - Cy) \\ P1z &= Cz + rP * (Pz - Cz) \end{aligned} \qquad (6)$$

$$\begin{aligned} Q1x &= Cx + rQ * (Px - Cx) \\ Q1y &= Cy + rQ * (Py - Cy) \\ Q1z &= Cz + rQ * (Pz - Cz) \end{aligned}$$

Note this is slightly different from Equation (1) since we set the parametric equations for the line passing thru' (Cx, Cy, Cz) and (Px, Py, Pz). The distance between the imaged points (P1x, P1y, P1z) and (Q1x, Q1y, Q1z) is given by:

$$\delta = \sqrt{(P1x - Q1x)^2 + (P1y - Q1y)^2 + (P1z - Q1z)^2} \qquad (7)$$

For each ground point (Px, Py, Pz) we introduce a pseudo-point defined by:

$$\begin{aligned} px &= rP * (Px - Cx) \\ py &= rP * (Py - Cy) \\ pz &= rP * (Pz - Cz) \end{aligned} \qquad (8)$$

and hence:

$$\delta = \sqrt{(px - qx)^2 + (py - qy)^2 + (pz - qz)^2} \qquad (9)$$

Thus by computing the distances between the imaged points, we can effectively compute the distance between its corresponding pseudo-points on the ground. Now we can apply the Theorem to compute the coordinates of the pseudo-point corresponding to the target on the ground by computing the distances between the imaged points and applying the algorithm delineated in Appendix 1. If the target has the coordinates (Tx, Tx, Tz), the pseudo-coordinates will be:

$$\begin{aligned} tx &= rT * (Tx - Cx) \\ ty &= rT * (Ty - Cy) \\ tz &= rT * (Tz - Cz) \end{aligned} \qquad (9)$$

From the determination of (tx, ty, tz) and eliminating rT, we will have two simultaneous linear equations with three variables Tx, Ty, Tz. This situation merely reinforces the fact that a true three dimensional picture cannot be deduced from a two-dimensional snapshot. However by taking consecutive snapshots temporally or simultaneous snapshots spatially, we can solve for the coordinates of the target as there will be more than two simultaneous linear equations specifying three variables.

We considered the imaged points as three-dimensional and the distances are also computed between three dimensional coordinates. However the images are two-dimensional and now we show that the two-dimensional distances (pixel distances) are the same as the three-dimensional distances in magnitude. Assume an x-axis and y-axis in the image plane with respect to which the two-dimensional distances are computed. We can rotate the X-Y-Z coordinates of the ground so that the its Y-axis is parallel to the assumed y-axis of the image plane. If the image plane is inclined at an angle θ to the X-Y plane, then X, Y and Z in the ground coordinates can be related to the image plane coordinates x and y:

$$\begin{aligned} X &= x \cos\theta + d_x \\ Y &= y + d_y \\ Z &= z \sin\theta + d_z \end{aligned} \tag{10}$$

where d_x , d_y and d_z are displacement constants. Define a matrix **D**:

$$\mathbf{D} = \begin{bmatrix} \cos\theta & 0 \\ 0 & 1 \\ \sin\theta & 0 \end{bmatrix} \tag{11}$$

The three dimensional distance Δ between two ground points separated by Δ_X, Δ_Y, and Δ_Z and the two-dimensional distance δ between its corresponding two image points separated by δ_x and δ_y are given by:

$$\begin{aligned} \Delta^2 &= \Delta_X^2 + \Delta_Y^2 + \Delta_Z^2 \\ \delta^2 &= \delta_x^2 + \delta_y^2 \end{aligned} \tag{12}$$

Define $\mathbf{\Delta}$ to be the column vector $[\Delta_X \ \Delta_Y \ \Delta_Z]^T$ and $\boldsymbol{\delta}$ column vector $[\delta_x \ \ \delta_y]^T$ and the following relation holds:

$$\mathbf{\Delta} = \mathbf{D}\boldsymbol{\delta} \tag{13}$$

from which :

$$\Delta^2 = \mathbf{\Delta}^T\mathbf{\Delta} = \boldsymbol{\delta}^T\mathbf{D}^T\mathbf{D}\boldsymbol{\delta} \tag{14}$$

Since $\mathbf{D}^T\mathbf{D} = \mathbf{I}$ it follows the three dimensional and two dimensional distances are the same.

The above procedure can be summarized as follows. Select landmarks (at least four) on the ground whose coordinates are known and compute the distance between the image point representing the target and the image points representing the landmarks. From these distances compute using the technique delineated in Appendix 1 (a computer program is given in Appendix 2), the target coordinates can be specified by means of three simultaneous equations. From different cameras and/or temporally separated and/or

viewpoint different snapshots from the same camera, generate more than two sets of three equations to solve in a least squared fashion for the coordinates of the target.

We conclude the paper by listing future venues of investigation and application areas. One of the interesting areas of videogrammetry has been the Hawk-Eye [2] and an interesting objective would be to eliminate radar speed trackers since they are very susceptible to direction in which the ball is traveling. Also mounting a few inexpensive digital cameras and providing an inexpensive processor for line calls in tennis courts will be welcome by most week-end tennis players. Traffic cameras can investigate accidents from the images captured by them and since there are landmarks, the presented techniques can be profitably used to determine vehicle speeds, and directions. Robotic vision might be other area where one can put specific landmarks (marked in a certain color to be easily picked up in images) to guide its motion. For surveillance, digital cameras can be positioned to track threats for homeland security.

[1] Reddi, S. S., "An Exact Solution to Range Computation with Time Delay Information for Arbitrary Array Geometries," IEEE Trans. Signal Processing, Jan. 1993.

[2] Hawk-Eye. Hawki.com and other internet resources (type Hawk-Eye Cricket in yahho search).

Appendix 1
To follow on Next Page

An Exact Solution to Range Computation with Time Delay Information for Arbitrary Array Geometries

Seenu S. Reddi

Abstract—**This correspondence presents an exact solution for the computation of the range of a target from the time delays observed at the receiver elements of an array with arbitrary geometry. The solution consists of solving a simple quadratic equation and hence can be implemented in real time for sonars and other target locating applications.**

I. Introduction

The range computation problem we solve in this correspondence can be stated as follows. There are $n + 1$ receivers located at coordinates $(0, 0, 0)$, (x_1, y_1, z_1), $\cdots$, (x_n, y_n, z_n) and there is a target at location (x_t, y_t, z_t). From the received emanations from the target, the time delays δt_i (computed by correlation or some other method) between the reference receiver at the origin and the ith receiver for the target signal are known. The problem is to compute (x_t, y_t, z_t) from δt_i and the geometry of the receivers. Though exact solutions have been given in the literature for linear geometries with three elements (for instance, see Hassab [1]), it is surprising that an exact solution can be found for arbitrary geometries. The computation (or estimation) of time delays from the received waveforms of the sensors will not be considered here, and the reader is referred to Hassab [1] for specific techniques of time delay estimation.

Manuscript received June 21, 1991; revised April 16, 1992.
The author is with Signal Research Laboratory, Irvine, CA 92714.
IEEE Log Number 9203364.

II. Approach

Consider Fig. 1 which shows a rectangular coordinate system with a target located at (x_t, y_t, z_t), the reference receiver at $(0, 0, 0)$, and n receivers at (x_i, y_i, z_i), $i = 1, 2, \cdots, n$. Define

$$R_i^2 = x_i^2 + y_i^2 + z_i^2, \qquad i = 1, 2, \cdots, n$$

$$R_t^2 = x_t^2 + y_t^2 + z_t^2$$

$$S_i^2 = (x_t - x_i)^2 + (y_t - y_i)^2 + (z_t - z_i)^2, \qquad i = 1, 2, \cdots, n. \tag{1}$$

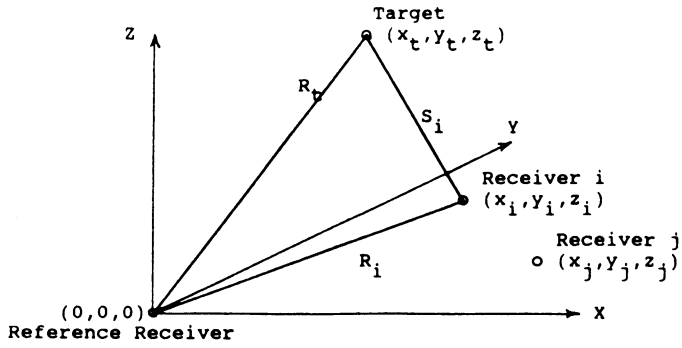

Fig. 1. Geometry of the target and receiving sensor system. The time delays are measured with respect to the reference receiver at receivers 1, 2, $\cdots$, n for the target.

We are given $\delta t_i = (R_t - S_i)/v$ for $i = 1, 2, \cdots, n$ and hence $\delta r_i = (R_t - S_i)$ where v is the velocity of propagation. (δt_i's are computed by cross correlation of the received waveforms at the receivers and are at best estimates.) To compute (x_t, y_t, z_t) from these δr_i's, we proceed as follows. It follows from (1)

$$R_t^2 - S_i^2 = x_t^2 - (x_t - x_i)^2 + y_t^2 - (y_t - y_i)^2 + z_t^2 - (z_t - z_i)^2. \quad (2)$$

The left-hand side (LHS) of the above equation becomes

$$(R_t + S_i)(R_t - S_i) = (2R_t - \delta r_i)\,\delta r_i \quad (3)$$

whereas the right-hand side (RHS) reduces to

$$2x_t x_i + 2y_t y_i + 2z_t z_i - R_i^2. \quad (4)$$

Thus we have

$$x_t x_i + y_t y_i + z_t z_i = R_t \delta r_i + (R_i^2 - \delta r_i^2)/2 \quad (5)$$

for $i = 1, 2, \cdots, n$.
Letting

$$W_i = (R_i^2 - \delta r_i^2)/2, \quad i = 1, 2, \cdots, n$$

$$W^T = [W_1, W_2, \cdots, W_n]$$

$$D^T = [\delta r_1, \delta r_2, \cdots, \delta r_n]$$

$$V^T = [x_t, y_t, z_t] \quad (6)$$

$$A = \begin{bmatrix} x_1 & y_1 & z_1 \\ x_2 & y_2 & z_2 \\ & \cdots & \\ x_n & y_n & z_n \end{bmatrix}$$

we can rewrite (5) as

$$AV = R_t D + W. \quad (7)$$

Define $B = (A^T A)^{-1} A^T$ (note A should be of rank 3, otherwise the array cannot locate the target), $C = BD$ and $Y = BW$. Then we have

$$V = R_t C + Y. \quad (8)$$

Equation (8) cannot be solved directly as (x_t, y_t, z_t) occur in the LHS as well as the RHS. However,

$$V^T V = R_t^2 C^T C + R_t (C^T Y + Y^T C) + Y^T Y \quad (9)$$

and hence noting $V^T V = R_t^2$:

$$R_t^2 (C^T C - 1) + R_t (C^T Y + Y^T C) + Y^T Y = 0. \quad (10)$$

The quadratic equation can be solved for R_t. Once R_t is known, (8) can be solved for $V^T = [x_t, y_t, z_t]$.

Thus (10) and (8) constitute an exact solution for the range computation problem. The steps involved in the solution can be summarized as follows.

1) For the given geometry of the receivers and the reference receiver at (0, 0, 0), compute R_i's, (the distances between the reference receiver and the ith receiver), compute A (defined as in (6)), and $B = (A^T A)^{-1} A^T$.

2) Estimate time delays between the waveforms received at the reference and the n receivers for the target of interest. From these delays and the velocity of propagation, compute δr_i's. From these δr_i's, compute the vectors W and D.

3) Compute vectors $C = BD$ and $Y = BW$.

4) Form the quadratic equation (10) in R_t using vectors C and Y. Solve for R_t.

5) To find the target coordinates (x_t, y_t, z_t), use (8).

III. Example

We will now present a simple example to illustrate the approach. Assume four receiver elements at (0, 0, 0), (1, 0, 0), (0, 1, 0), and (0, 0, 1) and a target at (2, 0, 0). By direct computation, we find $\delta r_1 = 1$, $\delta r_2 = 2 - \sqrt{5}$ and $\delta r_3 = 2 - \sqrt{5}$. Given that $\delta r_1 = 1$, $\delta r_2 = \delta r_3 = 2 - \sqrt{5}$, and that receivers are located at (0, 0, 0), (1, 0, 0), (0, 1, 0), and (0, 0, 1), we will compute the coordinates of the target.

We note $R_1^2 = R_2^2 = R_3^2 = 1$, $W^T = [0, 2\sqrt{5} - 4, 2\sqrt{5} - 4]$, and $D^T = [1, 2 - \sqrt{5}, 2 - \sqrt{5}]$. Also A and B are identity matrices, and hence $Y = W$ and $C = D$. Thus (10) becomes

$$(18 - 8\sqrt{5})R_t^2 - 8(9 - 4\sqrt{5})R_t + 8(9 - 4\sqrt{5}) = 0$$

i.e.,

$$R_t^2 - 4R_t + 4 = 0$$

from which we get $R_t = 2$. Using (8), we get $x_t = 2$, $y_t = z_t = 0$.

Reference

[1] J. C. Hassab, *Underwater Signal and Data Processing*. Boca Raton, FL: CRC, 1989.

Appendix 2

A Computer Program for Target Computation delineated in Appendix 1

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#define DIST(A, B, C, D, E, F)   sqrt((A-D)*(A-D)+(B-E)*(B-E)+(C-F)*(C-
F))
int invert(double c[3][3]);

main()
{
    double A[100][3];  // x, y, z coordinates
    double AA[3][3], B[3][100], R[100], W[100], D[100], C[3], Y[3];
    double err, cc, cy, yy, a, b, c, r1, r2, xt, yt, zt;
    int i, j, k;
    int N;
    double tx, ty, tz;
    double delta;

LX:
    tx = rand() % 1000;
    ty = rand() % 1000;
    tz = rand() % 1000;
    N = 5;
   printf("(%g %g %g)\n", tx, ty, tz);
LOOP:
    for (i = 0; i < N; i++) {
        for (j = 0; j < 3; j++) {
            A[i][j] = ((double) rand() / 32768.0) * 1000.0;
        }
    }

    for (i = 0; i < N; i++)
        R[i] = DIST(A[i][0], A[i][1], A[i][2], 0.0, 0.0, 0.0);

// compute deltas
    for (i = 0; i < N; i++)
        D[i] = DIST(A[i][0], A[i][1], A[i][2], tx, ty, tz) -
               DIST(0.0, 0.0, 0.0, tx, ty, tz);

// accuracy adjustment
    for (i = 0; i < N; i++) {
        delta = (double) (rand() % 10 - 5);
        D[i] = D[i] * (100.0 + delta) / 100.0;
    }

// compute AA matrix
    for (i = 0; i < 3; i++) {
        for (j = 0; j < 3; j++) {
            AA[i][j] = 0.0;
            for (k = 0; k < N; k++) {
                AA[i][j] += A[k][i] * A[k][j];
            }
```

```
        }
    }

// invert AA matrix
    invert(AA);

// compute B matrix
    for (i = 0; i < 3; i++) {
        for (j = 0; j < N; j++) {
            B[i][j] = 0.0;
            for (k = 0; k < 3; k++) {
                B[i][j] += AA[i][k] * A[j][k];
            }
        }
    }

// compute W matrix
    for (i = 0; i < N; i++) {
        W[i] = (R[i] * R[i] - D[i] * D[i]) / 2.0;
    }

// compute C and Y matrices
    for (i = 0; i < 3; i++) {
        C[i] = Y[i] = 0.0;
        for (j = 0; j < N; j++) {
            C[i] += B[i][j] * D[j];
            Y[i] += B[i][j] * W[j];
        }
    }

// compute cc = C * C, cy = C * Y, yy = Y * Y
    cc = cy = yy = 0.0;
    for (i = 0; i < 3; i++) {
        cc += C[i] * C[i];
        cy += C[i] * Y[i];
        yy += Y[i] * Y[i];
    }

// compute a, b, c
    a = cc - 1.0;
    b = 2.0 * cy;
    c = yy;

// compute roots r1 and r2
    r1 = (-b + sqrt(b*b - 4.0 * a * c)) / (2.0 * a);
    r2 = (-b - sqrt(b*b - 4.0 * a * c)) / (2.0 * a);

// compute xt, yt and zt
   xt = r1 * C[0] + Y[0];
   yt = r1 * C[1] + Y[1];
   zt = r1 * C[2] + Y[2];
   err = DIST(xt, yt, zt, tx, ty, tz);
   printf("soln 1: %d -> (%g %g %g) [%g %g %g] %g\n",
     N, xt, yt, zt, tx, ty, tz, err);

// compute xt, yt and zt
   xt = r2 * C[0] + Y[0];
```

```
   yt = r2 * C[1] + Y[1];
   zt = r2 * C[2] + Y[2];
   err = DIST(xt, yt, zt, tx, ty, tz);
   printf("soln 2: %d -> (%g %g %g) [%g %g %g] %g\n",
     N, xt, yt, zt, tx, ty, tz, err);
   N += 5;
   if (N <= 100)
       goto LOOP;
}

int invert(double c[3][3])
{
    double d[3][6], x ;
    int i, j, k ;

    for (i = 0 ; i < 3 ; i++) {
        for (j = 0 ; j < 3 ; j++) {
            d[i][j] = c[i][j] ;
            d[i][j+3] = (i == j) ? 1.0 : 0.0 ;
        }
    }

    /* Gaussian Elimination - Forward Stage */

    for (i = 0 ; i < 3 ; i++) {
        if (fabs(d[i][i]) < 1.0E-20) {
            printf("Singular Matrix !!!\n") ;
            return -1 ;
        }
        else {
            for (j = i, x = d[i][i] ; j < 6 ; j++)
                d[i][j] = d[i][j] / x ;

            for (j = i+1 ; j < 3 ; j++)
                for (k = 0, x = d[j][i] ; k < 6 ; k++)
                    d[j][k] = d[j][k] - x * d[i][k] ;
        }
    }

    /* Backward Substitution */

    for (i = 2 ; i > 0 ; i--)
        for (j = 0 ; j < i ; j++)
            for (k = 0, x = d[j][i] ; k < 6 ; k++)
                d[j][k] = d[j][k] - x * d[i][k] ;

    for (i = 0 ; i < 3 ; i++) {
        for (j = 0 ; j < 3 ; j++)
            c[i][j] = d[i][j+3] ;
    }
    return 0 ;
}
```