Manual for
The Trilateration Program

Willy Hereman and William S. Murphy, Jr.

MCS-91-11 September 1991

Prepared for: *The Bulldozer Project*
Thunder Basin Coal Company
Wright, Wyoming 82732
USA

Department of Mathematical and Computer Sciences
Colorado School of Mines
Golden, CO 80401-1887, USA
Phone: (303) 273-3860
Fax: (303) 273-3278

MANUAL FOR

THE TRILATERATION PROGRAM

developed by

WILLY HEREMAN & WILLIAM S. MURPHY, JR.

THE BULLDOZER PROJECT
THUNDER BASIN COAL COMPANY
P.O. BOX 406
WRIGHT, WYOMING 82732

DEPARTMENT OF MATHEMATICAL AND COMPUTER SCIENCES
COLORADO SCHOOL OF MINES
GOLDEN, COLORADO 80401-1887

December 6, 1991

Contents

1 Introduction

The problem deals with finding the unknown 3D-position of a point, given the distances from that point to a number of fixed points.

One may think of the following application: Determine the position of that piece of equipment (e.g. a bulldozer or drilling equipment) in an open mine with given ranges (slope distances) from that piece of equipment to a number of fixed beacons place on the rim of the mine. The x and y coordinates represent the horizontal position, and the z coordinate the altitude of the piece of equipment. The coordinates of the beacons are obtained by surveying techniques and are supposed to be exact.

In contrast to triangulation problems where angles are used to determine positions, in this problem only distances are measured. Surveyors invented the word *trilateration* for solving this type of problems.

The application for which the program was originally developed deals with the determination of the position of drills and bulldozers in an open pit coal mine operated by Thunder Basin Coal Company at Wright, Wyoming. In the manual we will refer to the company as TBCC.

In this manual we will discuss in detail the mathematical solution to this trilateration problem and its computer implementation in C. Since this document will primarily be used as a manual we start with the latter.

2 The Computer Program

Basic knowledge of Borland C++ is required to use the program. The user should know how to COMPILE and LINK files. To simplify compiling and linking, procedures can be written in batch files.

2.1 Brief Description of the Files in C-code

All the files of the trilateration program are written in Borland C++.
The program consists of the following files:

- xyz.c
- funcxyz.obj
- calib.exe

In addition, two sample data files are provided:

- beacon.dat
- calib.dat

All these files will be discussed in detail in what follows.

Also note that executable files will run on a PC even if C++ is not installed.

xyz.c Main program that calls the function *location*, which is located in the compiled version of the trilateration program, **funcxyz.obj**.

The distances from each beacon to the unknown equipment location are contained in **xyz.c**. Each time the distances are changed, **xyz.c** must be recompiled, and linked to **funcxyz.obj**. An alternate method of obtaining the distances is to modify the program **xyz.c** so that it will read the distances from a database or a data file. TBCC has modified **xyz.c** and included it in **tbcc.c** so that it reads the distances from a **Paradox** database. The description of **tbcc.c** is out of scope of the manual.

xyz.c sends the distances to the function *location*. The function *location* then reads the known beacon coordinates from the data file **beacon.dat**, and calculates the coordinates of the piece of equipment. The function *location* returns the equipment coordinates to **xyz.c**, if it was able to calculate them accurately. Otherwise, it returns an error code which is interpreted and displayed by **xyz.c**. If the error code is just a warning then the coordinates will be given with the message.

The four error codes are:

- Error, use a maximum of 24 beacons.
- Error, less than 4 positive radii.
- Warning, position is above lowest beacon.
- Error, unable to open file **beacon.dat**.

The actual C-code of **xyz.c** is given in Appendix I.

funcxyz.obj Contains the object code that reads in the beacon data from the file **beacon.dat**, and then calculates the coordinates of the position of the equipment.

calib.exe Executable version of the calibration program which reads in the beacon locations from the data file **calib.dat** and tests the effects a of a half a foot error on the radii for different positions in the mine. The data file **calib.dat** has the original experimental beacon locations, that can be changed with an editor. This calibration program is used to test proposed beacon locations to ensure that there are no BAD SPOTS within the perimeter of the mine. By bad spots we mean calculated positions for which the error on the ALTITUDE is larger than defined error range. In the development of the program we used 5 feet as the error range.

beacon.dat Is the data file with beacon locations as they are determined by the surveyor.

An example of the data file **beacon.dat** is as follows:

Total 8

1	475060.0	1096300.0	4670.0
2	481500.0	1094900.0	4694.0
3	482230.0	1088430.0	4831.0
4	478050.0	1087810.0	4775.0
5	471430.0	1088580.0	4752.0
6	468720.0	1091240.0	4803.0
7	467400.0	1093980.0	4705.0
8	468730.0	1097340.0	4747.0

In the first line *Total* refers to the total number of beacons, a number between 4 and 24 with no decimal point.

The remaining lines contain the beacon number followed by the easting, northing and elevation of the beacon. These locations need the decimal point notation. There can be extra spaces between the number and the coordinates. There can be spaces between the lines, but no extra comments are allowed. Of course, the number of beacons given by their coordinates must correspond to the number given to *Total* in the first line.

calib.dat Is the data file with beacon locations as they are determined by the surveyor, together with desired testing parameters.

An example of the data file **calib.dat** is as follows:

Total 8

Offset 40.0

Depth 600.0

Range 5.0

Grid 10

1	475060.0	1096300.0	4670.0
2	481500.0	1094900.0	4694.0
3	482230.0	1088430.0	4831.0
4	478050.0	1087810.0	4775.0

5	471430.0	1088580.0	4752.0
6	468720.0	1091240.0	4803.0
7	467400.0	1093980.0	4705.0
8	468730.0	1097340.0	4747.0

In the first line *Total* refers to the total number of beacons, a number between 4 and 24 with no decimal point.

In the second line *Offset* refers to the distance between the elevation of the lowest beacon and the (highest) point in the mine where the testing will start. In development and testing of the program, the offset was 40.0 feet (decimal notation!).

In the third line *Depth* refers to the distance between the elevations of the highest and the lowest test points in the mine. In testing the depth was 600.0 feet (decimal notation!). Including the offset, this means that we tested points 640 feet below the lowest beacon.

In the fourth line *Range* refers to the user defined acceptable error on the x, y and z coordinates. In testing we used 5.0 feet (decimal notation!).

In the fifth line *Grid* refers to a number between 1 and 10 (no decimal notation), which determines how many points in the mine will be tested. If grid is set to n, with $1 \leq n \leq 10$ then a 3D-grid containing n^3 points will be generated.

Examples:

Grid 1 calculates 1 points (1 * 1 * 1 = 1)

Grid 2 calculates 8 points (2 * 2 * 2 = 8)

Grid 3 calculates 27 points (3 * 3 * 3 = 27)

Grid 10 calculates 1000 points (10 * 10 * 10 = 1000)

The way the 3D grid is constructed is discussed in the Section dealing with the 'Calibration Program'.

The remaining lines contain the beacon number followed by the easting, northing and elevation of the beacons. These locations need the decimal point notation. There can be extra spaces between the number and the coordinates. There can be spaces between the lines, but no extra comments are allowed. Again the number of beacons that are used must be specified as *Total* in the first line.

2.2 Input and Output

INPUT Consists of the number of beacons beacon locations and the measured distances.

OUTPUT Is the position (x, y, and z) of the equipment together with messages when appropriate.

2.3 Using the Program

There are two different ways to use the program: Manual and Automatic.

MANUAL Type in the correct beacon locations by editing the file **beacon.dat**. The file **beacon.dat** must always be saved into the same directory as **xyz.c** and **funcxyz.obj**.

Type the distances (radii) from the beacons to the unknown equipment location into **xyz.c**. The actual C-code of **xyz.c** is given in Appendix I, where radii for 8 beacons are explicitly filled out.

Compile **xyz.c** and link it to **funcxyz.obj**. Every time **xyz.c** is updated, it must be recompiled, and linked to **funcxyz.c**.

The number of beacons specified in **beacon.dat** must correspond to the number of distances in **xyz.c**.

AUTOMATIC There are two different methods to use the program in an automated mode.

Method 1

Various ways could be used to enter the radii into the file **xyz.c** automatically. For instance the radii could be read from a file; or third party software could be purchased to read the radii from a data base. An alternate way to enter the radii automatically is to purchase or develop software that communicates directly with the beacons. All these options are application dependent and require the user to modify **xyz.c** by inserting the appropriate communications code.

TBCC uses method 2 discussed below.

Method 2

A simple method of calculating x, y, and z automatically is to extract the code from **xyz.c**, and insert it into a communications program. If this method is used, the program that the code is inserted into must be linked to **funcxyz.obj** at compile time.

Thunder Basin Coal Company inserted the program **xyz.c** into their multi purpose network program named **tbcc.c**. For TBCC's purposes the trilateration program will be used to determine the (horizontal) position of drills (only the x and y are needed), the 3D-position of bulldozers, etc.

When **tbcc.c** is compiled, it must be linked to **funcxyz.obj**. However, Thunder Basin Coal Company does not need to link **tbcc.c** with **xyz.c** because all of the code in **xyz.c** that is required to calculate the unknown x, y, and z is an integral part of **tbcc.c**.

Users of Method 2 should read the following subsection.

2.4 Extracting Code From xyz.c for use by a Communications Program

Let us discuss the contents of the file **xyz.c** as given in Appendix I.

If a communications program is going to be used to call the functions which calculate the unknown x, y, and z position, the following actions must be accomplished.

If the following include statements are not already part of the communications program, they should be inserted above **main**, and made global:

```
#include < stddef.h >
#include < stdlib.h >
#include < math.h >
#include < alloc.h >
```

The following function prototypes should be inserted into the communications program above **main**:

```
int location(double *distance, double *coordinate);
void *alloc1 (size_t n1, size_t size);
void free1 (void *p);
```

The following variable declarations, and memory allocations should be inserted into main if the function *location* will be called from **main**. Otherwise, they should be inserted into the function which will call the function *location*.

```
int message;
double *distance, *coordinate;
/* Allocate Space */
coordinate = (double *)alloc1(3, sizeof(double));
distance = (double *)alloc1(24,sizeof(double));
```

The communications program must receive the distances from the beacons to the equipment after the memory is allocated. The numbers listed below are examples of the correct format for the distances. The actual distances will be supplied by the communications program. These distances must be included in the function that calls *location*. The distances will be different for each unknown x, y, z position that is being calculated. The number of distances must match the number *Total* which is specified in **beacon.dat**. For example, if there are eight beacons, distance[0] thru distance[7] should be used, and the *Total* in the first line of **beacon.dat** should be: *Total* 8 (with no decimal point). If any of the required distances (in this example distance[0] thru distance[7]) are unknown, they must be entered as 0.0 with a decimal point. If additional beacons are used, the communications program must provide additional distances. The program will not provide correct results of the number of distances provided does not correspond to the *Total* in **beacon.dat**.

The correct format for the distances is as follows:

```
distance[0] =  5064.8377;    /* Beacon 1 Radius to Equipment */
distance[1] =  9774.7488;    /* Beacon 2 Radius to Equipment */
distance[2] = 10794.8361;    /* Beacon 3 Radius to Equipment */
distance[3] =  7376.2291;    /* Beacon 4 Radius to Equipment */
distance[4] =  3669.4236;    /* Beacon 5 Radius to Equipment */
distance[5] =  3526.9769;    /* Beacon 6 Radius to Equipment */
distance[6] =  5044.0810;    /* Beacon 7 Radius to Equipment */
distance[7] =  6168.8921;    /* Beacon 8 Radius to Equipment */
```

An example format for adding the distance that corresponds to the 9^{th} beacon is given below. Correct formats for the distances corresponding to beacons 9 thru 24 are found in **xyz.c** (see Appendix I).

Formats for beacons 9 thru 24 must be commented out unless there is a corresponding beacon that is in use. Likewise, if there are less than 8 beacons, then some of the above distance declarations should not be in the program.

```
distance[8] = ;          /* Beacon 9 Radius to Equipment */
```

Note: The label used in distance[ ] is one less than the beacon number!

The following statement calls the function that calculates the unknown x, y, and z coordinates. The statement must be included in the function that calls *location*, after the distances are assigned values. The variable *message* is an integer which describes the reliability of the result.

```
message = location(distance, coordinates);
```

The following switch handles the message which is returned by *location*. The switch must be included in function that calls *location*, after *location* is called.

```
switch(message){
    case(1):
    printf("  Error, use a maximum of 24 beacons. ");
    break;
    case (2):
    printf("  Error, less than 4 positive radii. ");
    break;
    case (3):
    printf("  Warning, Position is above lowest beacon. ");
    break;
    case (4):
    printf("  Error, unable to open file beacon.dat. ");
}
```

The following statements print out the x, y, and z coordinates (*Easting, Northing*, and *Elevation*) if there were at least 4 distances given, and no other errors were encountered. These statements must be included in the function that calls the function *location*, after the "switch" in order to display the results.

```
if((message==0) || (message==3))
{
    printf("Easting = %lf ", coordinate[0]);
    printf("Northing = %lf ", coordinate[1]);
    printf("Elevation = %lf ", coordinate[2]);
}
```

The following statements free up the allocated space, and must be included in the communications program that calls *location*, after the results are displayed and no longer needed for other applications.

```
/* Free allocated space */
free1((void *)distance);
free1((void *)coordinate);
```

Finally, when **tbcc.c** is compiled, it must be linked to **funcxyz.obj**.

2.5 Instructions for Calculating a Position via tbcc.c

- Ensure that **beacon.dat** is updated.
- Extract code from **xyz.c** for use by **tbcc.c**.
- Ensure that **tbcc.c** provides distances from beacons to the unknown location.
- Ensure that **funcxyz.obj** is linked to **tbcc.c**.
- Run the program.

2.6 Calibration Program

The program **calib.exe** is a program that is used to determine acceptable locations to position beacons. It establishes a rectangular solid test area that is broken down into a 3D grid pattern. The boundaries of the rectangle are calculated by determining the maximum and minimum northing, and easting of all beacons. The depth of the rectangular solid area that is tested is specified by the user.

Calib.exe reads data from the file **calib.dat** which is described in detail above.

This program must be executed and the output must be edited to analyze the proposed beacon positions. This program will output the calculated position, the exact position, and whether the point is out of tolerance.

The beacon locations specified in **beacon.dat** are for test purposes only. If the beacons in the mine are moved, **beacon.dat** must be updated as discussed above.

3 Mathematical Treatment

3.1 Statement of the Problem

The problem deals with finding the unknown 3D-position of a point, given the distances from that point to a number of fixed points.

The x and y coordinates could represent the horizontal coordinates, and the z the altitude of the unknown point. The coordinates of the fixed points (let us call them *beacons*) are supposed to be exact.

In contrast to triangulation problems where angles are used to determine positions, in this problem only distances are measured. This class of problems is commonly referred to as *trilateration* problems.

Questions that arise are:

- Is there a mathematical solution?
- What is theoretically the smallest number of beacons needed?
- Can the position of the bulldozer be determined "fairly accurately" even if the distances are inaccurate?
- What are the optimal positions of the beacons?
- What is the 'best' algorithm for the solution?
- Can the algorithm be translated into a fast C-program?
- What are the possible applications of the problem?

Some of these questions will be addressed in the text, but focus will be on a precise determination of the coordinates (x, y, z).

3.2 Sketch

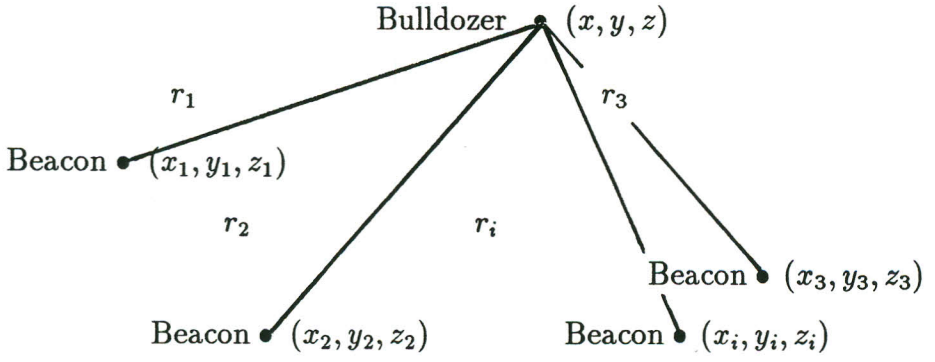

Fig. 1 Geometry of the Problem

3.3 A Mathematical Solution

The solution of this nonlinear problem is based on geometry, linear algebra and analysis.

LINEARIZATION

The coordinates (x_i, y_i, z_i) $(i = 1, 2, ..., n)$ of the n beacons are given whereas the coordinates (x, y, z) of the bulldozer are unknown. Let r_i be the *measured* (approximate) slope distance (range) from the bulldozer to the i^{th} beacon, and let $\hat{r}_i$ be the *exact* slope distance to the i^{th} beacon.

First let us ignore this distinction and suppose that the all radii are exact. The constraints are spheres with radii r_i $(i = 1, 2, ..., n)$,

$$(x - x_i)^2 + (y - y_i)^2 + (z - z_i)^2 = {r_i}^2. \tag{1}$$

The coordinates of the bulldozer must satisfy these n *nonlinear* equations. Trying to solve those by elimination of two of the coordinates would result in a highly nonlinear equation in the remaining coordinate, which has to be solved numerically. Furthermore, since the equations are quadratic many case for the signs would have to be considered. This is a difficult approach.

Let us use the j^{th} constraint as a *linearizing* tool.

Adding and subtracting x_j, y_j and z_j in (1) gives

$$(x - x_j + x_j - x_i)^2 + (y - y_j + y_j - y_i)^2 + (z - z_j + z_j - z_i)^2 = {r_i}^2 \tag{2}$$

with $(i = 1, 2, ..., j-1, j+1, ..., n)$

Expanding and regrouping the terms, leads to

$$\begin{aligned} & (x - x_j)(x_i - x_j) + (y - y_j)(y_i - y_j) + (z - z_j)(z_i - z_j) \\ = \; & \frac{1}{2}[(x - x_j)^2 + (y - y_j)^2 + (z - z_j)^2 \\ & -{r_i}^2 + (x_i - x_j)^2 + (y_i - y_j)^2 + (z_i - z_j)^2] \\ = \; & \frac{1}{2}[{r_j}^2 - {r_i}^2 + d_{ij}^2], \end{aligned} \tag{3}$$

where

$$d_{ij} = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2 + (z_i - z_j)^2} \tag{4}$$

is the distance between beacons i and j.

If we select $j = 1$, then $i = 2, 3, ..., n$ and we are left with a linear system of $(n - 1)$ equations in 3 unknowns:

$$(x - x_1)(x_2 - x_1) + (y - y_1)(y_2 - y_1) + (z - z_1)(z_2 - z_1)$$

$$= \frac{1}{2}[r_1{}^2 - r_2{}^2 + d_{21}^2] = b_{21} \tag{5}$$

$$\begin{aligned}&(x - x_1)(x_3 - x_1) + (y - y_1)(y_3 - y_1) + (z - z_1)(z_3 - z_1)\\ &= \frac{1}{2}[r_1{}^2 - r_3{}^2 + d_{31}^2] = b_{31}\end{aligned} \tag{6}$$

$$\vdots$$

$$\begin{aligned}&(x - x_1)(x_n - x_1) + (y - y_1)(y_n - y_1) + (z - z_1)(z_n - z_1)\\ &= \frac{1}{2}[r_1{}^2 - r_n{}^2 + d_{n1}^2] = b_{n1}.\end{aligned} \tag{7}$$

This system is easily written in matrix form $\mathbf{A}\vec{x} = \vec{b}$, with

$$\mathbf{A} = \begin{pmatrix} x_2 - x_1 & y_2 - y_1 & z_2 - z_1 \\ x_3 - x_1 & y_3 - y_1 & z_3 - z_1 \\ \vdots & \ddots & \vdots \\ x_n - x_1 & y_n - y_1 & z_n - z_1 \end{pmatrix}, \quad \vec{x} = \begin{pmatrix} x - x_1 \\ y - y_1 \\ z - z_1 \end{pmatrix}, \quad \vec{b} = \begin{pmatrix} b_{21} \\ b_{31} \\ \vdots \\ b_{n1} \end{pmatrix}. \tag{8}$$

Let $\tilde{\mathbf{A}}$ be any 3×3 matrix selected from $\mathbf{A}$. The robustness of the calculation will partly depend on the condition number $c(\tilde{\mathbf{A}}) = \|\tilde{\mathbf{A}}\|\,\|\tilde{\mathbf{A}}^{-1}\|$ of the coefficient matrix $\tilde{\mathbf{A}}$. The equations should be "well-conditioned" in order to be able to determine all three components of $\vec{x}$ with high accuracy. The change $\delta\vec{x}$ of the solution $\vec{x}$ resulting from the changes of $\tilde{\mathbf{A}}$ and $\vec{b}$ is given by [1]

$$\frac{\|\delta\vec{x}\|}{\|\vec{x}\|} \leq M\, c(\tilde{\mathbf{A}})\Big(\frac{\|\delta\vec{b}\|}{\|\vec{b}\|} + \frac{\|\delta\tilde{\mathbf{A}}\|}{\|\tilde{\mathbf{A}}\|}\Big), \tag{9}$$

where $M = \frac{1}{1-\alpha}$, with $\alpha = \|(\delta\tilde{\mathbf{A}})\tilde{\mathbf{A}}^{-1}\| < 1$. Both the data $\vec{b}$ and the position of the beacons (reflected in $\tilde{\mathbf{A}}$) can be changed to optimize the calculation of the position $\vec{x}$ of the bulldozer.

At this point we can draw some preliminary conclusions:

- The problem is linear. The system has $(n-1)$ equations in 3 unknowns.
- Theoretically only 4 beacons are needed to determine the unique position provided the beacons with labels 2, 3 and 4 are not co-linear.
- The least squares method for the linear problem is readily applicable.
- The robustness of the calculation will depend on the condition number of the coefficient matrix.

We now turn to two geometrical interpretations of the linear system.

Analytic Geometry

Let us give an interpretation of, say, the equation (5),

$$\begin{aligned}&(x-x_1)(x_2-x_1)+(y-y_1)(y_2-y_1)+(z-z_1)(z_2-z_1)\\ =\;&\frac{1}{2}[{r_1}^2-{r_2}^2+d_{21}^2]=b_{21}.\end{aligned}\tag{10}$$

The normal form of the plane (Fig. 2) through point $B_0\,(x_0,y_0,z_0)$ and with vector $\vec{N}(x_2-x_1,y_2-y_1,z_2-z_1)$, in the normal direction, is given by

$$(x-x_0)(x_2-x_1)+(y-y_0)(y_2-y_1)+(z-z_0)(z_2-z_1)=0.\tag{11}$$

Selecting the special point B_0 with

$$\begin{aligned}x_0 &= x_1+\frac{b_{12}}{d_{12}^2}(x_2-x_1)=\frac{1}{2d_{12}^2}[({r_2}^2-{r_1}^2)(x_1-x_2)+d_{12}^2(x_1+x_2)],\\ y_0 &= y_1+\frac{b_{12}}{d_{12}^2}(y_2-y_1)=\frac{1}{2d_{12}^2}[({r_2}^2-{r_1}^2)(y_1-y_2)+d_{12}^2(y_1+y_2)],\\ z_0 &= z_1+\frac{b_{12}}{d_{12}^2}(z_2-z_1)=\frac{1}{2d_{12}^2}[({r_2}^2-{r_1}^2)(z_1-z_2)+d_{12}^2(z_1+z_2)]\end{aligned}\tag{12}$$

the equation (11) reduces to (10). Three of these planes intersect in one point (the bulldozer position).

Trigonometry

Apply the cosine rule in the triangle $B\,B_1\,B_2$ in Fig. 3, to obtain

$${r_2}^2={r_1}^2+d_{12}^2-2\,\vec{v_1}\cdot\vec{v_2}=0\tag{13}$$

Since $\vec{v_1}=\vec{r}-\vec{r_1}$ and $\vec{v_2}=\vec{r_2}-\vec{r_1}$ with $\|\vec{v_1}\|=r_1$, $\|\vec{v_2}\|=d_{12}$, one gets

$$\begin{aligned}{r_2}^2 &= {r_1}^2+d_{12}^2-2[(x-x_1)(x_2-x_1)\\ &+\;(y-y_1)(y_2-y_1)+(z-z_1)(z_2-z_1)],\end{aligned}\tag{14}$$

which is nothing else than the first equation (5) of the system.

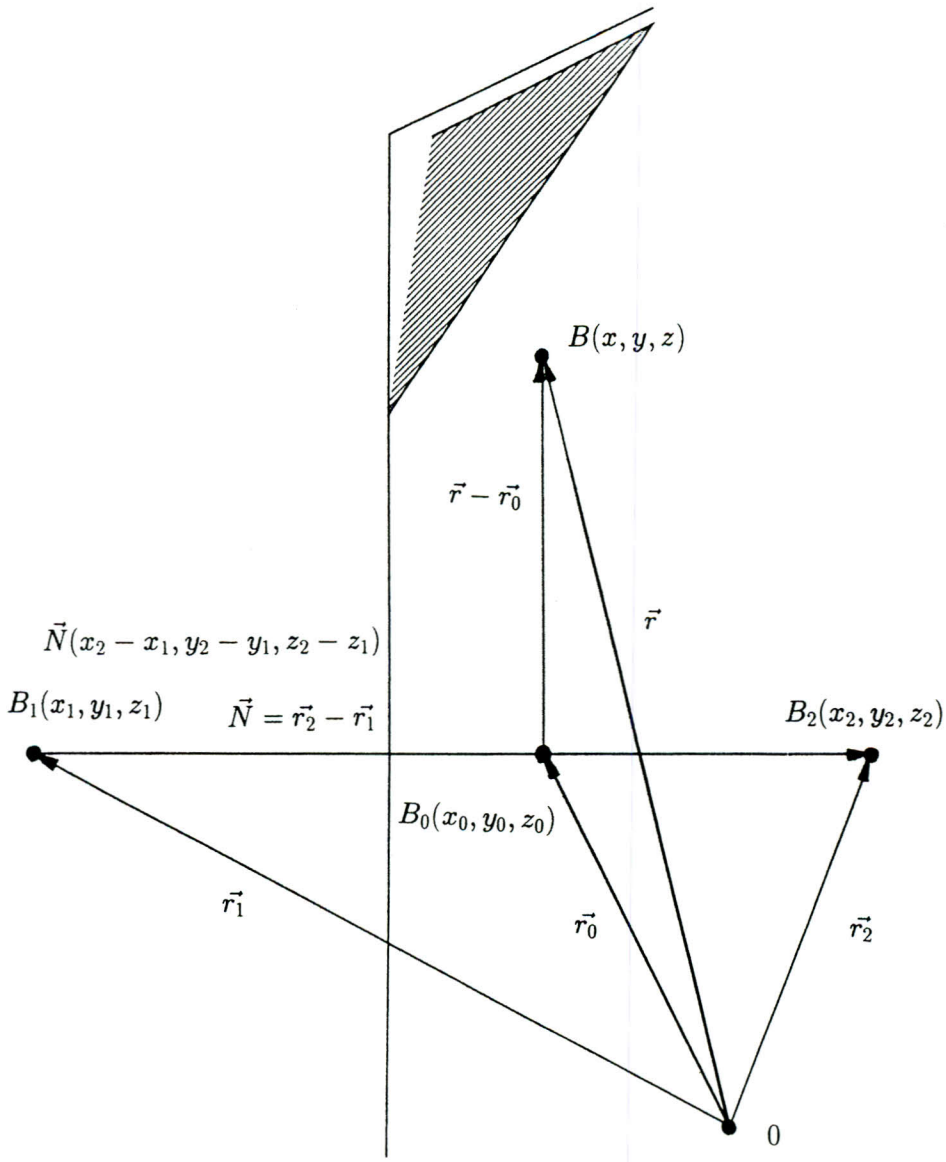

Fig. 2 Analytic Geometrical Interpretation

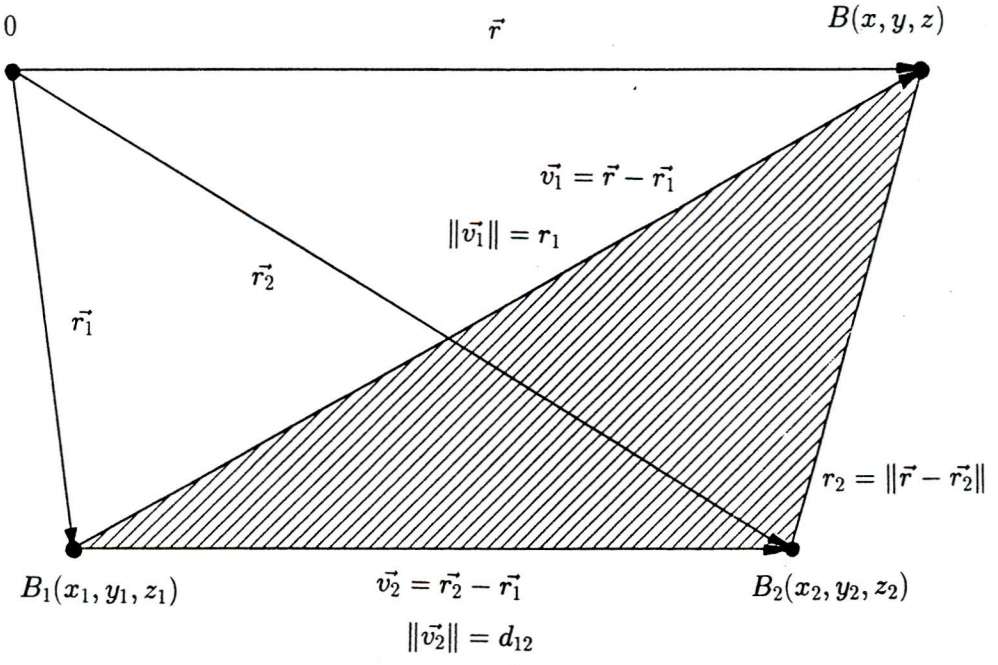

Fig. 3 Trigonometrical Interpretation

3.4 The Least Squares Method (Linear System)

In practice, the distances r_i are only approximate. Thus the problem requires the determination of $\vec{x}$ such that $\mathbf{A}\vec{x} \approx \vec{b}$. Minimizing the sum of the squares of the residuals,

$$S = \vec{r}^{\,T}\vec{r} = (\vec{b} - \mathbf{A}\vec{x})^T(\vec{b} - \mathbf{A}\vec{x}), \tag{15}$$

requires the solution of the *normal* equation [1]

$$\mathbf{A}^T\mathbf{A}\vec{x} = \mathbf{A}^T\vec{b}. \tag{16}$$

If $\mathbf{A}^T\mathbf{A}$ is non-singular then

$$\vec{x} = (\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T\vec{b}. \tag{17}$$

In general, one uses the normalized QR-decomposition of $\mathbf{A}$, i.e. $\mathbf{A} = \mathbf{QR}$, where $\mathbf{Q}$ is orthonormal matrix and $\mathbf{R}$ is upper-triangular matrix and the solution is then found from

$$\mathbf{R}\,\vec{x} = \mathbf{Q}^T\vec{b} \tag{18}$$

by back substitution.

It may happen that the matrix $\mathbf{A}^T\mathbf{A}$ is close to singlular even when the original matrix $\mathbf{A}$ was not close to singular. For situations like that, $\mathbf{QR}$ decomposition may overcome the problem. If not, there are other alternatives for solving the least squares problem accurately. We mention only two:

- Singular Value Decomposition (SVD). The optimal solution $\vec{x}_0$ is then given by $\vec{x}_0 = \mathbf{A}^+\vec{b}$. The pseudo-inverse $\mathbf{A}^+ = \mathbf{V}\mathbf{\Sigma}^+\mathbf{U}^H$ involves the unitary matrices occuring in the SVD of $\mathbf{A}$, this is $\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^H$.
- Schur Decomposition. In this case $\mathbf{A} = \mathbf{PTP}^H$, where $\mathbf{T}$ is upper-triangular and $\mathbf{P}$ is orthogonal if the eigenvalues of $\mathbf{A}$ are real.

For more details about these well-known techniques we refer to [1].

3.5 A Better Estimate for the Altitude 'z'

Two cases arise where the value of the z-coordinate obtained from (17) is not accurate even if the ranges are measured accurately. This is due to the geometry of the problem in cases where e.g. the bulldozer is close to the more or less common plane of the beacons. Furthermore, the magnitudes of the x and y coordinates is substantially larger than of the z coordinate. This difference in scale induces errors in the calculated z coordinate.

One way to obtain a better guess for z is to solve each of the given constraints (1) for z,

$$z \approx z_i \pm \sqrt{r_i^2 - (x - x_i)^2 - (y - y_i)^2}, \tag{19}$$

once x and y are known from (17). With $i = 1, 2, ..., n$ this gives n values for z which we denote by $z_{\{k\}}$ $(k = 1, 2, ..., n)$.

From now onwards we will only consider positions of the bulldozer that are lower than *all* the beacons. Consequently, only the $-$ sign in (19) is relevant. We could now take an average of the n values of $z_{\{k\}}$ obtained above. As is well-known, the arithmetical average will give the value of z that approximates the expected value in a least squares sense. But, could a weighted average improve matters? The answer is yes [3].

To show this, consider the error f_1 between the measured distance r_1 and the theoretical distance $\hat{r}_1$ from the bulldozer to beacon 1, i.e.

$$f_1 = \hat{r_1} - r_1 = \sqrt{(x - x_1)^2 + (y - y_1)^2 + (z - z_1)^2} - r_1. \tag{20}$$

Let $(\tilde{x}, \tilde{y}, \tilde{z})$ denote a 'good' guess for (x, y, x) and $(\delta x, \delta y, \delta z)$ the errors on these coordinates. Using a Taylor expansion we then obtain

$$\begin{aligned} f_1(x, y, z) &= f_1(\tilde{x} + \delta x, \tilde{y} + \delta y, \tilde{z} + \delta z) \\ &= f_1(\tilde{x}, \tilde{y}, \tilde{z}) + \left(\frac{\partial f}{\partial x}\right)_{x=\tilde{x}, y=\tilde{y}, z=\tilde{z}} \delta x + \left(\frac{\partial f}{\partial y}\right)_{x=\tilde{x}, y=\tilde{y}, z=\tilde{z}} \delta y \\ &+ \left(\frac{\partial f}{\partial z}\right)_{x=\tilde{x}, y=\tilde{y}, z=\tilde{z}} \delta z + ... \end{aligned} \tag{21}$$

Taking into account the explicit form of f_1 in (20), we have

$$\begin{aligned} f_1(x, y, z) &= f_1(\tilde{x}, \tilde{y}, \tilde{z}) + \frac{\tilde{x} - x_1}{f_1(\tilde{x}, \tilde{y}, \tilde{z}) + r_1} \delta x \\ &+ \frac{\tilde{y} - y_1}{f_1(\tilde{x}, \tilde{y}, \tilde{z}) + r_1} \delta y + \frac{\tilde{z} - z_1}{f_1(\tilde{x}, \tilde{y}, \tilde{z}) + r_1} \delta z + ..., \end{aligned} \tag{22}$$

where

$$f_1(\tilde{x}, \tilde{y}, \tilde{z}) = \sqrt{(\tilde{x} - x_1)^2 + (\tilde{y} - y_1)^2 + (\tilde{z} - z_1)^2} - r_1. \tag{23}$$

This result reveals that the weight factors should be proportional to the reciprocals of the distances. Therefore, a good approximation will be

$$z = \sum_{k=1}^{s} \frac{\frac{1}{r_k} z_{\{k\}}}{\frac{1}{r_k}}, \tag{24}$$

where the sum is only taken of these values of k for which the corresponding $z_{\{k\}}$ from (19) is real.

3.6 The Least Squares Method (Nonlinear System)

Recall that r_i denotes the *approximate* slope distance (range) from the bulldozer to the i^{th} beacon and that $\hat{r}_i$ stands for the *exact* slope distance to that beacon, i.e.

$$(x - x_i)^2 + (y - y_i)^2 + (z - z_i)^2 = \hat{r}_i^{\,2}. \tag{25}$$

Our task is to minimize the function

$$F(x, y, z) = \sum_{i=1}^{n} (\hat{r}_i - r_i)^2 = \sum_{i=1}^{n} f_i(x, y, z)^2, \tag{26}$$

with

$$f_i(x, y, z) = \hat{r}_i - r_i = \sqrt{(x - x_i)^2 + (y - y_i)^2 + (z - z_i)^2} - r_i. \tag{27}$$

In words: this is a nonlinear least squares method in which one minimizes the sum of the squares of the errors on the radii.

This is a fairly common problem in applied mathematics for which various algorithms are available [2]. Various different approaches can be taken, from simple to very complicated [4]. We will use Newton iteration to find the 'optimal' solution (x, y, z).

The initial guess for $(\tilde{x}, \tilde{y}, \tilde{z})$ is provided e.g. by the least squares method applied to the linear system. A 'better guess' for z can be obtained from solving the constraint equations and weighted averaging (see Section 1.5).

We only consider the case for which $F_{\text{min}} > 0$ and there for $n > 3$. Differentiating (26) with respect to x leads to

$$\frac{\partial F}{\partial x} = 2 \sum_{i=1}^{n} f_i \frac{\partial f_i}{\partial x}. \tag{28}$$

The formulae for the partials with respect to y and z are similar. Introducing the vectors $\vec{f}, \vec{g}$ and the Jacobian matrix $\mathbf{J}$ we obtain

$$\vec{g} = 2\mathbf{J}^T \vec{f}, \tag{29}$$

where

$$\mathbf{J} = \begin{pmatrix} \frac{\partial f_1}{\partial x} & \frac{\partial f_1}{\partial y} & \frac{\partial f_1}{\partial z} \\ \frac{\partial f_2}{\partial x} & \frac{\partial f_2}{\partial y} & \frac{\partial f_2}{\partial z} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_n}{\partial x} & \frac{\partial f_n}{\partial y} & \frac{\partial f_n}{\partial z} \end{pmatrix}, \quad \vec{f} = \begin{pmatrix} f_1 \\ f_2 \\ \vdots \\ f_n \end{pmatrix}, \quad \vec{g} = \begin{pmatrix} \frac{\partial F}{\partial x} \\ \frac{\partial F}{\partial y} \\ \frac{\partial F}{\partial z} \end{pmatrix}. \tag{30}$$

Defining the vector $\vec{x}$

$$\vec{x} = \begin{pmatrix} x \\ y \\ z \end{pmatrix}, \tag{31}$$

Newton iteration gives

$$\vec{x}_{\{k+1\}} = \vec{x}_{\{k\}} - (\mathbf{J}^T_{\{k\}}\mathbf{J}_{\{k\}})^{-1}\mathbf{J}^T_{\{k\}}\vec{f}_{\{k\}}, \tag{32}$$

where $x_{\{k\}}$ denotes the kth approximate solution and the subscript $\{k\}$ in $\mathbf{J}$ and $\vec{f}$ means that these quantities are evaluated at $\vec{x}_{\{k\}}$. Obviously $\vec{x}_{\{1\}} = (\tilde{x}, \tilde{y}, \tilde{z})^T$.

Using the explicit form of the function $f_i(x, y, z)$ we get

$$\mathbf{J}^T\mathbf{J} = \begin{pmatrix} \sum_{i=1}^n \frac{(x-x_i)^2}{(f_i+r_i)^2} & \sum_{i=1}^n \frac{(x-x_i)(y-y_i)}{(f_i+r_i)^2} & \sum_{i=1}^n \frac{(x-x_i)(z-z_i)}{(f_i+r_i)^2} \\ \sum_{i=1}^n \frac{(x-x_i)(y-y_i)}{(f_i+r_i)^2} & \sum_{i=1}^n \frac{(y-y_i)^2}{(f_i+r_i)^2} & \sum_{i=1}^n \frac{(y-y_i)(z-z_i)}{(f_i+r_i)^2} \\ \sum_{i=1}^n \frac{(x-x_i)(z-z_i)}{(f_i+r_i)^2} & \sum_{i=1}^n \frac{(y-y_i)(z-z_i)}{(f_i+r_i)^2} & \sum_{i=1}^n \frac{(z-z_i)^2}{(f_i+r_i)^2} \end{pmatrix}, \tag{33}$$

and

$$\mathbf{J}^T\vec{f} = \begin{pmatrix} \sum_{i=1}^n \frac{(x-x_i)f_i}{(f_i+r_i)} \\ \sum_{i=1}^n \frac{(y-y_i)f_i}{(f_i+r_i)} \\ \sum_{i=1}^n \frac{(z-z_i)f_i}{(f_i+r_i)} \end{pmatrix}. \tag{34}$$

In practice this type of iteration works fast, in particular when the matrix $\mathbf{J}^T\mathbf{J}$ is augmented by a diagonal matrix which effectively biases the search direction towards that of *steepest decent*, an improvement due to Levenberg and Marquardt. As the solution is approached such modifications can be expected to have a decreasing effect. We will not describe them here.

References

[1] NOBLE, B. and DANIEL, J.W., *Applied Linear Algebra*, Prentice-Hall, Englewood Cliffs, 3rd edition, 1988.

[2] MCKEOWN, J.J., *Specialized versus general-purpose algorithms for minimizing functions that are sums of squared terms*, Mathematical Programming **9** (1975) 57-68.

[3] GROSSMAN, W., *Grundzüge der Ausgleichungsrechnung nach der Methode der Kleinsten Quadrate nebst Anwendung in der Geodesy*, Springer Verlag, Berlin, 1969.

[4] MIKHAIL E.M., *Observations and Least Squares*, IEP-Dun-Donnelley, Harper and Row Publishers, New York, 1976

4 Appendix I: Code of xyx.c

```
#include <stddef.h>
#include <stdlib.h>
#include <math.h>
#include <alloc.h>

int location(double *distance, double *coordinate);
void *alloc1 (size_t n1, size_t size);
void free1 (void *p);

main()
{
int message;
double *distance, *coordinate;

coordinate = (double *)alloc1(3,sizeof(double));
distance = (double *)alloc1(24,sizeof(double));

distance[0] =    5064.8377;     /* Beacon 1 Radius to Equipment */
distance[1] =    9774.7488;
distance[2] =   10794.8361;
distance[3] =    7376.2291;
distance[4] =    3669.4236;
distance[5] =    3526.9769;
distance[6] =    5044.0810;
distance[7] =    6168.8921;

//      distance[8] =   ;
//      distance[9] =   ;
//      distance[11] =  ;
//      distance[12] =  ;
//      distance[13] =  ;
//      distance[14] =  ;
//      distance[15] =  ;
//      distance[16] =  ;
//      distance[17] =  ;
//      distance[18] =  ;
//      distance[19] =  ;
//      distance[20] =  ;
//      distance[21] =  ;
```

```
//      distance[22] =  ;
//      distance[23] =  ;
//      distance[24] =  ;

message = location(distance, coordinate);

switch(message)
{
case(1):

        printf("\n\nError, use a maximum of 24 beacons.\n");
break;

case (2):

printf("\n\nError, less than 4 positive radii.\n");
break;

case (3):

printf("\n\nWarning, Position is above lowest beacon.\n");
break;

case (4):

printf("\n\nError, unable to open file beacon.dat.\n");

}

if(((message==0)||(message==3)))
{
 printf("Easting =   %lf\n", coordinate[0]);
 printf("Northing =  %lf\n", coordinate[1]);
 printf("Elevation = %lf\n", coordinate[2]);
}

free1((void *)distance);
free1((void *)coordinate);

return 0;
}
```