Computational Aspects of the FO3DS Triangulation Algorithm

Lennart Damm
Robotics Laboratory
Department of Electrical Engineering
and Computer Science
The University of Michigan
Ann Arbor, Michigan

Abstract

Execution measurements are performed on a triangulation algorithm for a new fiber optical sensor. Active illumination is used and each emitting fiber generates a light spot on a 3-D surface. The objective is to identify the object via surface shape information. A set of four triangulation algorithms, one for each emitter, is mapped onto one to four processing units. Four emitters is the minimum number required for the sensor system.

The triangulation execution time for a minimum sensor head configuration (four algorithms) on the SUN is 26.7 milliseconds without FPA and 1.8 milliseconds with FPA (excluding input and output in both cases).

A uniprocessor NCUBE system gives the execution time 3.8 milliseconds for a single triangulation algorithm (including algorithm input and output times). The multiprocessor execution times for a system with four triangulation algorithms are 6.7 milliseconds with two processors, 7.7 milliseconds with three processors, and 3.8 milliseconds with four processors. Message passing in the form of algorithm inputs and outputs are included in these figures. Only the three processor case involves inter-processor message passing. This shows that a NCUBE hypercube type computer is not suitable for fast execution of the triangulation algorithm, where intra-algorithm messages are short. Only deterministic execution cases are considered.

The results show that the triangulation algorithm cannot be implemented on a hypercube because of the heavy penalty on setting up a channel for message passing. A SUN with FPA clearly gives better performance. This is true at least for the low level parallelizing scheme investigated, where a single algorithm is partitioned to run on more than one processor.

An analysis of sensitivity of triangulated data to deviations in the sensor head design parameters is performed. The results will be used as a basis for sensor head calibration.

ensm

UMR6757

Contents

1 Introduction.

Noncontact optical 3-D sensors that can measure the range, orientation, and shape of a surface for the purpose of identifying an object can enhance the performance of industrial robots. Potential applications range from assembly, inspection, and seam tracking to pick-and-place tasks.

This work is based on a method using optics and a conical sensor head emitter configuration (with no moving parts) proposed by [Kanade & Fuhrman]. Their method employed active illumination and triangulation to calculate 3-D points from 2-D detector values. A detector value was generated as a centroid of a light spot on the detector surface. Their sensor head contained a number of LED's arranged in rings. A light cone was defined by a ring of LED's, with a unique diameter and cone height combination. Several cones of this type were used to improve the depth of sight. The accuracy was 0.1 mm in spatial resolution and 1 degree in surface normal resolution.

This paper describes the triangulation algorithm for a conical sensor of the type described by [Kanade & Fuhrman]. However, a few new features have been included in this Fiber Optic 3-D Sensor (FO3DS). Optical fibers are used as both emitting and receiving elements instead of LED's and a camera respectively. A set of recursive predictive filters will be used to handle the dynamics of sensor/object intermotion. This replaces the statistical uncertainty analysis for sensor head design and performance enhancement suggested by [Kanade & Fuhrman].

The context of the triangulation algorithm is depicted in Figure 1. *Sensor calibration* is used to calibrate camera parameters such as lens distortion and detector nonlinearity. Systematic errors in the sensor head are modelled into the system by calibration when a new sensor head has been mounted on the robot arm and optionally during production runs. A test surface is used as calibration medium.

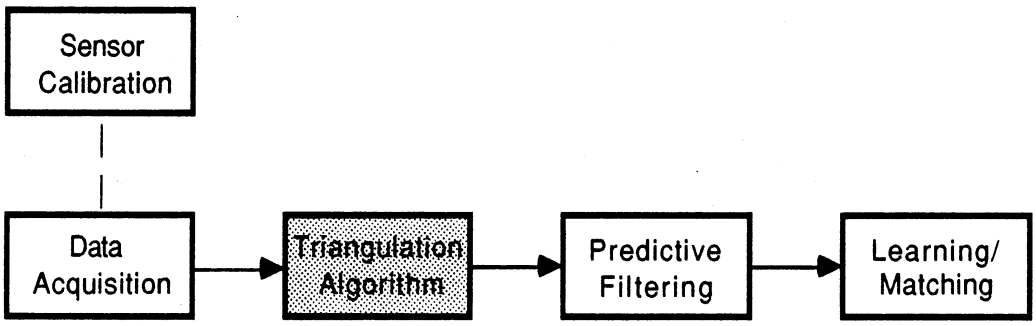

Figure 1: The FO3DS triangulation context.

Data acquisition is done through sampling and digitizing the analog signals from the lateral photo detector. For each sample the 2-D light spot value as well as amplitude information is available after pre-amplification.

Recursive predictive filtering gives optimal, measurement-based object surface spot position as well as spot velocity. As a by-product an improved signal-to-noise ratio is obtained. This is an important factor in a noise-prone system. Each emitting fiber generates a spot on the surface of the object and thus requires a predictive filter.

Learning/matching can be divided into an off-line object surface learning phase and a matching

phase, which takes place when trying to identify an object. The model matching is the last sensor phase before any robot motion or end-effector action can be taken. It is also the most complex phase. Object surface modelling can be done off-line manually. Alternatively a 3-D automated learning method can be adopted for the object surface learning task and a subset of the same learning method be used for object surface matching.

For the investigation of triangulation algorithm execution times a SUN 3/280 workstation (under UNIX) is used for sequential execution and a NCUBE/ten hypercube computer for sequential and parallel execution.

In this paper execution times and the impact of parallelization are analyzed. First one triangulation algorithm is run on a single processor. The division of the algorithm into separable parts is investigated. Then four (identical) algorithms are run on two to four processors. When run on multiprocessors the four algorithms are distributed among the available processors if this is feasible. The number of four algorithms are chosen because this is the minimum configuration for any FO3DS sensor system. The results are easily extrapolated to more than four algorithms and more than four processors.

It is not the intent to give execution times in exact, absolute values. Nor is high-lighting the performance of the computers of major interest. However, the upper bounds on execution times can be estimated. The triangulation algorithm will eventually be derived to run on an application-specific processor system (with parallel capability). Thus, this paper addresses the questions of execution bottlenecks and multiprocessor parallelization.

This paper is organized in the following way. Section 2 describes the triangulation algorithm to be used for measuring execution times. A geometric sensor head overview is followed by derivation of the triangulation equations. Aspects of computational overhead is discussed and execution speedup measures suggested.

The different executable entities of the triangulation algorithm are described in Section 3. This section also gives the results from execution time measurement runs. Finally a set of Gantt charts shows the speedup and processor utilization for the processors executing the triangulation code.

The computer memory requirements for the triangulation algorithm are briefly covered in Section 4.

Sensitivity of triangulated data as a function of design parameter deviations is presented in Section 5.

2 The triangulation algorithm.

2.1 FO3DS sensor head geometry.

The sensor head contains emitting fibers, lenses and receiving fibers. Figure 2 shows the geometrical model used for the triangulation procedure. The coordinate system has its origin on the optical axis where the receiving fiber bundle starts. The x-y plane defines the virtual image plane. The z axis is directed towards the receiving lens and the object.

The direction of the emitted light beam is defined by angle α. The baseline b is the distance

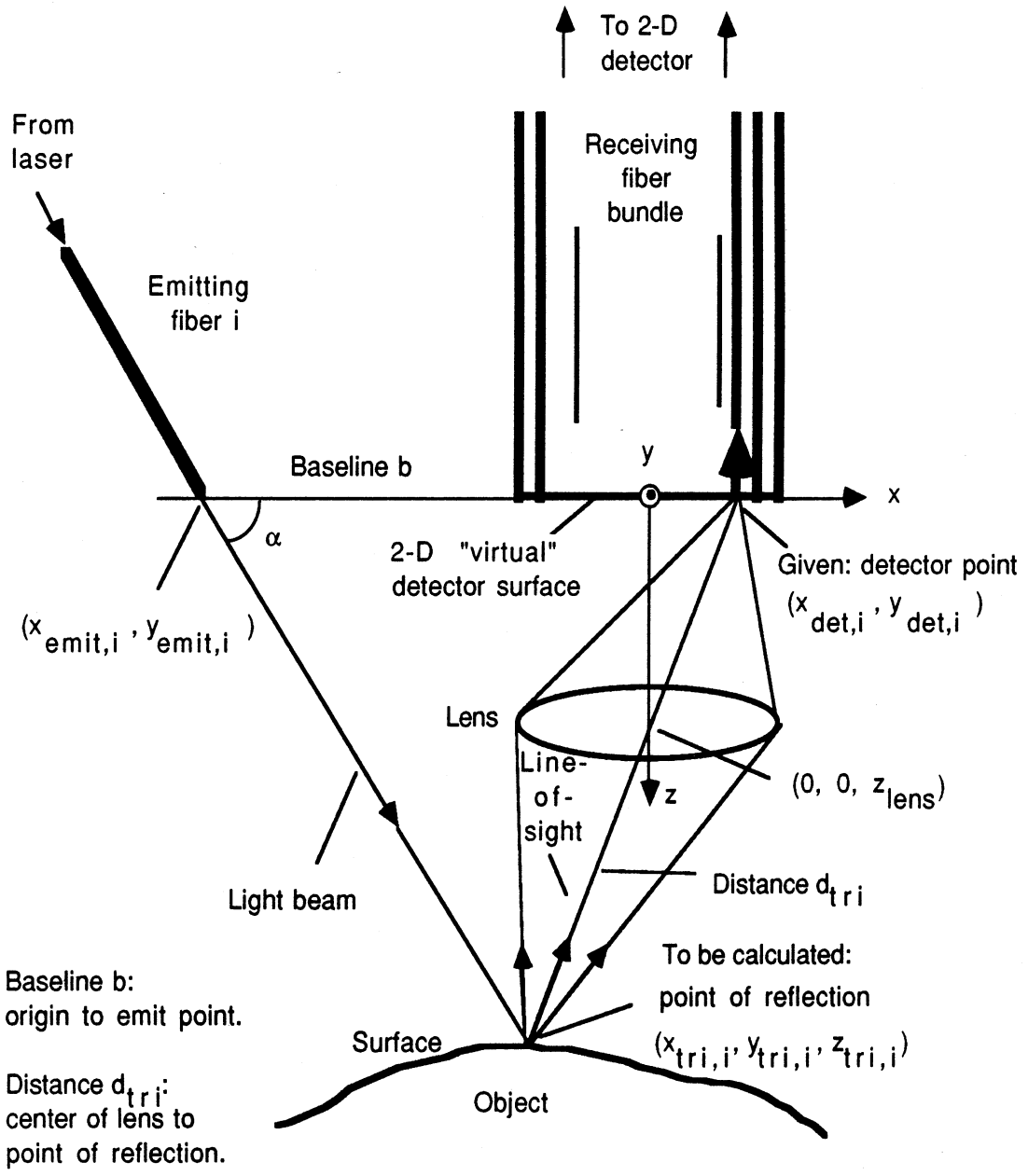

Figure 2: Geometric sensor head model overview – diffuse reflection case shown (not drawn to scale).

between the origin of the coordinate system and the emitter point. The line-of-sight, defined by the reflected light beam, starts at the point (x_{det}, y_{det}) and extends via the point $(0, 0, z_{lens})$ towards the point $(x_{tri}, y_{tri}, z_{tri})$. These two lines meet at the latter point, the top of the light cone.

The reflective surface can be matte, mirror-like or something in between. These three surface types give diffuse, specular and diffuse-specular reflected light beams respectively. An ideal surface should be matte as far as optical properties are concerned. By being matte the surface reflects light in all directions. Thus the problem of fore-shortening is minimized. The sensor can always pick up a signal. On the other hand, if a surface is specular, the surface normal must be almost perpendicular to the optical axis of the sensor (i.e. the z axis). Both diffuse and specular reflections can be handled by the FO3DS sensor. This is an important feature for applications which require a robot end-effector to traverse boundaries of surfaces with different reflective characteristics.

2.2 Deriving the triangulation equations.

Inputs to the triangulation algorithm are the point (x_{det}, y_{det}) and the emitter number (Figures 3 and 4). The use of the emitter number is described in section 2.3. Sensor head related constants are the total number of emitters n_{emit}, the sensor head angle α, the sensor head baseline b, and the distance z_{lens} from the center of the lens to the virtual image plane (the receiving fibers).

Algorithm outputs include the point $(x_{tri}, y_{tri}, z_{tri})$, the distance d_{tri} from the center of the lens to this point, and a set of credibility flags, which are defined later.

The FO3DS sensor contains a set of emitting fibers, each one defining three θ angles (Figure 3). Each of these fibers generate a point on the surface in 3-D space. The detector spot should formally be denoted $(x_{det,i,nT}, y_{det,i,nT})$, where i is the current emitter number and nT (n=0,1,2,...) is the current sampling time. The sampling period is the time between two activations of the same emitter. For simplicity the detector (virtual image plane) spot is denoted (x_{det}, y_{det}) and the triangulated (surface spot) point $(x_{tri}, y_{tri}, z_{tri})$ for emitter i at sampling time nT in this paper.

Similar triangles (Figure 3, shaded triangle in Side view 1) give the triangulated x_{tri} coordinate as a function of the detector coordinate:

$$\frac{x_{tri} - x_{det}}{z_{tri}} = -\frac{x_{det}}{z_{lens}} \tag{1}$$

which can be written as

$$x_{tri} = -x_{det}\left(\frac{z_{tri}}{z_{lens}} - 1\right) \tag{2}$$

An equivalent equation is easily derived for y_{tri}. By inspecting Figure 3 (shaded triangle in Side view 2 and Bottom view) two expressions can be derived for z_{tri}:

$$z_{tri} = \left(b - \frac{y_{tri}}{sin\theta_{det}}\right)tan\alpha \tag{3}$$

$$z_{tri} = \left(b - \frac{x_{tri}}{cos\theta_{det}}\right)tan\alpha \tag{4}$$

The θ angles, as defined by the (active) emitter, are (Figure 3):

$$\theta_{det} = arctan\frac{y_{det}}{x_{det}} \tag{5}$$

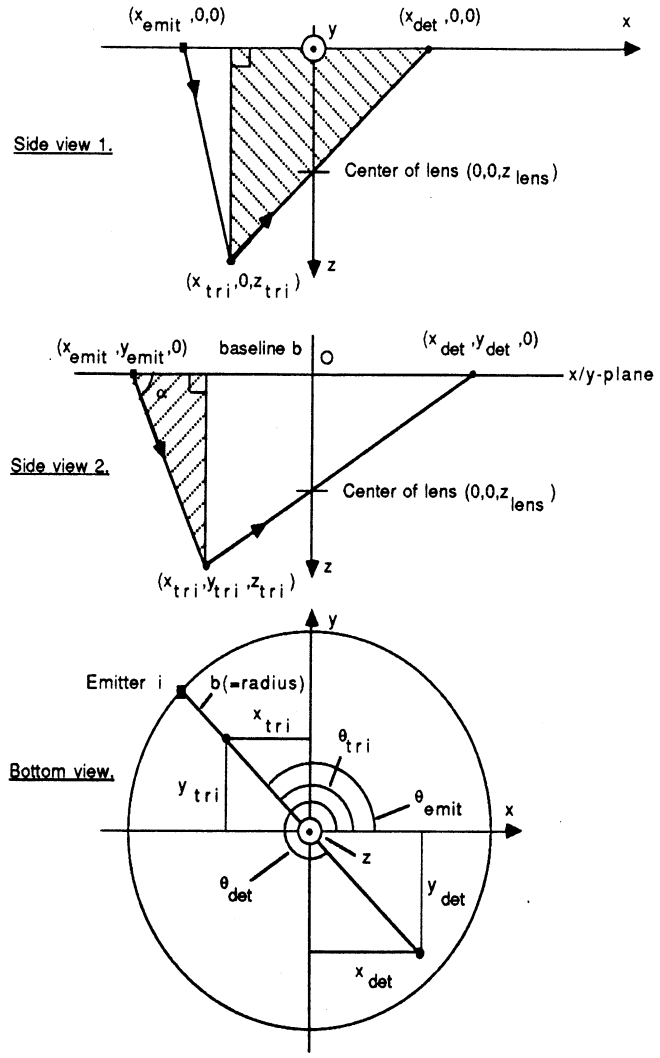

Figure 3: Geometry for 3-D triangulation. Shaded triangle in Side view 1 is used to derive x_{tri} and shaded triangle in Side view 2 is used to derive z_{tri}.

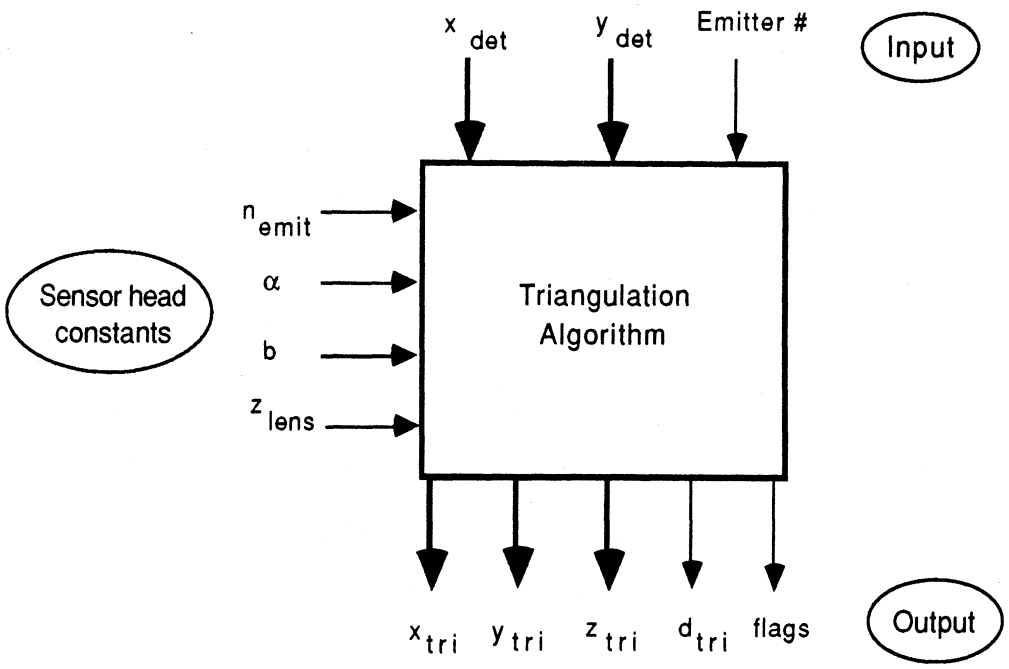

Figure 4: Triangulation algorithm variables and constants.

$$\theta_{emit,i} = \frac{i * 2\pi}{n_{emit}} + \theta_{const} \tag{6}$$

where z_{lens} = the distance from the center of the lens to the receiving fibers (the virtual image plane),
$b = \sqrt{x_{emit}^2 + y_{emit}^2}$ is the baseline,
emitter number $i = 0, 1, ..., (n_{emit} - 1)$,
n_{emit} = number of emitters in the ring,
θ_{const} = sensor head emitter fiber configuration dependent constant,
$\theta_{const} = 0$ gives first emitter location on positive x-axis.

In equations (2), (3), and (4) the angle θ_{det} is used. It can be argued that the θ_{emit} angleshould be used instead, since an emitter direction is more accurate than an indirectly calculated θ_{det} value [Kanade & Fuhrman]. The latter angle is subjected to lens distortion and detector surface non-linearity effects via the x_{det} and y_{det} values.

In this paper the line-of-sight (defined by θ_{det}) is used, not the emitter beam direction (defined by θ_{emit}). Uncertainties, such as lens distortion etc., are then no longer an integral part of the triangulation algorithm. Also simpler and more straight forward equations for the triangulation procedure is obtainable. The primary detector output is used directly to derive the triangulated spot position. By combining equations (2) and (3), and (2) and (4), equations (2) through (5) give the following set of equations:

$$\theta_{det} = arctan\frac{y_{det}}{x_{det}} \tag{7}$$

$$x_{tri} = \frac{x_{det}(z_{lens} - btan\alpha)cos\theta_{det}}{z_{lens}cos\theta_{det} - x_{det}tan\alpha} \tag{8}$$

$$y_{tri} = \frac{y_{det}(z_{lens} - btan\alpha)sin\theta_{det}}{z_{lens}sin\theta_{det} - y_{det}tan\alpha} \tag{9}$$

$$z_{tri} = (bsin\theta_{det} - y_{tri})\frac{tan\alpha}{sin\theta_{det}} \tag{10}$$

$$z_{tri} = (bcos\theta_{det} - x_{tri})\frac{tan\alpha}{cos\theta_{det}} \tag{11}$$

These equations make up the triangulation algorithm. The equations are subject to denominator constraints as explained in section 2.3.

2.3 Completing the algorithm.

The basic triangulation algorithm is defined by equations (7) through (11). In order to model a more production like triangulation algorithm a number of necessary features are added to the basic equations (Figures 4 and 5). The angle θ in Figure 5 is equal to either θ_{det} or θ_{emit} according to section 2.3.4.

2.3.1 Choice of z_{tri} equation.

The choice of z_{tri} equation, (10) or (11), is given by the θ_{det} value. To avoid mathematical singularities, equation (10) can be used to calculate z_{tri} for angles in the intervals $\frac{\pi}{4} < \theta_{det} < \frac{3\pi}{4}$ and $\frac{5\pi}{4} < \theta_{det} < \frac{7\pi}{4}$ (or some other intervals which give a non-singular solution), while equation (11) is used for all other θ_{det} angles.

2.3.2 Credibility testing.

If a non-modulated light source is used in the sensor system, spurious reflections (from illegal light sources) during sampling instances can give false alarms, i.e. irrelevant $(x_{tri}, y_{tri}, z_{tri})$ values. To counteract this problem certain geometric credibility tests are performed. These tests generate a set of credibility flags, which are passed on to other algorithms for evaluation and subsequent action.

The first credibility test compares the signs of x_{det} and x_{tri}. Similar signs are geometrically impossible since the line-of-sight goes through the center of the lens (Figure 2). Similar signs are caused by an illegal light source which is being activated during the sampling procedure (Figure 6c). The same sign test is applied to y_{det} and y_{tri} although this test may be redundant.

The second credibility test checks if the three points (x_{det}, y_{det}), $(0,0,0)$, and $(x_{tri}, y_{tri}, z_{tri})$ are located in the same plane. If a discrepancy cannot be explained by noise, an error is assumed. The cause again is an illegal light source.

The third credibility test is concerned with large values of x_{tri} or y_{tri}. This is simply to alert the system if the measured surface spot is far from the optical axis. The use of this information may be application dependent.

The last credibility test investigates the sign of z_{tri}. For distant objects the emitted light beam and the line-of-sight could be nearly parallel in an extreme case (imagine that the point of reflection in Figure 2 is located far to the right). System or measurement noise may then cause a negative z_{tri}

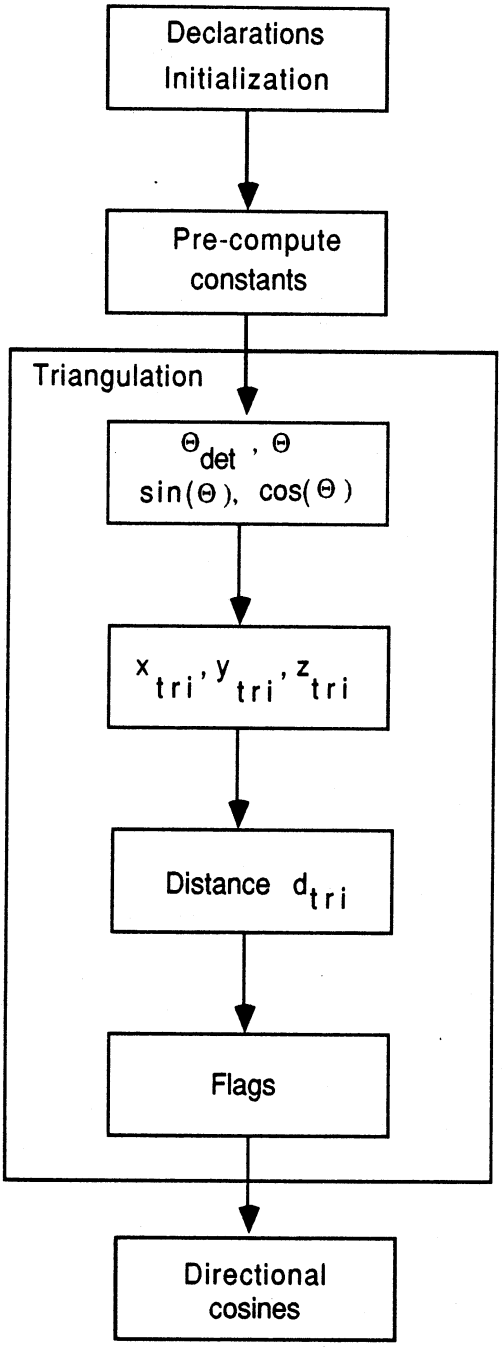

Figure 5: The triangulation algorithm.

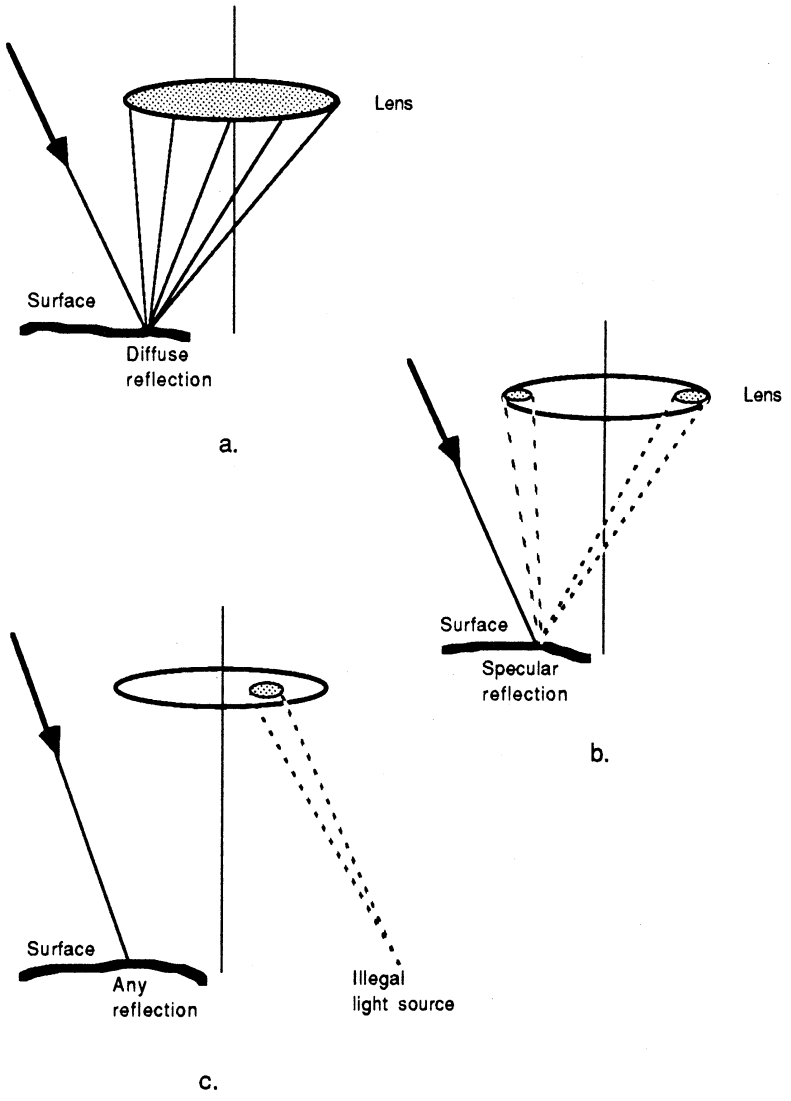

Figure 6: Diffuse, specular and illegal reflection cases.

to be accidentally obtained. This means that the system is made to believe that the two lines intersect behind the sensor instead of in front of it (viz. on the surface of the object).

2.3.3 Optional triangulation algorithm output.

"Post-triangulation" computations consist of calculation of three directional cosines. These define the d_{tri} direction. This direction is not considered to be vital but is included for completeness. It can be used together with the d_{tri} distance value by a host system for collision avoidance purposes.

2.3.4 Equation constraints.

Equation (7) must meet the constraint $x_{det} \neq 0$. Therefore it is rewritten as:

$\theta_{det} = arctan\frac{y_{det}}{x_{det}}$ if $x_{det} \neq 0$

$\theta_{det} = \frac{\pi}{2}$ if $x_{det} = 0$ and $y_{det} > 0$

$\theta_{det} = \frac{3\pi}{2}$ if $x_{det} = 0$ and $y_{det} < 0$

θ_{det} = not defined if $x_{det} = y_{det} = 0$.

The last case is handled in the following way. If $x_{det} = y_{det} = 0$ then $\theta_{emit,i}$ (or shorter θ_{emit}) is used instead of θ_{det} to calculate z_{tri}.

Equation (8) is valid only if $z_{lens}cos\theta_{det} - x_{det}tan\alpha \neq 0$, equation (9) only for $z_{lens}sin\theta_{det} - y_{det}tan\alpha \neq 0$, equation (10) only for $sin\theta_{det} \neq 0$, and equation (11) only for $cos\theta_{det} \neq 0$ (see also section 2.3.1).

2.3.5 Execution improvement using separation.

Execution of the basic triangulation equations (7) to (11) can be improved by separating out pre-computable parts. These parts are computed once at system start-up. The equations for on-line execution are simplified by defining the two constants:

$$c_{0,i} = tan\alpha \tag{12}$$

$$c_{1,i} = z_{lens} - b * c_{0,i} \tag{13}$$

where emitter number $i = (0, 1, 2, ..., (n_{emit-1}))$. This gives the number of computations for on-line processing as shown in Figure 7. The total number of operations per emitter (for each measuring cycle) is four divisions, 13 multiplications, five subtractions, one arctan calculation, two sin calculations, two cos calculations, and one square root calculation.

If θ_{emit} could be used instead of θ_{det}, the possibility of separating pre-computable operations would increase. However, this would mean that the inherent information contents of the detector values (x_{det}, y_{det}) would not be exploited.

One emitter:	Division	Multiplication	Subtraction	Addition	Other
Θ_{det}	1	-	-	-	arctan
x_{tri}	1	4	1	-	cos (2)
y_{tri}	1	4	1	-	sin (2)
z_{tri}	1	2	1	-	-
d_{tri}	-	3	2	2	$\surd$

Figure 7: Computations per emitter and measuring cycle.

2.3.6 Potential execution improvement using parallelism.

The triangulation algorithm can be decomposed into concurrently executable entities (low level parallelizing). For low level parallelizing we need to know the execution times for simple operations such as addition, multiplication etc. Also a set of complete triangulation algorithms can be distributed among a number of processors (high level parallelizing).

The equations (7) to (11) together with (12) and (13) can be used as a basis for parallelizing the triangulation algorithm. The equations can be depicted as a deterministic model [Hwang & Briggs]. Here a very low-level task structure is chosen. The tasks are defined for the computations of θ_{det}, x_{tri}, y_{tri}, z_{tri}, and d_{tri} (equations (7) to (13) and Figure 7). Apart from the arctan function (task T0) and the square root function (T24), the following operator tasks are defined: addition (tasks T19 and T23), subtraction (T5, T11, T15, T20, and T21), multiplication (T1-T4, T7-T10, T14, T16, T17, T18, and T22), and division (T6, T12, and T13). Task execution times are t_+ for addition etc. (Figure 8). The process graphs show the parallelizing potential of the different processes. Also they depict the number of computational levels required (as horizontal rows of task circles).

3 Execution time measurements.

Different number of emitting fibers are simulated as test cases to investigate the execution performance of the triangulation algorithm. The minimum number of emitters in a FO3DS sensor system is four. There is no upper limit, but an even number of emitters should be used since they work in pairs. Since there is a linear relation in the execution times for these cases can be extrapolated from the four-emitter case. Figure 7 gives the following number of computations for four emitters (defining one complete measuring cycle):

- 4 arctan functions
- 4 square root functions
- 4 sin operations
- 4 cos operations
- 16 divisions
- 52 multiplications
- 15 subtractions

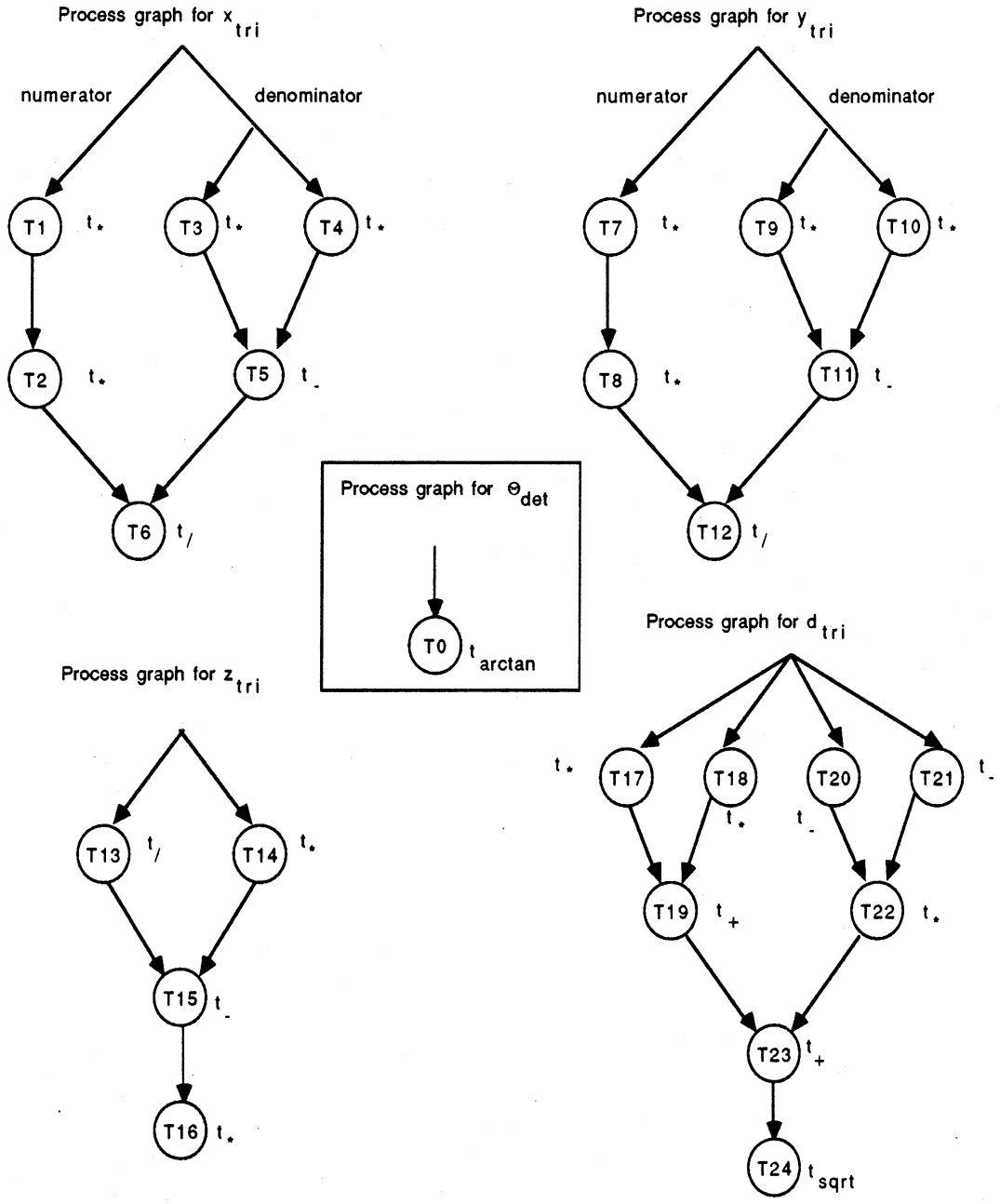

Figure 8: Process graphs for $\theta_{det}, x_{tri}, y_{tri}$ and z_{tri} (deterministic model). T_i are task numbers, execution times are denoted t_+ for addition operation, t_- for subtraction, t_* for multiplication, $t_/$ for division, t_{arctan} for arctan function, and t_{sqrt} for square root function.

SUN execution times in C language (microseconds).					
Operation	Without FPA	Normalized	FPA norm.	With FPA	Normalized
Addition	23.2	1	7	3.3	1
Subtraction	26.1	1	8	3.3	1
Multiplication	33.0	1.5	10	3.4	1
Division	89.8	4	27	6.6	2
$\sqrt{\ }$	113.6	5	34	28.6	9
sin(angle) 1)	292.2	13	89	40.5	12
sin(angle)	603.8	26	183	19.9	6
cos(angle)	697.2	30	211	20.5	6
tan(angle)	728.9	32	221	36.0	11
arctan(angle)	1083.8	47	328	29.9	9
Triangulation algorithm:					
Pre-computations	1194.8	52	362	57.5	17
Triangulation	5211.5	225	1579	338.0	102
Post-computations	272.8	12	83	61.2	19

1) Calculated as $\frac{y}{\sqrt{(x^2+y^2)}}$, equivalent for cos.

Figure 9: Measured execution times on the SUN.

- 8 additions
- conditionals
- setting credibility flags.

The emitter execution time must be much faster than the measuring cycle rate. A measuring cycle is defined by the activation of all emitters in sequence once.

3.1 Execution times on a sequential machine (SUN).

A SUN 3/280 (with FPA facility) was used to run the triangulation algorithm. The variables in the triangulation algorithm are declared as double-precision floating point. Execution times for the basic operations addition, subtraction, multiplication, and division are shown in Figure 9. Addition is used as the basis for normalization.

The "setitimer" and "getitimer" C routines [SUN] were used to measure execution times. Each test run encompassed enough iterations in a loop to make a total execution time of between 150 and 200 seconds. This was required because the SUN clock resolution time is 10 milliseconds. The maximum error in the execution time values should therefore be less than 0.1%. Corrections were made for loop overhead.

To improve the execution times on the SUN the C compiler in-line floating point code generation option was used together with the object code optimization option [SUN]. Measured execution times are given in Figure 9.

3.2 Execution times on a parallel machine (NCUBE).

The NCUBE/ten is a hypercube computer with a 7 MHz clock, a host processor, and 64 node processors. Only one of the nodes was used for the triangulation algorithm.

When using a parallel processor a certain amount of processing overhead due to communication must be taken into account. The time for an algorithm to run on a hypercube can be given by the general expression [Mudge & Abdel-Rahman]

$$\tau(N) = \tau_i + \tau_p + (1 - \alpha)\tau_c + \tau_o \tag{14}$$

where N indicates the number of processors in the cube, τ_i the time to input data to the cube, τ_p the time to perform the processing at a node, α the degree of transparency, τ_c the internode communication time, and τ_o the time to output data. The degree of transparency α is the internode communication time that can be overlapped with the node processing (using cache and DMA channels).

Figure 10 shows the NCUBE execution times for some basic operations and the triangulation algorithm. Replacing the arctan function with direct sin and cos calculations gives a gain of (1+24+24+24)-(7+7) = 59 units of time. The execution times were not improved by applying the C compiler floating point and optimization options.

Communication between NCUBE nodes is handled via message passing. Message passing time for n bytes is 466.40+(n-1)*3.14 microseconds [Buzzard]. All non-uniprocessor NCUBE runs include message passing between nodes . The ntime() C routine [NCUBE] was used to measure node program execution times.

Double-precision variables are eight bytes long and integer variables two bytes long. Each message passing session requires an additional eight bytes. Message passing for input takes 602 microseconds for two consecutive algorithms and 547 microseconds for a single algorithm (Figure 11). Inputs are (x_{det}, y_{det}) and *emitno*. This gives two doubles and one integer to be passed for each algorithm. Message passing for output takes 853 microseconds for two consecutive algorithms and 672 microseconds for a single algorithm. Outputs are $(x_{tri}, y_{tri}, z_{tri})$, directional cosines for $(x_{tri}, y_{tri}, z_{tri})$, distance lens-to-$(x_{tri}, y_{tri}, z_{tri})$, and a flag word. This gives seven doubles and one integer to pass. To summarize: input to an algorithm is 26 bytes and output from an algorithm is 66 bytes.

Figure 11 shows the execution times for a two-processor NCUBE (see also Figure 15). Message handling amounts to about 27% and 47% of the nominal code execution times for the two cases shown. However, it has been shown that these standard communication message passing times can be improved more than fivefold [Buzzard].

Since the NCUBE has no clock synchronization between nodes, execution times from Figures 10 and 11 are used to derive multiprocessor execution times.

Decomposition is dependent upon the impact of message passing between nodes, see Figure 12 for NCUBE message passing values. The eight byte overhead is included in the "# Bytes" column.

NCUBE execution times in C language		
Operation	Microsec.	Normalized
Addition	10.1	1
Subtraction	10.1	1
Multiplication	10.5	1
Division	12.6	1
$\sqrt{\ }$	49.3	5
sin 1)	69.9	7
sin	245.7	24
cos	239.8	24
tan	241.3	24
arctan	240.8	24
Triangulation algorithm:		
Pre-computations	288.2	29
Triangulation:	2172.3	215
(a) Θ, sin, cos	1305.5	129
(b) $(x, y, z, d)_{tri}$	336.7	33
(c) Flags.	530.1	53
Post-computations	157.6	16

1) Calculated as $\frac{y}{\sqrt{(x^2+y^2)}}$, equivalent for cos.

Figure 10: Measured execution times on uni-processor NCUBE.

NCUBE execution times in C language, 2 processors - 4 algorithms.		
Operation	Microsec.	Normalized
A. SISO		
Input message	602	0.11
Triangulation	5236	1.00
Output message	853	0.16
B. DIDO		
Input messages	1093	0.21
Triangulation	5236	1.00
Output messages	1344	0.26

Figure 11: Execution times for two-processor NCUBE. SISO = Single Input Single Output message sending, DIDO = Double Input Double Output message sending.

	Task	Execution time [μ sec]	Message passing (to next task): # Doubles, Integers	# Bytes	Estimated time [μ sec]
1	Input	-	2D + 1I	26	301 (SISO), 547 (DIDO)
2	Pre	289	5D	48	614
3	Θ, sin, cos	1306	4D + 1I	42	595
4	$(x, y, z)_{tri}$	337	4D	40	589
5	Flags	531	1I	10	495
6	Post	158	7D + 1I	66	427 (SISO), 672 (DIDO)

Figure 12: Intra-algorithm decomposition overview with inter-task message passing (NCUBE).

Doubles are represented by eight bytes and integers by two bytes. Some inter-task outputs are split up to more than one receiving task. Output from task 1 is split to task 2 (one integer) and task 3 (two doubles). Output from task 2 is divided between task 3 (two doubles) and task 4 (three doubles). Output from task 3 is divided between task 4 (three doubles and one integer) and task 5 (one double). In order to make use of intra-algorithm decomposition the message passing times should be an order of magnitude less than the task execution times involved (the NCUBE may meet these demands with extended communication [Buzzard]).

3.3 Evaluation.

3.3.1 Execution improvement by replacing the arctan function.

For simplicity all variables in the FO3DS triangulation algorithm are declared as double-precision. The arctan function is used to calculate the angle θ_{det} (equation 7). The arctan execution time is 47 normalized time units (i.e. comparable to 47 additions). Since the calculated θ_{det} values are to be used in only these sin and cos calculations, the latter functions can be computed directly as:

$$sin(\theta_{det}) = \frac{y_{det}}{\sqrt{x_{det}^2 + y_{det}^2}} \tag{15}$$

$$cos(\theta_{det}) = \frac{x_{det}}{\sqrt{x_{det}^2 + y_{det}^2}} \tag{16}$$

Using this method the execution time decreases from 47 to 13 units. The total execution time improvement using equations (15) and (16) is (4+47+26+30)-(13+13) = 81 units of time.

The calculation of θ_{det} (task T0) using the arctan function is replaced by (Figure 13): addition (T27, T32), multiplication (T25, T26, T30, T31), division (T29, T34), and square root (T28, T33).

When using the SUN floating point option the arctan replacement gives a gain of only (2+9+6+6)-(12+12) = 1 unit of time (values from Figure 9).

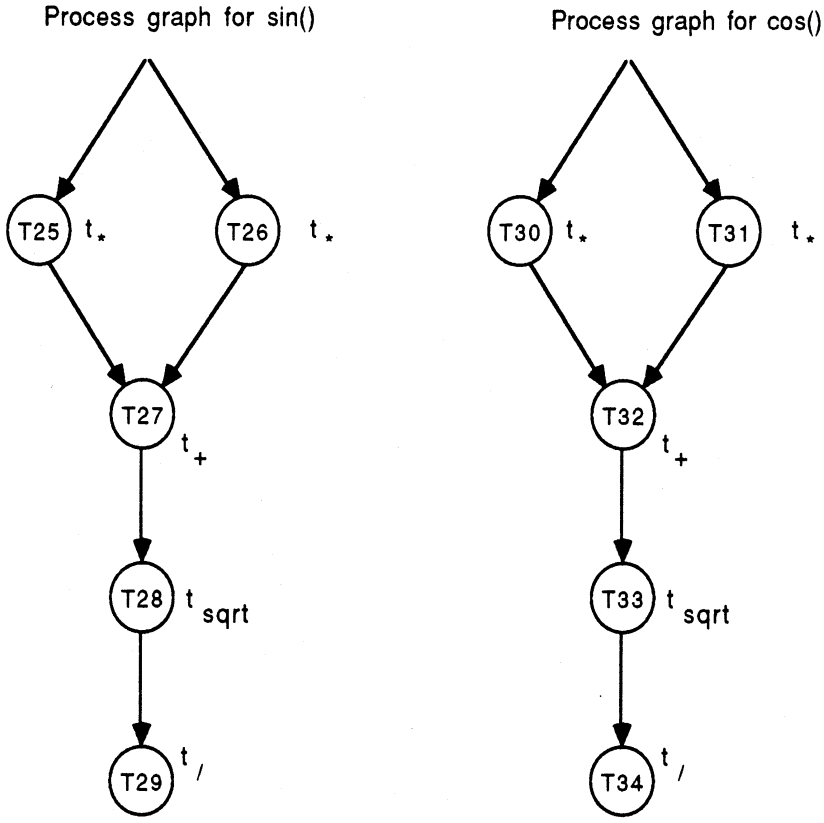

Figure 13: Process graphs for $sin(\theta_{det})$ and $cos(\theta_{det})$ computations.

3.3.2 Uniprocessor.

The sequential execution of the triangulation algorithm on a uniprocessor is shown in Gantt chart form in Figure 14. The SUN total execution time is 289 units (6.68 msec.), or equivalently 2024 FPA units, and with floating point accelerator 138 FPA units (0.46 msec.).

The NCUBE total execution time for one algorithm is 113 SUN units (2.62 msec.). To compute multiprocessor speedup, the average DIDO message passing time is added. This gives the execution time ET_1 = 166 SUN units (3.84 msec.).

3.3.3 Two processors.

Four algorithms are best grouped on two processors as shown in the Gantt chart in Figure 15. Here the white fields are message passing times. The best total execution time is ET_2 = 289 SUN units (6.69 msec.). DIDO speedup is $S_2 = 4 * ET_1 / ET_2 = 2.30$ and the mean utilization of the two-processor system is $U_2 = 1.00$. It does not pay off to distribute any algorithm to two or more processors since any message passing gives a substantial penalty.

3.3.4 Three processors.

With four algorithms and three processors it is necessary to distribute at least one algorithm over the available processors (Figure 16). White fields are message passing times. Black fields denote idle processor time. Task 4 is divided into three parts (A, B, and C). For simplicity the same input messages are used for tasks 2 and B as for tasks 3 and A (this gives a few extra bytes to pass). Because of the heavy impact of message passing it is hard to find a good solution. Let us assume that messages must be passed from A to B and from B to C. From A to B five pre-computed constants are passed. The message passing time is 614 microseconds. From B to C θ_{det}, θ, $\sin(\theta)$, $\cos(\theta)$, and a flag are passed. The message passing time is 595 microseconds.

The total execution time with three processors as shown in the example in Figure 16 is ET_3 = 333 SUN units (7.72 msec.). This should be compared to the non-message passing case for two processors, see figure 15, with total execution time 6.69 msec. The speedup is $S_3 = 1.99$ and the average processor utilization is only $U_3 = 0.75$.

3.3.5 Four processors.

Four processors and four algorithms is a trivial case. The total execution time is ET_4 = 166 SUN units (3.84 msec.), see Figure 16. The speedup is $S_4 = 4.00$ and the processor utilization $U_4 = 1.00$.

3.3.6 Summary.

Previous Gantt charts can be compiled into a bar chart for comparison, see Figure 17. The single algorithm NCUBE case is taken as a norm. All two, three, and four processor bars include four

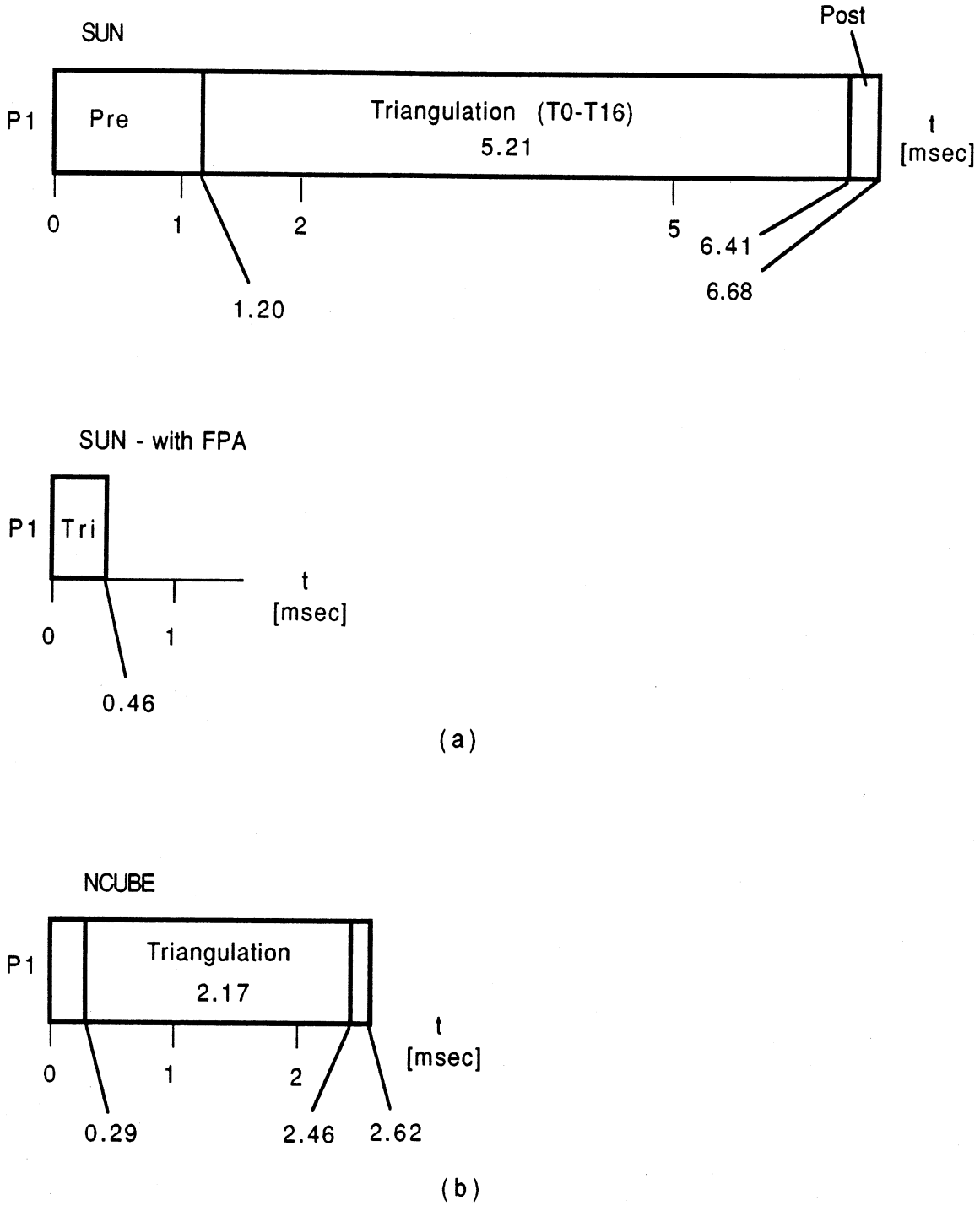

Figure 14: Uniprocessor system Gantt charts (a) SUN and (b) NCUBE.

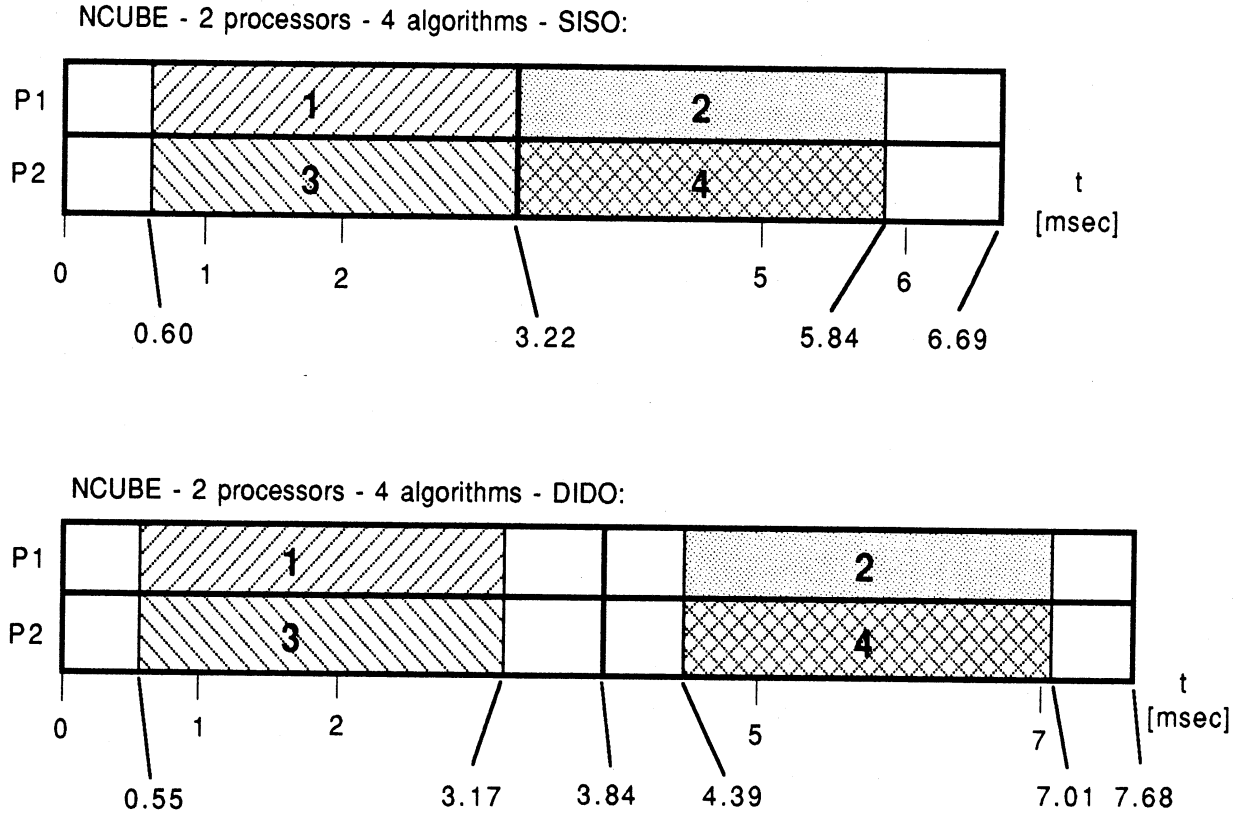

Figure 15: Two-processor system Gantt charts for four algorithms (NCUBE).

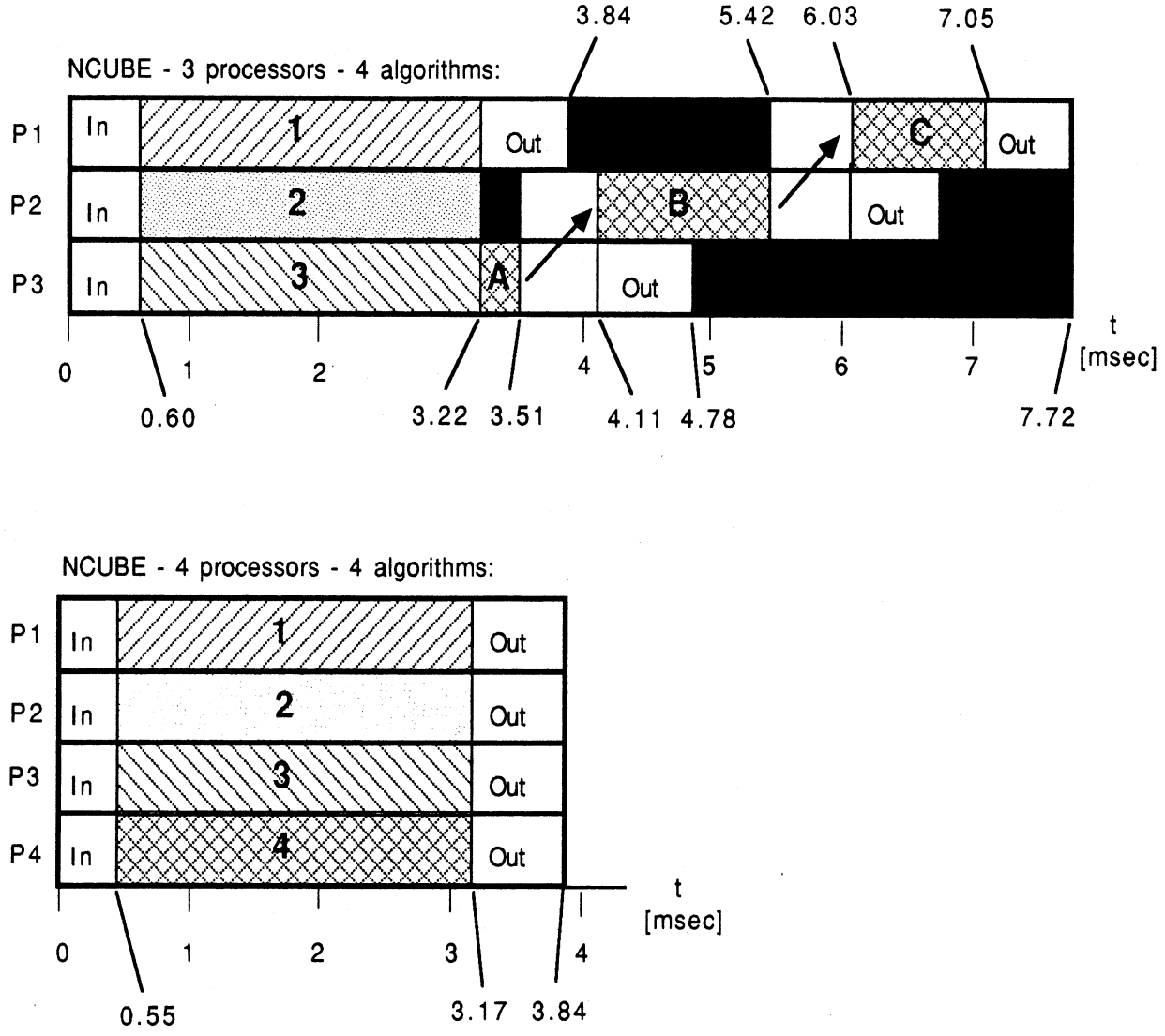

Figure 16: Multiprocessor system Gantt charts for four algorithms (NCUBE). Algorithm 4 contains parts A, B, and C.

(identical) algorithms. The white fields show message passing for input to algorithms and output from algorithms. The three processor system is included to show the drastic impact of message passing. Two, three, and four processors have relative execution times of approximately 2.55, 2.95, and 1.47 respectively. The two message passing sequences introduced in the three-processor case gives a considerable degradation from the two-processor case. Even by reducing message passing times to a minimum, as mentioned above, a NCUBE three-processor system will not outperform a NCUBE two-processor system for the algorithm in question. It should be pointed out that this conclusion is based on the assumption of a deterministic execution environment. However, worst case execution code length was sought by simulating suitable input data. Also, in all runs the slow arctan calculation was part of the executed code.

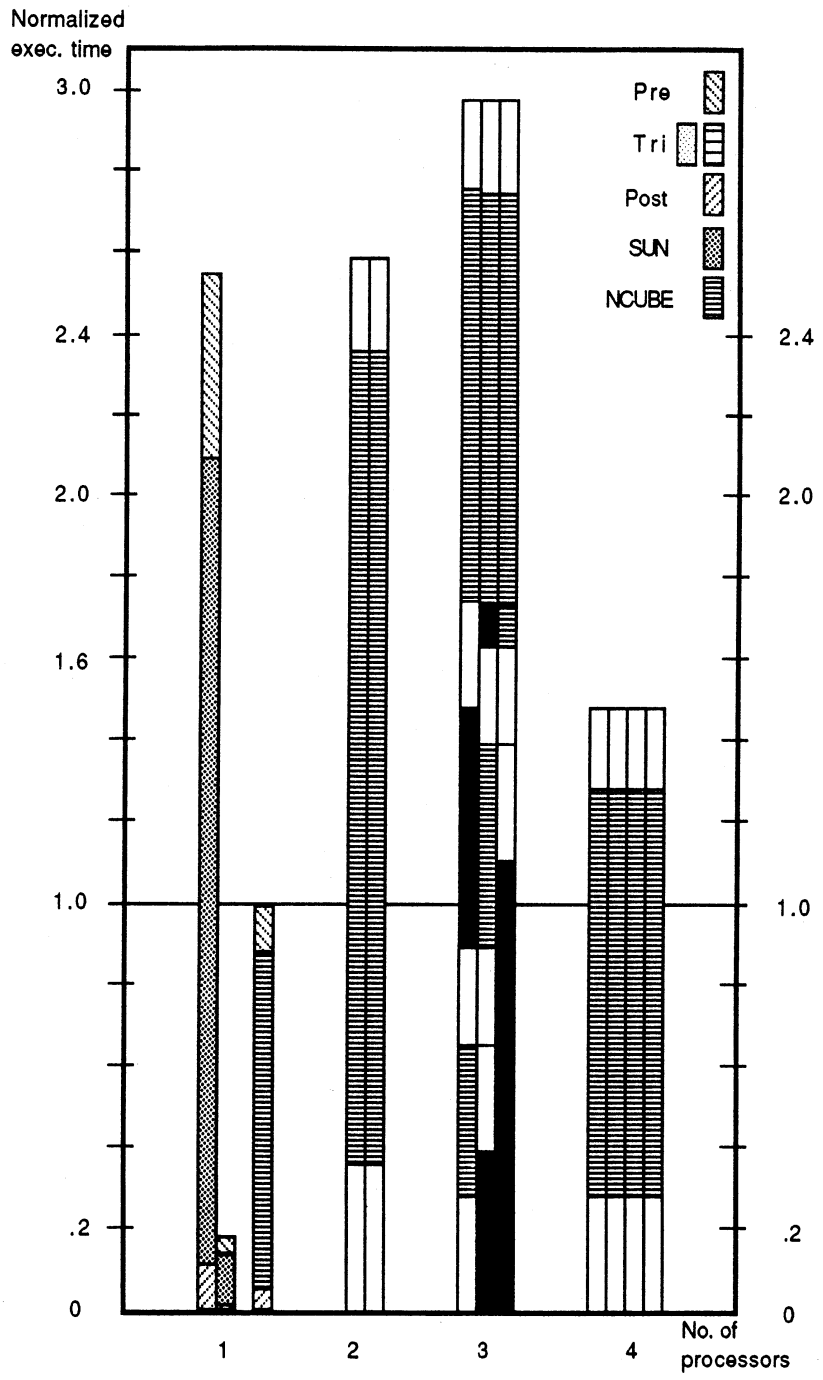

Figure 17: Summary of relative execution performance for triangulation algorithm (Gantt charts). One algorithm is used for uniprocessor cases, four identical algorithms are used for the two, three, and four processor cases. Execution times for the SUN are without and with FPA.

4 Memory requirements.

The memory requirements for the triangulation algorithm are summarized in Figure 18. The memory requirements are split into five parts: the program executable code, the local variables, the global variables, the data input buffer, and the data output buffer. The values are based on output for one triangulation algorithm from the "mapn" utility [NCUBE]. Estimated SUN memory requirements are less than 4 kbyte program and 224 byte data (no message passing). For the NCUBE node the memory requirements are less than 4.5 kbyte program and 380 byte data (including input/output message passing, but not inter-processor message passing). Of the program figures more than 65% are related to standard routines such as trigonometric functions.

	Program [bytes]			Data [bytes]				
	Triang. algorithm	Standard routines	Total	Local variables	Global variables	Input buffer	Output buffer	Total
SUN	1328	2715	4043	148	76	-	-	224
NCUBE	1500	2849	4349	180	76	58	66	380

Figure 18: Triangulation algorithm memory requirements for SUN and NCUBE node.

5 Design parameter sensitivity.

5.1 Method.

The sensitivity of triangulated data to deviations in sensor head design parameters is analyzed using design data from a prototype sensor head with four emitters. Nominal values are (definitions see Figures 2 and 3): cone angle α =71.5° (1.2479093 radians), baseline length b = 11.0 mm, lens position z_{lens} = 12.0 mm, and emitter angle θ_{emit} = 45° (0.7853981 radians). As inputs are chosen arbitrarily x_{det} = 2.0 mm, y_{det} = 1.5 mm, and n_{emit} = 0. This gives the nominal output x_{tri} = -9.2199 mm, y_{tri} = -6.9150 mm, z_{tri} = 67.3197 mm, and d_{tri} = 56.5074 mm from equations (7) to (11).

The deviation influenced $x_{tri}(\delta\alpha)$ is calculated, where $\delta\alpha$ is the parameter deviation (in this case the cone angle deviation). Then $\delta x_{tri} = x_{tri} - x_{tri}(\delta\alpha)$ is calculated, where x_{tri} is the nominal value.

The x_{tri} calculations above for $\delta\alpha$ are repeated for δz_{lens} and δb. The computations are repeated for y_{tri}, z_{tri}, and d_{tri}. Plots in this section show x_{tri}, y_{tri}, z_{tri}, and d_{tri} as a function of deviations in the design parameters. For cone angle α deviations the interval (-0.2,+0.2) radians is considered (approximately 11.5° maximum deviation). For lens position z_{lens} and baseline length b the deviation interval (-2.0,+2.0) mm is chosen.

5.2 Sensitivity of triangulated data to deviations in the cone angle.

The first two plots contain an asymptote (Figures 19 and 20). Curves are not shown (extrapolated) around the asymptotes since these areas are of minor interest. At $\delta\alpha$ = +0.115 radians (approximately 6.6°) all triangulated values approach infinity (Figure 19). The curve for distance d_{tri} is similar to the curve for z_{tri} to the left of the asymptote and therefore is not plotted. Figure 20 shows that positive α deviations are more critical than negative α deviations.

The z_{tri} value is more sensitive to cone angle deviations than the x_{tri} and y_{tri} values. This is natural since z_{tri} is a function of one of the other two coordinates according to equation (10) or (11). A more detailed plot (Figure 21) shows the range (-1.0,+1.0) mm. The curve with the negative slope is z_{tri}. It is the most sensitive coordinate and reaches the deviation $\pm$1 mm at cone angle deviations of approximately -0.002 radians (0.15°) and +0.0015 radians (0.09°).

5.3 Sensitivity of triangulated data to deviations in the lens position.

Distance d_{tri} and coordinate z_{tri} are most sensitive to lens position z_{lens} deviations (Figure 22). The distance d_{tri} reaches a deviation of $\pm$1 mm for z_{lens} deviations -0.085 mm and +0.09 mm (Figure 23).

5.4 Sensitivity of triangulated data to deviations in the baseline length.

Baseline length b deviations give similar sensitivities for z_{tri} and d_{tri} (Figure 24). Here z_{tri} (d_{tri}) deviations of $\pm$1 mm are reached for the baseline length b deviations $\pm$0.17 mm (Figure 25).

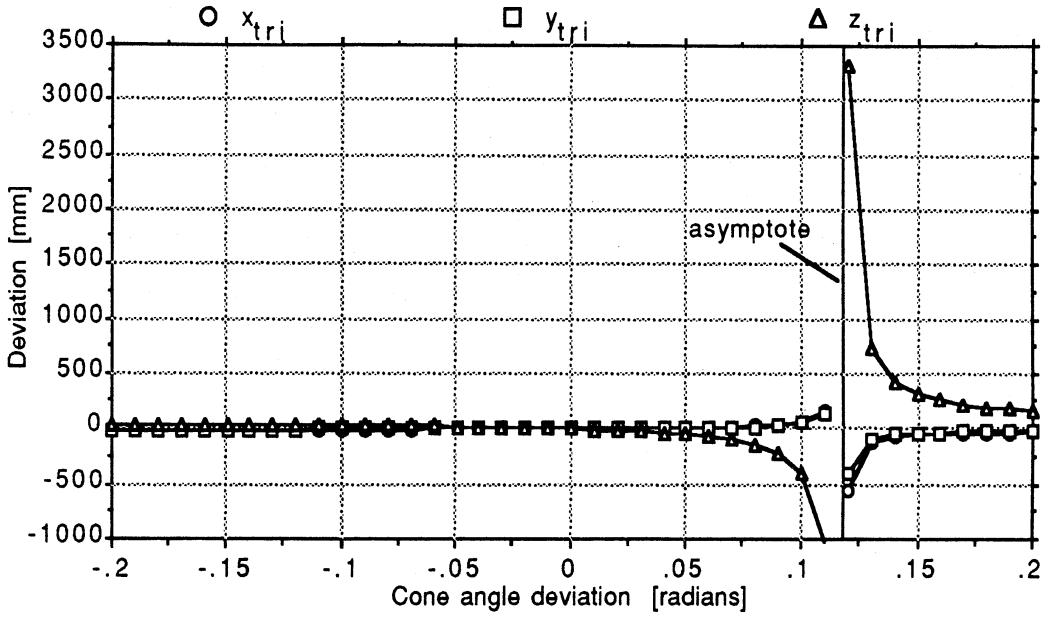

Figure 19: Sensitivity of triangulated data to cone angle α deviations - overview.

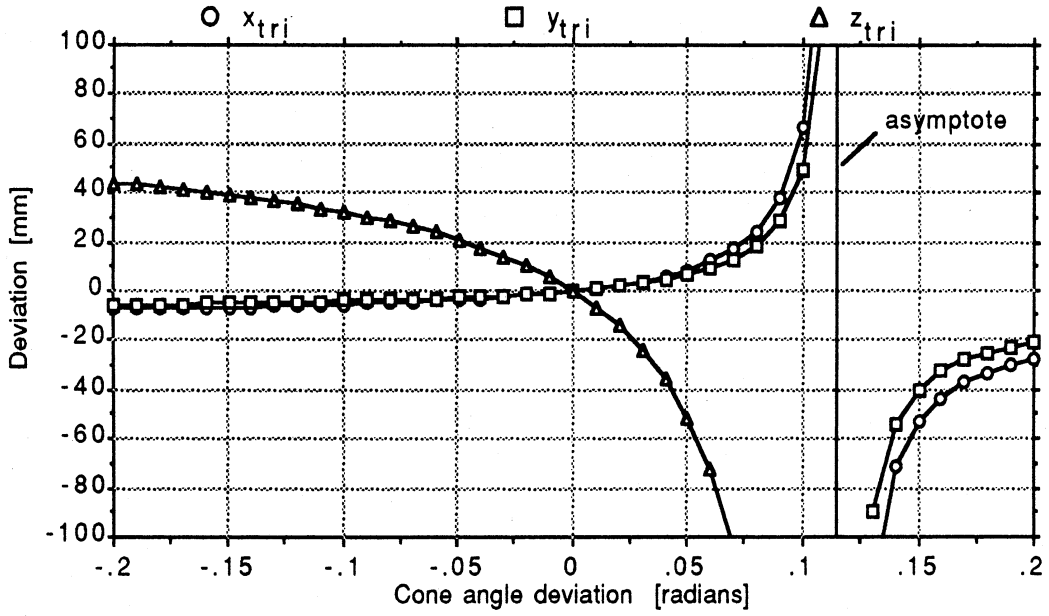

Figure 20: Sensitivity of triangulated data to cone angle α deviations.

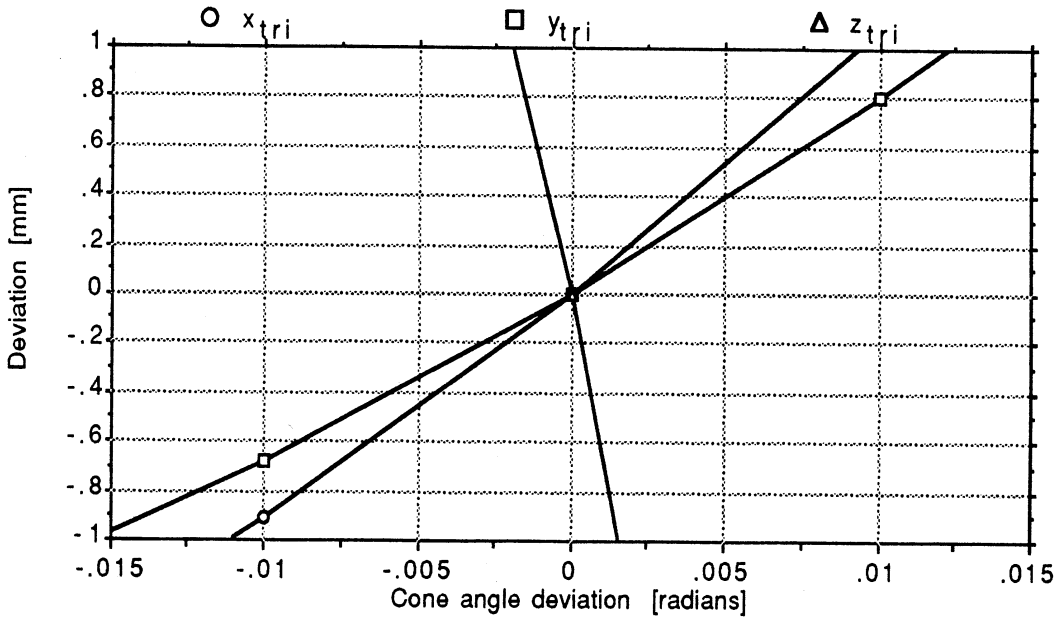

Figure 21: Sensitivity of triangulated data to cone angle α deviations - detail.

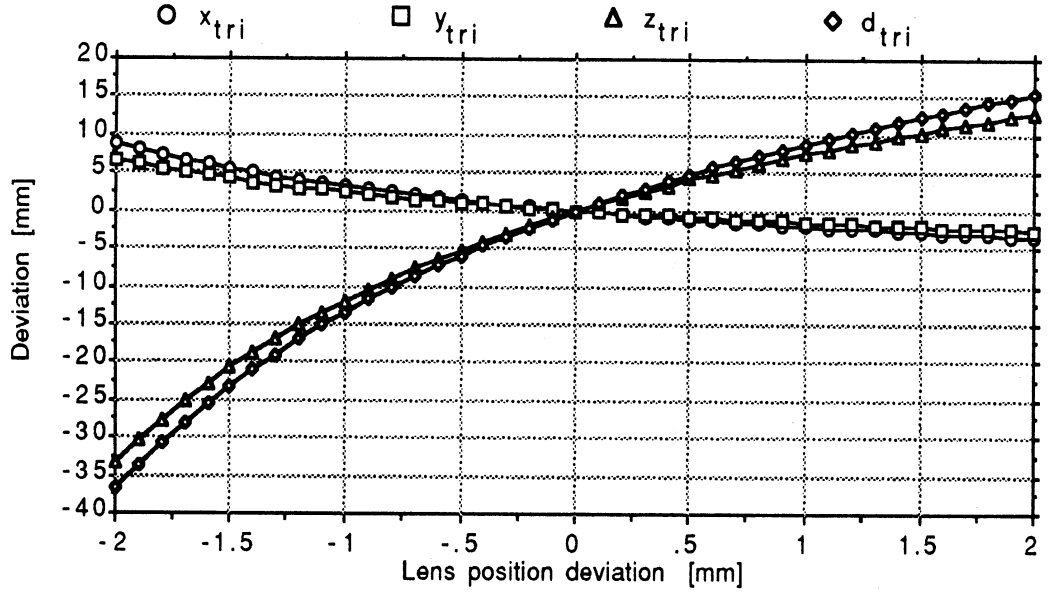

Figure 22: Sensitivity of triangulated data to lens position z_{lens} deviations.

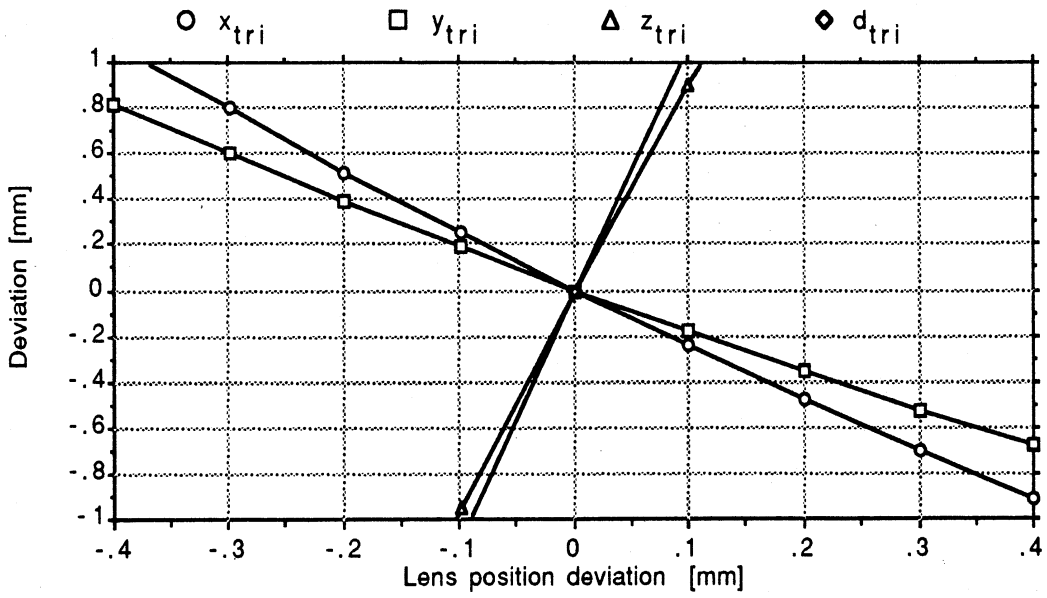

Figure 23: Sensitivity of triangulated data to lens position z_{lens} deviations - detail.

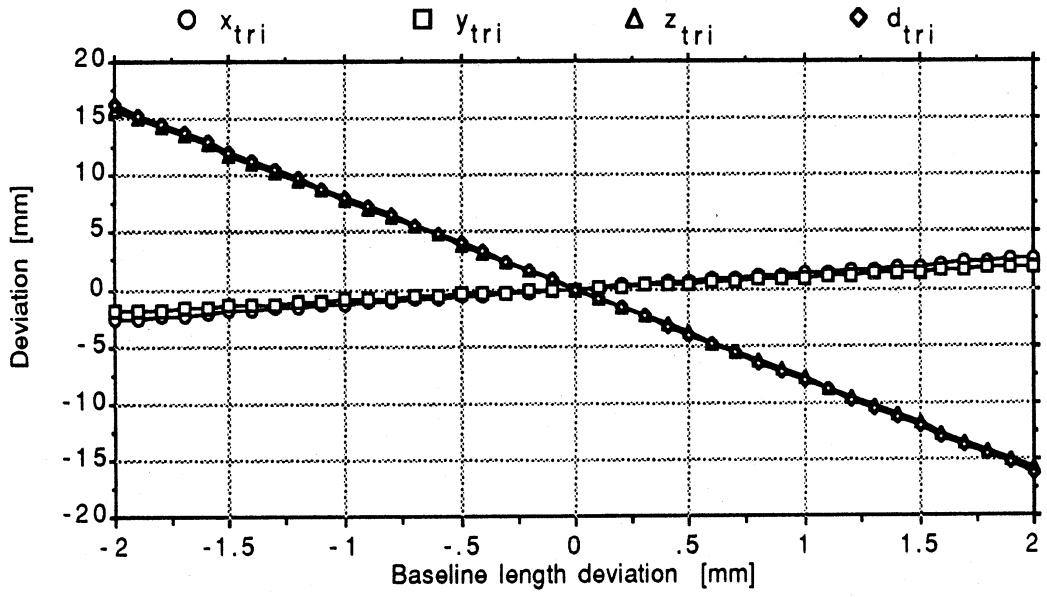

Figure 24: Sensitivity of triangulated data to baseline length b deviations.

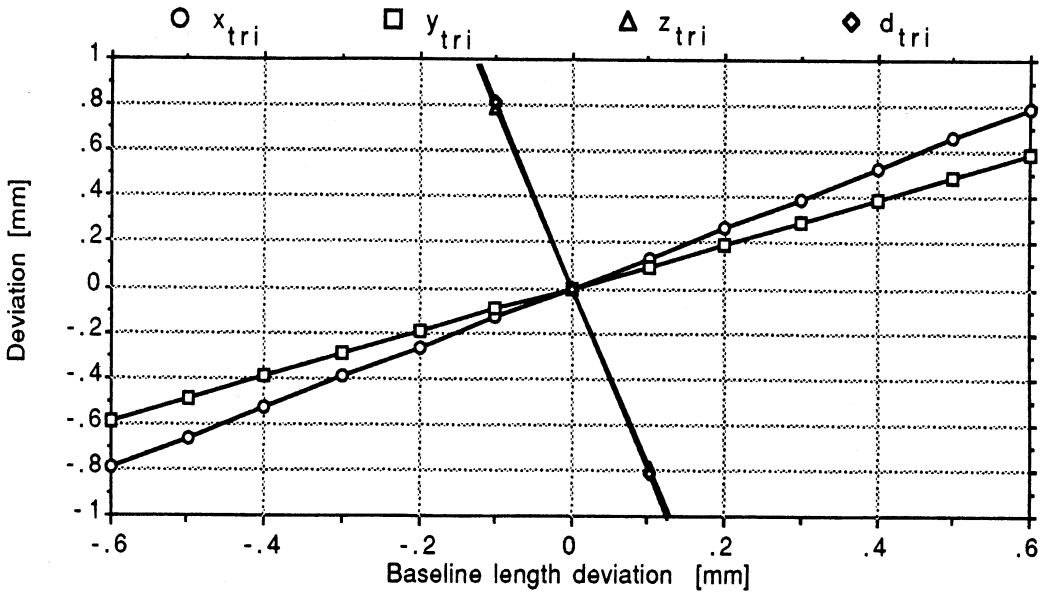

Figure 25: Sensitivity of triangulated data to baseline length *b* deviations - detail.

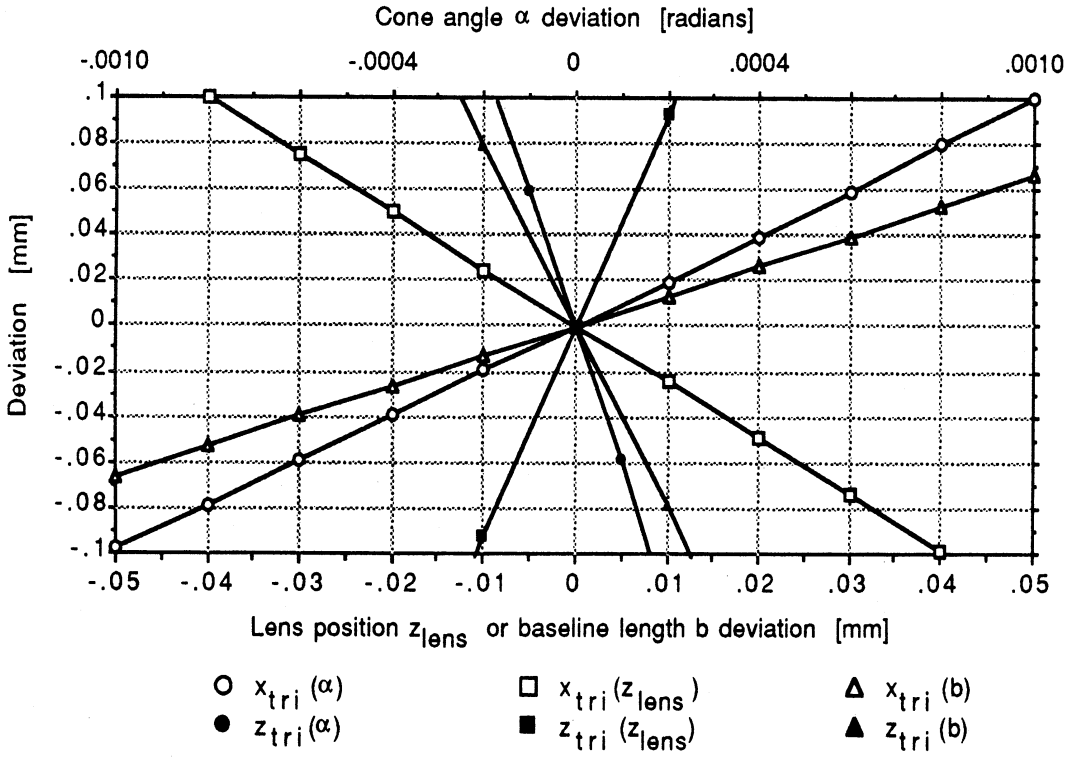

Figure 26: Sensitivity summary - fine detail.

5.5 Sensitivity of triangulated data to deviations in the emitter angle.

Since the emitter angle θ_{emit} may be used instead of the angle θ_{det} according to section 2.3.4, the design parameter θ_{emit} should also be analyzed. The only case when θ_{emit} is used is when $x_{det} = y_{det} = 0$. But then $x_{tri} = y_{tri} = 0$ according to equations (8) and (9). This gives an expression for z_{tri} which is independent of the θ_{det} angle (here equal to θ_{emit}) as is evident from equations (10) and (11). Thus the triangulated data are insensitive to emitter angle deviations.

5.6 Sensitivity summary.

For convenient comparison of sensitivity results sample curves are shown in Figure 26. Two curves for each design parameter are depicted, viz. coordinates x_{tri} and z_{tri}. Coordinate y_{tri} is less sensitive than x_{tri} (see previous graphs). The distance d_{tri} sensitivity is either the same as for the coordinate z_{tri}, or somewhat higher (see previous graphs).

The accuracy of the overall sensor system will be specified to be 0.1 mm. Output from the triangulation computations should be at least an order of magnitude better in accuracy. Therefore the curves in Figure 26 are shown for the fine detail ± 0.10 mm deviation range.

The results from the sensitivity analysis in this section will be used in a calibration procedure of the FO3DS sensor head.

6 References.

[Buzzard] G. Buzzard, "High Performance Communications on Hypercube Multiprocessors," Ph.D. thesis in preparation, The University of Michigan, Ann Arbor, MI.

[Hwang & Briggs] K. Hwang and F. Briggs, *Computer Architecture and Parallel Processing,* McGraw-Hill, 1985, Chapter 8.

[Kanade & Fuhrman] T. Kanade and M. Fuhrman, "A Noncontact Optical Proximity Sensor for Measuring Surface Shape," in *Three-Dimensional Machine Vision,* Kluwer 1987, pp. 151-192.

[Mudge & Abdel-Rahman] T.N. Mudge and T.S. Abdel-Rahman, "Vision Algorithms for Hypercube Machines," *Journal of Parallel and Distributed Computing,* Vol. 4, 1987, pp. 79-94.

[NCUBE] NCUBE Users Handbook, Version P2.1 October, 1987.

[SUN] SUN C Compiler User Command Manual, 20 August 1986.