FOA-R--96-00386-505--SE
April 1997
ISSN 1104-9154

Methodology report

Alexander Wahlstedt, Jesper Fredriksson,
Karsten Jöred and Per Svensson

Submarine Tracking by Means of Passive Sonobuoys

I. Design of a simulation model and system

Div. of Command and Control Warfare Technology
SE-581 11 LINKÖPING, SWEDEN

Defence Research Establishment
Division of Command and Control Warfare Technology
P. O. Box 1165
SE-581 11 LINKÖPING
SWEDEN

FOA-R--96-00386-505--SE
April 1997
ISSN 1104-9154

Alexander Wahlstedt, Jesper Fredriksson,
Karsten Jöred and Per Svensson

Submarine Tracking by Means of Passive Sonobuoys

I. Design of a simulation model and system

Distribution: FMV:Span, FMV:MML, KTH/NADA

FOA: MTC/OA, 64, 65, 73

0

Dokumentets utgivare Försvarets Forskningsanstalt Avdelningen för Ledningssystemteknik Box 1165 581 11 LINKÖPING	**Dokumentnamn och dokumentbeteckning** FOA-R--96-00386-505--SE	
	Dokumentets datum April 1997	**Projektnummer** E 1434
	Projektnamn (ev förkortat) Datafusion av data från undervattenssensorer	
Upphovsman(män) Alexander Wahlstedt, Jesper Fredriksson, Karsten Jöred och Per Svensson	**Uppdragsgivare** FM	
	Projektansvarig Per Svensson	
	Fackansvarig Per Svensson	
Dokumentets titel Följning av ubåt med hjälp av passiva hydrofonbojar I. Konstruktion av en simuleringsmodell och ett spelprogram		

Denna rapport innehåller lösningsförslag till delar av problemet att simulera upptäckt och följning av en fientlig ubåt med hjälp av sonobojar i en grund skärgårdsmiljö. De metoder som används för att beräkna ubåtens position - hyperboliska positionsbestämningsmetoden och sonarekvationen - finns beskrivna i denna rapport, såväl som några alternativa metoder. Vidare innehåller rapporten en beskrivning av simuleringssystemets målsättning. Delar av simuleringsprogrammet - i huvudsak baneditor, bojutläggning och positionsbestämning - har hittills implementerats i det objektorienterade språket Eiffel. Baneditorn PathEd används till att interaktivt definiera ubåtens bana. Bojutläggning och positionsbestämning görs i grafisk dialog med programmet SubTrack, version 1. Systemarkitekturen för PathEd och SubTrack version 1 finns beskriven i rapporten.

Nyckelord Ubåtsdetektion, hydrofonbojar, hyperboliska positionsbestämningsmetoden, sonarekvationen, datafusion		
Övriga bibliografiska uppgifter	**Språk** Engelska	
ISSN 1104-9154	**ISBN**	
	Omfång 47 sidor	**Pris**
	( ) Begränsad distribution	

Distributör (om annan än ovan)

Issuing organization	Document name and doc.ref.no.	
Defence Research Establishment Division of Command and Control Warfare Technology P.O. Box 1165 SE-581 11 LINKÖPING SWEDEN	FOA-R--96-00386-505--SE	
	Date of issue April 1997	**Project No.** E 1434
	Project name (abbrev. if necessary) Fusion of underwater sensor data	
Author(s) Kristian Johansson and Per Svensson	**Initiator or sponsoring organization** FM	
	Project manager Per Svensson	
	Scient. and techn. responsible Per Svensson	

Document title

Submarine Tracking by Means of Passive Sonobuoys.
I. Design of a simulation model and system

Abstract

This report treats part of the problem of optimal placement of sonobuoys with the purpose of detecting and tracking hostile submarines in a shallow water, archipelagic scenario. The methods - the Hyperbolic Fix Method and the Sonar Equation - used to estimate the position of a submarine and its stochastic variance using (simulated) passive sonobuoys are described. Finally, the function and architecture of the currently completed programs PathEd and SubTrack version 1, are described.

Key words

Submarine detection, Sonobuoys, Hyperbolic Fix Method, Sonar equation, Data fusion

Further bibliographic information	Language	
	English	
ISSN 1104-9154	**ISBN**	
	Pages 47 pages	**Price**
	( ) restricted distribution	

TABLE OF CONTENTS

1.0 Introduction

This report is an extended and edited version of Alexander Wahlstedt's Master of Science thesis, which was written for the Department of Numerical Analysis and Computing Science (NADA), KTH, Stockholm in March, 1996. Thesis supervisors were Per Svensson, Director of Research in Computer Science at FOA and Stefan Arnborg, Professor in Computer Science at NADA.

Per Svensson has been project leader and initiator of the study which deals with *the optimal placement of passive sonobuoys in order to detect and track a hostile submarine.* The study requires the development of a simulation model and system capable of simulating a submarine hunt using passive sonobuoys and of computing their optimal placement.

During the spring and summer of 1996, continued development of the simulation system took place. In this phase, the design and implementation of a first version of the system was completed. Involved in this work were initially Karsten Jöred and later Göran Neider, FOA. During the summer months, Jesper Fredriksson, an M Sc student in Engineering Physics at KTH, completed most of the implementation work of the first version of the system, called SubTrack Version 1. This version simulates a submarine moving in a two-dimensional area, and by use of simulated sonobuoys, estimates its position.

This report contains:

- a statement of the final goal of the study
- a study of methods and mathematical algorithms that could be used to (1) estimate the position of the submarine and (2) calculate the signal-to-noise ratio at the positions of the sonobuoys
- architecture and overall design of the Path Editor, used to interactively create the path of the target
- architecture and overall design of SubTrack version 1.

2.0 Summary

The final goal of the study is to determine possible benefits (if any) of using reactive planning and multi-sensor data fusion in the problem of determining and tracking the position of a submarine in archipelagic anti-submarine warfare (AASW). A computer program has been developed, which can simulate a submarine hunt using passive sonobuoys. The final version of the model is planned to include a mathematical optimization algorithm which calculates the optimal positions of the sonobuoys.

The first version of this program simulates a submarine that follows a predefined polygonal path contained in a two-dimensional gaming area. During the game, the user places sonobuoys at arbitrary locations within the area. The information acquired from the sonobuoys is used to calculate the position of the sub. This is done by use of the Hyper-

bolic Fix Method, and the signal-to-noise ratios at the positions of the sonobuoys is calculated by use of the Sonar Equation. The sonobuoy-position uncertainty and the sonobuoy-information uncertainty is taken into account, thus the area of an ellipse depicts the probable location of the sub.

Ongoing work includes development of a mathematical optimization algorithm that estimates the optimal positions of the sonobuoys. This requires that a method for predicting the (near) future position of the target is also developed.

3.0 Overview of Sonobuoy Use

The term sonobuoy [1] refers to a microphone that is deployed from a platform to become submerged in the water and that provides information about the local sound amplitude, as a function of frequency and time. In AASW, the buoys are usually anchored on the sea-floor, whereas in deep ocean ASW, the buoys are free-floating. Sonobuoys are used to estimate and track the position of underwater objects that emit sounds, in this case submarines. The platforms that carry this equipment are usually helicopters or airplanes, but they can also be surface ships. Here, we have studied *passive* sonobuoys; an advantage of these compared to *active* sonobuoys, which emit a sound signal, is that they do not reveal their presence. On the other hand, to measure the position of a target using passive buoys, one needs at least three buoys which hear the target simultaneously, and the position must be calculated from these signals using some kind of data fusion algorithm.

A sonobuoy registers not only the sound from a possible target, but all sounds above a certain level. Other sound sources could, for example, be boats, animals, waves or wind. The sound from such sources is called background noise and makes the signal of interest uncertain and hard to distinguish. To reduce background noise, the sonobuoy system needs so-called *integration time*. Up to a limit, the longer the integration time, the more accurate is the information from the system. However, there is always a possibility that the sonobuoy system will make a wrong decision; it may decide that there is a signal though there is none (false alarm), or it may decide that there is no signal when in fact there is one (no-detection possibility). Distinguishing the signal of interest from background noise has become increasingly difficult in recent years due to development of extremely quiet subs. In addition to this, in some environmental conditions a submarine could hide between layers of different temperatures or different salt concentrations which greatly influence the sound propagation and make the target even harder to detect. In a shallow-water archipelago, these problems are further compounded by strong reverberation effects, caused by sound reflection from the sea-floor, the surface, and islands, which leads to subsequent interference phenomena.

It may be difficult to decide where to horizontally and vertically place the sonobuoys because the volume where the sub could be located is usually far too large to be completely covered by the detection range of the buoys. In addition to this, the information provided by a single sonobuoy is - as mentioned above - not sufficient to estimate the position of a possible sub. It should be made clear that a single buoy cannot even deter-

mine a certain distance range for the sub. The reasons for this are (1) that the source level is usually not known and (2) that the sound propagation circumstances could significantly differ between positions that may be located only some hundred meters apart. For example, in good conditions, a 100 dB source level could be heard over a distance of several kilometers while the same source level may only be heard over a distance of a few hundred meters at a position where the conditions are poor.

The placement problem is a matter of timing as well. To have a chance to detect a submarine, the buoys must be dropped into the water soon after the first alarm of the presence of a possible sub. This, together with the fact that only a limited number of sonobuoys are available, suggests that the commander should devise and use a sonobuoy placement strategy. Other timing factors are the deployment time (the amount of time required to deploy and activate a buoy at a certain position) and the lifetime of a particular sonobuoy.

4.0 The Goal

The final goal of this study is to determine the applicability of reactive planning and multi-sensor data fusion in a Swedish archipelagic anti-submarine warfare scenario (AASW). Reactive planning means in this case that recent sensor information is used in the placement of further sonobuoys, and multi-sensor data fusion means that information from several buoys is combined in such a way that the combined information is more specific and therefore more useful than the unprocessed collection of separate pieces of sonobuoy information.

The study is intended to provide at least partial answers to the following questions:

1. Can the application of reactive planning and multi sensor data fusion contribute in solving the problem of submarine detection and tracking in shallow water, archipelagic scenarios?
2. If the study supports this concept, can we demonstrate convincingly and quantitatively what difference it could make, including showing that these methods can outperform a human decision maker?

4.1 Optimization Algorithm

To achieve this goal FOA has developed an algorithm that computes the optimal placement of the sonobuoys. By optimal we mean that for a given number of available buoys and a given minimum tracking accuracy, tracking time is as long as possible. The tracking problem would be trivial if one had sonobuoys enough to cover the entire possible area (even volume in cases where a two-dimensional view of the problem is inadequate) with a sufficiently dense grid of sensors, but granted that there are too few sonobuoys to do that, *when and where* should they be placed ? The answer is not necessarily to drop all buoys at one time; a better way would probably be to drop some of them first, then wait a while to be able to take into consideration the information received from the first drop, then apply that information to the placement of another (batch of) sonobuoy(s), and so on. Such an

approach is often called *adaptive planning*.

This optimization algorithm is part of a computer model, capable of simulating sonobuoys in a search for a submarine. The purpose is *not* to build a complete model which can immediately be used for real situations. Instead, the model is to be used for deciding *if* there are good reasons to develop a complete system, or if humans make more competent decisions than the optimization algorithm can make. The answer to this question might be found by simulating several submarine-hunt situations and comparing the results of human decisions with those of the optimization algorithm. If the conclusion is that the computer makes better decisions, then one should continue by developing a more realistic model. If the case is the opposite, the model might still be used to train sonobuoy operators. In the future, the model might also be used to train a submarine captain. For this, one would replace the fixed submarine path in Version 1 by a submarine navigation simulator.

As mentioned above, the first model is in many ways simplified as compared to a real situation. However, our hypothesis is that if an optimization approach works in an idealized model, then it is not unlikely that it could be made to work in a more complex and realistic model as well.

4.2 Relevance of the Model

To make the model relevant to the solution of the stated practical problem, it is necessary that realistic values for problem parameters be used. Such parameters include the following: the minimum time from first detection to first effective buoy deployment; the sensor detection ranges; the relation between speed and sound emission from the sub; the minimum signal integration time to achieve acceptable signal-to-noise ratios; the uncertainty values for buoy locations and time differences of arrivals; etc.

It is also important that the goal of having a passive, non-detectable system can be satisfied (at least to a working degree) and that the principle can still be used in disturbed conditions such as presence of intense surface traffic, multiple targets, bad weather conditions, etc.

In the model, the submarine's path is predefined, which means that the sub is modelled either as if it does not know that it is hunted, or at least, does not make any reactive maneuvers to avoid being followed. The question whether the sub could make use of information about the position of sonobuoys to plan effective evasive maneuvers is beyond the scope of this report.

This model does not attempt to attack all these problems, nor are we proposing or analysing a design for a future underwater surveillance system. Our goal is the more modest one to clarify, or at least illustrate, the possible effects of using multisensor data fusion and optimal sensor allocation in a simplified gaming scenario: a single target which moves along a predetermined path is to be followed as long as possible, given a limited number of available sonobuoys and prespecified tracking performance, i.e., a minimum position estimation error and a threshold tracking probability are given gaming conditions.

An open question to us is whether future submarines will be at all detectable by passive sonobuoys even within the short ranges we envision, a few hundred meters.

Apart from the issue of detectability, even with current submarine designs there is the following technical flaw in the concept: the lifetime of existing sonobuoys is limited because (1) the energy storage capacity versus consumption is low and (2) sonobuoys are preset to sink to the ocean floor after a predefined time, maximum a few hours. In AASW, this is not an ideal situation since the sub may stay silent and lay still on the seafloor for several days if necessary. Thus, with current sonobuoy design, the operator has to expend, say, three buoys every third hour, just to keep the silent submarine from escaping.

5.0 Simulation Model Version 1

Version 1 of the simulation model is designed to simulate a moving submarine in a two-dimensional, rectangular area. Using information from simulated passive sonobuoys, it estimates the probable position of the sub and represents this estimate conceptually and visually by an elliptical area. The Hyperbolic Fix Method (See Section 6.1 on page 10) is used for estimating the position of the sub, and the Sonar Equation (See Section 8.0 on page 17) is used to estimate the signal-to-noise ratio at the positions of the sonobuoys. The sound will reach the sonobuoys by a straight-line route, in other words, neither sound refraction nor reflections from features in the environment will be considered. Although this is a strong simplification of real conditions, we think it can be defended not only by referring to the model's limited ambitions, but also to the fact that we study a concept where the distance between target and sensor is typically much shorter than is usually assumed in sonar detection models.

In the model, time is discretized. For each time step the signal-to-noise ratio at the locations of the sonobuoys is computed; if the ratio is considered to be sufficiently high, and if the integration time is reached, the information from that sonobuoy is taken into account. Next, the time differences of arrivals (TDOA) for the sound reaching the sonobuoys is measured. Using this information, the most likely position together with its statistical error (its variance) will be estimated.

5.1 Submarine

The sub follows a predefined polygonal path (cf. Section 4.2 on page 8) with a predefined speed associated with each path segment. There is also an option to pause for a predefined time interval at a given point. The path of the sub will usually start and end at the border of the gaming area. Entry into the area is followed by a first detection which is being announced to the decision maker, thereby starting up the game.

The sound level of the sub is dependent on its speed (Figure 1), and the level decreases, as described by the Sonar Equation, on its way to the sonobuoys (Figure 2).

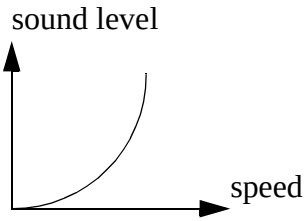

FIGURE 1. The sound level of the sub is dependent on its speed.

5.2 Sonobuoys

The user decides the arbitrary locations of the sonobuoys during run-time. Sonobuoys can not be retrieved for reuse during a game and only a limited number, which is set by the user, of buoys are available. The integration time is modelled by disregarding the information from a particular buoy during a certain number of time steps, and the signal level must be above a given fraction of the noise level to be detected (Figure 2). The noise level will be set by the game leader. The deployment time is modelled as the sum of the activation time and the delivery time. The activation time is a constant and the delivery time is dependent on the distance between the current position of the buoy delivery platform and the intended position for the sonobuoy. (Figure 3).

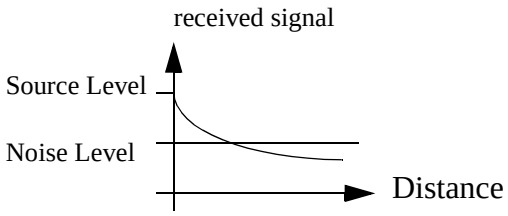

FIGURE 2. To be detected, a signal received by a sonar must be above a certain fraction of the noise level.

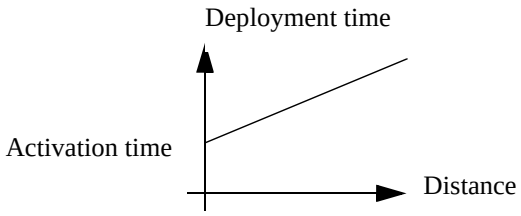

FIGURE 3. Deployment time = Delivery time + Activation time

6.0 Position Estimating Methods

In investigating methods for position estimation, we considered a few different approaches, which we discussed with experts in the field. We concluded that the Hyperbolic Fix Method would best fit our study. This and two other conceivable methods are explained in the following sections.

6.1 The Hyperbolic Fix Method

The Hyperbolic Fix Method is a method commonly used to calculate the positions of

underwater targets. It is based on measurements of the time differences between arrivals of the sound travelling from the target to each sonobuoy. This time difference can be estimated by computing the time difference which maximizes the cross-correlation between a time-domain sound pattern window for each pair of buoys. Using this, a system of equations can be set up whose solution is an estimate of the unknown position of the sub. To use this method, it is necessary that at least three sonobuoys hear the sub simultaneously.

When exactly three sonobuoys hear the sub, the target position can be found by solving a non-linear system of equations. This system of equations may be derived from the observation that to each pair of buoys corresponds a half-hyperbolic point set containing the position of the target (since a half-hyperbola is precisely the set of points whose distance to two given points have a given, constant signed difference). The system of equations expresses the requirement that the target position must coincide with the intersection of any pair of these half-hyperbolas.

When more than three sonobuoys hear the sub, a non-linear, overdetermined system of n equations can be formed. This system can be transformed into a linear system of n equations, plus one equation of the original type [2]. The linear system can be solved using a standard least squares method.

Figure 4 below illustrates the problem setup.

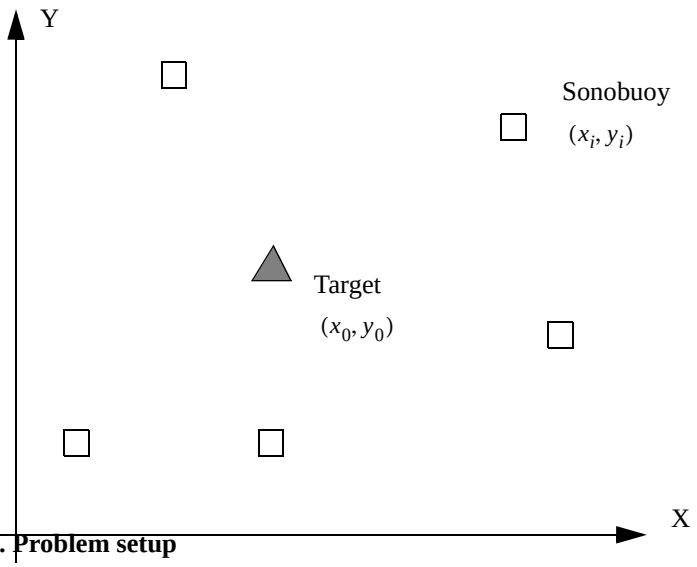

FIGURE 4. Problem setup

The problem setup will be the following:
The sonobuoys are numbered as i=1,2,...n.

(x_i, y_i)	= The position of the i:th sonobuoy
t_i	= The time of arrival for the sound reaching the i:th sonobuoy
(x_0, y_0)	= The unknown position of the target

t_0 = The unknown time when the sound was transmitted

c = The speed of sound

By expressing the distances in two ways, the following system of equations is obtained:

$$\begin{aligned}(x_1 - x_0)^2 + (y_1 - y_0)^2 &= c^2(t_1 - t_0)^2 \\ &\vdots \\ (x_n - x_0)^2 + (y_n - y_0)^2 &= c^2(t_n - t_0)^2\end{aligned} \qquad \text{(EQ 1)}$$

By subtracting the sum of all equations from all the others, a linear system results. By choosing as origin of the coordinate system the arithmetical mean of all sonobuoy positions, the equations are simplified. Below all coordinates are assumed to be expressed in this *average system*, as it will henceforth be called. A least-squares solution to this system is (see [2]):

$$x = \frac{1}{2} \cdot (A^T A)^{-1} \cdot (A^T \cdot b) \qquad \text{(EQ 2)}$$

where

$$x = \begin{bmatrix} x_0 \\ y_0 \\ ct_0 \end{bmatrix}$$

The (n x 3) -matrix A and the (n x 1) -column vector b are defined by:

$$A = \begin{bmatrix} x_1 & y_1 & -(c \cdot t_1) \\ x_2 & y_2 & -(c \cdot t_2) \\ \dots & \dots & \dots \\ x_n & y_n & -(c \cdot t_n) \end{bmatrix} \qquad b = \begin{bmatrix} x_1^2 + y_1^2 - c^2 t_1^2 \\ x_2^2 + y_2^2 - c^2 t_2^2 \\ \dots \\ x_n^2 + y_n^2 - c^2 t_n^2 \end{bmatrix}$$

Note the obvious fact that the average system will change whenever a buoy is added to or removed from the system of equations.

6.2 The Doppler Fix Method

Another method for estimating the position of the target is the Doppler Fix Method. As the name reveals, it is based on the doppler effect. Figure 5 depicts the measurement situation.

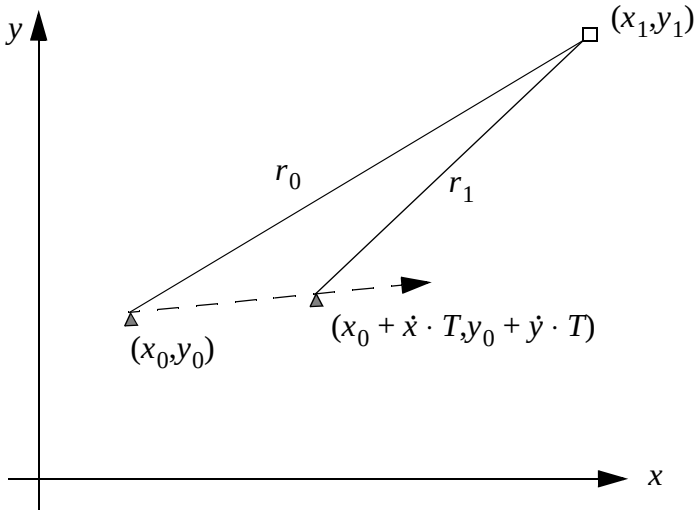

FIGURE 5. The Doppler Fix Method

(x_0,y_0)	= target's position at t=0
(x_1,y_1)	= buoy position
$\dot{x}$	= target's x-velocity
$\dot{y}$	= target's y-velocity
$(x_0 + \dot{x} \cdot T, y_0 + \dot{y} \cdot T)$	= target's position at t=T
r_0	= distance between target and buoy at t=0
r_1	= distance between target and buoy at t=T
f	= frequency of transmitted sound
f'	= frequency of received sound
c	= speed of sound

What frequency will the buoy register?

Assume that a wave crest is transmitted from the target at time t=0. This wave crest will reach the buoy at time:

$$t_1 = \frac{r_0}{c} \qquad \text{(EQ 3)}$$

T seconds later, the next wave crest is transmitted. It will reach the buoy at time:

$$t_2 = \frac{r_1}{c} + T \qquad \text{(EQ 4)}$$

(assuming that the target's velocity vector is constant during the period time).

The period time registered by the buoy will therefore be:

$$T' = t_2 - t_1 \qquad \text{(EQ 5)}$$

which yields the frequency:

$$f' = \frac{1}{t_2 - t_1} = \frac{1}{T + \frac{r_1 - r_0}{c}} \qquad \text{(EQ 6)}$$

where

$$r_0 = \sqrt{(x_1 - x_0)^2 + (y_1 - y_0)^2} \qquad \text{(EQ 7)}$$

and

$$r_1 = \sqrt{(x_1 - (x_0 + \dot{x} \cdot T))^2 + (y_1 - (y_0 + \dot{y} \cdot T))^2} \qquad \text{(EQ 8)}$$

There are five unknown variables, $x_0, y_0, \dot{x}, y, \dot{f}$, which means that five buoys must hear the target simultaneously. A system of equations can then be stated, one equation of the same kind as EQ 7 for each buoy.

The problem with using the Doppler Fix Method in this context is that the frequency differences are too small to be reliably detected. The following example illustrates this:

v	= 3 m/s
f	= 100 Hz
c	= 1500 m/s

Assume, for highest possible doppler shift, that the target is moving directly towards the buoy. This gives:

$$f' = \frac{1}{T + \frac{v \cdot T}{c}} = \frac{1}{T\left(1 + \frac{v}{c}\right)} = \frac{f}{1 + \frac{v}{c}} = \frac{f}{1 + \frac{1}{500}} \approx f \qquad \text{(EQ 9)}$$

The difference between f and f' is so small that it will drown in measurement errors. Therefore, the Doppler Fix Method cannot be used for passive sonobuoys and slow, low-frequency noise targets. For active sonobuoys, on the other hand, it is quite possible to use it.

6.3 The Cross Bearing Method

This technique is based on the combination of two or more direction, or *bearing*, measurements. Each bearing measurement can be accomplished by a cluster of two (or preferably three) passive sonobuoys which are placed relatively close to each other. The phase difference between the received signals is measured. From that information, the bearing to the target can be calculated. By combining bearing information from at least two such clusters, the position of the sub can be estimated. Thus, the computations are separated into two phases with different objectives, in contrast to the hyperbolic fix case. Used in a configuration where the sensors in a cluster are rigidly fixed with respect to each other, this technique has been reported to give good results, but such is not the case in our scenario. We have not analysed further the usefulness of this technique for our tracking problem.

7.0 Uncertainty Ellipse Computation

The least squares solution for the position of the target provides an estimate of the mean value of the statistical distribution for the submarine's position. To model also the variance, or mean square error, of the position, the uncertainty for (1) the position of the sonobuoys and (2) the time differences of arrivals need to be represented as stochastic variables. The uncertainty is modelled by assuming that these variables are Gaussian-distributed with zero mean:

$x_i + dx_i$ where $\mathrm{dx}_i \in N(0, s_1^2)$ = the ith buoy's x-position

$y_i + dy_i$ where $\mathrm{dy}_i \in N(0, s_1^2)$ = the ith buoy's y-position

$t_i + dt_i$ where $\mathrm{dt}_i \in N(0, s_2^2)$ = the arrival time of the transient to the ith buoy

The positions of the sonobuoys and the arrival times are now considered as normal distributed stochastic variables. From this follows that (x_0, y_0), the position of the target, is a two-dimensional stochastic variable, whose distribution is approximately normal (for a detailed derivation, see Appendix A):

$$f(z) = \frac{1}{2\pi\sqrt{detK}} e^{-0{,}5 z^T K^{-1} z}$$

where

$$z = \begin{bmatrix} x & y \end{bmatrix}^T$$

$$K = \begin{bmatrix} \sigma_{11}^2 & \sigma_{12} \\ \sigma_{21} & \sigma_{22}^2 \end{bmatrix}$$

σ_{11}^2 = the variance for dx_0

$\sigma_{12} = \sigma_{21}$ = the covariance between dx_0 and dy_0

σ_{22}^2 = the variance for dy_0

dx_0 and dy_0 can be approximately computed through their Taylor expansions:

$$dx_0 \approx \frac{\partial x_0}{\partial x_1} dx_1 + \ldots + \frac{\partial x_0}{\partial t_n} dt_n$$
$$dy_0 \approx \frac{\partial y_0}{\partial x_1} dx_1 + \ldots + \frac{\partial y_0}{\partial t_n} dt_n$$

where x_0 and y_0 are given from EQ 2. (EQ 10)

The position of the submarine will be depicted by an ellipse whose centre will be the computed position (x_0, y_0). The equation of the ellipse is: $(dx_0, dy_0)K^{-1}(dx_0, dy_0)^T < R^2$

This particular ellipse is chosen because there is no smaller area that contains the target with the same (or greater) probability. From the normal distribution assumption follows that this target containment probability p equals:

$$p = \int_{ellipse} f(z)dz = 1 - e^{-\frac{R^2}{2}} \quad \text{(EQ 11)}$$

or equivalently:

$$R = \sqrt{-2\log(1-p)}$$

The half-axes of the ellipse are (see Appendix B):
$a = R\sqrt{\lambda_1}$ and $b = R\sqrt{\lambda_2}$

where λ_1, λ_2 are the eigenvalues of the matrix K.

When $\sigma_{12} \neq 0$ the rotation angle α relative to the x-axis is calculated from:

$$\tan\alpha = \frac{\lambda_1 - \sigma_{11}}{\sigma_{12}} \quad \text{(EQ 12)}$$

To compute the covariance matrix K from measured data, we use the relation:

$$K = G \cdot C \cdot G^T \quad \text{(EQ 13)}$$

Here C is the known diagonal covariance matrix for ($\underline{dx}$, $\underline{dy}$,$\underline{dt}$) , and G is the Jacobian matrix of (EQ 11), which is computed using formulae given in reference [2] (although unfortunately there are misprints in some of the equations).

8.0 The Sonar Equation

The Sonar Equation [3,4] states the relationship between the emitted sound level, the received sound level, the environment conditions, and the sonar equipment. It has been used for calculations of the maximum range of sonar equipment since the World War II, and has thus been well tested in a large number of real situations. As will become obvious shortly, the sonar equation can only provide a rough estimate of the sonar range unless a number of environment dependent parameters can be determined with required accuracy. This is not always feasible.

In the simulation model, it is used to calculate the signal-to-noise ratio at the positions of the different sonobuoys. Information for a particular sonobuoy will be taken into account if the signal level is high enough compared to the background noise at its position. The equation is:

$$(SL - TL) - (NL - DI) = DT \qquad \text{(EQ 14)}$$

where

SL = *Source Level.* This is the sound level of the source, for example from a submarine.

TL = *Transmission Loss.* This is the loss of the sound level while it travels from the source to the sonobuoy.

NL = *Noise Level.* This is the level of the background noise at the position of the sonobuoy.

DI = *Directivity Index.* This denotes the noise reduction capabilities of a particular sonobuoy.

DT = *Detection Level*. This is the minimum level of the sound at the buoy position required to detect the sub.

The table below includes the definitions of each parameter.

TABLE 1. Definitions of the Sonar Parameters

Parameter	Symbol	Explanation	Definition
Source Level	SL	at target position	$10 \cdot \log \frac{\text{source intensity}}{\text{reference intensity}}$
Transmission Loss	TL	one meter from source and at receiver	$10 \cdot \log \frac{\text{signal intensity at source}}{\text{signal intensity at receiver}}$
Noise Level	NL	at receiver position	$10 \cdot \log \frac{\text{noise intensity}}{\text{reference intensity}}$
Directivity Index	DI	at receiver position	$10 \cdot \log \frac{\text{noise intensity}}{\text{reduced noise intensity}}$
Detection Threshold	DT	at receiver position	$10 \cdot \log \frac{\text{source intensity}}{\text{noise intensity}}$

The sound intensity is defined by power per unit area, that is:

$$I = \frac{P}{A}[W/m^2] \qquad \text{(EQ 15)}$$

The sonar parameters are expressed in decibels, which is convenient when the intensity fluctuations are high. The number of decibels for a sound with intensity I is $10\log\frac{I}{I_0}dB$, where $I_0 = 0{,}67 \cdot 10^{-18} W/m^2$ is the reference intensity.

DT stands for detection threshold, given as a signal-to-noise ratio at the position of the receiver. The detection threshold is the minimum signal-to-noise ratio required to detect the target.

Figure 6 shows where the different parameters appear.

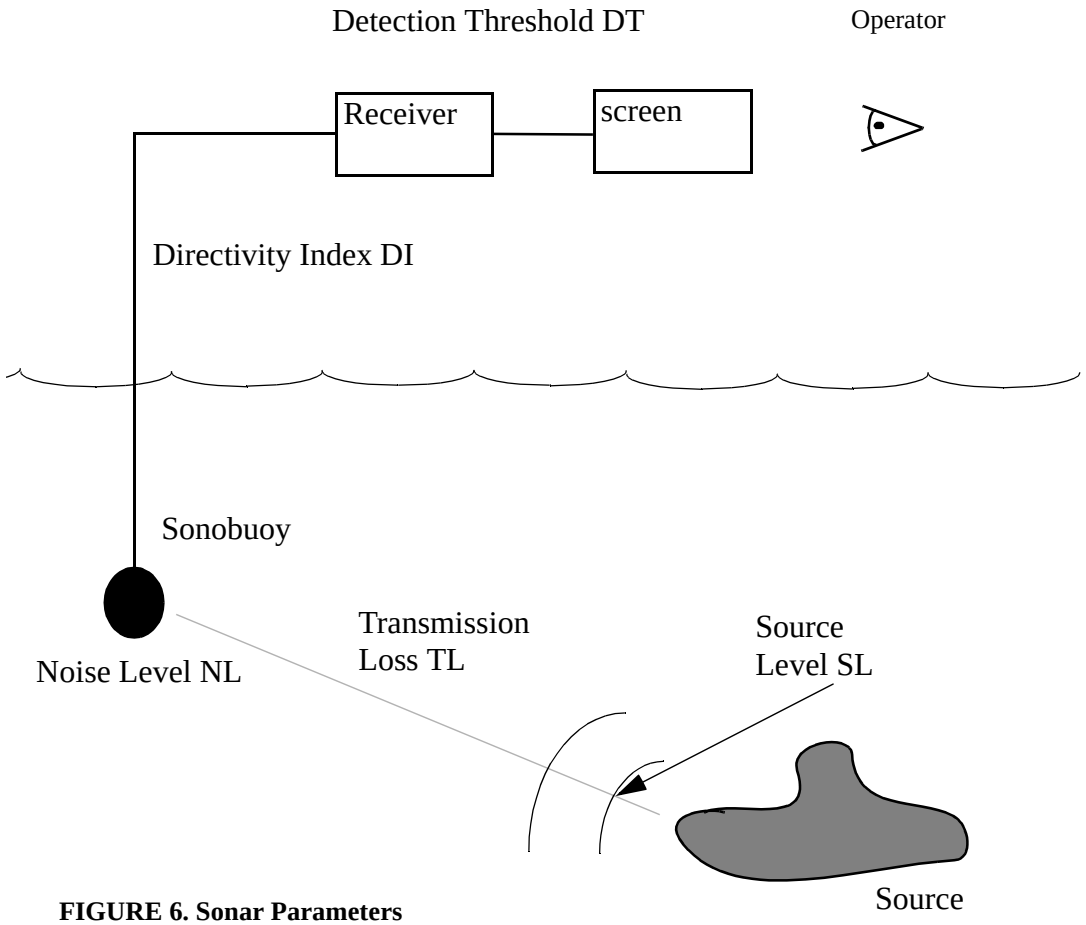

FIGURE 6. Sonar Parameters

Imagine, as the figure shows, a source (for example a submarine) that emits a sound with a certain level (SL dB). This sound travels through the water to the sonobuoy. On its way, the sound level decreases due to transmission loss (TL dB). The transmission loss mainly comes from the spreading of the intensity, but also from absorption. At the sonobuoy position, there is a certain noise level (NL dB). If this noise level is too high, it will be impossible to distinguish the source signal from the background noise. However, some sonar systems have noise reduction capabilities that lower the background noise by (DI dB). When the signal finally reaches the sonar operator, he has to make the decision whether there is a target or not. In some cases, the signal-to-noise ratio, DT, is so high that there is no doubt, but mostly that is a hard decision.

8.1 Derivation of the Sonar Equation

The source-to-noise ratio, in terms of decibels, can be expressed as:

$$(SL - TL) - (NL - DI)$$

This can be shown by:

$$
\begin{aligned}
&(SL - TL) - (NL - DI) \; = \\
&= \; 10\log\frac{I_{\text{target}}}{I_0} - 10\log\frac{I_{\text{target}}}{I_{\text{receiver}}} - 10\log\frac{I_{\text{noise}_1}}{I_0} + 10\log\frac{I_{\text{noise}_1}}{I_{\text{noise}_2}} \; = \\
&= 10\log\left(\frac{I_{\text{target}}}{I_0} \cdot \frac{I_{\text{receiver}}}{I_{\text{target}}} \cdot \frac{I_0}{I_{\text{noise}_1}} \cdot \frac{I_{\text{noise}_1}}{I_{\text{noise}_2}}\right) \; = \; 10\log\frac{I_{\text{receiver}}}{I_{\text{noise}_2}} \; = \; DT
\end{aligned}
\qquad \text{(EQ 16)}
$$

where $noise_1$ and $noise_2$ denote the noise intensity before and after the reduction, respectively.

This gives the Sonar Equation:

$$(SL - TL) - (NL - DI) \; = \; DT \qquad \text{(EQ 17)}$$

8.2 Usage of the Sonar Equation in the Simulation Model

To calculate the sound level at the buoy positions, *SL*, *NL*, and *DI* will be input parameters. *TL* will be computed through a relationship between *TL* and the distance *r* between the sonobuoy and the target (this relationship will be derived in the following chapter). The Sonar Equation will finally be solved for *DT*. If *DT* is “high enough”, the information from the buoy will be taken into account. What is high enough is determined by how sure one wants to be that there is a signal. If the sonar operator chooses a low detection level, then all targets will probably be detected, but the likelihood for false alarm is also high in that case. A high detection level, on the other hand, gives a small likelihood for false alarms, but some targets may also go undetected. This relationship is usually illustrated by receiver-operating-characteristic curves (ROC-curves). See Figure 7 .

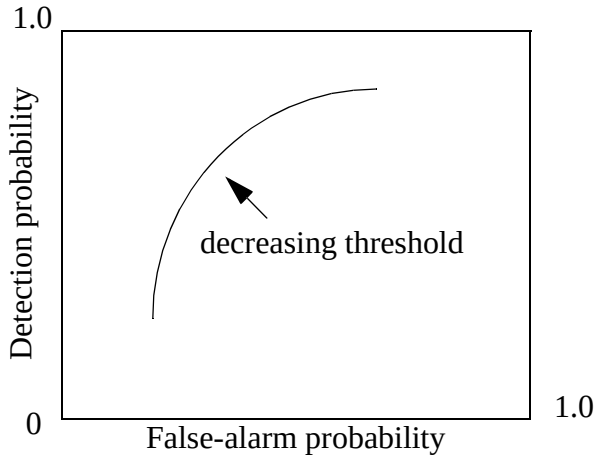

FIGURE 7. An ROC-curve

8.3 Transmission Loss

The transmission loss consists of two parts, loss due to spreading and loss due to absorption. These are discussed below.

8.3.1 Spreading

To calculate the spreading loss, assumptions must be made about how the sound is spread. The two most common models are spherical spreading and cylindrical spreading. In the Swedish archipelago, the conditions are usually somewhere in between, but sometimes even worse than spherical spreading.

8.3.1.1 Spherical Spreading

If the water is deep, allowing the sound to spread in all directions, spherical spreading can be assumed. This means that the sound power - which remains constant - is spread over a spherical surface. See Figure 8 .

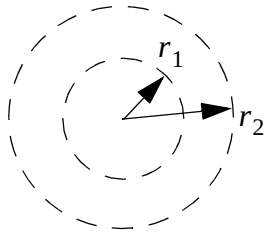

FIGURE 8. Spherical Spreading

The fact that the sound power remains constant gives the following equation:

$$P = 4\pi r_1^2 I_1 = 4\pi r_2^2 I_2 = \ldots \quad \text{(EQ 18)}$$

The transmission loss due to spherical spreading between the distances r_1 and r_2 is therefore:

$$TL = 10\log\frac{I_1}{I_2} = 10\log\frac{r_2^2}{r_1^2} = 20\log\frac{r_2}{r_1} \quad \text{(EQ 19)}$$

TL is the transmission loss in the case of spherical spreading between the distances r_1 and r_2.

8.3.1.2 Cylindrical Spreading

On the other hand, if the water is shallow, the spreading is limited by the surface and the bottom. The area over which the intensity is spread is then the area of a cylinder with increasing radius.

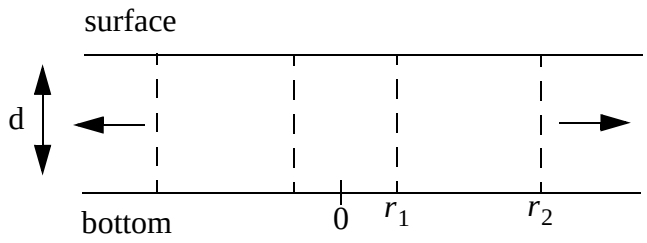

FIGURE 9. Cylindrical Spreading

Figure 9 gives the following expression for the transmission loss:

$$P = 2\pi r_1 \mathrm{d}I_1 = 2\pi r_2 \mathrm{d}I_2 = \ldots \Rightarrow TL = 10\log\frac{I_1}{I_2} = 10\log\frac{r_2}{r_1} \qquad \text{(EQ 20)}$$

The fact that the power P remains constant is used here as well.

8.3.2 Absorption

When the sound travels through the water, some of its energy is transformed to heat. This kind of transmission loss is called absorption. The intensity loss due to absorption can be modelled by the differential equation:

$$dI = -\beta I dr \qquad \text{(EQ 21)}$$

where dI denotes the change when the sound travels the distance dr. Integration between r_1 and r_2 gives:

$$I_2 = I_1 e^{-\beta(r_2 - r_1)} \qquad \text{(EQ 22)}$$

The *logarithmic absorption coefficient* α is defined by:

$$\alpha = \frac{10\log I_1 - 10\log I_2}{r_2 - r_1}$$

which gives the following relationship between β and α:

$$\alpha = 10\beta\log e \quad \text{(EQ 23)}$$

This can be seen through the logarithm of EQ 23.

α is strongly frequency dependent. The relationship is [5]:

$$\alpha(f) = \frac{0{,}1f^2}{1+f^2} + \frac{40f^2}{4100+f^2} + 2{,}75 \cdot 10^{-4} f^2 \quad \text{(EQ 24)}$$

To summarize all of above, the total transmission loss can be stated as:

$$TL = K \cdot \log r + r \cdot \alpha(f) \quad \text{(EQ 25)}$$

where $\alpha(f)$ is the part of the transmission loss due to absorption (this part is very small and is usually omitted). The other part, $K \cdot \log r$, comes from the fact that the sound intensity spreads over a larger and larger area while the sound travels through the water. K is a constant dependent upon how the sound spreads. In shallow water it can be assumed that the sound spreads over a cylindrical surface, giving a K equal to 10. On the other hand, if the water is deep, it can be assumed that the sound spreads over a spherical surface, giving a K equal to 20. In a shallow-water scenario, an empirical value K would lie somewhere between 10 and 20. To find the value of K in practice, experiments must be performed in which a sound with a known level is transmitted and measured at a known distance, using a hydrophone. In the simulation program, K is an input parameter.

9.0 The Path Editor

The Path Editor is an independent part of the submarine tracking simulator, which like the other components of the system, was implemented in the object-oriented language *Eiffel 3*. It provides a graphical interface for creating and editing the path of the submarine. The path has a polygonal shape, and is represented in the program as a list of point vertices and line segments.

9.1 Functionality

The user **adds** nodes to the path by left-clicking on a certain position on the **game area**. A node created in that manner will be added as the last node of the path. One may also **add** a new node before a node that already exists. In doing so, the user middle-click on the node that is to be followed by the new node. A right-click on a particular node **removes** it, and new segments are drawn appropriately. The user may also **change the position** of a node by keeping the left-mouse-button down over the node and drag it to the desired position.

A click on the **Open** button gives the user the opportunity to retrieve a path that was created during a former session. When the button is clicked, a window (the *FILE_BOX*) appears for selecting a file.

The **Save** button stores the path to a file. The data written to the file includes the following:

- the (x,y)-position of each node (upper left corner is the origin)
- the arrival time of the submarine at each node
- the hold time of the submarine at each node
- the departure time of the submarine from each node
- the speed of the submarine at each segment

These values can also be **displayed** on the screen during run-time by left-clicking on the node. To **set the hold time and the speed**, one just enters the values in the corresponding text fields (a keystroke is not necessary). The succeeding nodes and segments will get the chosen values.

The **Quit** button, of course, ends the session.

9.2 Architecture

Eiffel 3 is a modern language that supports all features - such as inheritance (multiple inheritance as well), dynamic binding, information hiding, templates and assertions - that are characterizing an object-oriented language. The architecture of the program includes six clusters, each containing architecturally related classes. These are ***state, windows, root_cluster, widgets, path_classes,*** and ***commands***. Figure 10, “The architecture of the PathEditor,” on page 25 depicts the system design using the BON notation [6].

9.2.1 The *root_cluster* cluster

includes three classes, *CONTROL*, *SHARED_CONTROL*, and *APPLICATION,* the latter being the root class. The first thing that happens when the system is started is that an object of the root class is created, and that its *make* routine is executed. This routine sets the first state, state one, which is the only state in this application. Thereafter, it calls the *init_windowing* routine in the class *WINDOWS*, which, in turn, creates the main window, *PERM_WIND1*, through a “once function”. The once function guarantees that there will only be one instance of the class *PERM_WIND1* during the session. A once function also provides a good way of sharing an instance of a class between multiple objects.

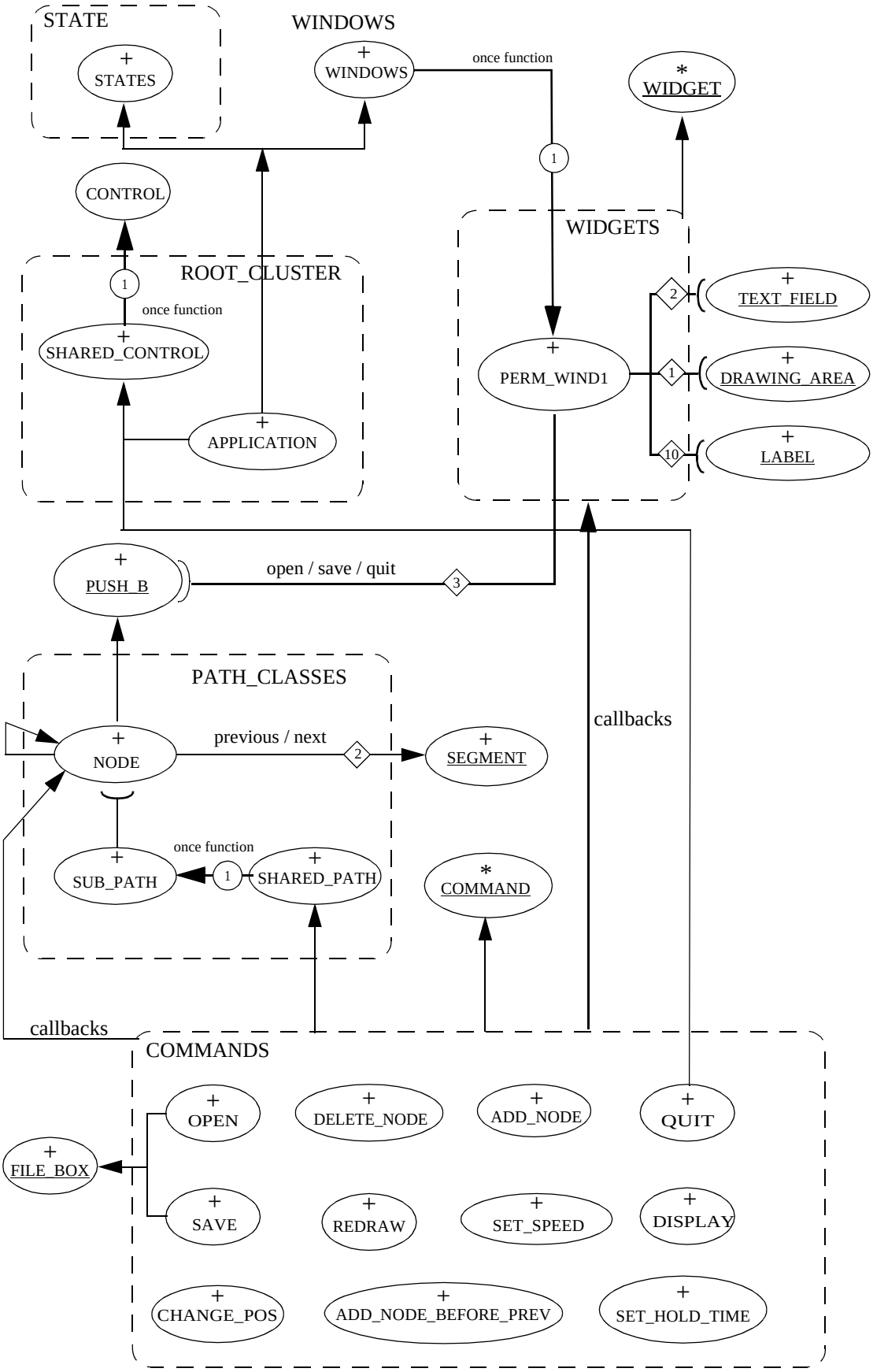

FIGURE 10. The architecture of the PathEditor

9.2.2 The *widget* cluster

includes four classes, *PERM_WIND1*, *TEXT_FIELD*, *DRAWING_AREA* and *LABEL*. All of these are descendants of the class *WIDGET*. A widget is a graphical object where different kinds of events occur. Such events could for example be a click on a mouse button, a key stroke, or a pointer motion. *TEXT_FIELD*, *DRAWING_AREA* and *LABEL* are children of *PERM_WIND1*, which means that *PERM_WIND1* creates instances of these classes, and that *PERM_WIND1* is responsible for redrawing and resizing these objects when an expose event has occurred. They also have equal life times.

9.2.3 The *command* cluster

includes the different command classes. All of these are descendants of the deferred class *COMMAND*, which defines a deferred routine *execute*. The command classes are clients of the widget classes. When a certain event occurs on a widget, a callback is sent to the execute routine of the appropriate command class. For example, when the left mouse button is clicked on the *DRAWING_AREA*, a callback is sent to the execute routine of the *ADD_NODE* class. The object of this class, in turn, calls the *add_node* routine of the unique instance of the *SUB_PATH* class. The command classes that needs access (not all of them) to the sub_path inherit from the class *SHARED_PATH*, which merely has a once function that creates the *SUB_PATH* instance. The *SHARED_PATH* class will, therefore, ensure that all the command classes refer to the same instance of the *SUB_PATH*.

9.2.4 The *path_classes* cluster

consists of three different classes, *SUB_PATH*, *NODE* and *SHARED_PATH*. The *SUB_PATH* class is an *abstract data type* (ADT). It provides an interface for operating on the different nodes of the path, and manages a linked list of *NODE* objects. The clients of the *SUB_PATH* class do not know how the path is implemented because that is hidden from them; only the *SUB_PATH* class has access to the features of the *NODE* class (with a few exceptions). The *NODE* class inherits from the *PUSH_B* class, which is a kind of a widget that is used for push buttons. This implies the possibilities for deleting and changing a node´s position by **clicking/dragging** it. The *COMMAND* classes (not all of them) that deal with those events are clients of the *NODE* class.

The *NODE* class is also a client of the *SEGMENT* class. The *SEGMENT* represents the edges between the nodes.

Finally, the class *CONTROL* together with the class *STATES* manages the states of the system. In this program, there is only one state. But there could be several states in a system. For different states, the same event may have different meanings. For example, a second state could be useful for this system if one wants to prohibit that a second path is retrieved if there already is a path defined for the gaming area. As soon as the user creates a node, the system would transit to the second state that removed the action for the open button. The *CONTROL* class also manages the **Quit** request.

10.0 The Sonobuoy Tracking Simulator version 1

(This documentation refers to version 1.30, unless otherwise stated, compiled 96-08-22)

The purpose of the program, called **SubTrack**, is to simulate the interaction between a target (submarine) and a number of passive sonobuoys in a Swedish archipelagic scenario. A predefined path of the submarine is loaded (visibility can be switched on/off) and then the user is to find the location of the submarine at each time-step by means of deploying sonobuoys into the sea.

10.1 Functionality

The user loads an unknown target path from a file (generated by PathEd, Section 9.0, “The Path Editor,” on page 23). When the user has started the simulation, the approximate initial position of the target is displayed in the shape of a circle. The user may then deploy either single buoys or buoy clusters in various patterns, at times and positions of his choice. For every time step, the program determines whether each buoy has detected (“heard”) the target or not. If at least four buoys hear the target, an area is displayed which represents the smallest area within which the probability of finding the target is 95 %. Unless this area is too large, it is presented as an ellipse whose size, shape, and attitude gives information about the uncertainty associated with the measurement of the target’s position.

To study in more detail the factors which influence the detection of the target, the user can request that the target and/or its path be displayed. The user may also change the game’s starting time, or freeze the time to make the position and noise of the sub constant. Using these features, the operator may try out different buoy patterns.

10.2 Contents

The documentation is structured around the clusters of the program, which are described in Section 10.4 on page 28.

A simplified diagram (Figure 11, “SubTrack clusters and classes,” on page 35) showing the basic design is included, as well as a somewhat more complex one (Figure 12, “Application clusters and classes,” on page 36).

The simplified diagram is discussed in Section 10.3, “Basic Design,” on page 27, while the more complex one is an illustration to Section 10.4, “Architecture,” on page 28.
It should be noted that not all classes are shown in fig 2. This is discussed in Section 10.6, “Comments to the diagrams,” on page 33.

10.3 Basic Design

When the path has been loaded it is read into the class *PATH_DATA* of cluster *target*. The path consists of segments between pairs of nodes (class *NODE*, also in cluster *target*).

At each time step [1] the target's position (kept in class *TARGET_DATA* of cluster ***target***) and the status of deployed buoys (kept in class *BUOY_DATA* of cluster ***buoy_management***) are updated, and then class *SIGNAL_RECEIVER* of cluster ***measurement*** performs an estimation of the target's position from the data of the sonobuoys that are hearing the submarine (provided by class *BUOY_CLUSTER*).

This information is visualized on the screen by an elliptical area containing the target with a predefined probability.

If all measurements performed were exact then the estimate of the target's position would be exact as well (the area would shrink into a point). Uncertainty in positioning the buoys and in measurements of sound arrival times is taken into account in the class *OBSERVATIONS*, depicting the smallest area maximizing the probability of containing the target, that is, an ellipse.

In order to estimate a position at least three buoys have to "hear" the the target. In version 1.30 at least four buoys are required, since a different algorithm has to be used in this case.

10.4 Architecture

The architecture of the program includes seven clusters. These are the ***root_cluster***, which initializes the program, ***sim_management***, which contains the simulation time and the logger, ***target***, which contains all classes that has to do with the target, ***buoy_management***, which controls the deployment of buoys, ***measurement***, which performs calculations on the information from the buoys, ***app_commands***, containing commands that are invoked by callback from windows, and ***app_widgets***, which contains the X11 graphics widgets that are to be used (mostly windows).

10.4.1 The *root_cluster* cluster

Contains only two classes, *APP* and *SHARED_CONTROL*.

In *APP*, the initial state and all possible states are defined. Even if only one state is used in the program, it is useful to use a state transition to accomplish a quit program command. That is the purpose of the statement *control.put (state1, exit_from_application,* "Quit"); Further on, game-parameters are read from a file (*subtrack.ini*), and then the windowing environment is started, and thereby the whole simulation.

The purpose of the class *SHARED_CONTROL* is to create a shared object to manage state transitions. No states other than *Quit* are used, but everything is dependent on the clock, where states are defined by boolean variables (as in *Stopped, Initialized* etc).

1.This updating takes place in the class *START* in cluster ***app_commands***, explained below, Section 10.4.6 on page 30.

10.4.2 The *sim_management* cluster

The classes of this "miscellaneous-cluster" are *CLOCK, TIME_DATA, LOGGER, PARAMETERS, SETUP_DATA, DICTIONARY, PAIR_OF_DOUBLES,* and *PAIR.*
CLOCK implements a simple stop-watch, using system time acquired from the external C-function *My_time*. *TIME_DATA* contains one object of type *CLOCK*, that can be shared by all the other classes through the Eiffel-specific once-function.

LOGGER creates and keeps a log for the target and the buoys. The data is saved in two files : *target.log* and *buoys.log.*

PARAMETERS loads game parameters from the setup-file *subtrack.ini.* The loading of parameters are divided into three categories: *colors, sonar_eq* and *buoy_parameters*. The first and last are read into arrays and lists, and features like *std_dev_1_* are then defined as items in the list/array.

SETUP_DATA contains one object of type *PARAMETERS*, that can be shared by all the other classes through the Eiffel-specific once-function.

DICTIONARY is used for keeping tables of functions (x,f(x)), where f might be, for example, *velocity_to_noise*.

PAIR_DOUBLE contains a concretization of the generic ADT *PAIR.*

PAIR contains a simple Scheme-like implementation of the generic ADT *PAIR.*

10.4.3 The *target* cluster

This cluster contains the target, its path and the nodes that make up the path.

Its classes are *TARGET, SUBPATH, NODE,* and *TARGET_DATA.*

TARGET contains the target, a descendant of *SCREEN_OBJ*. Here all sorts of features of the target is defined, such as the representation on-screen, and routines for updating it's position.

SUBPATH handles reading and storage of a target track (path) consisting of a number of points (nodes) connected by straight lines.

NODE describes the nodes which (pairwise together) define the segments of a subpath.

TARGET_DATA contains one object of type *TARGET*, that can be shared by all the other classes through the Eiffel-specific once-function. It also contains one object of type *SUBPATH*, that was previously placed in a separate class.

10.4.4 The *buoy_management* cluster

This cluster contains various classes concerned with the buoys, their deployement and such.

Its classes are *BUOY, BUOY_INVENTORY, BUOY_DEPLOYER, BUOY_REVOKER, BUOY_SCHEDULER, BUOY_MENU,* and *BUOY_DATA.*

A buoy is a descendant of *SCREEN_OBJ*, and the class *BUOY* contains its specific features.

The class *BUOY_INVENTORY* keeps track of the buoys that have been deployed.

BUOY_DEPLOYER is used (by callback) to deploy a buoy.

BUOY_REVOKER is used (by callback) to remove a buoy.

BUOY_SCHEDULER handles deployment of buoys from the *BUOY_MENU*.

BUOY_MENU defines a menu showing information about a specific buoy.

The class *BUOY_DATA* contains one object of type *BUOY_INVENTORY,* that can be shared by all the other classes through the Eiffel-specific once-function.

10.4.5 The *measurement* cluster

The position of the target is estimated here, from the information in “hearing” buoys. The actual calculations are performed in external C-modules, using a freeware library of matrix functions.

Its classes are *SIGNAL_RECEIVER, OBSERVATIONS, QR_SOLVER,* and *BUOY_CLUSTER.*

SIGNAL_RECEIVER contains the *position_estimator,* which uses *QR_SOLVER* to estimate the position of the target.

OBSERVATIONS contains features for simulating the uncertainty of measurements performed on the target. Uncertainty is modelled by assuming measurement errors to be Gaussian-distributed with zero mean (this is simulated in the class *BUOY* when positioning the sonobuoy).

QR_SOLVER is the Eiffel encapsulation of the *meschach*-routines, available from Netlib, for matrix operations. solve1 returns a least-squares approximation of the solution to the overdetermined system of equations **Ax=b-r**, using QR-factorisation, and solve2 solves the exact equation **Ax=b**.

BUOY_CLUSTER is the same as *get_cluster* in *BUOY_INVENTORY,* but it is performed on the shared object contained in the class *BUOY_DATA*.

10.4.6 The *app_commands* cluster

These are commands that are mostly used by callbacks from the widgets, that is, the windows. All of these are descendants of (exactly) one of the following: *COMMAND, CMD,* or *POPUP_CMD*.

The classes of this cluster are *START, STOP, CONT, LOAD, SAVEG, RUN, QUIT, EXIT, REDRAW, GET_CLICK, TIME_ENTRY, SHOW_STATUS, SHOW_PATH,* and *SHOW_TARGET.*

START is the heart of the simulation, where a kind of 'main loop' is kept by constantly setting new *TIMER*-based callbacks to the same routine (execute). A lot of checking has to be done here, perhaps this can be made more readable later on, for example by some sort of status-flag.

Signals to execute :

- 0 = (re-)start (sent by callback from *PERM_WIND1* or from *BUOY_SCHEDULER*)
- 1 = continue until restarted (sent recursively by *START*)

This is needed because of the recursive nature of the routine.

All the updating is performed from here, every time-step. The time-steps are defined by the delay time of each *TIMER*-based callback to this routine (set recursively by the same routine).

In contrast to class *START,* the class *STOP* is really simple; it just stops the simulation time.

Similar to class *STOP,* the class *CONT* resumes the simulation time.

LOAD describes the loading of a path (created by **PathEd**) from disk.

SAVEG defines the saving of a simulation to disk (not a path - that is performed by the **PathEd**). To perform this it takes a snapshot of the simulation and saves the buoys in file buoys.data, which is a non-editable file, and the rest of the data in an ascii-file *subtrack001.sav.* This should perhaps be changed so that the user can specify a save-file (and thus can have multiple savegames stored).

RUN - the naming might be confusing - is the dual of class *SAVEG*. It should retrieve a previously saved game from disk. This has not yet been implemented.

QUIT - as the name indicates, contains the code needed to quit the program.

EXIT - to quit the program, giving the user an opportunity to save the simulation.
This has not been implemented, and is probably unnecessary.

REDRAW - to redraw the target track at expose events (hidden or moved window).

GET_CLICK sets the active text_field to be the one that is clicked. Needed for the *BUOY_SCHEDULER* to register the buoy coordinates in a menu.

The *TIME_ENTRY* command acquires information from *TEMP_WIND2,* and sets the

simulation time accordingly.

SHOW_STATUS - when the time menu is shown this command sets the status label to either *Stopped* or *Running*. Now in color!

SHOW_PATH - command to switch path display on/off.

SHOW_TARGET - command to switch target display on/off.

10.4.7 The *app_widgets* cluster

This cluster contains widgets, mainly windows, that have been designed for this program. The classes of this cluster are *PERM_WIND1*, *TEMP_WIND1*, *TEMP_WIND2*, *TEMP_WIND3*, *TEMP_WIND4*, and *TEMP_WIND5***.**

PERM_WIND1 is the main window, containing the gaming area, tracking display and menus.

TEMP_WIND1 is used to Show/Hide the Target/Track -window.

TEMP_WIND2 defines the window for time-data display.

TEMP_WIND3 defines the window for buoy data display.

TEMP_WIND4 defines the window for message display.

TEMP_WIND5 defines the window for buoy deployment.

10.5 External Code

To perform some of the mathematical operations (read: matrix algebra) it was necessary to incorporate some external code. This code is written in C, and utilizes a freeware linear algebra library called *Meschach*, available from Netlib on Internet.

The classes *QR_SOLVER* and *OBSERVATIONS* both interface to the C-module *Eq_solver.c*, where these operations are performed.

In addition, there is no system time to be acquired from the Eiffel libraries, therefore we used a very simple C-function to do this. The class *CLOCK* benefits from this C-function, named *My_time.c*.

10.5.1 The modules

Comments to the modules are copied from the source code and edited.

10.5.1.1 Module My_time.c

This one is very simple. It #includes *time.h*, where the function *ctime*, which gets the system time, can be found.

10.5.1.2 Module Eq_solver.c

This module is meant to include external functions called from Eiffel.

- *qr_solver* solves the Ax=b eq, using QR-factorisation and least-squares.
- *create_c_data* allocates space for the static variables, and
- *destroy_c_data* deallocates the same space.
- *A_set_mn* sets the contents of the matrix A,
- *b_set_m* sets contents of b, while
- *x_get_m* gets the results of the calculation performed.

If these functions are declared as external "C" routines in Eiffel they should work okay, although somewhat crude. The crudeness comes from the fact that one cannot pass matrices directly as arguments from Eiffel to C and vice versa.

There ought to be a check for memory leaks on these procedures. Or will they be taken care of by the Eiffel/C compiler?

One might argue that there should be two different modules to take care of (1) the solution of the equation system, and (2) the computation of the covariance matrix, but it was decided that it would be easier, under the circumstances, to keep them in the same bowl, so that there wouldn't be so much sending parameters back and forth.

10.6 Comments to the diagrams

The diagrams were made in BON-notation, using the *EiffelCase* development environment.

Not every class (or even every cluster) is shown even in Figure 12, "Application clusters and classes," on page 36, and this is because those classes are mainly concerned with either the window-system (all windows are "globalized" by once-functions and put in the class *WINDOWS* of cluster *app_windows* - which is not visible on the diagrams), or with the startup-parameters set in a class *SETUP_DATA* in the cluster *sim_management*.

These "globalized" objects are added to those needed for the simulation (eg. *PATH_DATA*, *TARGET_DATA*, *BUOY_DATA*, and *TIME_DATA*). The classes containing these objects are then inherited into every class that needs data from them, and while this makes things easy for the programmer, it tends to clutter images showing relations between classes and between clusters. Therefore it was decided not to incorporate all of these "globalized" objects into the diagrams, since the way the window management works and setup-parameters are distributed really isn't the issue here.

(The other “globalized” objects are there though, each of them named with *_DATA*-suffixes.)

10.7 Known Bugs

If, when choosing the path, a non-**PathEd**-file is chosen, a precondition exception will occur.

After a path is fully traversed by the target, it should not be asked to continue (with the ***cont*** command). That will cause a precondition exception.

If, for some reason, the input data to *qr_solver.c* is degenerate, singular matrices may be generated, and the system will crash. There should be some check in the C-module to avoid this.

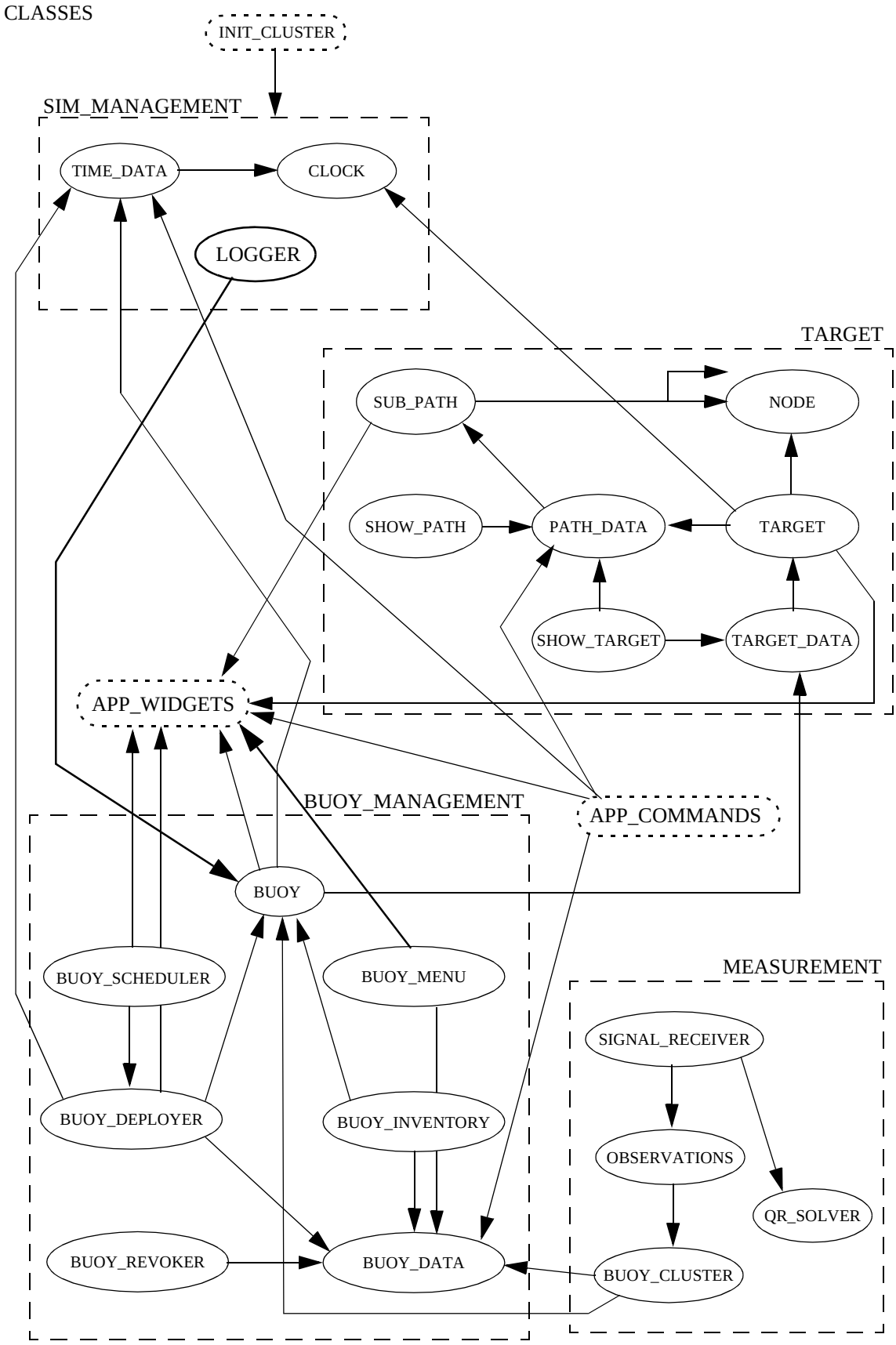

FIGURE 11. SubTrack clusters and classes

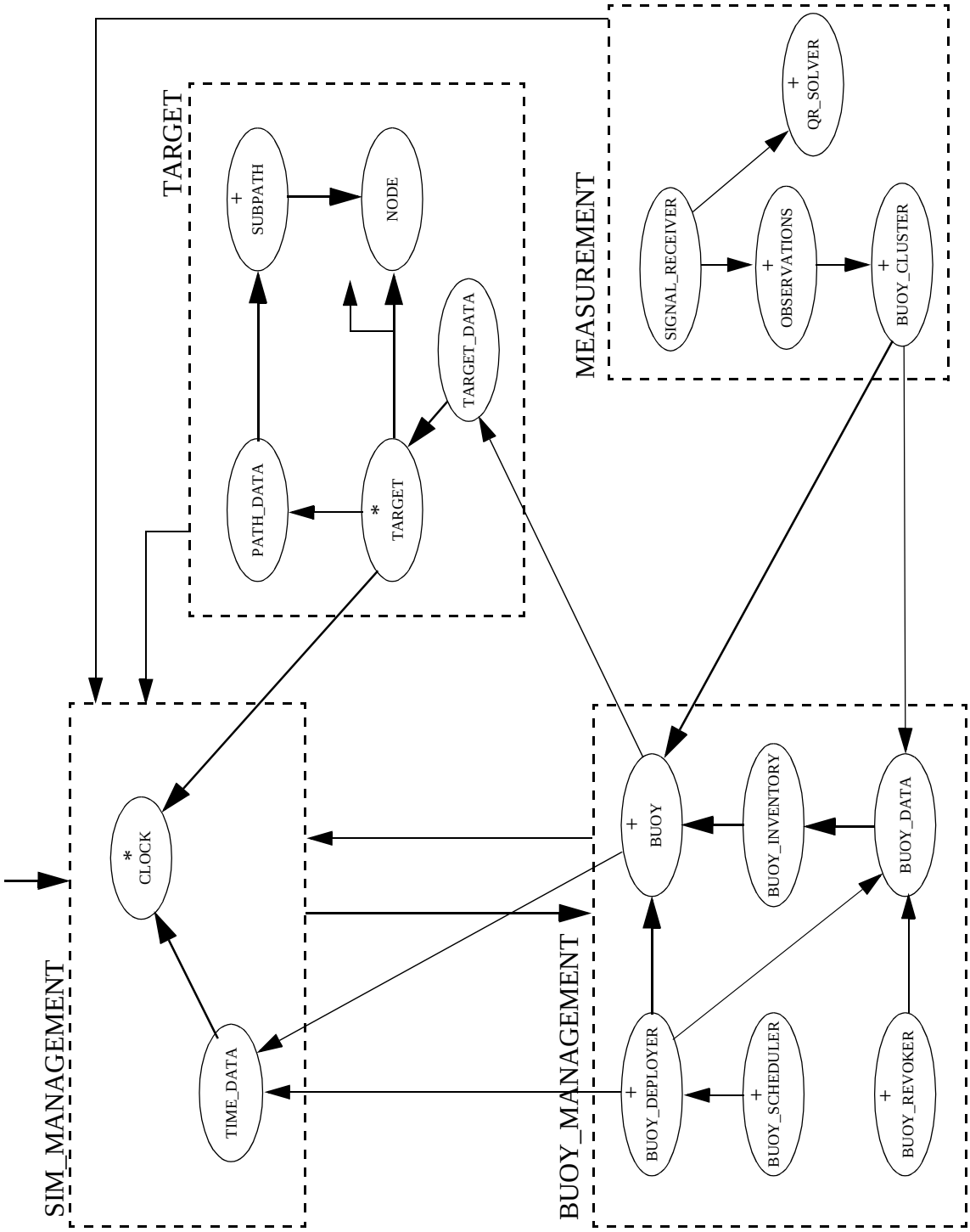

FIGURE 12. Application clusters and classes

11.0 Future work

A second version of the submarine tracking simulator is now in an advanced development stage. It includes a buoy deployment optimization algorithm. Methods for predicting the future position of the target, based on Kalman filtering, have been implemented and are used to support the buoy deployment algorithm. The computational methods developed for the second version will be described in a forthcoming report.

A detailed model would simulate the sound propagation in a three-dimensional archipelago environment including sea-bottom topography, islands, reverberation effects, multiple targets, etc. Such a model is, however, seemingly not within the current state-of-the-art, and if it were to be developed, would certainly be an extremely complex piece of software.

Other, more easily achievable extensions are:

The path of the sub could be changed during run time, or the program could be connected to a submarine vehicle simulator. False alarms as well as no-detection possibilities would be modelled by generating randomly false alarms and no-detection occasions. Other kinds of detection equipment, such as active sonobuoys and magnetic sensors, could conceivably also be added to the model.

12.0 References

1. NAVY INTERNATIONAL, September 1990, 307-312.
2. C. BLIXT, FOA RAPPORT C 20814-2.2 (1991).
3. R.J. URICK, Principles of Underwater Sound. McGraw-Hill Book Company (1983).
4. W.S. BURDIC, Underwater Acoustic System Analysis. Prentice Hall (1991).
5. R. NIELSEN, Sonar Signal Processing. Artech House (1991).
6. K. WALDEN and J. NERSON, Seamless Object-Oriented Software Architecture. Prentice Hall (1995).

Appendix A. On the multivariate normal distribution and error ellipses.

A.1 The normal distribution

A one-dimensional random variable with normal distribution with mean μ and variance = σ^2, is usually denoted: $N(\mu, \sigma^2)$

A.2 Multivariate normal distributions

If x_1,... x_n are n independent N(0,1) random variables, these can be combined into a n-dimensional stochastic vector X.

An n-dimensional multivariate normal distribution can be defined as an affine linear non-singular transformation of such a vector X consisting of n independent, equally distributed normal random variables, all with a N(0,1)-distribution, i.e:

$$Y=AX+\mu$$

Here:

$$X=\begin{bmatrix} x_1 \\ x_2 \\ \dots \\ x_n \end{bmatrix};\ \mu = \begin{bmatrix} \mu_1 \\ \mu_2 \\ \dots \\ \mu_n \end{bmatrix};\ Y=\begin{bmatrix} y_1 \\ y_2 \\ \dots \\ y_n \end{bmatrix}$$

and A is a constant non-singular (n x n)-matrix.

The so-called *joint density functions* for X and Y are, respectively:

$$f_x=\frac{1}{(2\pi)^{n/2}}\ e^{-\frac{1}{2}(X^TX)}$$

and:

$$f_Y=\frac{\sqrt{\det(C^{-1})}}{(2\pi)^{n/2}}\ e^{-\frac{1}{2}(Y-\mu)^TC^{-1}(Y-\mu)}$$

where T stands for transpose.

(See for example: Howard G. Tucker, An introduction to Probability and Mathematical Statistics, Academic Press 1963, or any elementary textbook on the subject).

A.3 The covariance matrix C

In the density function for Y the matrix C is the so called *covariance matrix* of Y:
$C = E[(Y-E[Y])*(Y-E[Y])^T]$, i.e. it contains at position (i,j) the number
$E[(y_i - \mu_i)*(y_j - \mu_j)]$ (here E[.] means expectation).

For C we have the relation $C= A*A^T$, which is evident from $X= A^{-1} *(Y-\mu)$;
$X^T*X= (Y - \mu)^T *(A^{-1})^T *A^{-1}*(Y - \mu)= (Y - \mu)^T *(A*A^T)^{-1} *(Y - \mu)$.

A.4 The mapping X-->Y

Let S_R denote the *sphere* with radius r around the origin in $(x_1, x_2, \ldots x_n)$-space.
This sphere is mapped by $Y=AX+\mu$ onto an *ellipsoid* E_R.

The equation for the volume contained in the sphere in X-space can be written:

$$X^T*X \leq R^2$$

Which ellipsoid in Y-space corresponds to this sphere? From $Y= A*X+\mu$ one gets
$X = A^{-1}(Y-\mu)$.

This is substituted into the equation of the sphere:

$$X^T*X = [A^{-1}(Y-\mu)]^T*[A^{-1}(Y-\mu)] \leq R^2$$

which can be simplified to:

$$(Y-\mu)^T*(AA^T)^{-1}(Y-\mu) \leq R^2 \quad (\text{since } (A^{-1})^TA^{-1} = (A^T)^{-1}A^{-1} = (AA^T)^{-1})$$

or, expressed in more detail:

$E_R= \{Y \mid (Y-\mu)^T C^{-1} (Y-\mu) \leq R^2\}$, where $C= AA^T$ is Y's covariance matrix.

The probability that the stochastic vector X belongs to the set S_R equals the probability for Y being situated within E_R. For the case when X is two-dimensional, the probability p that X belongs to S_R in the xy-plane is : $p = 1- \exp(-R^2/2)$.

This follows from:

$$p = \mathrm{P}[X \in S_R] = \frac{1}{2\pi}\int_{S_R} e^{-1/2(x_1^2+x_2^2)}dx_1dx_2 = \frac{1}{2\pi}\int_{r=0}^{R}\int_{\varphi=0}^{2\pi} e^{-\frac{r^2}{2}}r\,dr\,d\varphi$$

$$p = \int_0^R re^{-\frac{r^2}{2}}dr = -e^{-\frac{r^2}{2}}\Big|_0^R = 1 - e^{-\frac{R^2}{2}}$$

This means that a two-dimensional normally distributed random variable which has a density function like that of f_Y above satisfies that its values lie within the ellipse E_R with probability precisely 1- exp(-R^2/2).

A.5 Example of corresponding values of R and probability levels

Some values of the relation p = 1-exp(-R^2/2):

Table 2:

p	R
0.80	1.79
0.90	2.15
0.99	3.03

A.6 Error propagation

Now suppose that the variables x_0 and y_0 are given by continuously differentiable functions $f_1$and f_2 of the independent variables $x_1,\ldots x_n$, $y_1,\ldots y_n$, $t_1,\ldots t_n$,

$x_0 = f_1(x_1, \ldots x_n, y_1,\ldots y_n, t_1,\ldots, t_n)$
$y_0 = f_2(x_1, \ldots x_n, y_1,\ldots y_n, t_1,\ldots, t_n)$

Assume further that these variables are impaired by errors that can be regarded as independent random variables with zero means.

Assume also that they all have a normal distribution and that the variances of the errors in $x_1, \ldots y_n$ are = σ_1^2 , and for $t_1,\ldots t_n = \sigma_2^2$
By a Taylor expansion of:

$x_0 + \delta x_0 = f_1(x_1+\delta x_1, \ldots, x_n+\delta x_n, y_1+\delta y_1,\ldots, y_n+\delta y_n, t_1+\delta t_1,\ldots, t_n+\delta t_n)$

we get expressions of the form:

$$x_0 + \delta x_0 = f_1(x_1 \ldots x_n, y_1 \ldots y_n, t_1 \ldots t_n) +$$
$$\left(\delta x_1 * \frac{\partial}{\partial x_1} f_1 + \delta x_2 * \frac{\partial}{\partial x_2} f_1 + \ldots + \delta t_n * \frac{\partial}{\partial t_n} f_1\right) +$$
$$+ \text{ residual terms of higher order}$$

and correspondingly for y_0.

From this linearization one realizes that the error in the computed position (x_0,y_0) is (approximately) a linear combination of the normally distributed errors in x_i, y_i and t_i. The errors in x_0 and y_0 therefore become (approximately) normally distributed, but not independent. The relation can be written:

$$\begin{bmatrix} \delta x_0 \\ \delta y_0 \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{13n} \\ a_{22} & a_{22} & \cdots & a_{23n} \end{bmatrix} \begin{bmatrix} \delta x_1 \\ \delta x_2 \\ \cdots \\ \delta t_n \end{bmatrix}$$

where the matrix $A = (a_{ij})$ is made up of the partial derivatives from the Taylor expansion. (See Appendix C for derivative values).

Introduce the variables $\xi_1,\ldots\xi_{3n}$ through the normalization:
$\xi_1=\delta x_1/\sigma_1,\ldots\ \xi_n=\delta x_n/\sigma_1;\ \xi_{n+1}=\delta y_1/\sigma_1,\ldots\ \xi_{2n}=\delta y_n/\sigma_1;\ \xi_{2n+1}=\delta t_1/\sigma_2,\ldots\ \xi_{3n}=\delta t_n/\sigma_2;$

These variables then all have a N(0,1) distribution.

Set further: $\eta_1= \delta x_0$, $\eta_2= \delta y_0$, $Y^T =(\eta_1,\eta_2)$, $X^T= (\xi_1,\ldots\xi_{3n})$,
and let the matrix G describe the transformation from X to Y.

G will then implicitly contain the standard deviations $\sigma_1$and σ_2 as shown by:
G = A*B,
where B is a diagonal matrix, whose 3n elements consist of 2n σ_1, followed by
n σ_2, $(\sigma_1,\ldots\sigma_1, \sigma_2,\ldots\sigma_2)$.

The matrix $C = B*B^T$ is the covariance matrix for $(\delta x_1,\ldots, \delta y_1,\ldots, \delta t_1,\ldots,\delta t_n)$.
It is a diagonal matrix with variances σ_i^2 in the diagonal.

The transformation Y= G*X will map a sphere S_R in the 3n-dimensional X-space onto an ellipse E_r in the two-dimensional Y-space, with $r \neq R$ in general.

This is recognized by the fact that matrix G, which we assume has rank 2, can be augmented to a non-singular mapping. The ellipse E_r is then seen to be a projection to two dimensions of that ellipsoid which is the image of the sphere S_R (that the projection of an ellipsoid is

an ellipse is regarded a well known fact).

Define the probability p through:

$$p= P[Y \in E_R]= P[X \in S_R]= P[\xi_1^2 + \ldots + \xi_{3n}^2 <= R^2]$$

Under the given assumptions this probability is given by the distribution function of a variable with a *Chi-square* distribution with 3n degrees of freedom.

If for simplicity we assume that all variances are equal, $Var(x_i)= Var(y_i)=Var(t_i)=\sigma^2$, and if $F_m(x)$ is the distribution function of a χ^2– distributed variable with 3n degrees of freedom, the value of p can be expressed in terms of $F_m(.)$:

$$p = P[(\delta x_1 / \sigma)^2 + \ldots + (\delta t_n / \sigma)^2 \leq R^2]= F_{3n}(R^2)$$

Some examples of p-levels and corresponding limits (not used in the sequel though):

Table 3: Probabilities of chi-square variable

n	m	p	R^2	R
4	12	0.90	18.5	4.30
4	12	0.99	26.2	5.12
10	30	0.90	40.3	6.35
10	30	0.99	50.9	7.13

Here n equals the number of sonar buoys, and 3n is the number of space- and time coordinates which are input to the position computation.

However, it is difficult to calculate in this way the dimensions of the error ellipse corresponding to a given level of probability. Below another method is given to accomplish this.

A.7 The error ellipse equation

As already mentioned the sphere S_R is mapped onto the ellipse E_r, where R and r are in general *different*. To determine the image E_r by purely geometrical means is difficult, and a simpler way is to use directly that $Y^T= (\delta x_0, \delta y_0)$, which as being a linear combination of independent normal stochastic variables itself becomes normally distributed.

Since the X-variables all have zero mean, the same holds for Y, i.e., $E[\eta_1] = E[\eta_2] = 0$. The covariance matrix K for $(\delta x_0, \delta y_0) = (\eta_1,\eta_2) = Y^T$ becomes:

$$K = Cov[\eta_1,\eta_2] = E[Y*Y^T] = E[(ABX)*(ABX)^T] =$$

$= E[ABXX^TB^TA^T]=AB*E[XX^T]*B^TA^T$

$= A(BB^T)A^T = ACA^T$

because $E[X*X^T]$ is the unit matrix, due to the fact that the covariances are = 0 and the variances are = 1 for the variables in X.

Thus the covariance matrix K for Y is known:

$$K = \begin{bmatrix} \sigma_{11} & \sigma_{12} \\ \sigma_{21} & \sigma_{22} \end{bmatrix}$$

where $\sigma_{11}^2 =$ $Var[\delta x_0]$;
$\sigma_{22}^2 =$ $Var[\delta y_0]$;
$\sigma_{12} = \sigma_{21} =$ $E[\delta x_0 * \delta y_0]$;

Finally the error ellipse is given by the expression:

$$\begin{bmatrix} \eta_1 & \eta_2 \end{bmatrix} K^{-1} \begin{bmatrix} \eta_1 \\ \eta_2 \end{bmatrix} \leq R^2$$

Note: To determine the dimensions of the error ellipse it is thus not necessary to worry about the relations between input data and the size of R in the equation above. R is instead computed directly from the probability level p.

Appendix B. Size and orientation of error ellipse.

B.1 Introduction

Suppose that an ellipse is given by a positive definite quadratic form defined by the inverse of a symmetric matrix K:

$$\begin{bmatrix} x, & y \end{bmatrix} \mathbf{K}^{-1} \begin{bmatrix} x \\ y \end{bmatrix} = R^2$$

This situation occurs in connection with stochastic error ellipses, where x and y are independent normal random variables.

$$K = \begin{bmatrix} \sigma_{11} & \sigma_{12} \\ \sigma_{21} & \sigma_{22} \end{bmatrix}$$

Here $\sigma_{11} > 0$, $\sigma_{22} > 0$, $\sigma_{12} = \sigma_{21}$ and:

$$\sigma_{11} \times \sigma_{22} - \sigma_{12} \times \sigma_{21} > 0$$

The equation to determine the eigenvalues of K are:

$$p(\lambda) = det(K - \lambda E) = 0$$

$$p(\lambda) = \begin{bmatrix} \sigma_{11} - \lambda & \sigma_{12} \\ \sigma_{21} & \sigma_{22} - \lambda \end{bmatrix} = (\sigma_{11} - \lambda)(\sigma_{22} - \lambda) - \sigma^2{}_{12}$$

The roots are λ_1 och λ_2, where λ_1 is the larger eigenvalue,
and as $p(0)=\sigma_{11}*\sigma_{22}-\sigma_{12}*\sigma_{21} > 0$ and $\lambda_1+\lambda_2 = (\sigma_{11} + \sigma_{22})/2 > 0$, we have $\lambda_1 > \lambda_2 > 0$.

B.2 Connection between the matrix K and the ellipse

Let x designate a two-dimensional vector.

Let Q be the symmetric operator defined by matrix K^{-1}.
The quadratic form can then be written as an inner product: (x,Qx),
and the ellipse is given by the expression $(x, Qx) = R^2$, where R defines the size.
The eigenvalues of Q are the inverted eigenvalues of K, and as Q(x)= λ*x =>

$Q^{-1}(x) = \lambda^{-1} * x$, we see that Q has the same eigenvectors as K.
The eigenvector (ξ_1, η_1) which belongs to eigenvalue λ_1 is given by equation:

$$(\sigma_1 - \lambda_1) * \xi_1 + \sigma_2 * \eta_1 = 0$$

which can be written as the inner product:

$$(\sigma_1 - \lambda_1,\ \sigma_2) * (\xi_1, \eta_1) = 0$$

From this follows that the eigenvector for λ_1 is parallel to: $(\sigma_2, \lambda_1 - \sigma_1)$.

B.2.1 Determination of length and direction of the major semiaxis

B.2.1.1 Length of the major semiaxis

Assume x to be a unit vector (length= 1).
The point on the ellipse corresponding to that direction will equal t*x for some positive number t. In the direction of the major axis t attains its maximum value (=a).
Thus we have: $(tx, Qtx) = R^2$.
Now it is a well known fact (see e.g. Shilov, Linear Algebra) that (x, Qx) attains its smallest and largest values for the eigenvectors. Since as already shown, if x is an eigenvector for the eigenvalue λ of K, then x is also an eigenvector with the eigenvalue $1/\lambda$ for Q and we get:
.

$$(x, Qx) = (x, \lambda^{-1}x) = \lambda^{-1}$$

From this follows: $(tx, Qtx) = t^2 * (x,Qx) = t^2/\lambda = R^2$, which shows that the greatest eigenvalue ($\lambda_{1)}$ of K gives maximum t-value and thus the length of the major semi-axis is given by the expression:

$$a = R\sqrt{\lambda_1}$$

B.2.1.2 The direction of the major semiaxis

According to what has been said above, the eigenvector belonging to the eigenvalue λ_1^{-1} of Q points in the direction of the major axis. This vector is also the eigenvector belonging to the eigenvalue λ_1 of the operator with matrix K.
The direction of that vector has been determined to be: $(\sigma_2, \lambda_1 - \sigma_1)$.
We get the following formulae for the determination of major axis direction α,
which can be assumed to lie in the interval $-90° < a <= 90°$:

B.2.1.2.1 Case $\sigma_{12} \neq 0 (-90° < a < 90°)$

$$\tan\alpha = \frac{\lambda_1 - \sigma_{11}}{\sigma_{22}}$$

B.2.1.2.2 Case: $\sigma_{12} = 0$:

- If $\sigma_{11} > \sigma_{22}$:$\alpha = 0°$
- If $\sigma_{11} < \sigma_{22}$:$\alpha = 90°$