# Noen konsepter for datafusjon for undervannsakustisk sensornettverk

Reinert Korsnes

Forsvarets forskningsinstitutt (FFI)

18. november 2009

FFI-rapport 2009/01786

103901

P: ISBN 978-82-464-1668-7
E: ISBN 978-82-464-1669-4

## Emneord

Sensornettverk

Datafusjon

Akustikk

## Godkjent av

| | |
|---|---|
| Connie E. Solberg | Prosjektleder |
| Jan Erik Torp | Avdelingssjef |

## Sammendrag

Denne rapporten beskriver prinsipper for tracking av fartøyer som passerer et undervannsakustisk sensornettverk hvor nodene måler retninger til lyd. Dataene fra nodene kan inkludere målefeil og tvetydigheter. Estimater for posisjoner til lydkilder er derfor ikke entydige posisjoner men angir heller et sett av mulige lokasjoner for lydkildene.

## English summary

The present report describes principles for tracking targets (vessels) passing by an underwater acoustic sensor network where the nodes measure sound directions. The data from the nodes include errors and ambiguities. Estimates for target positions may reflect uncertainty and give wide sets of possible locations for sound sources.

# Innhold

# 1 Innledning

Første del av denne rapporten diskuterer estimering av situasjonsbilder ut fra samtidige retnings-data fra lyttebøyer som registrerer lyd i havet (i første omgang bare en samtidig lydkilde). Antall samtidige målinger avhenger av deteksjons-rekkevidde. Poenget er å estimere posisjoner via krysspeilinger og å angi usikkerheter på disse estimatene. Rapporten viser også enkle muligheter for å estimere posisjoner og lydspektra for lydkilder basert på målte lyd(-spektra) ved nodene.

Rapporten viser prinsipielt muligheter med sensornettverk og som enkelt-sensorer ikke ville ha angående sporing av lydkilder. Begrensninger på båndbredde gjør at sporing (målfølging/tracking) basert på akustisk kommunikasjon helst (ihvertfall i komplekse situasjoner) utføres samkjørt med klassifisering for å redusere tvetydigheter og oppnå situasjonsbilder som er konsistente over tid. Dette er forskjellig fra målfølging via radar der mål kan følges tett i tid og hvor store båndbredder er tilgjengelige.

# 2 Statistikk på retningsdata

Fordelinger av retningsdata er noe forskjellig fra fordelinger av tall på tallinja. Vi ønsker ofte at fordelinger og estimater på tallinja er skift-invariante og ellers tilfredsstiller similaritetskrav på den måten at resultatene ikke avhengige av tilfeldig brukte fysiske enheter og referanser. Tilsvarende er det ønskelig at estimater basert på retnings-data (eller periodiske data) er rotasjons-invariante. I praksis betyr dette at valg av referanse-retninger ikke må påvirke estimatene basert på retningsdataene. Vi ønsker også at "lokalt" kan retnings- og lineære data og modeller for dette håndteres likt (asymptotisk). Følgende metrikk sikrer dette:

$$d(\theta_1, \theta_2) = ||\exp(i\theta_1) - \exp(i\theta_2)|| \tag{2.1}$$

der $||\cdot||$ betegner lengde (eller norm) til argumentet som er en vektor i planet. En slike definisjon gir også enkel og sikker programmering.

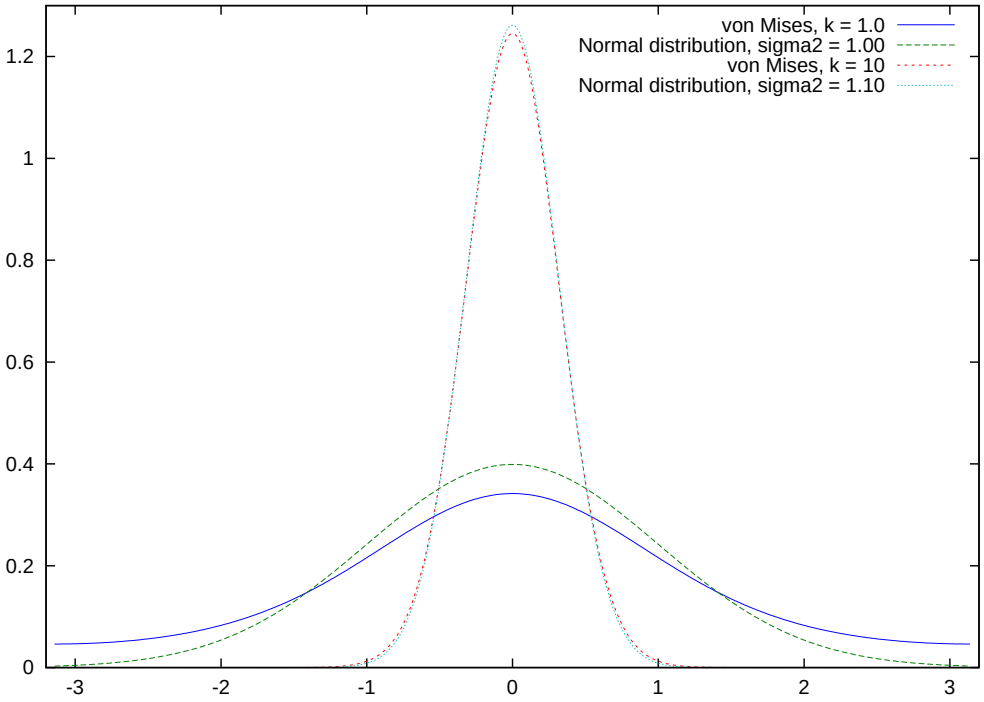
Von Mises (tetthets-)fordelingen bygger på prinsippene nevnt ovenfor. Den kan sees på som en 'vinkel-versjon' av normal-fordelingen og har form:

$$f(\theta|\mu, \kappa) = \frac{e^{\kappa \cos(\theta-\mu)}}{2\pi I_0(\kappa)} \tag{2.2}$$

hvor $I_0$ er den modifiserte Bessel funksjonen av orden 0 (kan her sees på som en normaliseringsfaktor). Legg merke til at ved å sette inn $\cos(\theta - \mu) \simeq 1 - \frac{1}{2}(\theta - \mu)^2$ i ligning 2.2 så får en normalfordelingen (for verdier av $\theta$ i nærheten av $\mu$):

$$f(\theta|\mu, \kappa) \simeq \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(\theta-\mu)^2}{2\sigma^2}} \tag{2.3}$$

hvor $\kappa = 1/\sigma^2$ (eller kanskje bedre $\sqrt{k}$) er et mål på hvor mye fordelingen "konsentreres" om middelverdien $\mu$. Figur 2.1 viser eksempler på von Mises fordelinger sammen med tilhørende normalfordelinger.




*Figur 2.1: Eksempel på von Mises fordelinger sammenlignet med normal-fordelinger.*

# 3 Likelihood estimat av posisjoner til lydkilde gitt målinger av retninger

Maximum likelihood estimering (MLE) er velkjent statistisk metode for å tilpasse matematiske modeller til data. Gitt en lydkilde i havet med posisjonene $z_1, z_2, \ldots, z_n$ ved de respektive $n$ tidspunktene $t_1, t_2, \ldots, t_n$. Anta at $m$ romlig fordelte sensorer (noder) registrerer retningen til denne lydkilden. La $f_{z_1,z_2,\ldots,z_n}(\theta_1, \theta_2, \ldots, \theta_m)$ beskrive (tetthets-) fordelingen av retningene som hver av de $m$ nodene registrerer. Anta så at denne (tetthets-)fordelingen $f$ har form:

$$
\begin{aligned}
f_{z_1,z_2,\ldots,z_n}(\theta_1, \theta_2, \ldots, \theta_m) = \\
& f_{z_1,\kappa,\mu_{1,1}}(\theta_1) f_{z_2,\kappa,\mu_{1,2}}(\theta_1) \ldots f_{z_n,\kappa,\mu_{1,n}}(\theta_1) \\
& f_{z_1,\kappa,\mu_{2,1}}(\theta_2) f_{z_2,\kappa,\mu_{2,2}}(\theta_2) \ldots f_{z_n,\kappa,\mu_{2,n}}(\theta_2) \\
& \ldots \\
& f_{z_1,\kappa,\mu_{m,1}}(\theta_m) f_{z_2,\kappa,\mu_{m,2}}(\theta_m) \ldots f_{z_n,\kappa,\mu_{m,n}}(\theta_m)
\end{aligned} \tag{3.1}
$$

hvor (jmf Ligning 2.2)

$$f_{z_t,\kappa,\mu_{i,t}}(\theta_i) = \frac{e^{\kappa\cos(\theta_i-\mu_{i,t})}}{2\pi I_0(\kappa)} \tag{3.2}$$

der $\mu_{i,t}$ er vinkelen (retningen) for Node i til posisjonen $z_t$ ($= \arg(z_t - z_{\text{Node i}})$, der $z_{\text{Node i}}$ er posisjonen til Node i). Ligning 3.1 forutsetter uavhengighet hvilket kan være tvilsomt for observasjoner tett i tid. Naturlig støy er typisk korrelatert i tid ("rød støy").

For observasjon av retninger til en lydkilde ved bare ett tidspunkt ($n = 1$) blir Ligning 3.1 slik:

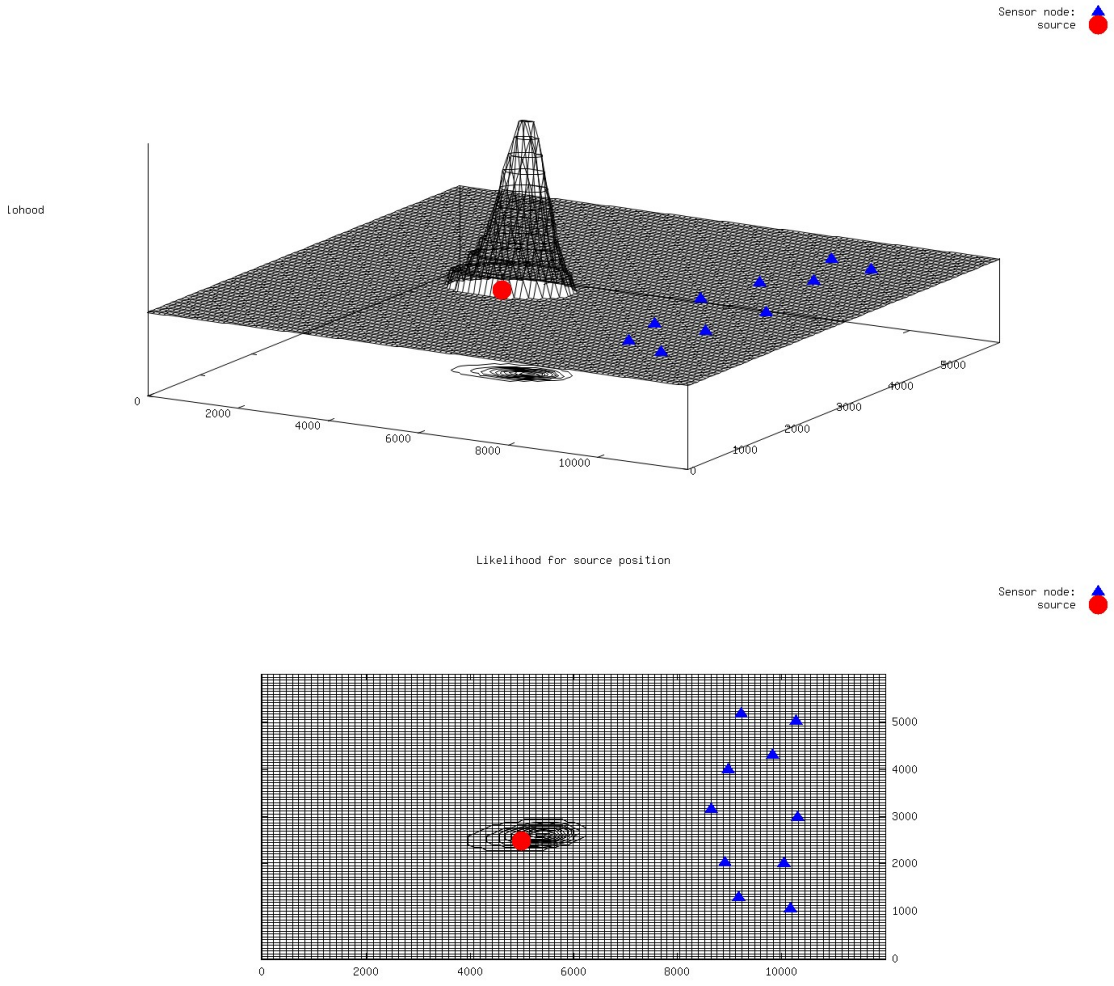
$$f_z(\theta_1, \theta_2, \ldots, \theta_m) = f_z(\theta_1|\kappa, \mu_1) f_z(\theta_2|\kappa, \mu_2) \cdots f_z(\theta_m|\kappa, \mu_m) \tag{3.3}$$

hvor $\mu_i = \arg(z - z_{\text{Node i}})$. La $(\theta_1, \theta_2, \ldots, \theta_m)$ ved dette høvet betegne observerte retninger ved respektive Node nummer $1, 2, ..., m$. $f_z(\theta_1, \theta_2, \ldots, \theta_m)$ betegner da 'likelihood' for den mulige posisjonen $z$ til den aktuelle lydkilden. Figur 3.1 viser et eksempel på likelihood verdier ved forskjellige mulige posisjoner til lydkilder gitt observasjoner fra 10 noder (for gitt $\kappa = 100$). Figur 3.2 illustrerer de aktuelle målingene.

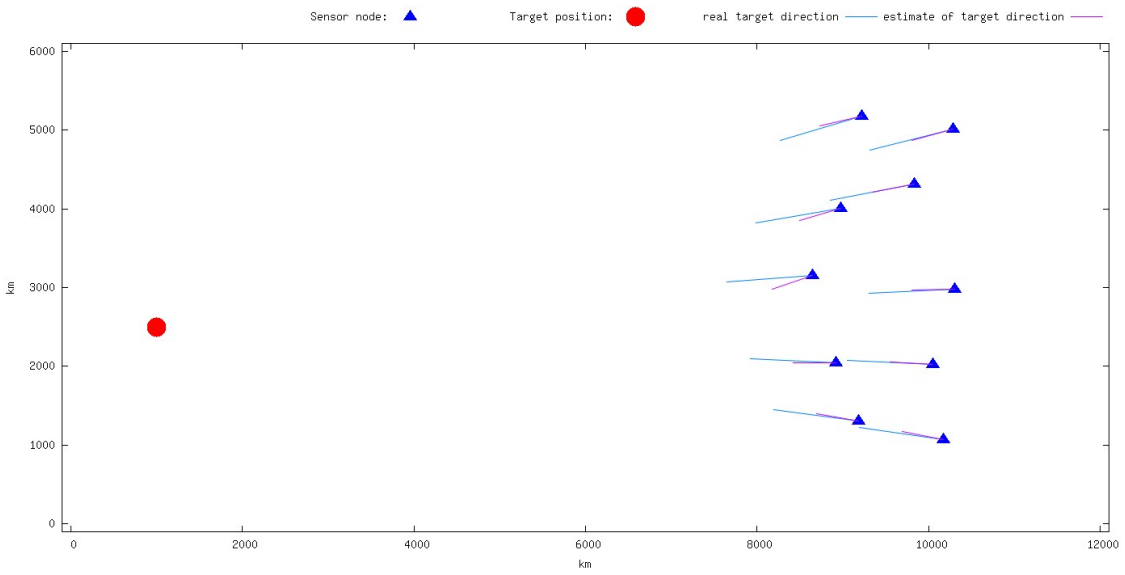
En generalisering av metoden ovenfor er at nodene kan rapportere for eksempel to samtidige retninger. Figur 3.3 illustrerer en slik situasjon. En "likelihood" funksjon som ovenfor kan da parameteriseres ved at for hvert punkt i planet (for det aktuelle området innen rekkevidde) får knyttet til seg de observerte vinklene som er mest "likely". Figur 3.4 illustrerer en "likelihood-funksjon" som kan fremkomme på denne måten.

# 4 Estimater av rette track

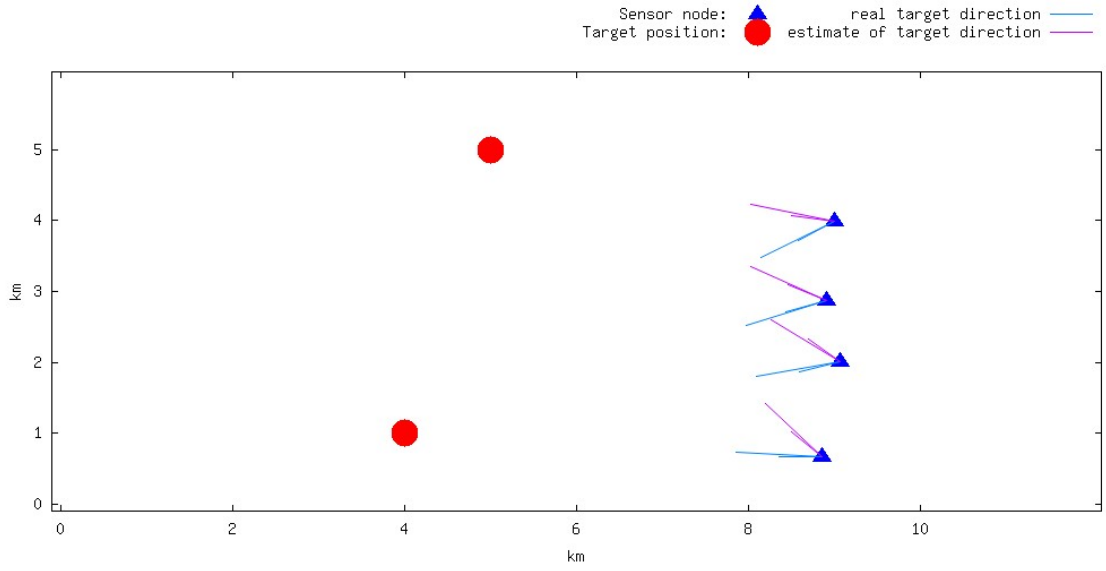
Her forklares en metode for å estimere posisjoner til objekter basert på antagelse om konstant (vektor-)hastighet til lydkilder med konstant lyd-intensitet (dvs konstante lydkilder som bevegelser seg i rette linjer med samme farten hele tiden). Figur 4.1 viser en mulig målt tidsserie for lyd ved en node for en slik lydkilde. Spredning og demping er ved dette høvet i henhold til Ligning 5.1. Denne modellen for spredning/demping av lyd kan ha store feil. Imidlertid så vil målt lydintensitet som oftest avta ved avstanden til lydkilden (ihvertfall i åpen hav). Dvs med stor sannsynlighet er tidspunktet for maksimum målt lydintensitet sammenfallende med når lyd-kilden er nærmest. Dette kan brukes til å følge lydkilder som beveger seg gjennom sensor-feltet. Hver node registrerer tidspunkt for når den måler maksimum lyd-intensitet. Denne informasjonen blir så sendt til en fusjonsentral som kan konstruere en figur som ligner på Figur 4.2. Passeringstidene $t1, t2, t3$ for når lydkilden er nærmest de forskjellige nodene er altså da direkte tilgjengelige (sentralt). La $I1, I2, I3$ være lyd-intensiteten tilhørende $t1, t2, t3$ (målt ved nodene Node1, Node2 og Node3). Avstandene (normalt) til tracket blir da (over-)bestemt av forholdene $I1/I2$, $I1/I3$ og $I2/I3$ (jmf Ligning 5.1). Dette bestemmer fart og retning til objektet. Legg merke til at slike estimater kan kombineres med estimater for retninger produsert ved enkelt-nodene.



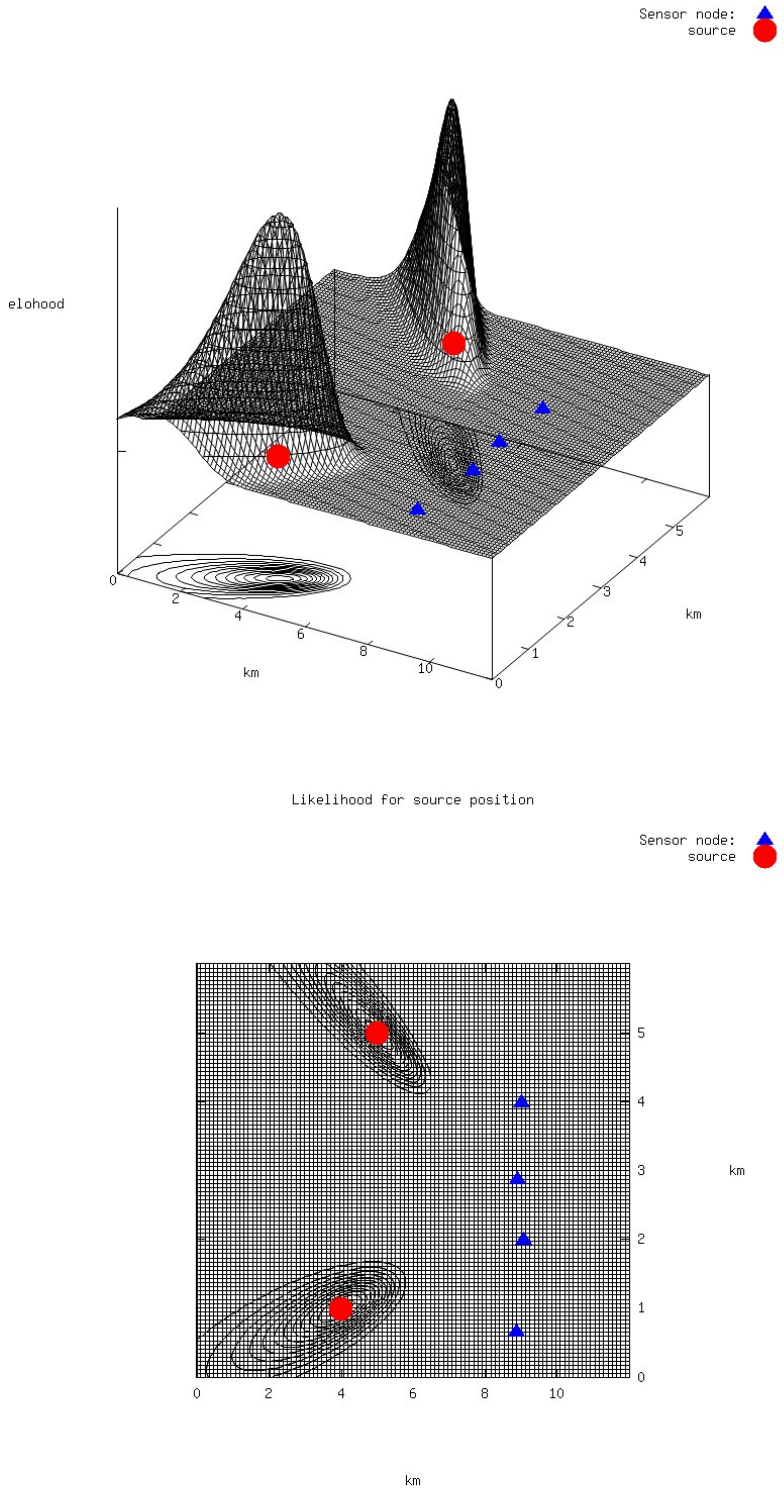

*Figur 3.1: Eksempel på likelihood verdier for forskjellige posisjoner gitt observasjoner fra 10 noder (gitt $\kappa = 100$). 3d plot med to forskjellige synsvinkler.*



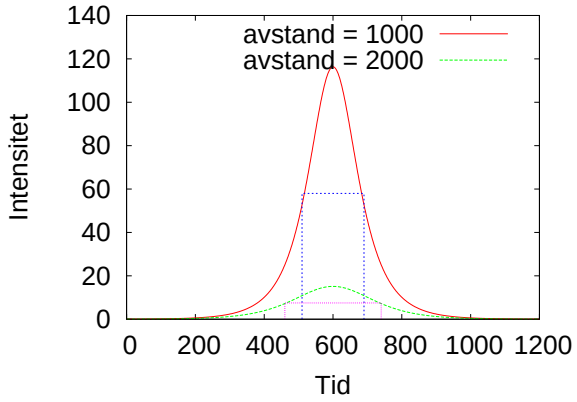

*Figur 3.2: Observasjon av retninger og som inkluderer målefeil ($\kappa = 100$).*



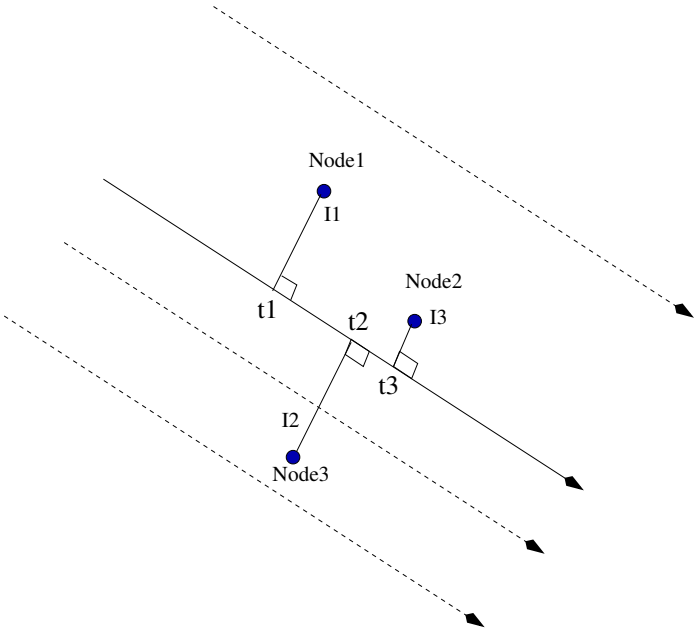

*Figur 3.3: Observasjon av to samtidige retninger med målefeil feil som ovenfor ($\kappa = 100$).*




*Figur 3.4: "Blandet likelihood-funksjon" for posisjonen til to lydkilder basert på målinger av to vinkler ved hver node ($\kappa = 100$).*




*Figur 4.1: Mulig variasjon i registrert lydintensitet ved noder som registrerer konstant lydkilder som går i rette linjer med konstant fart. Lydintensitet fordeles over lengre tid med økt avstand til track (bredere intervall med verdi over halvparten av maksimum verdi). Her er breddene slik at $(L/l)^{-1.5} \simeq 2$ (samme som forholdet mellom avstandene til trackene) der $L$ og $l$ er henholdsvis lengste og korteste "halverings-bredde".*




*Figur 4.2: Estimering av track til konstant lydkilde som beveger seg jevnt gjennom sensornettverk i rett linje. Fart og retning til trackene blir bestemt av tidspunktene til maksimum registrert lydintensitet. Prikket linje angir mulig track (basert på tidspunkter for maksimum lydintensitet ved nodene). Forhold mellom maksimum lydintensiteter $(I1, I2, I3)$ bidrar til å gi estimater for avstander til trackene (fra nodene).*

Anta at lyden blir spredt horisontalt isotropt og homogent men at en ellers ikke kjenner lagdelingen i vannmassene og at alle nodene er på samme dyp. Legg da merke til at dersom nodene kan kommunisere (for eksempel via en dialog) seg imellom tidspunktene for når de eventuelt måler samme lyd-intensitet, så kan slike rette track (som nevnt ovenfor) bestemmes ut fra tidspunkter for maksimum lydhastighet og tidspunktene for (eventuell) lik lydintensitet (uten at en antar kompliserte modeller for spredning av lyd i vannet). Dette kan så videre brukes til å parameterisere/bestemme akustiske forhold i farvannet. Legg også merke til at antagelse om horisontalt isotrope og homogene vannmasser også kan brukes til å estimere spredning og svekkelse av lyd i vann via en iterativ prosedyre der en iterasjonen går ut på å oppnå konsistente observasjoner nodene imellom. Figur 4.1 illustrerer at lyd blir spredt i tid (økt "standardavvik") ved økt avstand. Slikt "standardavvik" kan brukes til adaptive tilnærmelser ("læring") over tid samt estimering av situasjons-bilder der en altså krever at lyd-opptakene (for eksempel spredning i tid) ved nodene er konsistente.

# 5 Romlig fordeling av lydintensitet - randomiserte metoder

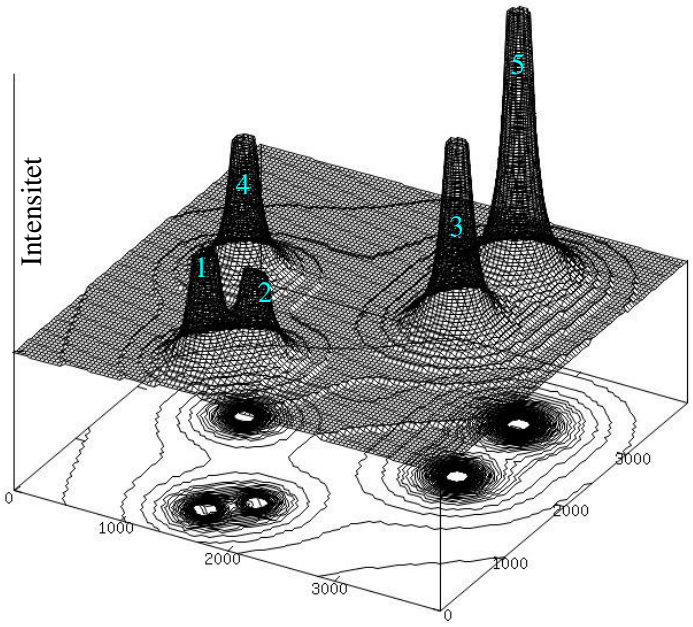
Anta at nodene i sensornettverket måler lydintensitet og rapporterer dataene inn til en sentral for datafusjon. Lyd kan spre seg på kompliserte måter i havet. Absorpsjon og hastighet (ca 1500 m/s) avhenger her av salinitet, trykk og temperatur. Absorpsjon avhenger også av frekvens. Variasjon i hastighet påvirker spredning av lyd (jmf Snell's lov). Refleksjon fra bunn kompliserer ytterligere. En "praktisk formel" for spredning av lyd i vann er at energien (grunnet spredning) avtar med avstanden $r$ fra kilden som $r^{-1.5}$ (Stojanovic (2006)). Dette kan sees på som et 'kompromiss' mellom lik (isotrop) spredning i alle retninger i et tre-dimensjonalt rom ($r^{-2}$), og tilsvarende spredning bare horisontalt/sylindrisk ($r^{-1}$).

Figur 5.1 illustrerer et eksempel på romlig fordeling av lydintensitet (energiflukser) skapt av fem lydkilder. Intensiteten $I$ fordeler seg ved dette høvet i henhold til modellen:

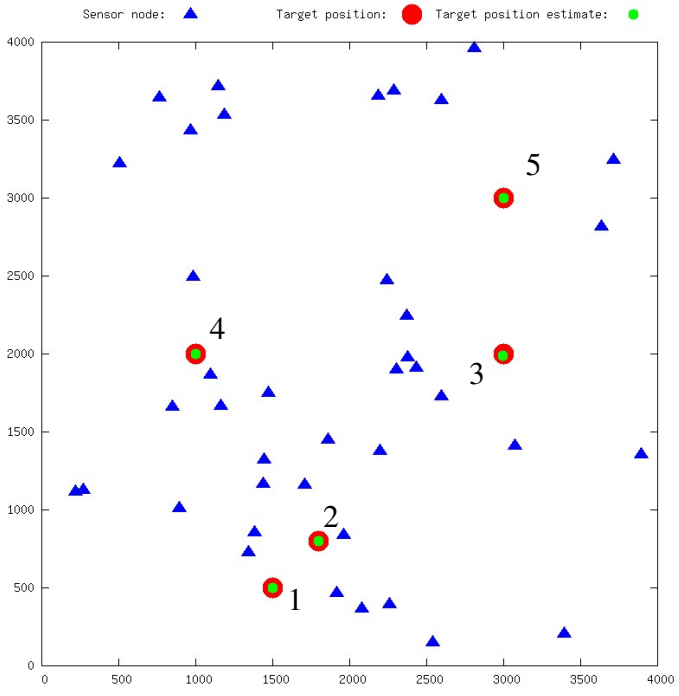
$$I = E * r^{-1.5} \exp(-\mu r) \tag{5.1}$$

der $E$ er intensiteten en enhet avstand fra punktkilden, $r$ er avstand og $\mu$ angir demping (varmeproduksjon i vannet). De absolutte verdiene er her normalisert slik at $E_5 = 100$. Kildene er i punktene $z_1 = (1500.0, 500.0)$, $z_2 = (1800.0, 800.0)$, $z_3 = (3000.0, 2000.0)$, $z_4 = (1000.0, 2000.0)$ og $z_5 = (3000.0, 3000.0)$. De tilhørende (relative) verdiene for $E$ er: $E_1 = 0.4E_5$, $E_2 = 0.3E_5$, $E_2 = 0.7E_5$ og $E_2 = 0.5E_5$.

Anta at total lydintensitet (for gitte frekvensbånd) måles ved noder inne i dette lyd-feltet (Figur 5.1). Anta også at en kjenner presist hvordan lyd sprer seg og dempes i vannet. Forholdene mellom de målt lydintensitet forskjellige steder gir da mulighet til å beregne posisjonene til lydkildene (forutsatt tilstrekkelig data-dekning). Figur 5.2 gir resultatet fra en slik beregning der en antar fordeling av lydintensitet slik som illustrert i Figur 5.1. En har her antatt som kjent posisjonen til node nummer 2 og 5 (som kunne tenkes tilhøre observerte overflate-fartøyer eller egne lyd-kilder). Data-programmet




*Figur 5.1: Simulert total intensitet av lyd fra fem punkt-kilder innen et* $4000 \times 4000\, km^2$ *flatt område ved gitt dybde.*



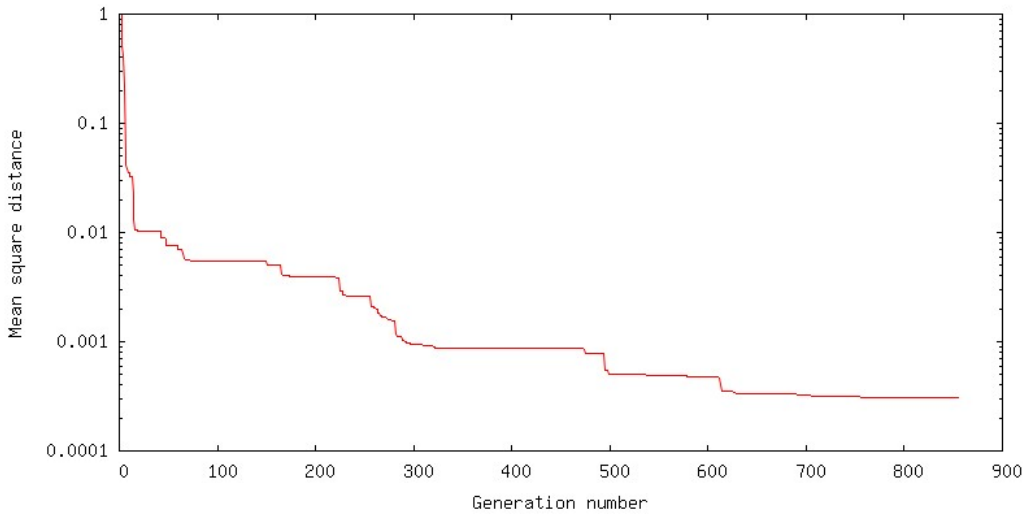

*Figur 5.2: Estimerte posisjoner for lydkilder (grønne punkter) tilhørende lydfelt gitt ved Figur 5.1 og som måles av 30 sensor-noder med tilfeldig utplassering i området. Posisjonene til lydkilde 2 og 5 er kjente.*

som beregnet posisjonene for Figur 5.2, er eksperimentelt og gjengitt i Appendix Appendix A for å vise hvor enkle slike beregninger kan gjøres.

Metoden for denne test-beregningen baserer seg på (noe forenklet) idéer om 'evolusjon', 'genetisk programmering' og randomisert problemsøsing. Dette gir muligheter til å lage enkle programmer som gir estimater for posisjoner til lydkilder i vann (ved dette høvet mindre enn 200 program-linjer i Ada 2005 inkludert program-linjer for lesing og utskrift). Denne programmerings-teknikken er velkjent fra mange sammenhenger slik som innen industri, transport, operativ-systemer og forskning (Eiben and Smith 2003, Motwani and Raghavan 1995). Eiben and Smith (2007) brukes forøvrig som lærebok ved Universitetet i Oslo.

Metoden finner optimale punkter (maksima/minima) for funksjoner/løsninger. Dersom det er flere samtidige krav (kvalitetskriteria) for løsninger i den aktuelle problemstillingen, så kalles dette "multi-objektiv" optimering hvor det produseres mengder av (flere) mulige løsninger. Hver enkelt løsning i denne mengden er optimal med hensyn til en av optimaliserings-kriteriene.

Deterministisk beregning blir ofte regnings-krevende og kompliserte spesielt dersom de skal inkludere bruk av mange forskjellige (gjerne usikre) informasjoner. Ved dette høvet gjelder det å finne posisjoner til mulige lydkilder slik at predikerte forhold mellom lydintensitet ved forskjellige lokasjoner blir mest mulig lik forholdene mellom observert lydintensitet. Det aktuelle analyse-programmet, som beregnet posisjonene for Figur 5.2, brukte minste kvadrats avstand som mål på 'likhet'. Figur 5.3 viser et eksempel på utvikling av dette målet som funksjon av generasjon (iterasjon) nummer




*Figur 5.3: Minste kvadrats avstand som funksjon av generasjon nummer.*

i evolusjonen (som fører til løsningen vist i Figur 5.2. Mange forskjellige mål vil i praktisk gi like resultater.

Bruk av idéer om 'evolusjon' betyr at en 'avler' på en 'bestand' av løsnings-forslag hvor ønskelige gener fremmes via rekombinering og mutasjoner. I eksemplet ovenfor kan posisjoner og lyd-

styrke (til lydkilder) sees på som ”gener”. Slik kan en effektivt søke etter optimale punkter i store mengder/rom av mulige løsninger (søkerom). Tradisjonelle optimaliserings-teknikker (der f.eks. den deriverte av minste kvadrats avstand settes lik null) kan til sammenligning bli kompliserte og vanskeligere å kontrollere.

En metode som ”avler” frem en bestand av forslag til løsninger på et problem, kan sies å akkumulere informasjon om endelige løsninger. Forskjellige forslag til løsninger kan være gode på sine spesielle måter. Effektiviteten til denne søke-metoden avhenger av hvor fort det akkumuleres informasjon om løsninger. Aktuelle variable i slike søk er størrelse på bestand (løsnings-forslag), mutasjons-rate og kommunikasjon av genetisk informasjon innen bestanden. Er mutasjons-raten for høy, så akkumuleres lite informasjon i søket som da blir mye likt tilfeldig søk. Er mutasjons-frekvensen for lav, så akkumuleres heller ikke informasjon så effektivt (dette gir unødvendig repetisjon i søkene). Rekombinering og seleksjon av ”gener” innen populasjonen må også avbalanseres for effektiv akkumulering av informasjon. For intensiv seleksjon gjør at informasjon går tapt og (motsatt) for lite intensiv seleksjon reduserer genetisk drift mot løsninger.

Følgende argumenter kan fremmes for ’evolusjonære’ algoritmer sammenlignet med alternativer:

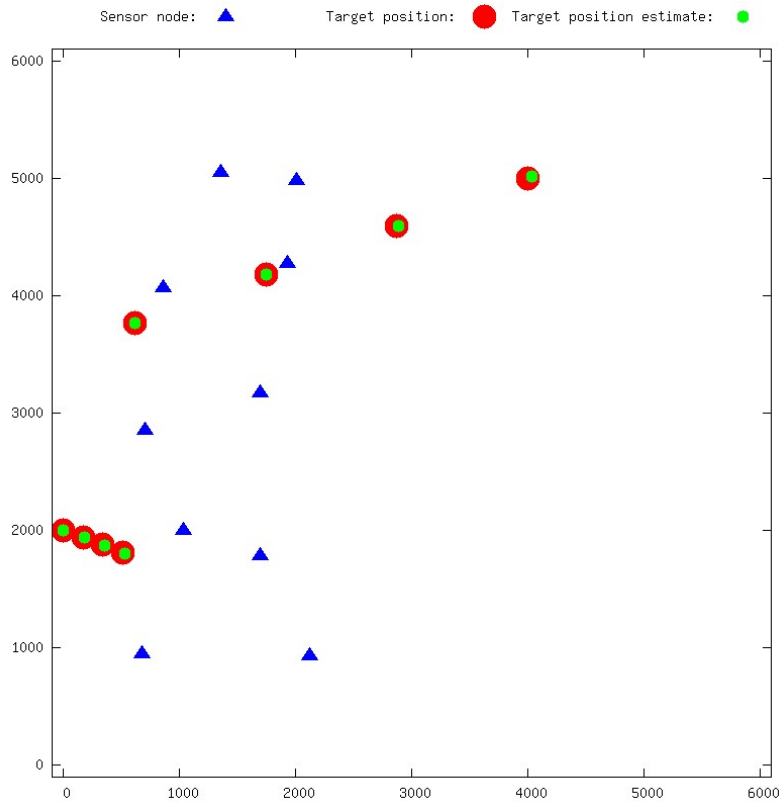
- Enkel, intuitiv, enkel å parallellisere.
- Kan direkte utnytte forskjellig informasjon og målinger fra forskjellige steder/tidspunkter. Eksempel: et fartøy har gjerne likt lydbilde over tid eller det er gjerne sammenheng mellom fart og lyd. Et helhetlig situasjonsbilde kan ha slike konsistens-sjekker og gi positive/negative ’poeng’ i henhold til dette.
- Data/kunnskap om lydkilder kan kan ha usikkerheter. Det er ’kanskje’ et fartøy i området med kjent lyd-profil. Slik informasjon kan brukes direkte.
- Lyd-kilder kan følges i tid og målinger ved forskjellige avstander kan utnyttes.

Metoden ovenfor gir direkte estimater av utgangs-energi (intensitet) for de enkelte lyd-kildene. En kan tenke seg (frekvens) filtrering i sensor-nodene slik at nodene sender (enkle) energi-spektra til en sentral for datafusjon. Slik kan denne sentralen beregne energi-spektra for de forskjellige lyd-kildene og eventuelt klassifisere dem. Svekking av lyd i vann avhenger av frekvens og som det kan kompenseres for. Modell for svekking av lyd kan også brukes innen estimering av lydkilder (intensitet for forskjellige frekvenser samt posisjoner). Poenget her er at datafusjon gir et sett av mulige konsistente forslag til ’forklaringer’ på observert lyd.

Absorpsjon av lyd i vann øker med frekvens. Dette kan utnyttes ved at posisjoner til lydkilder gjerne kan estimeres først ut fra de signifikante delene av et spekter med de høyeste frekvensene (korteste bølgelengdene). En kan også tenke seg å tillegge de høyeste frekvensene størst vekt i en samlet analyse (hvor alle spektral-verdiene brukes i en og samme analyse slik som beskrevet ovenfor). Denne vekten kan også avhenge av usikkerheten til spektral-verdiene slik at de sikreste dataene tillegges størst vekt.

# 6 Tracking

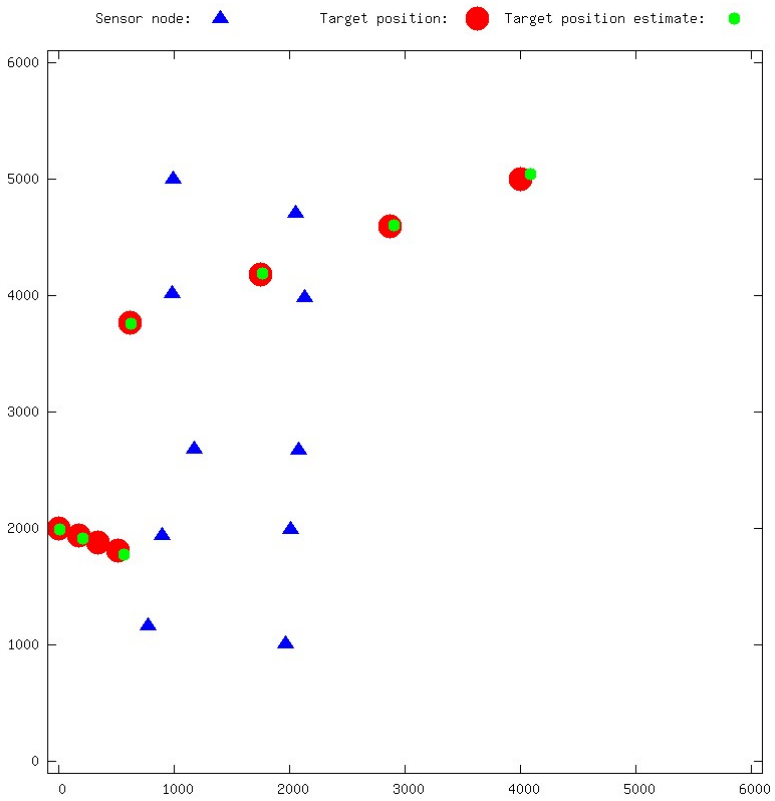
Figurene 6.1 og 6.2 viser eksempler på (simulert) ”tracking” av to objekter basert på målt lyd-




*Figur 6.1: Eksempel på simulert tracking av to samtidige objekter ved å sample posisjon hvert minutt. Øverste og nederste track har henholdsvis 20 m/s og 3 m/s.*

intensitet. Programmet i Appendix Appendix A er brukt for denne data-analysen. Dersom det er få noder som plasseres i rette linjer, så kan det lett oppstå tvetydigheter. Figur 6.2 viser at den aktuelle metoden kan gi usikre estimater ved slike høve. Imidlertid så kan tilleggs-informasjon (retninger) rette opp dette slik at resultatet likevel blir presist til slutt. Lydstyrken knyttet til det øverste tracket er konstant 5 ganger (energi) intensiteten for nederste track. Analyseprogrammet gir også dette innen 5 prosent presisjon (for noen få kjøre-eksempler som ga presise resultater).

En kan tenke seg uavhengig estimering av posisjoner for forskjellige spektral-komponenter for så å produsere et helhetlig ”bilde” som inkluderer tracking av objekter med gitte lyd-spektra samt estimater av parametre for akustiske forhold i vannet. Dette inkluderer estimering/parametrisering av svekking (absorpsjon) av lyd (avhengig av frekvens) for å få helhetlige/konsistente ”situasjons-bilder”.




*Figur 6.2: Eksempel på simulert tracking av to samtidige objekter ved å sample posisjon hvert minutt. Øverste og nederste track har henholdsvis 20 m/s og 3 m/s. Ett av posisjonsestimatene er mye feil grunnet uheldig plassering i forhold til sensor.*

## Referanser


Eiben, A. E. and Smith, J. E.: 2003, *Introduction to Evolutionary Computing (Natural Computing Series)*, Springer.

Eiben, A. E. and Smith, J. E.: 2007, *Introduction to Evolutionary Computing*, Springer.

Motwani, R. and Raghavan, P.: 1995, *Randomized algorithms*, Cambridge University Press, New York, NY, USA.

Stojanovic, M.: 2006, On the relationship between capacity and distance in an underwater acoustic communication channel, *WUWNet '06: Proceedings of the 1st ACM international workshop on Underwater networks*, ACM, New York, NY, USA, pp. 41–47.

# Appendix A

## A.1 Program som illustrerer randomiserte metoder

Seksjonene A.2, A.3 og A.4 nedenfor viser kilde-koden (Ada 2005) for programmet som estimerte posisjonene illustrert i Figur 5.2 i denne rapporten. Poenget med å inkludere denne kilekoden her er å vise hvor overraskende enkle randomiserte metoder kan være (sammenlignet med deterministiske). Det aktuelle kjøre-eksemplet ga estimater på posisjoner inne 100 m presisjon etter noen sekunder. Høyere presisjon krever mer kjøretid.

## A.2 Analyseprogrammet analysis2.adb

```
1  -- Author: Reinert Korsnes, FFI
2  -- Date:       10.10.2008
3  -- Revision: 10.11.2008 - Organizing source layout
4
5  with Ada.Numerics.Generic_Elementary_Functions;
6  with Ada.Numerics.Float_Random;
7  with Ada.Numerics.Discrete_Random;
8  with Text_IO;
9  with Ada.Containers.Vectors;
10 use  Text_IO,Ada.Numerics,Ada.Numerics.Float_Random,Ada.Containers;
11 with sensormodel1;
12 use  sensormodel1;
13 procedure analysis2 is
14
15   use sensormodel1.Int_Io;
16   use sensormodel1.Real_Io;
17   use sensormodel1.E_F;
18
19   subtype Parent_Number_t is Positive range 1..200;
20   parent : array(boolean) of Parent_Number_t;
21   package Parent_Number_Random is new
22         Ada.Numerics.Discrete_Random(Result_Subtype => Parent_Number_t);
23   package Boolean_Random is new
24         Ada.Numerics.Discrete_Random(Result_Subtype => boolean);
25
26   subtype Intensity_t is Real range 0.0 .. 2000.0;
27
28   type ESource_t is
29      record
```

```
30          z    : z_t;
31          s    : Intensity_t;
32          v    : Vector_t;
33       end record;
34
35    type Search_t is array(1..2) of ESource_t;
36    SSource1 : Search_t;
37
38    type Population_t is
39       record
40          SSource : Search_t;
41          dis     : Real;
42       end record;
43
44    package Population_P is new Vectors(Positive,Population_t);
45    use Population_P;
46
47    function "<" (Left,Right : Population_t) return Boolean is
48    begin
49        return Left.dis < Right.dis;
50    end "<";
51    package Sorting_Population is new Population_P.Generic_Sorting("<");
52    use Sorting_Population;
53
54
55    type Nodet_t is new Node_t with
56       record
57          sn,sm : Sound_t;
58       end record;
59
60    type Nodes_t is array(inode_t) of Nodet_t;
61
62    G_float   : Float_Random.Generator;
63    G_Parent  : Parent_Number_Random.Generator;
64    G_boolean : Boolean_Random.Generator;
65
66    function erandom(a,b : Real) return Real is
67    begin
68        return Real(Random(Gen => G_float))*(b-a) + a;
69    end erandom;
70
```

```
71    procedure fit1(Node : in out Nodes_t; SSource: Search_t; dis : out Real) is
72       r : Real;
73       sum_simsound : Real;
74    begin
75       dis := 0.0;
76       for t in time_t loop
77           sum_simsound := 0.0;
78           for i in inode_t loop
79               Node(i).sm(t) := 0.0;
80               for k in SSource'range loop
81                   r := length(SSource(k).z(t) - Node(i).z);
82                   Node(i).sm(t) := Node(i).sm(t) + intensity(r,SSource(k).s);
83               end loop;
84               sum_simsound := sum_simsound + Node(i).sm(t);
85           end loop;
86           for i in inode_t loop
87               Node(i).sm(t) :=    Node(i).sm(t)/sum_simsound;
88               dis := dis + (Node(i).sn(t) - Node(i).sm(t))**2;
89           end loop;
90       end loop;
91    end fit1;
92
93    Population1,Parents1,Children1 : Population_P.Vector;
94    Child1    : Population_t;
95    Node : Nodes_t;
96
97    sum_measurements : Real;
98
99    ds          : Intensity_t;
100   dx          : x_t;
101   dy          : y_t;
102   p_mutation : Real range 0.0..1.0;
103
104   dis : Real := Real'last;
105   b           : Boolean;
106
107   nodes2_file,measurements2_file,esource2_file   : File_Type;
108
109   m           : Natural := 0;
110
111 begin
```

```
112
113     Float_Random.Reset (Gen => G_float);
114     Parent_Number_Random.Reset (Gen => G_Parent);
115     Boolean_Random.Reset (Gen => G_boolean);
116
117     Open(nodes2_file,In_File,"nodes2.dat");
118     for i in Node'range loop
119         Get(nodes2_file,Node(i).z);
120         Put("␣i,z␣:␣");Put(Integer(i),4);Put(Node(i).z);New_Line;
121     end loop;
122     Close(nodes2_file);
123
124     Open(measurements2_file,In_File,"measurements2.dat");
125     for t in time_t loop
126         Put("␣t:␣");Put(Integer(t),4);
127         for n in inode_t loop
128             Get(measurements2_file,Node(n).s(t));
129             Put(Node(n).s(t),2,6,0);
130         end loop;
131         New_Line;
132     end loop;
133     Close(measurements2_file);
134
135     for t in time_t loop
136         sum_measurements := 0.0;
137         for i in inode_t loop
138             sum_measurements := sum_measurements + Node(i).s(t);
139         end loop;
140         for i in inode_t loop
141             Node(i).sn(t) := Node(i).s(t)/sum_measurements;
142         end loop;
143     end loop;
144
145     SSource1(SSource1'last).s :=  100.0;
146     while Population1.Length <= 100_000 loop
147         for k in SSource1'range loop
148             for t in time_t loop
149                 SSource1(k).z(t) := (erandom(x_t'first,x_t'last),erandom(y_t'fi
150             end loop;
151             if k < SSource1'last then
152                SSource1(k).s := erandom(Intensity_t'first,Intensity_t'last);
```

```
153             end if;
154          end loop;
155          fit1(Node,SSource1,dis);
156          Population1.Append(Population_t'(SSource1,dis));
157       end loop;
158
159       Sort(Population1);
160       Delete(Population1,Parent_Number_t'last+1,Population1.Length);
161       Parents1     := Population1;
162       Children1    := Empty_Vector;
163       Population1  := Empty_Vector;
164
165    Iterate:
166       while m < 100 loop
167
168          Case m Mod 4 is
169             when 0 => -- wild:
170                ds := Intensity_t'last;
171                dx := x_length;
172                dy := y_length;
173                p_mutation := 0.25;
174             when 1 => -- moderate:
175                ds := Intensity_t'last/10.0;
176                dx := x_length/10.0;
177                dy := y_length/10.0;
178                p_mutation := 0.05;
179             when others => -- conservative:
180                ds := Intensity_t'last/100.0;
181                dx := x_length/100.0;
182                dy := y_length/100.0;
183                p_mutation := 0.025;
184          end Case;
185
186       Make_Children1:
187          while Children1.Length <= 10_000 loop
188
189             parent(true)  := Parent_Number_Random.Random(Gen => G_Parent);
190             parent(false) := Parent_Number_Random.Random(Gen => G_Parent);
191
192             for k in Search_t'range loop
193
```

```
194                b := Boolean_Random.Random(Gen => G_boolean);
195                Child1.SSource(k).s := Parents1.Element(parent(b)).SSource(k).s
196                if k < Search_t'last
197                         and Random(Gen => G_float) < Float(p_mutation) then
198                   Child1.SSource(k).s :=
199                     erandom(Real'Max(Child1.SSource(k).s-ds,Intensity_t'first)
200                             Real'Min(Child1.SSource(k).s+ds,Intensity_t'last )
201                end if;
202
203                for t in time_t loop
204
205                    b := Boolean_Random.Random(Gen => G_boolean);
206                    Child1.SSource(k).z(t) := Parents1.Element(parent(b)).SSour
207
208                    if Random(Gen => G_float) < Float(p_mutation) then
209                       Child1.SSource(k).z(t) :=  Child1.SSource(k).z(t)
210                                                + (erandom(-dx,dx),erandom(-dy,d
211                    end if;
212
213                end loop;
214
215             end loop;
216
217             fit1(Node,Child1.SSource,Child1.dis);
218             Children1.Append(Child1);
219
220          end loop Make_Children1;
221
222          Population1 := Parents1 & Children1;
223          Sort(Population1);
224          Delete(Population1,Parent_Number_t'last+1,Population1.Length);
225          Parents1   := Population1;
226
227          Population1   := Empty_Vector;
228          Children1     := Empty_Vector;
229
230          Put("* dis : ");Put(Parents1.First_Element.dis,3,8,0);
231          Put(" m = "); Put(Integer(m),4); New_Line;
232
233          if Parents1.First_Element.dis < 0.999*dis then
234             for t in time_t loop
```

```
235                 for k in Parents1.First_Element.SSource'range loop
236                     Put( Parents1.First_Element.SSource(k).z(t));
237                     Put( Parents1.First_Element.SSource(k).s,6,1,0);
238                 end loop;
239                 New_Line;
240             end loop;
241             dis := Parents1.First_Element.dis;
242             m := 0;
243         end if;
244
245         m := m + 1;
246
247     end loop Iterate;
248
249     Create(esource2_file,Out_File,"esource2.dat");
250     for k in Parents1.First_Element.SSource'range loop
251         for t in time_t loop
252             Put(esource2_file,Parents1.First_Element.SSource(k).z(t));
253             New_Line(esource2_file);
254         end loop;
255         New_Line(esource2_file);
256     end loop;
257     Close(esource2_file);
258
259 end analysis2;
```

### A.3 Spesifikasjon av hjelpeprogrammer (sensormodel1.ads)

```
1  -- Author: Reinert Korsnes, FFI
2  -- Date:       15.10.2008
3  -- Revision: 07.11.2008 - Organizing source layout
4
5  package body sensormodel1 is
6
7      procedure Put(Data_File : File_Type;z : Vector_t) is
8      begin
9          Put(Data_File,z.x,6,1,0);
10         Put(Data_File,z.y,6,1,0);
11     end Put;
12
13     procedure Get(Data_File : File_Type;z : out Vector_t) is
```

```
14    begin
15       Get(Data_File,z.x);
16       Get(Data_File,z.y);
17    end Get;
18
19    procedure Put(z : Vector_t) is
20    begin
21       Put(z.x,6,1,0);
22       Put(z.y,6,1,0);
23    end Put;
24
25    function "-" (Left,Right : Vector_t) return Vector_t is
26    begin
27       return (Left.x - Right.x, Left.y - Right.y);
28    end "-";
29
30    function "+" (Left,Right : Vector_t) return Vector_t is
31    begin
32       return (Left.x + Right.x, Left.y + Right.y);
33    end "+";
34
35    function length(z : Vector_t) return Real is
36    begin
37       return sqrt(z.x**2 + z.y**2);
38    end length;
39
40    function intensity(r,E : Real) return Intensity_t is
41       r1 : Real := Real'Max(10.0,r);
42    begin
43       return E*r1**(-1.5)*exp(-r1/1000.0);
44    end intensity;
45
46    function "*" (Left : Vector_t;Right : Real) return Vector_t is
47    begin
48       return (Left.x*Right,Left.y*Right);
49    end "*";
50
51    function "*" (Left : Real;Right : Vector_t) return Vector_t is
52    begin
53       return (Left*Right.x,Left*Right.y);
54    end "*";
```

```
55
56 end sensormodel1;
```

### A.4 Realisering av hjelpeprogrammer (sensormodel1.adb)

```
1  -- Author: Reinert Korsnes, FFI
2  -- Date:       15.10.2008
3  -- Revision: 07.11.2008 - Organizing source layout
4
5  package body sensormodel1 is
6
7     procedure Put(Data_File : File_Type;z : Vector_t) is
8     begin
9        Put(Data_File,z.x,6,1,0);
10       Put(Data_File,z.y,6,1,0);
11    end Put;
12
13    procedure Get(Data_File : File_Type;z : out Vector_t) is
14    begin
15       Get(Data_File,z.x);
16       Get(Data_File,z.y);
17    end Get;
18
19    procedure Put(z : Vector_t) is
20    begin
21       Put(z.x,6,1,0);
22       Put(z.y,6,1,0);
23    end Put;
24
25    function "-" (Left,Right : Vector_t) return Vector_t is
26    begin
27       return (Left.x - Right.x, Left.y - Right.y);
28    end "-";
29
30    function "+" (Left,Right : Vector_t) return Vector_t is
31    begin
32       return (Left.x + Right.x, Left.y + Right.y);
33    end "+";
34
35    function length(z : Vector_t) return Real is
36    begin
```

```
37          return sqrt(z.x**2 + z.y**2);
38       end length;
39
40       function intensity(r,E : Real) return Intensity_t is
41          r1 : Real := Real'Max(10.0,r);
42       begin
43          return E*r1**(-1.5)*exp(-r1/1000.0);
44       end intensity;
45
46       function "*" (Left : Vector_t;Right : Real) return Vector_t is
47       begin
48          return (Left.x*Right,Left.y*Right);
49       end "*";
50
51       function "*" (Left : Real;Right : Vector_t) return Vector_t is
52       begin
53          return (Left*Right.x,Left*Right.y);
54       end "*";
55
56    end sensormodel1;
```