SeisDrift

Autonom seismisk datainnsamlingsenhet

Hovedprosjekt HiB våren 2005 – Erik Grindheim

Oppdragsgiver: Institutt for Geovitenskap, UiB

INNHOLDSFORTEGNELSE

1. SAMMENDRAG

I forbindelse med en Arktisk ekspedisjon (kalt HOTRAX) hvor Universitetet i Bergen deltar, skal det lages en prototyp av et utstyr som kan plasseres på et isflak og samle inn refleksjonsseismiske data. Tanken er at isflaket driver med havstrømmen. Utstyret kalles **SeisDrift**, og skal overføre data til Bergen via satellitt-telefonsystemet Iridium.

Institutt for Geovitenskap ønsker "proof og concept". Arbeidet dreier seg om en prototyp. Det viktigste blir ikke å lage en optimalisert enhet med lav vekt og lite strømforbruk, det må først og fremst bare lages noe som fungerer!

Jeg har arbeidet med denne enheten i vår, og i sommer fortsetter jeg både før og under ekspedisjonen. I august/september tar vi SeisDrift med til Nordpolen for å eksperimentere.

Ekspedisjonen starter i Dutch Harbor (Alaska), sør for Beringstredet, i månedsskiftet juli/august. Transportmidlet blir US Coast Guard sin nyeste isbryter "Healy". I løpet av den to måneder lange ekspedisjonen i ishavet skal SeisDrift prøves ut, og man vil finne ut om konseptet fungerer. Ekspedisjonen ender etter planen i Tromsø den 30. september.

Jeg vil takke følgende personer for hjelp og støtte med oppgaven i vår:

- ***Solfrid Sjåstad Hasund***
  Studiekoordinator på elektronikklinjen ved HiB, og min veileder på prosjektet.

- ***Lars Edgar Berg***
  Leder ved Institutt for Elektrofag, og en "kløpper" på programmering!

- De involverte ved Institutt for Geovitenskap, UiB

2. INNLEDNING

2.1 Oppdragsgiver

Oppdragsgiveren er *Institutt for Geovitenskap*, ved Universitetet i Bergen (UiB).

Adresse: Institutt for Geovitenskap, UiB
Allégaten 41,
5007 BERGEN

De ansvarlige personene på UiB er:

Overingeniør	Ole Petter Meyer,	5558 3421
Professor	Yngve Kristoffersen,	5558 3407

2.2 Problemstilling

Oppdragsgiver skal delta på et tokt i Arktis i august og september 2005, og ser dette som en god anledning til å prøve ut et konsept de har hatt i tankene en tid. Konseptet går ut på å lage en autonom seismisk datainnsamlingsbøye som kan plasseres på et isflak som driver med havstrømmen. Oppdragsgiver er interessert i:

- å få vurdert problemstillinger knyttet til fremstilling av en autonom seismisk datainnsamlingsenhet for isdekte havområder
- å komme i gang med fremstillingen av en prototyp av denne enheten

2.3 Hovedidé

Enheten, heretter kalt **SeisDrift**, skal plasseres på isflak hvor det borres hull for seismisk kilde og -mottaker. Enheten driver med isflaket og vil utløse den seismiske kilden og foreta registreringer hver 50. meter, på basis av posisjonsinformasjon fra GPS mottaker. Måledata lagres i enheten og overføres også i komprimert form til datamaskin annet sted via Iridium satellitt modem. Også posisjonsinformasjon overføres via satellittsambandet.

For den seismiske kilden ser det ut til at en såkalt "Sparker" er best egnet. Sparkeren er i prinsippet to elektroder som via en elektronisk bryter er forbundet med polene i et høyspenning kondensatorbatteri. Når avfyringskriterier er oppfylt aktiveres denne bryteren og akustisk energi spres i vannet som et resultat av den lysbuen som oppstår mellom elektrodene.

3. DEFINISJON AV OPPGAVEN

3.1 Kravspesifikasjon

- SeisDrift skal kunne operere alene i opptil 3 uker.
- SeisDrift må strømforsynes av batterier, solceller eller lignende.
- SeisDrift må tåle det arktiske klimaet, og kunne operere i de temperaturene som er aktuelle i Arktis i sommerhalvåret.
- Det skal benyttes en såkalt ”Sparker” som seismisk kilde.

3.2 Målsetning for oppgaven

- I samarbeid med oppdragsgiver fremstille en overordnet kravspesifikasjon av systemet.
- I samarbeid med oppdragsgiver (eventuelt alene) spesifisere de enkelte moduler som konstruksjonen kan deles opp i.

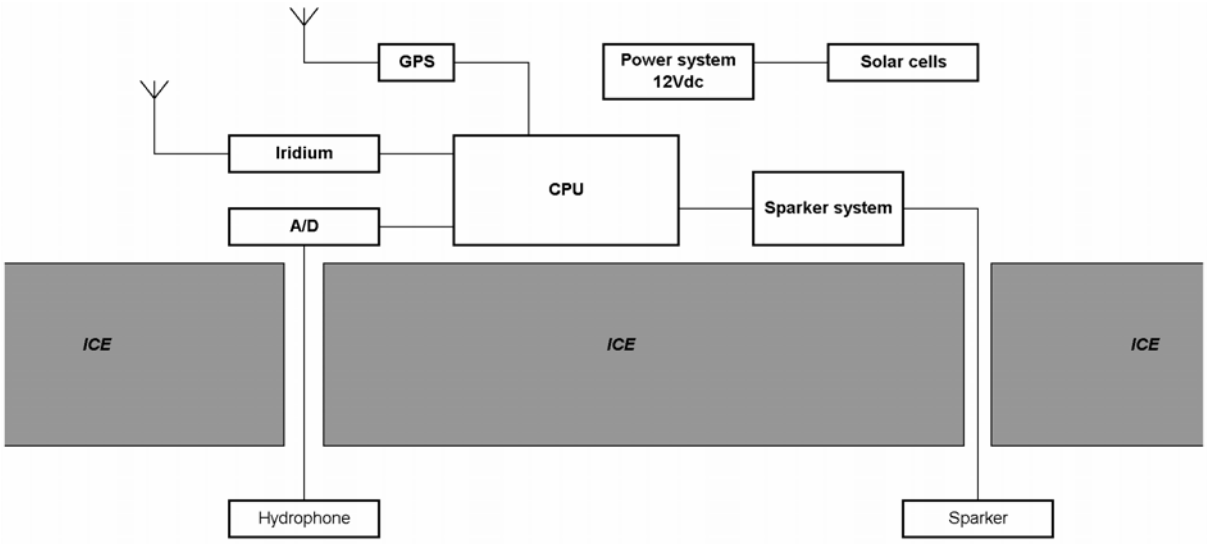
Konstruksjon og -beskrivelse i forbindelse med dette:

I samarbeid med oppdragsgiver velge ut den modul som kan egne seg for konstruksjon, og deretter fremstille først konstruksjonsbeskrivelse og siden selve modulen.

Prosjektet skal videreføres i sommer, og under selve toktet i august/september. Jo mer arbeid man får gjort i vår, dess bedre.

4. ANALYSE

For å danne seg et overblikk over SeisDrift sin virkemåte har jeg laget dette blokkskjemaet som viser den prinsipielle oppbygningen av systemet:



(Det skal altså borres to hull i hull i isen – ett for kilden og ett for hydrofonen.)

En sentral enhet, et CPU-kort styrer det hele. CPU-enheten får posisjonsdata fra GPS og beregner når det er tid for å avfyre neste skudd. Avfyring av et skudd skjer ved at Sparker-enheten genererer akustisk energi fra lysbuen som oppstår ved overslag mellom to elektroder. Sparker-enheten trekker mer energi enn noen annen del i systemet.

Når et skudd er avfyrt lytter A/D-modulen på de reflekterte akustiske signalene fra havbunnen, og sender disse målingene inn til CPU-enheten, hvor de lagres. Det skal gå an å overføre data mellom UiB og SeisDrift over et satellittmodem, Iridium.

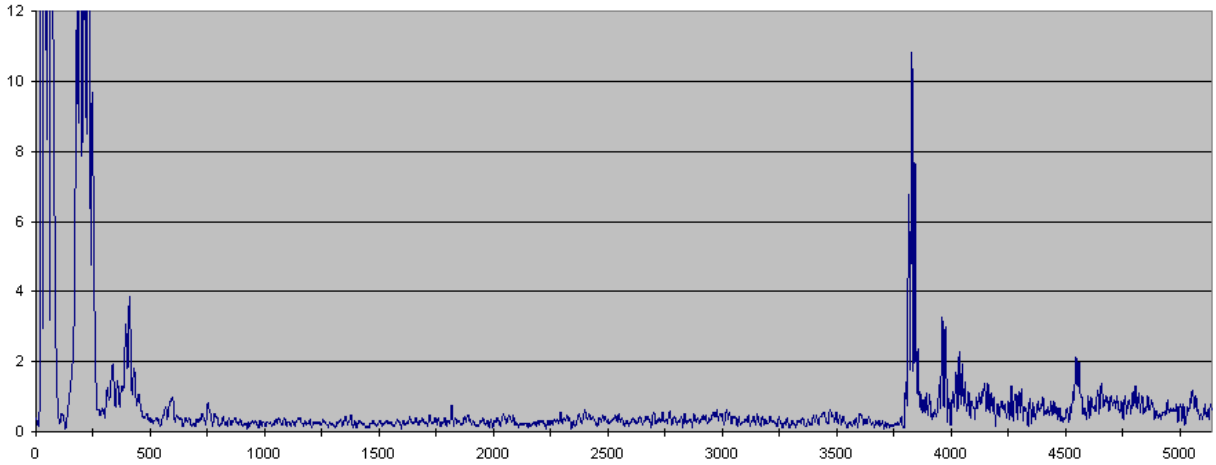
Strømforsyningen skal basere seg på solceller og batteri. Et vanlig "hyttestrømsanlegg" skal benyttes. For å unngå at det evt. legger seg snø på panelet kan det snues opp-ned. Snøen på isflakene har en albedo på over 50%, så hvis panelenes effekt dobles kompenseres tapet med å snu panelet.

4.1 Bunndeteksjon

En aktuell problemstilling er å begrense datamengden som skal overføres over satellitt. Seismiske data er egentlig bare en mengde "lydopptak" nedi sjøen. Opptakene startes idet den seismiske kilden avfyres, og varer en gitt tid, ut ifra hvor dypt vannet er og hvor lang penetrasjon man forventer nedi sedimentene. For å slippe å lagre den stillheten som er før vi mottar den første refleksjonen fra havbunnen kan man detektere hvor i datasettet nyttedataene begynner, og forkaste starten av filen (de første samplingene). En annen mulighet er å bare starte recordingen forsinket – en viss tid etter hvert skudd-tidspunkt. Denne siste metoden er mest aktuell om man har omtrent samme dybde hele tiden, og hvis denne er kjent på forhånd.

Jeg arbeidet mye med å finne en god metode for å detektere havbunnen i seismiske datasett i februar, og kom tilslutt frem til at den beste måten å gjøre dette på var ved å korrelere datasettene med en syntetisk generert puls. Teknikken går ut på at den syntetiske pulsen "glir" over datasettet, mens den i hver eneste posisjon foldes (*digital convolution*) med kurven fra det seismiske signalet. I det den syntetiske pulsen overlapper med en lignende kurveform fra det seismiske signalet får vi store utslag i resultatet fra folde-operasjonen. Så poenget er at den syntetiske pulsen må ha en kurveform som ligner sånn noenlunde på den første pulsen som reflekteres fra havbunnen.

Bildet på neste side viser et eksempel med absoluttverdien av foldeoperasjonene, hvor bunnen detekteres omtrent ved sample nr. 3800:



Vi kan altså kutte vekk alle samples frem til 3800 (stillhet), og ta vare på resten som er nyttedata med refleksjoner fra de seismiske lagene under havbunnen.

Dette temaet har generell interesse utover prosjektet SeisDrift, så jeg laget en egen webside hvor jeg beskriver mine forsøk i detalj. Siden er skrevet på engelsk for å passe bedre med oppdragsgiverens profil, i et internasjonalt miljø, og heter ***Seafloor detection in seismic data***, se http://www2.geo.uib.no/erik/seisdrift/seafloor-det/seafloor-det.html

Disse websidene er lagt ved helt bakerst i denne rapporten.

4.2 Kompresjon av seismiske datafiler

Seismiske data lagres veldig ofte i **`.su`** –filformat. Filene inneholder i tillegg til selve de seismiske dataene (flyttall) også *Headers* som forteller posisjon, tidspunkt, samplerate, og andre aktuelle parametre for datainnsamlingen (i form av integer-verdier, 2 eller 4 bytes).

Slike filer lar seg komprimere, og jeg gjorde tester på alle data vi samlet inn i Vesterålen sommeren 2003. Seismiske datafiler med filstørrelser fra 8 kB til 90 MB (totalt 323 MB) ble pakket i **`.zip`** –filformat, og resultatet var en kompresjon på 45−55 %, gjennomsnittet var 50%.

CPU-kortet som brukes, LogicFlex (mer om CPU-kortet senere), kan kjøre en del enkle DOS-programmer(!), og PKZIP.EXE (fra den tiden DOS regjerte i dataverdenen) fungerte greit i versjon 2.24g. Versjon 2.50 og nyere bruker mer ressurser, og er ikke aktuell på den nevnte CPU-kortet.

Så da finnes det en mulighet til å halvere datastørrelsen ved å komprimere de seismiske datafilene.

Les mer om **`.su`** −filformatet hos oppdragsgiver: http://www2.geo.uib.no/seismic-unix/

4.3 Satellittkommunikasjon

I polare strøk trengs det spesielt utstyr for å etablere pålitelig kommunikasjon med omverdenen. Mulige systemer er Iridium, Orbcomm og Argos. Inmarsat er et system med geostasjonære satellitter, og fungerer derfor bare innenfor ±70° bredde – ikke i polare strøk. Etter å ha undersøkt litt sammen med oppdragsgiver fant vi ut at vi ville gå for Iridium. Det er flere grunner til dette, f.eks. er Iridium mye mer energi-effektiv (i størrelsesorden 10-100 ganger!) enn Orbcomm og Argos med ca 20 $^{Joule}/_{kB}$ for større filer. Argos er enveis kommunikasjon, og mest for små datamengder. Orbcomm har elendig dekning i polare strøk, med de fleste satellittene i baner med 45° inklinasjon. Bare to satellitter har inklinasjon på 70° og gir dekning nær polpunktene. Iridium har ikke disse problemene og er relativt enkelt å ta i bruk. Her er et overslag på overoverføringstiden med Iridium:

Samplerate:	1000 Hz => 1 ms
Samples per skudd:	ca 4 sekund, la oss si 2^{12} = 4096 ms => 4096 samples
Rådata per skudd:	4096 × 24 bits = 98304 bits = 12288 bytes
Rådata per time:	12 × 12288 bytes ≈ 150 kB ($5\ ^{min}/_{skudd}$ <=> 0,32 knop & 50 m)
Tid @ 2400 bps:	(150×10^3 bytes × 8 $^{bits}/_{byte}$) / (2400 bps × 60 $^{sec}/_{min}$) ≈ 8,3 minutt

Hvis det overføres data fra SeisDrift hver time, så viser dette (meget grove) overslaget at overføringen tar i størrelsesorden 8 minutt. Da er det ikke tatt hensyn til kompresjon, og overhead for posisjonsinformasjon osv. Datamengdene vi ønsker å overføre er mye større enn det som er "vanlig" i andre bøyer som samler inn miljødata og overfører via satellitt. En typisk bøye sender gjerne noen få målinger per døgn, og hver melding er gjerne noen få bytes!

Dersom det skulle bli problemer å overføre de relativt store datamengdene som vi ønsker, så ser oppdragsgiver for seg følgende mulige alternative løsning:
SeisDrift kan prosessere de seismiske dataene på egen hånd, og generere f.eks. en liten **`.jpg`** –bildefil som viser den seismiske seksjonen (bilde av sedimentlagene, basert på tolkning av seismiske data). Da kan man altså overføre bare bilder av selve måledataene, slik at man får et visst overblikk og vet at SeisDrift fungerer og er operativ. Selve måledataene overføres ikke, og man er da avhengig av å senere plukke opp bøyen og laste ned de seismiske dataene manuelt, i ettertid. Det er selvfølgelig å foretrekke å laste ned rådata-filene på "vanlig måte", og det blir det vi tar sikte på å få til.

Oppdragsgiver har kjøpt inn et Iridium-modem fra NAL Research Corporation, modell *A3LA-DG*. Modemet er bokstavelig talt en liten "black box" som går på 4,4 volt, kommuniserer over RS-232 med et (utvidet) AT-kommandosett, og som tilkobles en liten ekstern antenne (1,6 GHz). Modemet har en SIM-kort holder, akkurat som på en vanlig GSM-telefon, og her må man ha et Iridium SIM-kort klargjort for datatjenester (det finnes også Iridium SIM-kort som bare kan brukes for tale). Modemet trekker 1 A under sending, 0,18 A i stand-by, og 0 A (!) hvis vi velger å skru det helt av når man ikke overfører data. Bakdelen med det er at da går det ikke an å ringe opp SeisDrift –systemet utenfra. Eventuelt kan man til faste forhåndsbestemte tider skru på modemet i stand-by, slik at SeisDrift kan kontaktes utenfra for endring av konfigurasjon osv.

Det finnes fem måter å overføre data med Iridium-modemet:

- Dial-Up Data
- Direct Internet Connection (eller Direct NIPRNET Connection)
- Short Message Service (SMS)
- Short-burst data (SBD)
- Router-Based Unrestricted Digital Internetworking Connectivity Solution (RUDICS)

SMS er som vi kjenner det fra det vanlige GSM-nettet en liten tekstmelding på maks 160 stk 7-bits tegn, dvs. 140 bytes med data. SBD er også en slags tekstmelding, men størrelsen er her ca 1,9 kB. For å sende data som SBD til Iridium-modemet går det an å sende e-mail til en fast adresse, hvor tittellinjen inneholder telefonnummeret til Iridium-modemet, og legge ved dataene (opptil 1,9 kB) som et filvedlegg. *Direct Internet Connection* gir tilgang til Internet via en Iridium gateway. Man må da implementere en hel protokollstakk for å ta dette i bruk. Evt. kan man bruke *MS Windows dial-up networking*, hvis man bruker en PC. Dette er foreløpig ikke aktuelt på SeisDrift. Dial-Up datatjenesten er enkel og effektiv og gjør at man kan ringe opp et annet modem som er tilkoblet det vanlige telefonnettet (PSTN).

Man kan lese mer om de ulike datatjenestene som støttes av Iridium-modemet på websiden til produsenten:

http://www.nalresearch.com/QuickReference.html

4.4 Tid & Sted

For å tidsstemple dataene, og samtidig bake inn posisjonsinformasjonen trengs det GPS-posisjonering. Skuddene skal også trigges på avstand; 50 m. Enten kan det brukes en vanlig Garmin GPS som sender ut NMEA setninger på RS-232, ellers kan vi dra nytte av at Iridium-modemet har innebygd GPS-mottaker som kan nåes gjennom det utvidete AT-kommandosettet. Frem til nå har vi brukt en egen Garmin GPS 35-HVS.

Med *Matematiske metoder III* friskt i minnet har jeg funnet frem til formler for å regne ut avstanden mellom to posisjoner på jorden. Jorden betraktes som en kuleflate. Som nevnt brukes dette for å trigge skuddene hver 50. meter. Dessuten har jeg også funnet ut hvordan man regner ut sann retning (*bearing*) mellom to punkter på jorden:

$$avst = R \cdot \arccos\left[\sin(latFm)\cdot\sin(latTo) + \cos(\Delta lon)\cdot\cos(latFm)\cdot\cos(latTo)\right]$$

$$retn = \left[360 + \frac{180}{\pi}\cdot\arctan_2(x,y)\right]\%360 \qquad \text{(Grader øst for nord, 0−359)}$$

$$x = \sin(\Delta lon)\cdot\cos(latTo)$$
$$y = \cos(latFm)\cdot\sin(latTo) - \sin(latFm)\cdot\cos(latTo)\cdot\cos(\Delta lon)$$

Vinkler og trigonometriske funksjoner opererer med radianer, og symbolene har følgende betydning:

latFm	Breddegrad for startposisjon	(*i radianer!*)
latTo	Breddegrad for endeposisjon	(*i radianer!*)
Δlon	Forskjellen i lengdegrad	(*i radianer!*)
R	Jordens radius	(≈ 6375 *km i Arktis*)
x & *y*	Hjelpevariabler	

4.5 Seismisk kilde

Oppdragsgiver ville bruke en ”Sparker” som seismisk kilde. Sparkeren er i prinsippet to elektroder som via en elektronisk bryter er forbundet med polene i et høyspennings-kondensatorbatteri. Når avfyringskriterier er oppfylt aktiveres denne bryteren og akustisk energi spres i vannet som et resultat av den lysbuen som oppstår mellom elektrodene. Her er link til en filmsnutt som viser en Sparker-elektrode under avfyring:

http://www2.geo.uib.no/sparker/docs/idac02c.mpeg

Det er ikke så mange produsenter som lager Sparkere og oppdragsgiver hadde mest tro på det engelske firmaet *Applied Acoustic Engineering Ltd*. Allerede tidlig i januar var undertegnede med oppdragsgiver til England for å besøke dette firmaet. Vi fikk sett utstyret deres, og fikk også en demonstrasjon av Sparkeren i drift. I tiden etter besøket vårt var det mye e-mail kommunikasjon mellom oppdragsgiver, meg og *Applied Acoustic*. Og den 03. mars ble det foretatt endelig bestilling på et custom design av modell CSPA4800. Vi ville ha en enhet som opererte direkte fra 12 volt (ikke 230 Vac), og vi hadde også en rekke andre endringer. For eksempel lager de et RS-232 kontroll-interface for oss, noe som ikke er med på en standard enhet. Vi ville ha mest mulig kondensatorkapasitet (dvs. mest mulig energi i skuddet) uten at dette gikk særlig ut over fysisk størrelse og vekt, og fikk øket standardspesifikasjonen fra 3000 Joules til 4800 Joules. Denne energien avfyres med 3500 volt.

Leveranse skulle etter avtalen ha funnet sted i slutten av mai, men *Applied Acoustic* er litt forsinket, og vi venter fortsatt på Sparkeren. Mer informasjon om firmaet og produktene deres finnes her:

http://www.appliedacoustics.com/mainframe/subbottom_profiling/subbottom_profiling.htm

4.6 Strømforsyning

Det skal brukes et solcellesystem av den typen som benyttes i hyttestrømsanlegg for å strømforsyne SeisDrift. Systemet gir en spenning på 12-15 volt. Den mest energikrevende komponenten i systemet er Sparkeren. La oss gjøre et grovt effekt-overslag basert på at denne avfyres med 4800 Joules hvert 5. minutt. (Det kommer til å gå fra minst 4 til over 20 minutt mellom hvert skudd, avhengig av hvor fort isen driver, avfyring skal skje hver 50. meter.)

Akustisk effekt Sparker:	$^{4800J}/_{(5\times60sec)}$ =	16 W

Tilført elektrisk effekt, Sparker:		100 W
CPU-kort, A/D, GPS, Iridium stand-by:	1 A × 15 V =	15 W
Iridium dataoverføring:	1 A × 15 V × $^{8}/_{60}$ =	2 W
Total gjennomsnittlig elektrisk effekt blir:		117 W

Her har jeg ikke tatt hensyn til at Iridium-modemet og CPU-kortet skal kjøres på 5 volt, noe som kan gi en besparelse på ca 4 watt ved bruk av switch-mode step-down spenningsregulator. Dermed må vi ha en gjennomsnittlig elektrisk effekt på ca 115 W. Hvis vi skal bruke et solcellepanel som står opp-ned på et isflak i Arktis, hvor vi har sol hele døgnet (midnattssol om sommeren), så bør dette ha en toppeffekt som er minst 200W.

Denne hjemmesiden beskriver laderegulatoren i et solcellebasert hyttestrømsanlegg:

http://home.no.net/camu/regulato.htm

Og på websiden til firmaet GETEK finner vi litt spesifikasjoner:

www.getek.no
http://www.mamut.com/homepages/Norway/1/18/getek/subdet46.htm

4.7 CPU-kort og A/D-omformer

Både oppdragsgiver og undertegnede har tidligere erfaring med *JK Microsystems* sine *single board computers*, som kjører en begrenset versjon av DOS. Til disse kortene kan det kjøpes tilleggskort som gir (enda flere) DOS-kompatible serieporter og en parallellport. Imidlertid kan det være greit å prøve nytt utstyr, så vi bestemte oss for å forsøke et CPU-kort fra *Persistor Instruments Incorporated*. Disse kan også levere en passende 24-bits A/D-omformer (add-on board) som er laget for dette CPU-kortet. Fra før har oppdragsgiver liggende en 24-bits A/D-omformer som er laget for å koble seg på parallellporten på en PC. Hvis denne kan brukes i SeisDrift, så har vi den muligheten.

Her er linker til produsentene av CPU-kortene og sistnevnte A/D-omformer:

http://www.persistor.com
http://www.jkmicro.com
http://www.symres.com/products/par8ch.htm

5. DESIGN

5.1 Ta i bruk Persistor CPU-kort

Det ble kjøpt inn systemkomponenter fra *Persistor Instruments Incorporated*. Komponentene inkluderte blant annet selve CPU-kortet, modell ”CF2”, *Compact-Flash* kort for datalagring, 24-bits A/D-omformer, utvidelseskort med flere serieporter (UART’er), og tilhørende

utviklingsverktøy (IDE) for Windows-miljø. Denne software-pakken heter *Metrowerks CodeWarrior* – samme som brukes for å utvikle applikasjoner for håndholdte datamaskiner (f.eks. *Palm*).

Etter å ha installert CodeWarrior, kompilert et "*Hello World!*" program på kortet og kommet halvveis gjennom *Getting Started Guide* ble det dessverre klart at Persistor Instruments Inc. har rot i versjonene av manualer og software revisjoner som ligger på Internet. Dette rotet er graverende, og ga et så dårlig inntrykk at det virket for risikabelt å basere SeisDrift på Persistor sine proprietære løsninger. Det ble besluttet å heller benytte et CPU-kort fra *JK Microsystems*, kalt LogicFlex.

Link til Persistor modell CF2:
http://www.persistor.com/pii/products/cf2.html

5.2 CPU-kort fra JK Microsystems

CPU-kortet fra *JK Microsystems*, heter LogicFlex. Dette kortet kan kjøre en del gode gamle DOS-programmer direkte, da det med visse begrensninger er en 386 PC i miniatyr. Oppdragsgiver hadde allerede et slikt CPU-kort liggende, og en 512 MB DiskOnChip –enhet ble kjøpt inn for å kunne lagre måledata på Flash-minne. Minnet dukket opp som egne stasjonsbokstaver i "DOS". Et problem som de ikke nevner på websidene sine er at man ikke kan ha større partisjoner enn 31 MB. Dermed måtte 512 MB deles opp i mange, mange partisjoner som hver fikk sin stasjonsbokstav; C:\, D:\, E:\ og så videre. Vi bruker vel nesten hele alfabetet..!

For å få nok RS-232 linjer og for å kunne kommunisere med A/D-omformeren som har parallellport-tilkobling ble det kjøpt inn et utvidelseskort med fire ekstra serieporter (COM3, COM4, COM5 og COM6), og en parallellport (LPT1). Siden denne parallellporten skulle fungere i *bidirectional* modus og være som en vanlig parallellport på en DOS PC ville vi bruke A/D-omformeren fra *Symmetric Research*, modell PAR4CH. Drivere for Windows, DOS, Linux etc. er tilgjengelig på firmaets hjemmeside www.symres.com. Jeg lastet ned driverfiler med C-rutiner for å kontrollere A/D-konverteren. Planen var å først rekompilere eksemplene som fulgte med driverne, og få tatt i bruk A/D-omformeren på en vanlig DOS test-PC. Det viste seg imidlertid å være svært vanskelig å rekompilere disse driverfilene, og jeg satt lenge og stanget hodet i veggen uten å komme videre. Jeg fikk hjelp fra kompetente folk på HiB uten å komme videre, og til slutt måtte jeg be oppdragsgiver om hjelp. Ikke før vi klarte å få tak i en eldgammel versjon av MS Visual C (ver. 1.52) greide vi å ta i bruk driverne. Endelig kommuniserte jeg med A/D-omformeren!

Neste skritt var å gjøre kjøre de samme testprogrammene på CPU-kortet. Og her oppdaget vi til vår store fortvilelse at parallellporten rett og slett ikke fungerer i BPP-mode! Det gikk an å skrive til de 8 bitene på datalinjene til parallellporten, pinne 2 tom. 9, men det var også alt. En mengde små tredjeparts test-utilities for DOS ble prøvd, men alle krasjet da de forsøkte å kommunisere med parallellporten. Produsentens kommentar: "*...it should work.*"

Mye tid og krefter ble kastet bort på denne løsningen. Nå har vi anskaffet oss et A/D-kort med RS-232 grensesnitt og vil bruke dette i stedet. Kortet er et evalueringskit fra Texas Instruments. Mer info om dette, og om LogicFlex CPU-kortet:

http://focus.ti.com/docs/toolsw/folders/print/ads1254evm.html

http://www.jkmicro.com/products/logicflex.html

5.3 Design av software

Software utvikles i Borland C++ versjon 4.52. Debug-informasjon og kontroll av programmet skjer via *console-porten*, dvs. COM2 som er kablet opp mot Hyperterminal på laptop'en min. Programmet kompileres til en **`.EXE`** –fil på laptop'en og denne overføres til CPU-kortets flash-disk vha. Xmodem –protokollen.

CPU-kortet har mange generelle IO-pinner. En del av disse brukes for å styre en vanlig (dvs. HD-44780 kompatibel) LCD-modul på 4×20 alfanumeriske tegn. Driveren for denne leveres av *JK Microsystems*, og lastes inn i oppstarten, før programmet mitt. Når CPU-kortet booter opp avslutter det med å kjøre filen **`STARTUP.BAT`**. I denne filen har jeg altså lagt inn LCD-driveren og deretter min **`.EXE`** –fil.

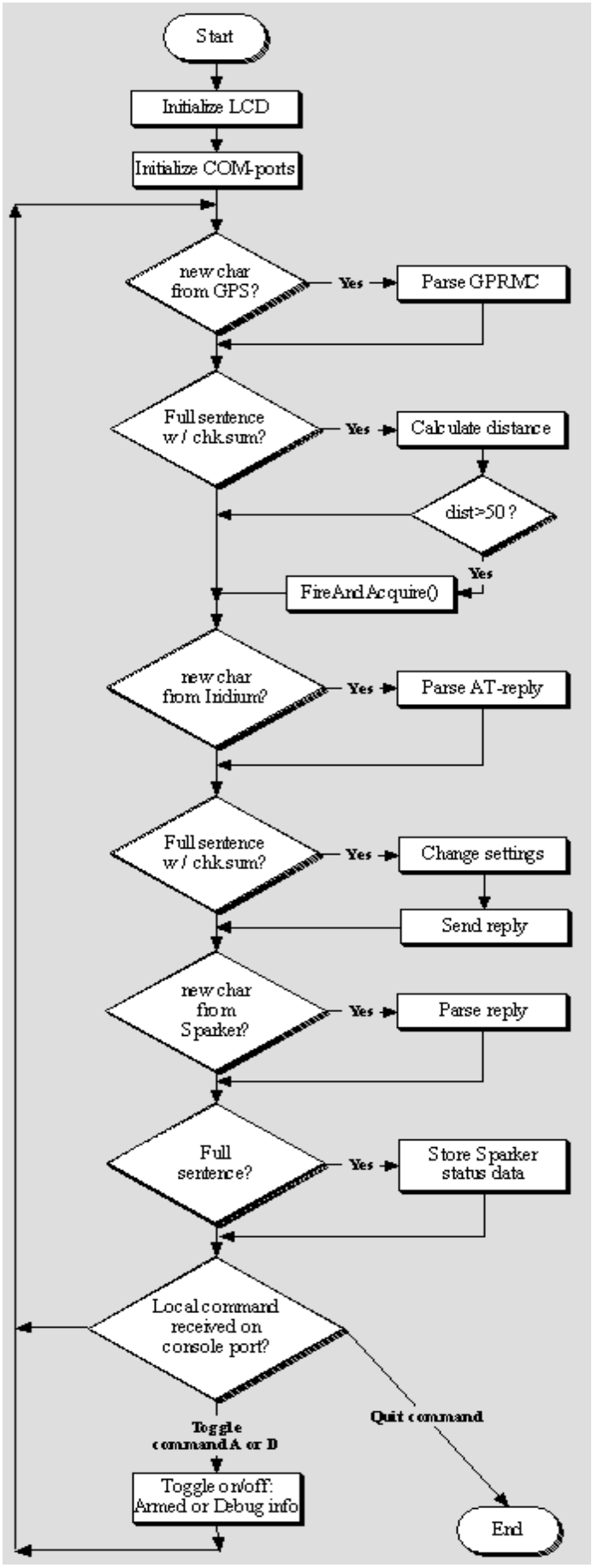
Programmet kommuniserer gjennom flere COM-porter, og er interrupt-basert. En hovedløkke gjennomløpes, og dersom det kommer data inn i bufferne for RS-232 data (interrupt-rutinene trigges i bakgrunnen, og tar seg av dette) så vil programmet reagere på den informasjonen som kommer inn.

Programmet er langt i fra ferdig enda. Status nå, er at interrupt-rutinene virker, alt som har med å motta data fra GPS og trigge på avstand fungerer. Kommunikasjon med A/D-omformeren jobbes det med i disse dager (se neste avsnitt), og kommunikasjonen med Iridium-modemet er ikke på plass.

Informasjon skrives ut til console-porten (COM2) og til LCD. Via console-porten kan man toggle av og på innstillingene for avfyring (*Armed*) og innstillingen for å gi ut debug-informasjon. Man kan også avslutte programmet.

Kildekoden som fortsatt er midt i en utviklingsfase er lagt ved bak i denne rapporten. Versjonen er:

`ver. 1.0.4  (26.05.2005)`


Start
Initialize LCD
Initialize COM-ports
new char from GPS?
Yes
Parse GPRMC
Full sentence w/ chk.sum?
Yes
Calculate distance
dist>50 ?
Yes
FireAndAcquire()
new char from Iridium?
Yes
Parse AT-reply
Full sentence w/ chk.sum?
Yes
Change settings
Send reply
new char from Sparker?
Yes
Parse reply
Full sentence?
Yes
Store Sparker status data
Local command received on console port?
Toggle command A or D
Toggle on/off: Armed or Debug info
Quit command
End

5.4 Reverse engineering

A/D-omformeren vi nå bruker er et evalueringskort fra Texas Instruments. Dette kortet er ment for å kobles til COM-porten på en Windows-PC, og det følger med et demo-program som er fullt av bugs. Kommunikasjonsprotokollen som brukes er i følge dokumentasjonen en proprietær protokoll som kalles CSR-232:

> *"Communication with the host is performed using TI's CSR-232 protocol, which is designed for efficient system control over point-to-point serial links."*

Texas Instruments ville ikke hjelpe til med å skaffe denne protokollen, og på Internet "eksisterer den ikke". Imidlertid viser det seg at CSR er en forkortelse for "*Control and Status Registers*", og dette finnes det noen bruddstykker av informasjon om på Internet (selv om ingen har hørt om Texas sin variant, CSR-232). Så da var det bare å sette i gang å eksperimentere med demo-programmet, og finne ut av det selv.

For å overvåke kommunikasjonen på COM-portene til en Windows-PC finnes det et ypperlig program som heter *Advanced Serial Port Monitor*. Programmet startes før A/D-demo-programmet, og kobler seg til den aktuelle COM-porten, deretter legger det seg "skjult" i bakgrunnen og lar andre programmer få tilgang til COM-porten. Hele tiden ligger *Advanced Serial Port Monitor* i "spy mode", som de kaller det, og snapper opp det som blir overført til og fra COM-porten. Her er screenshots og informasjon om dette programmet:

http://www.aggsoft.com/serial-port-monitor/

Det viste seg at alle meldinger til A/D-kortet var bygget opp på følgende form:

```
Adresse + Register + Data + CRC
```

Adressene var 8 bytes lange, og det var kun tre ulike adresser som ble brukt:

```
Adresse1: #61#00#01#FF#FF#00#00#00 (buss 0110.0001.00)
Adresse2: #60#00#01#FF#FF#00#00#00 (buss 0110.0000.00)
Adresse3: #48#00#01#00#00#00#00#00 (buss 0011.0000.00)
```

Første delen av adressen (de 10 første bits) refererer trolig til en såkalt "buss-adresse", men dette får ingen praktisk betydning for vårt vedkommende.

Etter adressen kommer en enkelt byte; "Register". Dette er også en adresse internt i A/D-kort systemet. "Data" er et felt på enten en eller fire bytes. Tilslutt kommer en 16-bits CRC sjekksum (2 bytes). Total lengde på en melding til A/D-kortet blir derfor enten 12 eller 15 bytes.

A/D-kortet kvitterer de aller fleste meldinger med å gi ut en konstant streng på fem bytes:

```
#F0#00#00#E3#52
```

Nedenfor gjengis det jeg har funnet ut så langt, om kommunikasjonen med A/D-omformeren:

```
Fra PC
Fra AD-kort

=========================================================================

Adresse1 + #00 + #00#00#00#00 + #0B#AB      Bare 0'ere (reset, initialisering?)
(svar)
Adresse1 + #04 + #00#00#AA#AA + #64#B8      Antall samples (32 bits integer)
                                                (potens av 2, fra 8 til 32768)
(svar)
Adresse1 + #10 + #00#00#00#01 + #1F#D0      Kanal nr.:      1 = 0x01
                                                2 = 0x02
                                                3 = 0x04
                                                4 = 0x08
                                            (Scopet på win-programmet viser samme signal på kanalene
                                             1 og fire, selv om bare 1 er tilkoblet!)
(svar)
Adresse1 + #18 + #00#00#00#01 + #1D#FD      ??
(svar)
Adresse2 + #0D + #00 + #8B#56               Averaging: No. of samples averaged.
                                            0x00 = none (=1)
                                            0x10 = 2
                                            0x20 = 4
                                            0x30 = 8
                                            ....
                                            0xF0 = 32768
                                            Eff. samplerate blir dividert med dette tallet!
(svar)
Adresse2 + #0E + #03 + #EE#66               ??
(svar)
Adresse1 + #14 + #45#BB#80#00 + #41#B8      Samplerate før averaging: float (single)
(svar)
Adresse2 + #0F + #03 + #DD#57               Kommando:
                                            Start konvertering...?
"Noe" fra AD-kort: 28 bytes data, verdier ikke avh. av antall samples (en konstant streng?)
#F0#00#00#E3#52#E1#04#02#00#00#00#01#F2#81#E1#04#02#00#00#00#03#D2#C3#E1#00#01#87#20
Adresse3 + #00 + #5F + #D4#98               Kommando:
                                            Gi data..?
header[2] + data[3*no_of_samples] + tail[2] data fra AD-kort.... (3 bytes per sample)
        (8 samples gir 28 bytes, 16 samples gir 52 bytes, og 32 samples gir 100 bytes, etc...)

        header[2] = {0xC8, (samples*3-1)}   // 0xC8 er 200d. Hva betyr denne verdien?
        tail[2] = CRC, 16 bits              // bare gjetning, vet ikke sikkert...

        Hver sample ser ut til å være et 24-bits binærtall på toerkomplement-form.

        I Windows-programmet er man nødt til først å sette samplerate. Dette er trolig ikke
        nødvendig hvis man sender kommandoene ovenfor, der samplerate er inkludert.
```

Informasjonen ovenfor holder for å komme i gang med å kommunisere med A/D-konverteren. Puslespillet er nesten komplett, og de bitene som mangler trengs ikke. Så lenge vi bare vet hva vi skal sende, og hva vi skal forvente å få tilbake, så er det ikke så farlig hva den bakenforliggende meningen med absolutt alle felter måtte være. Jeg vet hvor dataene ligger, og formatet på tallene. SeisDrift CPU-kortet skal bare "herme" etter oppførselen til demo-programmet som fulgte med A/D-kortet.

6. KONKLUSJON

Det eneste som er ordentlig testet ut så langt er det som går på GPS-posisjonsdata og trigging på distanse. Mye viktig testing gjenstår før SeisDrift ferdig til å plasseres på isflaket. For eksempel dataoverføringen over Iridium-modemet.

I forhold til fremdriftsplanen fra forprosjektet så kan man slå fast at ting har tatt lengre tid enn jeg håpet på. Årsakene til det kan være flere:

- Kommunikasjonen med produsenten av Sparkeren var ikke tydelig nok, slik at det dro ut lenge før vi endelig kunne bestille en Sparker. (Og denne er enda ikke levert.)

- I starten gikk det med en del tid på et CPU-kort som vi etter hvert innså at vi ikke torde å bruke, grunnet det inntrykket som produsenten gir med å ha utdaterte websider og rot i versjonene på både software og manualer. Skal man basere seg på proprietære løsninger så må i alle fall produsenten være "til å stole på" for fremtidig support og tilgang på hardware/software.

- Det å rekompilere DOS-drivere som er laget med MS Visual C (fra år 1994) viste seg å bli et enormt hodebry, og det tok lang tid å få det til. På toppen av det hele, så viste det seg å være forgjeves, da printerporten på CPU-kortet rett og slett ikke oppførte seg likt som en vanlig printerport i DOS og dermed ikke kunne brukes. Produsenten mente at printerporten skulle være kompatibel.

Likevel har vi selvfølgelig fått gjort en god del arbeid som måtte gjøres. Selv om skoleprosjektet med HiB nå er over, er SeisDrift-prosjektet bare i startgropen. Et godt grunnlag er lagt for videre utvikling av SeisDrift. Fremover i sommer skal det jobbes for fult med de gjenstående oppgavene. Dette arbeidet fortsetter også på selve ekspedisjonen i august og september parallelt med testing og utprøving, ved siden av "ordinær" seismisk datainnsamling.

Målsetningen har hele tiden vært rund i kantene på dette prosjektet. Jeg har hatt nok av oppgaver å gripe fatt i, og har gjort så mye jeg kan. Slik skulle det være, og slik ble det.

Erik Grindheim
Høgskolen i Bergen, 06.06.2005

VEDLEGG

- Kildekoden for CPU-kortet, slik den ser ut i dag, 11 sider
- Websiden om deteksjon av havbunnen i seismiske data, 4 sider, http://www2.geo.uib.no/erik/seisdrift/seafloor-det/seafloor-det.html

```
/*============================================================================

      S E I S D R I F T   C O M P U T E R   S O F T W A R E

                        H O T R A X   2 0 0 5

                     University of Bergen, Norway
                        Dept. of Earth Science


   Name:
       sd.cpp

   Description:
       Experimental version of SeisDrift computer software


   Rev    Date          By  Description
   ------------------------------------------------------------------
   1.0.4  May 26, 2005  EG  Test version

============================================================================*/

#include <dos.h>
#include <stdio.h>
#include <conio.h>
#include <math.h>
#include <cstring.h>

#define USING_LCD

#define constVERSION    "ver. 1.0.4"
#define constDATE       "26.05.2005"

#define COMPORT1 0x3F8  // GPS (GPRMC-sentence)
/////// COMPORT2 0x2F8  // Console port
#define COMPORT3 0x3E8  // A/D-board
#define COMPORT4 0x2E8  // Sparker system
/////// COMPORT5 0x368  // not used
//#define COMPORT6 0x268    // Iridium modem

#define INTVECT1 0x0C   // Com1, IRQ 4
/////// INTVECT2 0x0B   // Com2, IRQ 3
#define INTVECT3 0x0D   // Com3, IRQ 5
#define INTVECT4 0x0E   // Com4, IRQ 6
/////// INTVECT5 0x76   // Com5, IRQ 14
#define INTVECT6 0x0F   // Com6, IRQ 7

#define LCD CMD 160             // "command-mode" for LCD driver
//#define Deg Char Symbol   223 // Degree-symbol (normally ASCII #248 at a PC)
#define LCD Line1 128           //  =10000000b =addr. 00h at LCD-screen, Line 1
#define LCD Line2 192           //  =11000000b =addr. 40h at LCD-screen, Line 2
#define LCD Line3 148           //  =10010100b =addr. 14h at LCD-screen, Line 3
#define LCD Line4 212           //  =11010100b =addr. 54h at LCD-screen, Line 4
////////////////////////////////////////////////////////////////////////////
void interrupt (*oldport1isr)(...);
void interrupt COMPORT1INT(...);
void interrupt (*oldport3isr)(...);
void interrupt COMPORT3INT(...);
void interrupt (*oldport4isr)(...);
void interrupt COMPORT4INT(...);
void interrupt (*oldport6isr)(...);
void interrupt COMPORT6INT(...);
void FireAndAcquire(unsigned int Duration);
#ifdef USING LCD
void WriteLCD Text(char* DispText, unsigned char RowNo, unsigned char ColNo);
void InitializeLCD();
#endif
void InitializeCOM1();
void InitializeCOM3();
void InitializeCOM4();
//void InitializeCOM6();
void GPRMCparser();
void CleanUp();
////////////////////////////////////////////////////////////////////////////
//const double PI = 3.14159265358979323846264338327950;
const double DEG TO RAD = 0.01745329251994329576923690768488 6;  // pi/180
const double EARTH RADIUS = 6375000.0;  // meters
////////////////////////////////////////////////////////////////////////////
typedef struct
{
```

```
    unsigned char hh, mm, ss, day, month, year;
    double  degNorth, degEast;
} GPRMC type;
////////////////////////////////////////////////////////////////////////////
//unsigned int is 16 bits  ==>  0 to 65535
unsigned int buffer1In=0, buffer3In=0, buffer4In=0, buffer6In=0,
             buffer1Out=0, buffer3Out=0, buffer4Out=0, buffer6Out=0,
             shotCounter=0, samplingTime=8*1024;
unsigned char rxCheckSum, calcCheckSum, strPos=0, trigDist=50, ch, ch3, ch4,
              ch6, armed=1, debugOutputFlag=0, buffer1[1025], buffer3[1025],
              buffer4[1025], buffer6[1025];
double curDist;
GPRMC type prevRMC, newRMC;
#ifdef USING_LCD
FILE    *lcd;   // Stream for LCD data
#endif
////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////
void main(void)
{
    unsigned char c;

#ifdef USING_LCD
    InitializeLCD();
#endif
    InitializeCOM1();
    InitializeCOM3();
    InitializeCOM4();
//  InitializeCOM6();

////////////////////////////////////////////
    printf("\nInterrupt driven program that receive\n");
    printf("GPRMC-data and trig on distance.\nPress ESC to quit \n");
    printf("\n%s\n%s\n", constVERSION, constDATE);
    printf("\nPress D to toggle debug information on/off.\n");
    printf("Press A to toggle the ARMED parameter (Shooting on/off)\n\n");
    //printf("\nPress any key to kontinue...\n\n");
    //getch();

    do
    {
        if (buffer1In != buffer1Out)                    // Serial data waiting
            GPRMCparser();                              // Parse it!

        if ((strPos==53)&&(calcCheckSum==rxCheckSum))   // GPRMC-data ready!
        {
            ////////////////////////////////////////////////////////////////
            // Calculate distance:
            // Accuracy:  <5cm , the typical error is 1cm.
            // There is no reason to calculate with long double precision.
            curDist = EARTH_RADIUS * acos(
                                          cos(DEG_TO_RAD*(newRMC.degEast-prevRMC.degEast))*c
os(DEG_TO_RAD*newRMC.degNorth)*cos(DEG_TO_RAD*prevRMC.degNorth)+sin(DEG_TO_RAD*newRMC.degNorth
)*sin(DEG_TO_RAD*prevRMC.degNorth)
                                          );
            ////////////////////////////////////////////////////////////////
            if(debugOutputFlag)
            {
                printf("Tid: %.2u:%.2u:%.2u,  Dato: %.2u.%.2u.20%.2u\n", newRMC.hh,
                    newRMC.mm, newRMC.ss, newRMC.day, newRMC.month, newRMC.year);
                printf("Position: %i %f   %i %f\n", (int)newRMC.degNorth,
                    60 * (newRMC.degNorth - (double)((int)newRMC.degNorth) ),
                     (int)newRMC.degEast, 60 * (newRMC.degEast - (double)((int)newRMC.degEast)
) );
                printf("Distance: %f\n\n", curDist);
            }
#ifdef USING_LCD
            //Update LCD lines 1 and 2:
            fprintf(lcd, "%c%c%.2u:%.2u:%.2u  %.2u.%.2u.20%.2u", LCD_CMD, LCD_Line1, newRMC.hh
, newRMC.mm, newRMC.ss, newRMC.day, newRMC.month, newRMC.year);
            fprintf(lcd, "%c%c%8.2f/%.2u", LCD_CMD, 9+LCD_Line2, curDist, trigDist);
#endif

            if ( (curDist>trigDist) && armed )  //curDist=50.014, trigDist=50  ==>  TRIG!
                FireAndAcquire(samplingTime);   // Shoot!

            strPos++;           // =54  FINISHED! Start looking for new GPRMC...

        } //if ((strPos==53)&&(calcCheckSum==rxCheckSum))

/*
```

```
        if (buffer3In != buffer3Out)                    // Serial data waiting
        {
            ch3 = buffer3[buffer3Out];
            buffer3Out++;
            //if (buffer3Out == 1024) {buffer3Out = 0;}
            buffer3Out &= 0x03FF;   //Rollover:  1024=0
            printf("\nMottok dette tegnet: %c paa COM3\n\n",ch3);
        }
*/

        if (buffer4In != buffer4Out)                    // Serial data waiting
        {
            ch4 = buffer4[buffer4Out];
            buffer4Out++;
            //if (buffer4Out == 1024) {buffer4Out = 0;}
            buffer4Out &= 0x03FF;   //Rollover:  1024=0
            printf("\nMottok dette tegnet: %c paa COM4\n\n",ch4);
        }


        if (buffer6In != buffer6Out)                    // Serial data waiting
        {
            ch6 = buffer6[buffer6Out];
            buffer6Out++;
            //if (buffer6Out == 1024) {buffer6Out = 0;}
            buffer6Out &= 0x03FF;   //Rollover:  1024=0
            printf("\nMottok dette tegnet: %c paa COM6\n\n",ch6);
        }


        if (kbhit())
        {
            c = getch();
            outportb(COMPORT1, c);
            //Test:
  //            outportb(COMPORT3, '3');
  //            outportb(COMPORT3, c);
            outportb(COMPORT4, '4');
            outportb(COMPORT4, c);
            //outportb(COMPORT6, '6');
            //outportb(COMPORT6, c);
            ////////////////////
            switch (toupper(c))
            {
                case 'D' :  debugOutputFlag = !debugOutputFlag;
                            printf("Debugging mode: %u\n\n", debugOutputFlag);
                            break;
                case 'A' :  armed = !armed;
                            printf("Armed or Unarmed: %u\n\n", armed);
                            break;
            }
        }//if(kbhit())

    } while (c !=27);

    CleanUp();
}//main
////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////
void FireAndAcquire(unsigned int Duration)
{
//  unsigned int Total, Remain, NextPt, Newpts;

    //Skrive til LCD, be AD-konverteren om data, og lagre til fil.
    prevRMC = newRMC;
    shotCounter++;
    if (debugOutputFlag)
        printf("TRIGGED!  Shot no. <%05u>\n\n", shotCounter);
#ifdef USING LCD
    //Update LCD lines 3 and 4:
    fprintf(lcd, "%c%c %.2u:%.2u:%.2u", LCD_CMD,11+LCD_Line3,prevRMC.hh,prevRMC.mm,prevRMC.ss)
;
    fprintf(lcd, "%c%c<%05u", LCD_CMD, LCD_Line4, shotCounter);
#endif
    // IO-pin FIRE enabled...:
    // ...
//---------------------------   C O L L E C T   D A T A   ------------------------------
//Send streng 1:
    // Test:
    char txStr[15] = {'a','b','c','d','e','f','g','h','i','j','k','l','m','n','o'};
    //char cntr;
```

```
    for(char cntr=0; cntr<15; cntr++)
    {
        outportb(COMPORT3, txStr[cntr]);
    }

//Motta svar 1:
/*  if (buffer3In != buffer3Out)                    // Serial data waiting
    {
        ch3 = buffer3[buffer3Out];
        buffer3Out++;
        //if (buffer3Out == 1024) {buffer3Out = 0;}
        buffer3Out &= 0x03FF;   //Rollover:  1024=0
        printf("\nMottok dette tegnet: %c paa COM3\n\n",ch3);
    }  */


//------------------------   E n d   o f   d a t a   c o l l e c t i o n  -----------------
}
////////////////////////////////////////////////////////////////////////////
#ifdef USING LCD
void WriteLCD_Text(char* DispText, unsigned char RowNo, unsigned char ColNo)
{
    unsigned char line;
    switch (RowNo)
    {
        case 2 :    line=LCD_Line2;
                    break;
        case 3 :    line=LCD_Line3;
                    break;
        case 4 :    line=LCD_Line4;
                    break;
        default :   line=LCD_Line1;
                    break;
    }
    fprintf(lcd, "%c%c", LCD_CMD, line+ColNo-1);    // Locate the cursor...
    fwrite(DispText, strlen(DispText), 1, lcd);     // ..and write the string
}
#endif
////////////////////////////////////////////////////////////////////////////
#ifdef USING LCD
void InitializeLCD()
{
    lcd=stdprn;
    setbuf(lcd, NULL);                          /* disable buffering */
    // LCD setup is described at page 7 (of 16) in this document:
    //                                              http://www.grindheim.net/lcd1.pdf
    fprintf(lcd,"%c%c",LCD_CMD,0x28);   // 4-bit interface, 2 line mode
    fprintf(lcd, "%c%c%c%c", LCD_CMD, 0xA3, LCD_CMD, 0xA2 );  // For the driver!
    fprintf(lcd,"%c%c",LCD_CMD,0x08);   // DispCtrl: Disp, Cursor, Blink    OFF
    //fprintf(lcd,"%c%c",LCD_CMD,0x0C); // DispCtrl: Disp ON, no cursor or blink
    //fprintf(lcd,"%c%c",LCD_CMD,0x0F); // DispCtrl: Disp, Cursor, Blink    ON
    fprintf(lcd,"%c%c",LCD_CMD,0x06);   // Entry mode: Cursor shift, Increase
    fprintf(lcd,"%c%c",LCD_CMD,0x01);   // Clear & Home display

    WriteLCD_Text("SEISDRIFT COMPUTER", 1, 2 );
    WriteLCD_Text("==================", 2, 2 );
    WriteLCD_Text(constVERSION,         3, 6 );
    fprintf(lcd,"%c%c",LCD_CMD,0x0C);   // DispCtrl: Disp ON, no cursor or blink
    delay(1000);
    WriteLCD_Text(".         ", 4, 1);
    WriteLCD_Text("..        ", 4, 1);
    WriteLCD_Text("...       ", 4, 1);
    WriteLCD_Text("....      ", 4, 1);
    WriteLCD_Text(".....     ", 4, 1);
    WriteLCD_Text("......    ", 4, 1);
    WriteLCD_Text(".......   ", 4, 1);
    WriteLCD_Text("........  ", 4, 1);
    WriteLCD_Text("......... ", 4, 1);
    WriteLCD_Text("..........", 4, 1);
    WriteLCD_Text("...........", 4, 1);
    WriteLCD_Text("............", 4, 1);
    WriteLCD_Text("............", 4, 1);
    WriteLCD_Text("..............", 4, 1);
    WriteLCD_Text("...............", 4, 1);
    WriteLCD_Text("................", 4, 1);
    WriteLCD_Text("................", 4, 1);
    WriteLCD_Text("................", 4, 1);
    WriteLCD_Text("................", 4, 1);
    WriteLCD_Text("                ", 4, 1);
    // Scroll away! (to the right)
```

```
    for (unsigned char aa=20; aa>0; aa--)
    {
        WriteLCD_Text(" ", 1, aa);
        WriteLCD_Text(" ", 2, aa);
        if (aa==13)
            WriteLCD_Text("  ", 3, 12);
        else
            WriteLCD_Text(" ", 3, aa);
        WriteLCD_Text(" ", 4, aa);
        fprintf(lcd, "%c%c",LCD_CMD,0x1F);  // DispCtrl: Scroll one to the right
        delay(10);
    }//for
    fprintf(lcd,"%c%c",LCD_CMD,0x01);   // Clear & Home display
    fprintf(lcd,"%c%c",LCD_CMD,0x08);   // DispCtrl: Disp, Cursor, Blink    OFF
    delay(500);

    WriteLCD_Text("University of Bergen", 1, 1);
    WriteLCD_Text(constDATE, 2, 6);
    WriteLCD_Text("Department of", 3, 1);
    WriteLCD_Text("Earth Science", 4, 8);
    fprintf(lcd,"%c%c",LCD_CMD,0x0C);   // DispCtrl: Disp ON, no cursor or blink
    delay(2000);

    //Writes the "normal operation" display template:
    //           ############################
    //           ####00:00:00  xx.xx.20xx####
    //           ####Distance:   xx.xx/xx####
    //           ####Last shot:  00:00:00####
    //           ####<00000>   619MB free####
    //           ############################
    fprintf(lcd,"%c%c",LCD_CMD,0x08);   // DispCtrl: Disp, Cursor, Blink    OFF
    WriteLCD_Text(  "00:00:00  xx.xx.20xx", 1, 1);
    WriteLCD_Text(  "Distance:   xx.xx/xx", 2, 1);
    WriteLCD_Text(  "Last shot:  00:00:00", 3, 1);
    WriteLCD_Text(  "<00000>   xxxMB free", 4, 1);
    delay(500);
    fprintf(lcd,"%c%c",LCD_CMD,0x0C);   // DispCtrl: Disp ON, no cursor or blink
}
#endif
////////////////////////////////////////////////////////////////////////////
void InitializeCOM1()
{
    outportb(COMPORT1 + 1 , 0);     // Turn off interrupts - Port1
    oldport1isr = getvect(INTVECT1);    // Save old Interrupt Vector for later recovery
    setvect(INTVECT1, COMPORT1INT);

    outportb(COMPORT1 + 3 , 0x80);  // SET DLAB ON
    outportb(COMPORT1 + 0 , 0x18);  // Set Baud rate - Divisor Latch Low Byte
                        //  0x03 =  38,400 BPS
                        //  0x01 = 115,200 BPS
                        //  0x02 =  57,600 BPS
                        //  0x06 =  19,200 BPS
                        //  0x0C =   9,600 BPS
                        //  0x18 =   4,800 BPS  <--- This one!
                        //  0x30 =   2,400 BPS
    outportb(COMPORT1 + 1 , 0x00);  // Set Baud rate - Divisor Latch High Byte
    outportb(COMPORT1 + 3 , 0x03);  // 8 Bits, No Parity, 1 Stop Bit
    outportb(COMPORT1 + 2 , 0xC7);  // FIFO Control Register
    outportb(COMPORT1 + 4 , 0x0B);  // Turn on DTR, RTS, and OUT2

    // Set Programmable Interrupt Controller:
    outportb(0x21,(inportb(0x21) & 0xEF));      // COM1 (IRQ4) - 0xEF
                                                // COM2 (IRQ3) - 0xF7
                                                // COM3 (IRQ5) - 0xDF
                                                // COM4 (IRQ6) - 0xBF
    outportb(COMPORT1 + 1 , 0x01);              // Interrupt when data received
//  enable();
}
////////////////////////////////////////////////////////////////////////////

//void interrupt (*oldport1isr)(...);

////////////////////////////////////////////////////////////////////////////
void interrupt COMPORT1INT(...)     // Interrupt Service Routine (ISR) for COM1
{
// disable();
 int c;
 do {
        c = inportb(COMPORT1 + 5);  //Read the Line Status Register
        if (c & 1)                  //Bit 0: Data Ready?
        {
            buffer1[buffer1In] = inportb(COMPORT1);
```

```
            buffer1In++;
            //if (bufferIn == 1024) {bufferIn = 0;}
            buffer1In &= 0x03FF;    //Rollover:  1024=0
        }
    }while (c & 1);     //Repeat loop if there were data..
 outportb(0x20,0x20);   //Peripheral Interrupt Controller 1: Enable Interrupts
// enable();
}
////////////////////////////////////////////////////////////////////////////
void InitializeCOM3()
{
    outportb(COMPORT3 + 1 , 0);         // Turn off interrupts - Port3
    oldport3isr = getvect(INTVECT3);    // Save old Interrupt Vector for later recovery
    setvect(INTVECT3, COMPORT3INT);

    outportb(COMPORT3 + 3 , 0x80);  // SET DLAB ON
    outportb(COMPORT3 + 0 , 0x18);  // Set Baud rate - Divisor Latch Low Byte
                        //  0x03 =  38,400 BPS
                        //  0x01 = 115,200 BPS
                        //  0x02 =  57,600 BPS
                        //  0x06 =  19,200 BPS
                        //  0x0C =   9,600 BPS
                        //  0x18 =   4,800 BPS  <--- This one!
                        //  0x30 =   2,400 BPS
    outportb(COMPORT3 + 1 , 0x00);  // Set Baud rate - Divisor Latch High Byte
    outportb(COMPORT3 + 3 , 0x03);  // 8 Bits, No Parity, 1 Stop Bit
    outportb(COMPORT3 + 2 , 0xC7);  // FIFO Control Register
    outportb(COMPORT3 + 4 , 0x0B);  // Turn on DTR, RTS, and OUT2

    // Set Programmable Interrupt Controller: (enable IRQ)
    outportb(0x21,(inportb(0x21) & 0xDF));      // COM1 (IRQ4) - 0xEF
                                                // COM2 (IRQ3) - 0xF7
                                                // COM3 (IRQ5) - 0xDF
                                                // COM4 (IRQ6) - 0xBF
    outportb(COMPORT3 + 1 , 0x01);              // Interrupt when data received
//  enable();  //Denne må være der for at Com3 skal gi interrupt!
}
////////////////////////////////////////////////////////////////////////////

//void interrupt (*oldport3isr)(...);

////////////////////////////////////////////////////////////////////////////
void interrupt COMPORT3INT(...)     // Interrupt Service Routine (ISR) for COM3
{
// disable();
 int c;

// printf("\nInt.! (COM3)\n");

 do {
        c = inportb(COMPORT3 + 5);  //Read the Line Status Register
        if (c & 1)                  //Bit 0: Data Ready?
//      if (c & 0x1F)               //Bit 0: Any interrupts?
        {
            buffer3[buffer3In] = inportb(COMPORT3);
            buffer3In++;
            //if (buffer3In == 1024) {buffer3In = 0;}
            buffer3In &= 0x03FF;    //Rollover:  1024=0
        }
//  }while (c & 0x1F);      //Repeat loop if there were data..
    }while (c & 1);     //Repeat loop if there were data..

// printf("\nInt. slutt\n");

 outportb(0x20,0x20);   //Peripheral Interrupt Controller 1: Enable Interrupts (eller: End of
Interrupt..?)
// enable();
}
////////////////////////////////////////////////////////////////////////////
void InitializeCOM4()
{
    outportb(COMPORT4 + 1 , 0);         // Turn off interrupts - Port4
    oldport4isr = getvect(INTVECT4);    // Save old Interrupt Vector for later recovery
    setvect(INTVECT4, COMPORT4INT);

    outportb(COMPORT4 + 3 , 0x80);  // SET DLAB ON
    outportb(COMPORT4 + 0 , 0x18);  // Set Baud rate - Divisor Latch Low Byte
                        //  0x03 =  38,400 BPS
                        //  0x01 = 115,200 BPS
                        //  0x02 =  57,600 BPS
                        //  0x06 =  19,200 BPS
                        //  0x0C =   9,600 BPS
```

```
                        //  0x18 =   4,800 BPS  <--- This one!
                        //  0x30 =   2,400 BPS
    outportb(COMPORT4 + 1 , 0x00);  // Set Baud rate - Divisor Latch High Byte
    outportb(COMPORT4 + 3 , 0x03);  // 8 Bits, No Parity, 1 Stop Bit
    outportb(COMPORT4 + 2 , 0xC7);  // FIFO Control Register
    outportb(COMPORT4 + 4 , 0x0B);  // Turn on DTR, RTS, and OUT2

    // Set Programmable Interrupt Controller: (enable IRQ)
    outportb(0x21,(inportb(0x21) & 0xBF));      // COM1 (IRQ4) - 0xEF
                                                // COM2 (IRQ3) - 0xF7
                                                // COM3 (IRQ5) - 0xDF
                                                // COM4 (IRQ6) - 0xBF
    outportb(COMPORT4 + 1 , 0x01);              // Interrupt when data received
}
////////////////////////////////////////////////////////////////////////////

//void interrupt (*oldport4isr)(...);

////////////////////////////////////////////////////////////////////////////
void interrupt COMPORT4INT(...)     // Interrupt Service Routine (ISR) for COM4
{
// disable();
 int c;

// printf("\nInt.! (COM4)\n");

 do {
        c = inportb(COMPORT4 + 5);  //Read the Line Status Register
        if (c & 1)                  //Bit 0: Data Ready?
//      if (c & 0x1F)               //Bit 0: Any interrupts?
        {
            buffer4[buffer4In] = inportb(COMPORT4);
            buffer4In++;
            //if (buffer4In == 1024) {buffer4In = 0;}
            buffer4In &= 0x03FF;    //Rollover:  1024=0
        }
//  }while (c & 0x1F);      //Repeat loop if there were data..
    }while (c & 1);     //Repeat loop if there were data..

// printf("\nInt. slutt\n");

 outportb(0x20,0x20);   //Peripheral Interrupt Controller 1: Enable Interrupts (eller: End of
Interrupt..?)
// enable();
}
////////////////////////////////////////////////////////////////////////////
/*void InitializeCOM6()
{
    outportb(COMPORT6 + 1 , 0);         // Turn off interrupts - Port6
    oldport6isr = getvect(INTVECT6);    // Save old Interrupt Vector for later recovery
    setvect(INTVECT6, COMPORT6INT);

    outportb(COMPORT6 + 3 , 0x80);  // SET DLAB ON
    outportb(COMPORT6 + 0 , 0x18);  // Set Baud rate - Divisor Latch Low Byte
                        //  0x03 =  38,400 BPS
                        //  0x01 = 115,200 BPS
                        //  0x02 =  57,600 BPS
                        //  0x06 =  19,200 BPS
                        //  0x0C =   9,600 BPS
                        //  0x18 =   4,800 BPS  <--- This one!
                        //  0x30 =   2,400 BPS
    outportb(COMPORT6 + 1 , 0x00);  // Set Baud rate - Divisor Latch High Byte
    outportb(COMPORT6 + 3 , 0x03);  // 8 Bits, No Parity, 1 Stop Bit
    outportb(COMPORT6 + 2 , 0xC7);  // FIFO Control Register
    outportb(COMPORT6 + 4 , 0x0B);  // Turn on DTR, RTS, and OUT2

    // Set Programmable Interrupt Controller: (enable IRQ)
    outportb(0x21,(inportb(0x21) & 0x7F));      // COM1 (IRQ4) - 0xEF
                                                // COM2 (IRQ3) - 0xF7
                                                // COM3 (IRQ5) - 0xDF
                                                // COM4 (IRQ6) - 0xBF
    outportb(COMPORT6 + 1 , 0x01);              // Interrupt when data received
//  enable();  //Denne må være der for at Com6 skal gi interrupt!
}
*/
////////////////////////////////////////////////////////////////////////////

//void interrupt (*oldport6isr)(...);

////////////////////////////////////////////////////////////////////////////
/*void interrupt COMPORT6INT(...)       // Interrupt Service Routine (ISR) for COM6
{
```

```
// disable();
 int c;

// printf("\nInt.! (COM6)\n");

 do {
        c = inportb(COMPORT6 + 5);  //Read the Line Status Register
        if (c & 1)                  //Bit 0: Data Ready?
//      if (c & 0x1F)               //Bit 0: Any interrupts?
        {
            buffer6[buffer6In] = inportb(COMPORT6);
            buffer6In++;
            //if (buffer6In == 1024) {buffer6In = 0;}
            buffer6In &= 0x03FF;    //Rollover:  1024=0
        }
//  }while (c & 0x1F);      //Repeat loop if there were data..
    }while (c & 1);     //Repeat loop if there were data..

// printf("\nInt. slutt\n");

 outportb(0x20,0x20);   //Peripheral Interrupt Controller 1: Enable Interrupts (eller: End of
Interrupt..?)
// enable();
}
*/
////////////////////////////////////////////////////////////////////////////
void GPRMCparser()
{
            ch = buffer1[buffer1Out];
            buffer1Out++;
            //if (bufferOut == 1024) {bufferOut = 0;}
            buffer1Out &= 0x03FF;   //Rollover:  1024=0
            switch (strPos)
            {
                case 1 :    if(ch=='G')
                                strPos++;       // =2
                            else strPos=0;      // =0 (reset)
                            break;              //
                case 2 :    if(ch=='P')
                                strPos++;       // =3
                            else strPos=0;      // =0 (reset)
                            break;              //
                case 3 :    if(ch=='R')
                                strPos++;       // =4
                            else strPos=0;      // =0 (reset)
                            break;              //
                case 4 :    if(ch=='M')
                                strPos++;       // =5
                            else strPos=0;      // =0 (reset)
                            break;              //
                case 5 :    if(ch=='C')
                                strPos++;       // =6,    "$GPRMC" received
                            else strPos=0;      // =0 (reset)
                            break;              //
                case 6 :    if(ch==',')
                                strPos++;       // =7,    "$GPRMC," received
                            else strPos=0;      // =0 (reset)
                            break;              //
                ////////////////////////////////////////////////////////////
                case 7 :    strPos++;           // =8,    "$GPRMC," received
                            newRMC.hh = 10*(ch&0x0F);
                            break;              //
                case 8 :    strPos++;           // =9,      HH hours received
                            newRMC.hh += (ch&0x0F);
                            break;              //
                case 9 :    strPos++;           // =10,     Mx minutes received
                            newRMC.mm = 10*(ch&0x0F);
                            break;              //
                case 10 :   strPos++;           // =11,     MM minutes received
                            newRMC.mm += (ch&0x0F);
                            break;              //
                case 11 :   strPos++;           // =12,     Sx minutes received
                            newRMC.ss = 10*(ch&0x0F);
                            break;              //
                case 12 :   strPos++;           // =13,     SS minutes received
                            newRMC.ss += (ch&0x0F);
                            break;              //
                case 13 :   if(ch==',')         // Wait for comma...
                                strPos++;       // ... and move on (=14)
                            break;              // "$GPRMC,HHMMSSxxx,"
                ////////////////////////////////////////////////////////////
                case 14 :   if(ch=='A')         // VALIDITY:  A:ok, V:invalid
```

```
                strPos++;       // =15,    "$GPRMC,HHMMSS,A"
            else strPos=0;      // =0 (reset)
            break;              //
case 15 :   if(ch==',')
                strPos++;       // =16,    "$GPRMC,HHMMSS,A,"
            else strPos=0;      // =0 (reset)
            break;              //
////////////////////////////////////////////////////////////
case 16 :   strPos++;           // =17,     Lx degrees latitude
            newRMC.degNorth = 10*(ch&0x0F);
            break;              //
case 17 :   strPos++;           // =18,     LL degrees latitude
            newRMC.degNorth += (ch&0x0F); //
            break;              //
case 18 :   strPos++;           // =19,     Mx minutes latitude
            newRMC.degNorth += ((double)(ch&0x0F))/6.0;
            break;              //
case 19 :   strPos++;           // =20,     MM minutes latitude
            newRMC.degNorth += ((double)(ch&0x0F))/60.0;
            break;              //
case 20 :   if(ch=='.')
                strPos++;       // =21,  "$GPRMC,HHMMSS,A,DDMM."
            else strPos=0;      // =0 (reset)
            break;              //
case 21 :   strPos++;           // =22,     MM.m minutes lat.
            newRMC.degNorth += ((double)(ch&0x0F))/600.0;
            break;              //
case 22 :   strPos++;           // =23,     MM.mm minutes lat.
            newRMC.degNorth += ((double)(ch&0x0F))/6000.0;
            break;              //
case 23 :   if(ch!=',')
            {
                strPos++;           // =24,     MM.mmm minutes
                newRMC.degNorth += ((double)(ch&0x0F))/60000.0;
            }
            else
                strPos = 27;    // comma received ==> move on
            break;              //
case 24 :   if(ch!=',')
            {
                strPos++;           // =25,     MM.mmmm minutes
                newRMC.degNorth += ((double)(ch&0x0F))/600000.0;
            }
            else
                strPos = 27;    // comma received ==> move on
            break;              //
case 25 :   if(ch!=',')
            {
                strPos++;           // =26,     MM.mmmmm minutes
                newRMC.degNorth +=((double)(ch&0x0F))/6000000.0;
            }
            else
                strPos = 27;    // comma received ==> move on
            break;              //
case 26 :   if(ch==',')         // Wait for comma...
                strPos++;       // ... and move on (=27)
            break;              //
case 27 :   strPos++;
            if(ch=='S')
                newRMC.degNorth *= -1;
            break;
case 28 :   if(ch==',')
                strPos++;       // =29, next field is longitude
            else strPos=0;      // =0 (reset)
            break;              //
////////////////////////////////////////////////////////////
case 29 :   strPos++;           // =30,     Lxx degrees long
            newRMC.degEast = 100*(ch&0x0F);
            break;              //
case 30 :   strPos++;           // =31,     Lx degrees longitude
            newRMC.degEast += 10*(ch&0x0F);
            break;              //
case 31 :   strPos++;           // =32,     LL degrees longitude
            newRMC.degEast += (ch&0x0F);
            break;              //
case 32 :   strPos++;           // =33,     Mx minutes longitude
            newRMC.degEast += ((double)(ch&0x0F))/6.0;
            break;              //
case 33 :   strPos++;           // =34,     MM minutes longitude
            newRMC.degEast += ((double)(ch&0x0F))/60.0;
            break;              //
case 34 :   if(ch=='.')
```

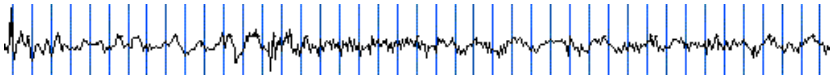
```
            {
                strPos++;       // =35,  "$GPRMC,HHMMSS,A,DDMM.MMxxx,N,DDDMM."
            }                   //
            else strPos=0;      // =0 (reset)
            break;              //
case 35 :   strPos++;           // =36,     MM.m minutes lon.
            newRMC.degEast += ((double)(ch&0x0F))/600.0;
            break;              //
case 36 :   strPos++;           // =37,     MM.mm minutes lon.
            newRMC.degEast += ((double)(ch&0x0F))/6000.0;
            break;              //
case 37 :   if(ch!=',')
            {
                strPos++;           // =38,     MM.mmm minutes
                newRMC.degEast += ((double)(ch&0x0F))/60000.0;
            }
            else
                strPos = 41;    // comma received ==> move on
            break;              //
case 38 :   if(ch!=',')
            {
                strPos++;           // =39,     MM.mmmm minutes
                newRMC.degEast += ((double)(ch&0x0F))/600000.0;
            }
            else
                strPos = 41;    // comma received ==> move on
            break;              //
case 39 :   if(ch!=',')
            {
                strPos++;           // =40,     MM.mmmmm minutes
                newRMC.degEast += ((double)(ch&0x0F))/6000000.0;
            }
            else
                strPos = 41;    // comma received ==> move on
            break;              //
case 40 :   if(ch==',')         // Wait for comma...
                strPos++;       // ... and move on (=41)
            break;              //
case 41 :   strPos++;           // =42
            if(ch=='W')
                newRMC.degEast *= -1;
            break;              //
////////////////////////////////////////////////////////////
case 42 :                   // First comma after lon. hemisphere
case 43 :                   // Second comma after lon. hemisph.
case 44 :   if(ch==',')     // Third comma after lon. hemisphere
                strPos++;   //
            break;          //
////////////////////////////////////////////////////////////
case 45 :   strPos++;           // =46
            newRMC.day = 10*(ch&0x0F);
            break;              //
case 46 :   strPos++;           // =47
            newRMC.day += (ch&0x0F);//      DD date received
            break;              //
case 47 :   strPos++;           // =48
            newRMC.month = 10*(ch&0x0F);
            break;              //
case 48 :   strPos++;           // =49
            newRMC.month += (ch&0x0F);//    DDMM date received
            break;              //
case 49 :   strPos++;           // =50
            newRMC.year = 10*(ch&0x0F);
            break;              //
case 50 :   strPos++;           // =51
            newRMC.year += (ch&0x0F); //    DDMMYY date received
            break;              //
////////////////////////////////////////////////////////////
case 51 :   if(ch=='*')         // Wait for asterisk...
                strPos++;       // ... and move on (=52)
            break;              // Next: CheckSum
////////////////////////////////////////////////////////////
case 52 :   rxCheckSum = (ch>64) ? ((0x07&ch)+9) : (0x0F&ch);
            rxCheckSum <<= 4;
            strPos++;           // =53
            break;              //
case 53 :   rxCheckSum += (ch>64) ? ((0x07&ch)+9) : (0x0F&ch);
            break;
////////////////////////////////////////////////////////////
default :   if(ch=='$')
            {
                strPos = 1;
```

```
                                //calcCheckSum=0x00; // Starting after '$'
                                calcCheckSum=0x67;  // Starting after '$GPRMC,'
                            }
                            break;              //
            }//switch
            if ((7<strPos)&&(strPos<52))        // Between '$GPRMC,' and '*'
                calcCheckSum ^= ch;             // Calculate checksum
}
////////////////////////////////////////////////////////////////////////////
void CleanUp()
{
    outportb(COMPORT1 + 1 , 0);             // Turn off interrupts - COM1
    outportb(0x21,(inportb(0x21) | 0x10));  // MASK IRQ4 using PIC
    outportb(COMPORT3 + 1 , 0);             // Turn off interrupts - COM3
    outportb(0x21,(inportb(0x21) | 0x20));  // MASK IRQ5 using PIC

    outportb(COMPORT4 + 1 , 0);             // Turn off interrupts - COM4
    outportb(0x21,(inportb(0x21) | 0x40));  // MASK IRQ6 using PIC
//  outportb(COMPORT6 + 1 , 0);             // Turn off interrupts - COM6
//  outportb(0x21,(inportb(0x21) | 0x80));  // MASK IRQ7 using PIC
                                            // COM1 (IRQ4) - 0x10
                                            // COM2 (IRQ3) - 0x08
                                            // COM3 (IRQ5) - 0x20
                                            // COM4 (IRQ6) - 0x40
                                            // COM6 (IRQ7) - 0x80
    setvect(INTVECT1, oldport1isr);         // Restore old interrupt vector
    setvect(INTVECT3, oldport3isr);         // Restore old interrupt vector
    setvect(INTVECT4, oldport4isr);         // Restore old interrupt vector
    setvect(INTVECT6, oldport6isr);         // Restore old interrupt vector

#ifdef USING_LCD
    fprintf(lcd,"%c%c",LCD_CMD,0x08);   // DispCtrl: Disp, Cursor, Blink    OFF
    fprintf(lcd,"%c%c",LCD_CMD,0x01);   // Clear & Home display
    WriteLCD_Text("SEISDRIFT COMPUTER", 1, 2 );
    WriteLCD_Text("==================", 2, 2 );
    fprintf(lcd,"%c%c",LCD_CMD,0x0C);   // DispCtrl: Disp ON, no cursor or blink
#endif

    printf("Shutting down...");
#ifdef USING_LCD
    delay(500);
    for (unsigned char i=255; i>1; i>>=1  )
    {
        WriteLCD_Text("SHUTTING DOWN!", 3, 4 );
        delay(i);
        WriteLCD_Text("              ", 3, 4 );
        delay(i);
    }
    WriteLCD_Text("SHUTTING DOWN!", 3, 4 );
    delay(500);
    fprintf(lcd,"%c%c",LCD_CMD,0x01);   // Clear & Home display
    fclose(lcd);                        // We opened lcd as a file, close it
#endif
}
////////////////////////////////////////////////////////////////////////////
```



SeisDrift
Seafloor detection in seismic data

University of Bergen (UoB)
Dept. of Earth Science
Allé gt. 41, N-5007 Bergen, Norway
Tel: (+47) 5558 3600
Fax: (+47) 5558 3660
http://www2.geo.uib.no/

14 Feb 2005: Latest news

- A usable method is found. The method is described below, and is based on correlation.

CONTENTS

1. Introduction

There are many possible approaches to the problem of identifying the seafloor from analysis of seismic data. One possibility would be to simply search for the largest (rectified) amplitudes in the traces. This method is not usable, however, because it's susceptible to noise-pulses, DC-offsets and other possible phenomena in the data.

We need something more robust. Therefore I did some tests using autocorrelation on the data. Letting each trace "slide" over itself while integrating the product of the two curves gave a clear indication of the time difference between the shot and the pulse echoed back from the seabed.

A better way to identify the seafloor is by correlation with a synthetic pulse. This algorithm needs less processing power while providing better results than what is possible with autocorrelation.

2. Test data

All the tests described here are performed on seismic data collected at a survey in 2003.

Parameters:

- Data-file: OBS2003line-g-1.su
- Traces used: 1 through 6
- Sample-rate: 1000 Hz
- Log-file: t-g-1.log
- Log-sheet: OBS2003-line-G.pdf

The relevant data from the .su -file was imported to Excel. A spreadsheet provides an easy way to work with the data and shows graphs for the correlation and other interesting results. Seismic data is very often stored in the **.su** file format, explained at this page: Seismic Unix (SU) data format

3. Autocorrelation

Definitions: Autocorrelation and Convolution

Correlation is mathematically based on convolution. Autocorrelation can be used when pulses with the same shape needs to be located within a particular signal waveform. In our application we have traces* (signal waveforms) which contain the seismic source signature in the beginning of the trace, followed by usually a longer period of "silence", and finally a strong echo from the seabed preceding the weaker seismic data.

*We are using one-channel recording, so there is exactly one trace per shot-point.

Drawbacks of using autocorrelation:

- Requires A LOT of floating-point operations to be carried out for each value of the input argument. This is due to the high number of data-points in each trace, compared to the short duration of the pulse we are looking for.
  (The argument to the autocorrelation function is a delay-time, in seconds or samples, usually referred to by the Greek letter "beta".)
- Slowly fluctuating DC-offsets in the trace data will produce large errors on the results. The autocorrelation curve in *Figure 1* should have had values around zero, except for where there actually is some correlation (beta from 0 to 500, and from 3750 to 3850). DC-offsets in the trace-data caused this effect which makes is very hard to locate the seabed location.
- Requires the source signature to be recorded in the beginning of each trace! And, furthermore, it should not be clipped! This means that the signal gain must be much lower than what we would like it to be for other obvious reasons.

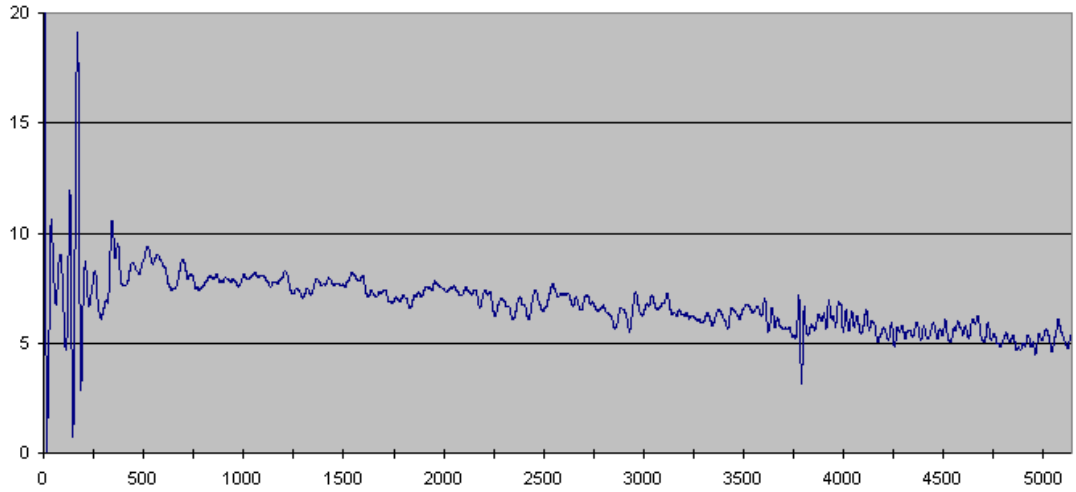


Fig. 1: Autocorrelation of trace 1 in OBS2003line-g-1.su

---

4. Correlation with a synthetic pulse

Definition: Cross-Correlation

If we construct a synthetic pulse and convolve it with the signal from the recorded traces we avoid all of the above mentioned drawbacks related to autocorrelation. To cancel the effect of a DC-component on the trace waveform it is important that the integral of the synthetic signal is zero. The synthetic pulse that I have used is composed of three halves of a sine wave, each with a different amplitude. Figure 2 shows the graph of the real pulse echoed back from the seabed (average of six time-adjusted traces, with reversed sign, DC removed), and the synthetic pulse used for the detection:

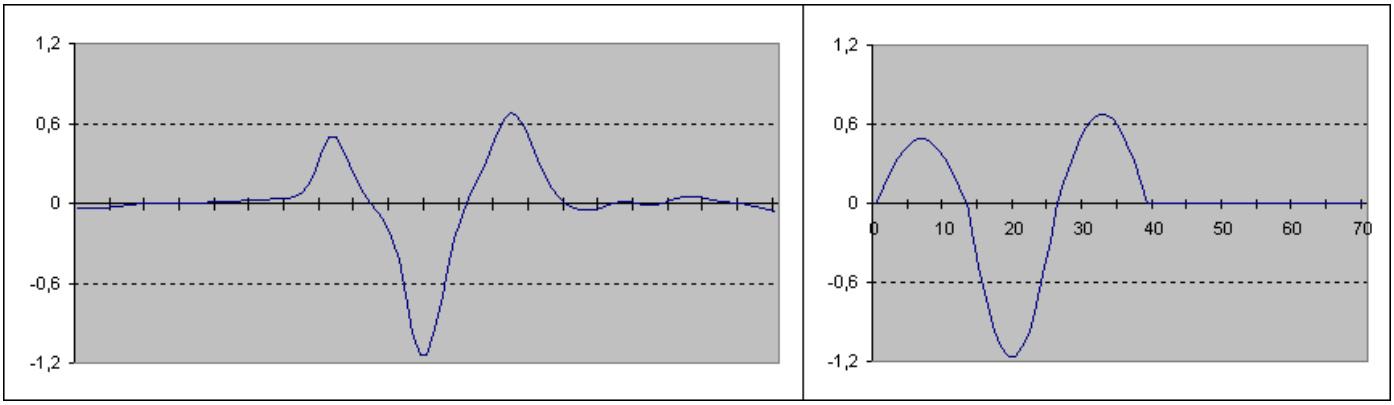


Fig. 2: A typical real pulse echoed from the seabed (DC removed), and the synthetic pulse used for detection

The characteristic properties of the synthetic pulse are the period T and the amplitudes A_1, A_2 and A_3. These properties must be chosen to match the waveform of the echoed pulse from the seabed, while at the same the keeping the integral equal to zero. The synthetic pulse used here has the following properties:

- T = 26ms = 26 samples
- A_1 = 0.49
- A_2 = 1.17
- A_3 = 0.68

The correlation function yields results around zero when there is no match between the trace-waveform and the synthetic detection pulse. When the argument has a value that causes coincidence with the synthetic pulse there will be a relatively large (positive or negative) deflection from zero. If the result from the correlation function is rectified we get the following curves (six adjacent traces with different colors):

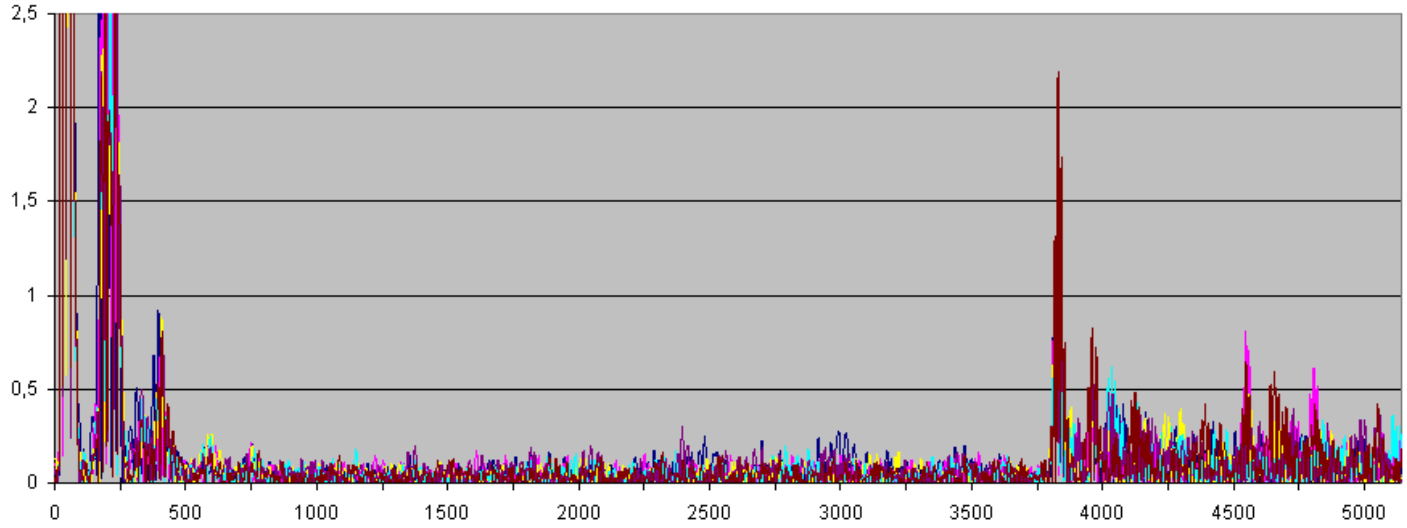


Fig. 3: The six rectified correlation curves

A zoomed display reveals that the correlation-curves are very identical around the place where the seabed is detected:

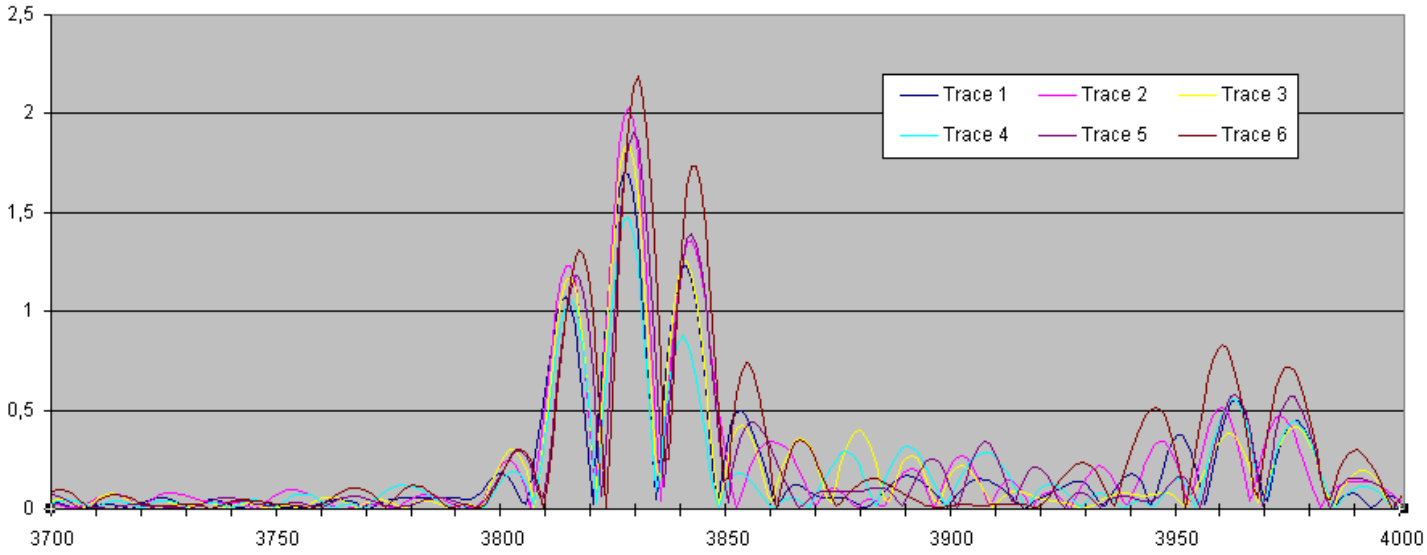


Fig. 4: Details of the correlation curves

Since the curves are almost identical (they are based on adjacent traces) they can be added. This reduce the random noise, and improves the S/N-ratio:

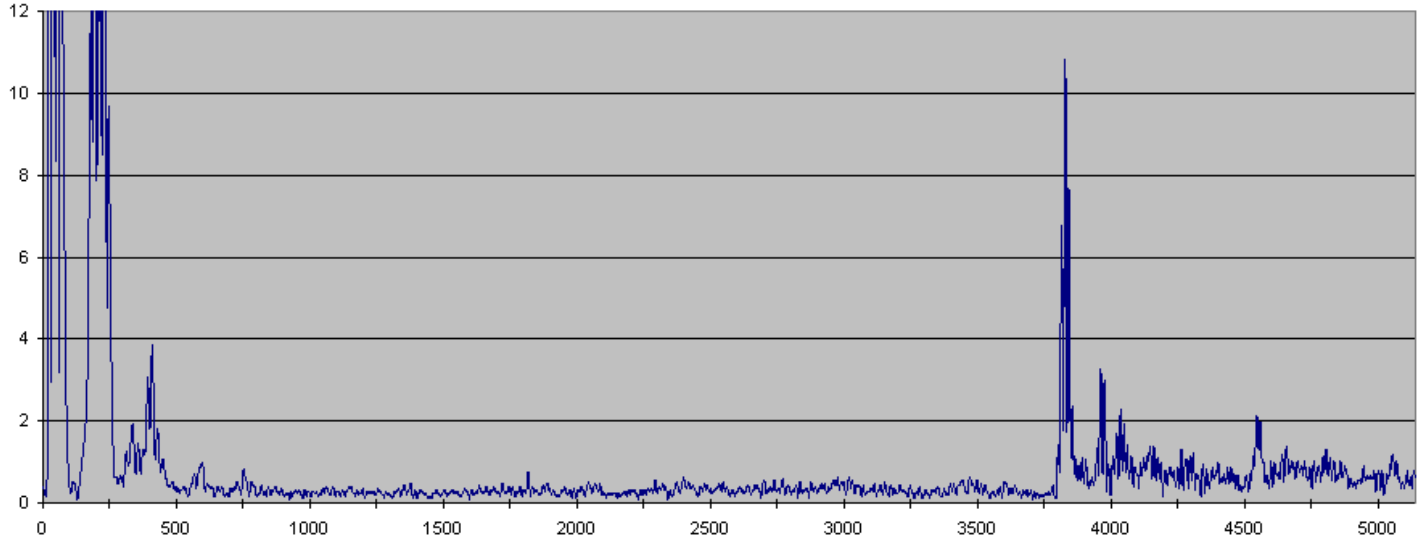


Fig. 5: The sum of six rectified correlation curves

And here is the zoomed version:

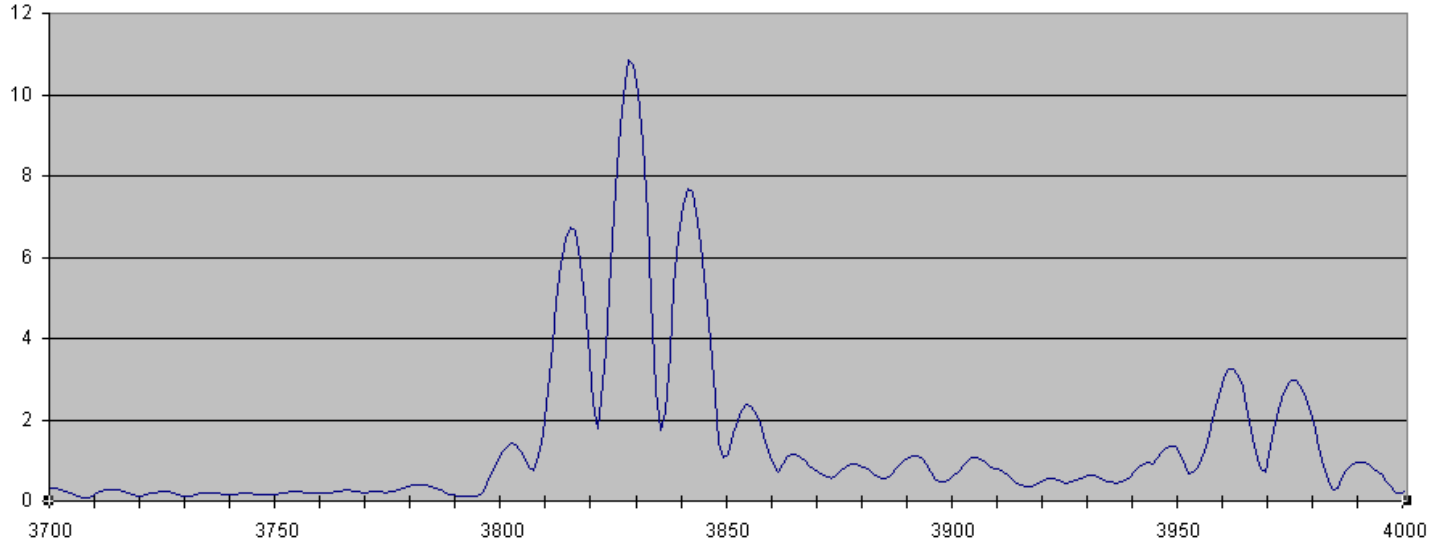


Fig. 6: Details of the summed curves

Correlation seems to be a reliable way to detect the seabed in seismic traces. If we know the minimum depth we also know the minimum two-way traveltime (TWT) for the signals bounced back from the seabed. For example, if we are operating at depths beyond 1000 meters, the TWT is more than 1.34 seconds and the bottom must be found somewhere after 1.34 seconds in the traces' data. The test-data used above are sampled at 1000 Hz, meaning that we must search the correlation curves for the highest rectified magnitude when the argument is varied from 1340 to *max*. (In this case, *max* is 12287 - the number of samples in the traces minus one.)

The maximum correlation for the six traces are found where *beta* is approximately 3828. This can be clearly seen from *Figure 4* and *Figure 6*:

Trace no.:	Seafloor detected: *beta* =	Seafloor depth: (from headers in **.su** -file)
1	3827	2845
2	3828	2846
3	3828	2845
4	3828	2845
5	3829	2847
6	3830	2848
Average	3828,33	2846

The real depth is extracted from the header-data field gwdep in OBS2003line-g-1.su. A change of one unit in *beta* corresponds to 1ms TWT, or approximately 75cm depth.

---

5. Software

So far, all experiments have been carried out in a spreadsheet. The next task is to implement the correlation algorithm in C-language, adapted to the Persistor CF2 single board computer.

---

6. Links and keywords

Links:

Here is a really neat Cross Correlation project done by Brian Mileshosky. It involves the concepts of radar images and Matlab.

http://www.dkdinst.com/articles/SLIDING_CORRELATORS.doc

http://astronomy.swin.edu.au/~pbourke/analysis/correlate/

http://www.eece.unm.edu/signals/Cross_Correlation/cross_correlation.html

Some keywords for Google:

correlation detection methods; sliding correlator; cross-correlation; first-arriving pulse (FAP)

==========

Erik Grindheim, 14.02.2005

==========