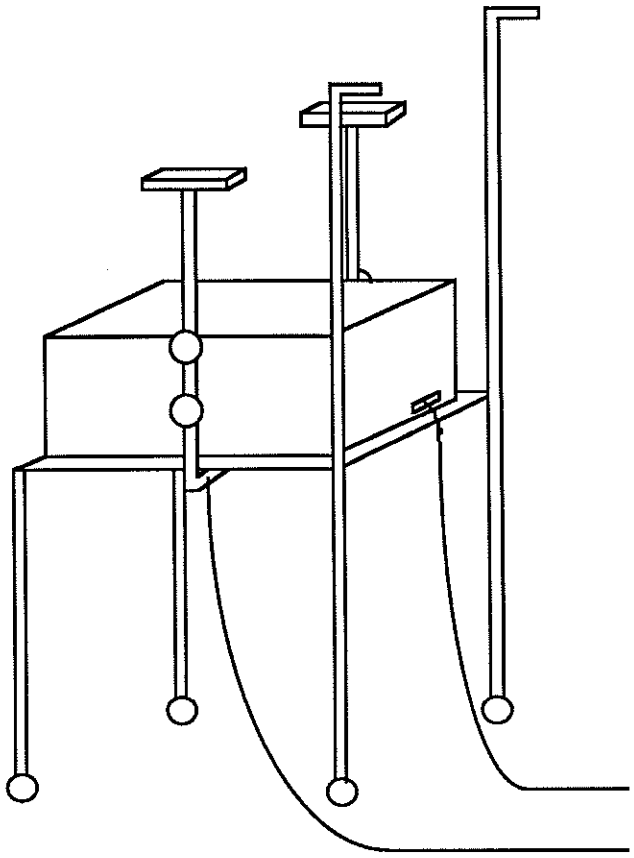
ISSN 0280–5316
ISRN LUTFD2/TFRT--5464--SE

Mätning och Identifiering

av

Postural Reglering



Gunilla Höglund
Johan Lidman

Institutionen för Reglerteknik vid Tekniska Högskolan i Lund
Oktober 1992

Department of Automatic Control
Lund Institute of Technology
P.O. Box 118
S-221 00 Lund Sweden

Document name
MASTER THESIS

Date of issue
October 1992

Document Number
ISRN LUTFD2/TFRT--5464--SE

Author(s)
Gunilla Höglund and Johan Lidman

Supervisor
Rolf Johansson and Måns Magnusson

Sponsoring organisation

Title and subtitle
Measurement and Identification of Postural Control. (Mätning och identifiering av postural reglering).

Abstract

Postural control, the ability of maintaining balance, is an important prerequisite of almost all human activity. Therefore disturbances in postural control may be a severe handicap. There have been difficulties in measuring human balance in an objective way, but through research done at the University Hospital of Lund methods have been developed that make it possible to measure balance on upright standing patients. This has been done, in an automatic control system, by identifying the human body as an inverted pendulum.

There has been a need for making measurement on patients who are incapable of standing upright without support. We have therefore tried to develop a method by which balance is measured in a seated position. The equipment consists of a specially designed chair where forces and torque of a force plate and elbow-rests are measured and further treated by computer. The patient is given an electrical or vibrational stimulus so that a disturbance in the vestibular system is simulated.

The equipment has been tested on a group of 12 healthy subjects in whom an effect of supplied stimuli has been noted, above all as an compensatory backward leaning when vibrational stimuli are used. However, we have not been able to prove the identification of the system with an inverted pendulum, and therefore a further development is motivated.

Key words

Classification system and/or index terms (if any)

Supplementary bibliographical information

ISSN and key title
0280–5316

ISBN

Language
Swedish

Number of pages
114

Recipient's notes

Security classification

Innehållsförteckning

Innehållsförteckning 1
Sammanfattning 2
Bakgrund 3
Problemformulering 4
Teori 5
Medicinsk teori 5
Mekanisk modell 7
Utrustning 10
Stolen 11
Avvikelse i mätningarna 12
Trådtöjningsgivare 13
Förstärkare 14
Datorn och programvara 14
Postcon3 programmets uppbyggnad och funktion 15
Postcon3 handhavande av programmet 16
Försöksuppställning 17
Stimulering av nackmuskulaturen med vibratorer 17
Galvanisk stimulering av balansnerven 17
Stimuleringssignal 17
Genomförande 18
Identifieringsmetoder 19
Parametrisk identifiering 19
Ickeparametrisk identifiering 19
Testresultat 20
Diskussion 22
Syfte 22
Armstödsmätning 22
Sittställning och stimuli 23
Slutsats 24

Tillkännagivanden

Referenser

Appendix

1 **Armstödsmätning felkällor**
2 **Trådtöjningsgivare kopplad i helbrygga**
3 **Nedböjning av armstöd**
4 **Förstärkaren**
5 **Kalibreringsprotokoll**
6 **Användarhandledning program**
7 **Programlista**
8 **Mätdata**

Sammanfattning

Postural kontroll, förmågan att upprätthålla balans, är en viktig förutsättning för nästan all mänsklig aktivitet. Därför kan störningar i den posturala kontrollen utgöra ett gravt handikapp. Svårigheter har funnits att på ett objektivt sätt mäta människans balansförmåga, men genom forskning på bl a Lunds lasaretts balanslaboratorium har man utarbetat metoder som gör det möjligt att mäta balansen på stående patienter. Detta har möjliggjorts genom att reglertekniskt identifiera systemet människokroppen med en inverterad pendel.

Ett behov finns att göra mätningar även på patienter som är oförmögna att stå upp självständigt. Vi har därför försökt utveckla en metod där balansförmågan mäts i sittande ställning. Utrustningen består av en specialutrustad stol där krafter och moment från sittplatta och armstöd mäts och behandlas vidare av en dator. Man tillför patienten en elektrisk eller vibratorisk stimulus för att på så sätt simulera en störning av balanssystemet.

Utrustningen har prövats på en grupp av 12 friska försökspersoner på vilka en påverkan av påförda stimuli har kunnat detekteras, framförallt i form av en kompensatorisk bakåtlutning vid vibrationsstimulering. Dock har vi ej kunnat styrka systemets identifiering med en inverterad pendel varför en vidareutveckling är motiverad.

Bakgrund

Det som händer då man lyfter ena armen för att klia sig i pannan, kan vid en första anblick verka nästan löjeväckande trivialt. Det visar sig efter en djupare studie att de olika musklernas samverkan, i sin påverkan av skelettets ben som tillsammans skapar armens svepande precisionsrörelse, innehåller en stor portion avancerad styr- och reglerteknik. Den här samverkan som sker så fort vi rör delar av vår kropp mot gravitationskrafterna kallas med ett samlingsnamn postural kontroll. Det är med hjälp av detta system vi upprätthåller balansen.

Postural kontroll är en förutsättning för nästan all mänsklig aktivitet. Därför är skador på den posturala kontrollen ett svårt handikapp, även för en människa med en kanske i övrigt funktionsduglig kropp. Det har sedan en tid tillbaka funnits olika metoder att analysera och mäta hur människor håller balansen. De metoder som hittills har använts bygger på att människor kan stå upp. Problemet är att många människor som har skador på den posturala kontrollen är så skadade att de ej kan stå upp, t ex neurologiskt skadade människor. Därför är det av stort intresse att kvantifiera postural kontroll i sittande ställning.

Problemformulering

Vår uppgift var att konstruera och göra en första utvärdering av en utrustning för att mäta balansen hos människor som ej kan stå upp, närmast neurologiskt skadade patienter. Utrustningen skall bestå av en stol med sittplatta och armstöd, där krafter och moment kan mätas med hjälp av trådtöjningsgivare. Till dessa skall det finnas förstärkare vars utsignal direkt kan kopplas till I/O-kortet hos en IBM-kompatibel persondator. I datorn skall mätsignalerna lagras på ett sådant sätt att de kan analyseras av programmet matlab.

Till vår hjälp fanns från början en modifierad vågstol och en kraftplatta, tidigare använd som ståplatta, med tillhörande förstärkare och Modula2-baserat mätdatasystem tillgängliga. Det problem vi ställdes att lösa kan delas in i fyra delar.

- Utforma armstöden och utrusta dessa med trådtöjningsgivare för att kunna mäta de krafter och moment armstöden utsetts för.

- Bygga instrumentförstärkare för att kunna mäta töjningen hos trådtöjningsgivarna och anpassa dess utsignalnivå till datorns signalingångar.

- Göra mätsignaler från instrumentförstärkarna tillgängliga för mjukvaran Matlab utgående från det befintliga mätdatasystemet. Dessutom implementera en kalibreringsrutin för att förenkla nolljusteringen av instrumentförstärkarna.

- Göra mätningar med vibrationsstimulus och galvanik på ett tiotal personer och utvärdera dessa mätserier.

Teori

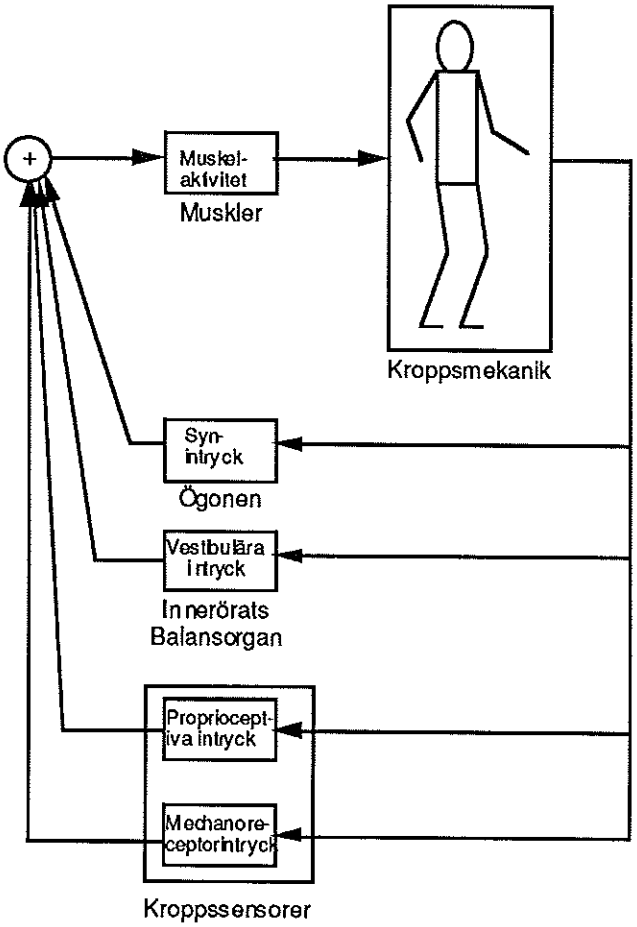
Ett av problemen, och tjusningen, vid forskningen kring postural kontroll är att den är en tvärvetenskap. Det finns en rent medicinsk sida eftersom det handlar om människokroppen och det visar sig även, att man kan se hela den posturala kontrollen som ett reglerproblem. De bägge vetenskaperna har olika storheter, angreppssätt och frågeställningar. Här nedan kommer tyngdpunkten att läggas vid den reglertekniska situationsbeskrivningen med dess beskrivningssätt och storheter.

Medicinsk teori

De insignaler människokroppen använder för att hålla balansen kan delas in i följande fyra huvudgrupper:

- *Visuella insignaler.* De visuella insignalerna är de synintryck ögonen bidrar med.
- *Signaler från vestibularisorganen.* I vardera örat sitter ett balansorgan, kallat vestibularisapparaten. Den består av otholitorganen och båggångarna. Otholitorganen kan känna linjära accelerationer, t ex tyngdkraften, och båggångarna vinkelaccelerationer, såsom huvudvridningar [12].
- *Det proprioceptiva systemet*, som har två typer av givare. Den första typen, muskelspolarna [7], sitter i kroppens alla muskler och känner av hur snabbt muskeln drar ihop respektive töjer sig samt i viss mån även muskelns längd(muskelspolen mäter muskelns hastighet och position). Den andra typen, kallad Golgis senorgan [7], sitter i muskelns senor och känner av den spänning senan utsätts för (Golgis senorgan registrerar den kraft muskeln utvecklar).
- *Signaler från hudens mekanoreceptorer.* I den hårlösa huden finns olika typer av receptorer som känner av tryck mot och töjning av huden [7].

I figur 1 visas en schematisk bild av den posturala kontrollen med in- och styrsignaler.



figur 1 Modell av återkopplingen med de olika styr- och observer-storheterna för den posturala kontrollen.

Styrsignalen människokroppen har till sitt förfogande för att reglera balansen är en kombination av olika stora stimulanser till de muskler som skall koordineras att utföra korrektionen. Utsignalen från systemet är den position kroppens olika lemmar har.

Ett sätt att försöka skaffa sig en modell av hur ett system fungerar är att excitera systemet och studera återverkningen i utsignalen. Det finns flera olika metoder att excitera människans balanssystem varav den här rapporten har utnyttjat två.

- Genom att störa det proprioceptiva systemet med fastsatta vibratorer vid de muskelgrupper som är viktiga för balanshållningen tillfogas de ovan beskrivna muskelspolarna och senorganen en störsignal.

- Stimulera balansnerven genom att en elektrisk ström tillförs via elektroder strax bakom öronen. Denna ström ändrar impulsfrekvensen som då balansnerven överför till hjärnan.

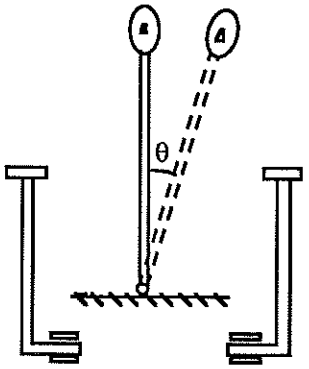
I båda fallen används statistiska systemidentifieringsmetoder.

Mekanisk modell

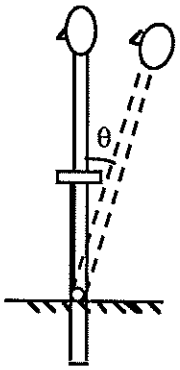
Den mekaniska modell som valts för att beskriva den sittande människokroppen illustreras i figur 2 och 3. Kroppen approximeras här med en inverterad pendel som kan falla i antingen lateral- eller sagittalled[1] . Den approximation som görs för att denna modell skall gälla är att kroppen antas bete sig som en stel kropp som vrider sig kring ett fast vridcentrum. Tröghetsmomentet kring vridcentrum ges nu av (se [11]):

$$J\frac{d^2\theta}{dt^2}=mgl\sin\theta(t) \qquad J=ml^2$$

där m är kroppens massa i kg.
l är avståndet från vridcentrum till kroppens masscentrum i meter
θ är den inverterade pendelns vinkelavvikelse från vertikalled.
J är tröghetsmomentet som för en homgen stav är ml^2



figur 2 Modell av en sittande människa som rör sig i lateralled.



figur 3 Modell av en sittande människa som rör sig i saggitalled

Den ovan beskrivna modellen är instabil. För att stabilisera den inverterade pendeln läggs ett stabiliserande moment,$M_{stab}(t)$ till (människan kan ju förbli upprätt sittande). Dessutom införs vridmomentet $M_{stör}(t)$ för att beskriva skillnaden mellan verkligt och uppmätt vridmoment. Systemet ser nu ut som:

$$J\frac{d^2\theta}{dt^2}=mgl\sin\theta(t)+M_{stab}+M_{stör} \qquad J=ml^2$$

För att kunna göra en identifiering av systemet måste även reglermekanismen, d v s det sätt den posturala kontrollen försöker hålla kroppen upprätt, modelleras. M_{stab} är den styrstorhet med

[1] Sagittalled är riktning framåt-bakåt
Lateralled är riktning sida till sida

vilket detta sker. Den regulator som väljs är en PID-regulator som återkopplar vinkeln, θ. Man kan visa att systemet blir instabilt om θ endast återkopplas med en P-regulator. Likaså kan man visa att det räcker med en PD-regulator för att stabilisera systemet. PD-regulatorn kan dock ej reglera bort felet helt utan kvar blir ett litet stationärt fel. En sittande människa kan naturligtvis reglera bort felet fullt ut. Läggs en I-del till elimineras det stationära felet. Således approximeras sättet att hålla balansen med en PID-regulator. De olika P, I och D komponenterna väljs som följer.

P: $-mgl\sin\theta(t)-kJ\theta(t)$

D: $-\eta J\dot{\theta}(t)$

I: $-\rho J\int_{t_0}^{t_1}\theta(t)dt$

Då nackmuskulaturen och/eller balansnerven stimuleras är tanken att en felsignal introduceras hos vinkelläget och vinkelhastigheten. I fallet med vibrationsstimulering är kopplingen medicinskt förankrad där lägesåterkoppling och hastighetsåterkopplingen motsvarar muskelspolarna. Då stimuleringen sker med hjälp av galvanisk stimulering av balans-nerven är det dock oklart var felsignalen adderas till systemet. Stimuleringssignalen kallad V(t) viktas in i P- och D-delen med konstanterna b_1 och b_2.

P: $-mgl\sin\theta(t)-kJ\theta(t)+b_1V(t)$

D: $-\eta J\dot{\theta}(t)+b_2V(t)$

Med vridmomemtsjämvikt och den ovan framresonerade regulatormekanismen fås nu de två ekvationerna:

$$J\frac{d^2\theta}{dt^2}=mgl\sin\theta(t)+M_{stab}+M_{stör} \qquad J=ml^2$$

$$M_{stab}=-mgl\sin\theta(t)-kJ\theta(t)+b_1V(t)-\eta J\dot{\theta}(t)+b_2V(t)-\rho J\int_{t_0}^{t_1}\theta(t)dt$$

$$\Rightarrow$$

$$J\frac{d^2\theta}{dt^2}=mgl\sin\theta(t)-mgl\sin\theta(t)-kJ\theta(t)+b_1V(t)-\eta J\frac{d\theta(t)}{dt}+b_2V(t)-\rho J\int_{t_0}^{t_1}\theta(t)dt+M_{stör}(t)$$

$$\Rightarrow$$

$$J\frac{d^2\theta}{dt^2}+\eta J\frac{d\theta(t)}{dt}+kJ\theta(t)+\rho J\int_{t_0}^{t_1}\theta(t)dt=(b_1+b_2)V(t)+M_{stör}(t)$$

Uttrycket Laplacetransformeras och överföringsfunktionen från

$M_{stör}(t)$ och $V(t)$ till $\theta(t)$ blir:

$$Js^2\theta(s)+\eta Js\theta(s)+kJ\theta(s)+\rho\frac{1}{s}\theta(s)=(b_1+b_2)V(s)+M_{stör}(s)$$

$$\Rightarrow$$

$$\theta(s)=\frac{\frac{1}{J}(b_1+b_2)s}{s^3+\eta s^2+ks+\rho}V(s)+\frac{\frac{1}{J}s}{s^3+\eta s^2+ks+\rho}M_{stör}(s)$$

Den inverterade pendeln antas svaja kring sitt jämviktsläge med små vinkelutslag. För att få ett linjärt system görs approximationen $\sin\theta\approx\theta$. Överföringsfunktionen från $M_{stör}(t)$ och $V(t)$ till $M_{stab}(t)$ blir:

$$M_{stab}\approx(Js^2-mgl)\theta(s)+M_{stör}$$

$$\Rightarrow$$

$$M_{stab}\approx\frac{1}{s^3+\eta s^2+ks+\rho}\left[(Js^2-mgl)\left(\frac{1}{J}(b_1+b_2)sV(s)+\frac{1}{J}sM_{stör}(s)\right)-M_{stör}(s)(s^3+\eta s^2+ks+\rho)\right]=$$

$$=\left[J=ml^2\right]=\frac{1}{s^3+\eta s^2+ks+\rho}\left[(b_1+b_2)\left(s^2-\frac{g}{l}s\right)V(s)-\left(-s^3+s^3+\eta s^2+ks+\rho+\frac{g}{l}s\right)M_{stör}(s)\right]=$$

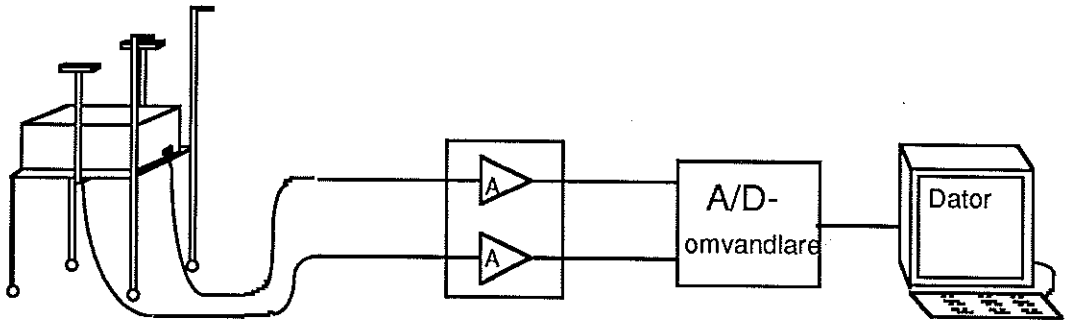
$$=\frac{1}{s^3+\eta s^2+ks+\rho}\left[(b_1+b_2)\left(s^2-\frac{g}{l}s\right)V(s)-\left(\eta s^2+\left(k+\frac{g}{l}\right)s+\rho\right)M_{stör}(s)\right]$$

Detta är den linjäriserade överföringsfunktionen som beskriver hur $M_{stör}(t)$ och $V(t)$ överförs till $M_{stab}(t)$ då vinkeln θ är liten. Den har validerats för stående personer med vibrationsstimulering på vadmuskulaturen [8]. Diskretiseras överföringsfunktionen fås en tredje ordningens ARMAX-process. Det är denna diskretiserade modell som mätvärdena under avsnittet Testresultat kommer att försöka anpassas till.

Utrustning

Den utrustning som behövs för att undersöka balansrubbningar hos patienter som är oförmögna att stå upp men som är så pass friska att de trots allt självständigt kan sitta upp, är i grunden en stol utrustad med bl a en kraftplatta [4]. Kraftplattan är utvecklad och utprovad på Vestibularislaboratoriet vid Lasarettet i Lund i samarbete med Institutionen för Reglerteknik samt Institutionen för Hållfasthetslära. Den är en konstruktion som kan mäta krafter och moment i de tre rumsdimensionerna. Eftersom det finns patienter som är så sjuka att de inte kan sitta utan stöd för armarna har armstöd med mätutrustning tillfogats.

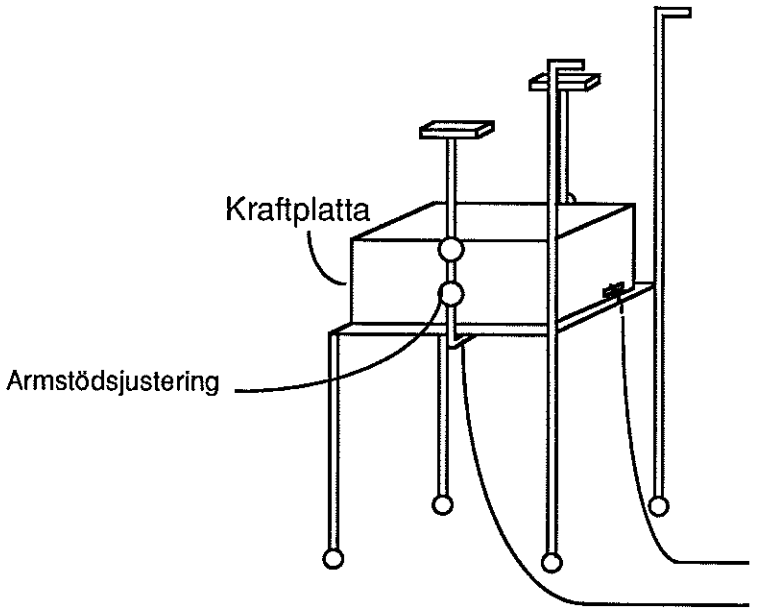
Mätutrustningen består av trådtöjningsgivare med tillhörande instrumentförstärkare. Signalerna från armstöden och kraftplattan A/D-omvandlas och mottas av en dator som lagrar och behandlar data. Se figur 4.



Figur 4. Utrustning vid balansundersökning av sittande personer.

Stolen

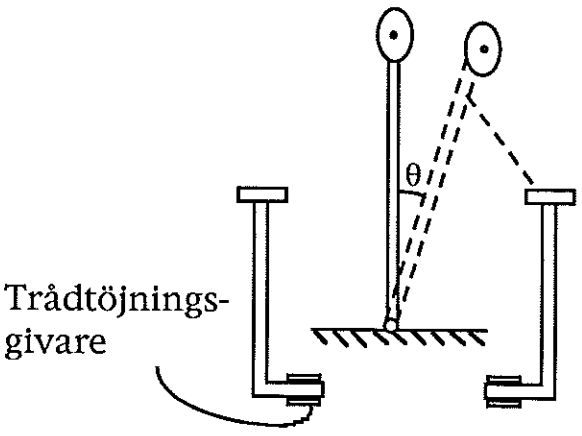
Den stol som används är en modifierad, mobil vågstol. Stolen är försedd med, i höjdled, justerbara armstöd. Däremot saknas ryggstöd såsom framgår av figur 5.



Figur 5. Stol med kraftplatta och armstöd utrustad med givare.

Vid försöken rör sig patienten i lateral- och sagittalled. Vid rörelse i sagittalled detekteras denna enbart i kraftplattan. Vid en rörelse i lateralled kan patienten även ta stöd mot armstöden och en nedböjning av dessa kommer att kunna detekteras. För att få ett korrekt mätvärde från armstöden, d v s veta hur krafterna och momenten fördelar sig i horisontalled respektive vertikalled antas följande modell.

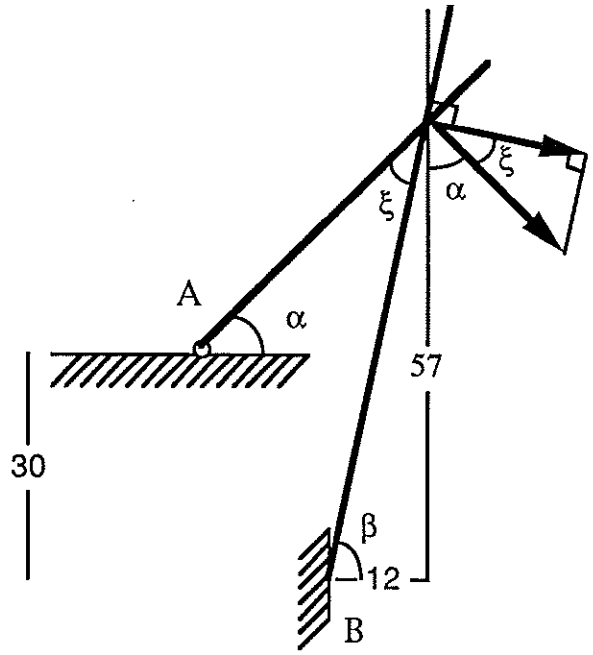
Kroppen identifieras som en inverterad pendel placerad enligt figur 6. Axlar och armar antas stelt oledade. När pendeln faller ur sin upprätta position och vidrör armstödet bidrar detta med ett stabiliserande moment och pendeln återgår till sitt ursprungsläge.



Figur 6. Försöksperson som stödjer mot det givarförsedda armstödet.

Med valda dimensioner på balkar och med placering av trådtöjningsgivare enligt bilden ovan, erhålls en utsignal från armstöden som motsvarar tillsammans med signalerna från kraftplattan, med ett visst fel, det stabiliserande momentet.

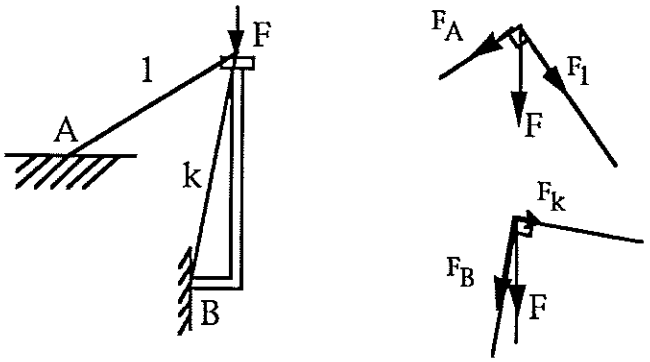
Avvikelse i mätningarna



Figur 7. Ur stolens trigonometri kan avvikelsen i mätningen beräknas.

Det moment som detekteras i armstödet, i figur 7 punkt B (givaren), är det moment som uppkommer pga kraften vertikalt mot den resulterande momentarmen. Det moment som egentligen är av intresse för modellen, är det som återfinns vid kraftplattan, A. Eftersom förhållandet mellan dessa två moment ej är proportionellt så är det ej möjligt att enkelt kompensera för det fel som mätningen ger.

Felet kan beräknas genom följande trigonometriska resonemang.



Figur 8. Kraftverkan på armstöd. Kraftkomposanter i A- och B riktning.

En kraft som, enligt Figur 8, appliceras på armstödet kan delas upp i komposanterna F_A och F_l. Det bidragande stabiliserande momentet från armstödet kan då skrivas:

$M_A = l * F_l + 0 * F_A$
Det moment som faktiskt mäts i armstödet är dock:

$M_B = k * F_k + 0 * F_B$

Ett mått på hur bra det sökta momentet verkligen mäts är således hur bra F_l överförs på F_k. Det är också intressant att veta hur stor överhörningen från F_A till F_k är. Ur figur 7 och 8 kan utläsas att :

$F_k = F_l \cos \xi + F_A \sin \xi$ där $\xi = \beta - \alpha$.

cos ξ talar om hur bra F_l överförs på F_k och sin ξ hur stor överhörningen från F_A till F_k är.

Armstödshöjden justeras vid försöken till 57 cm vilket tillsammans med övriga mått enligt Figur 7 ger:

β = arctan(57/12) =76.2°
α = arctan(30/30) = 45°
$\xi = \beta - \alpha$ = 31.2°

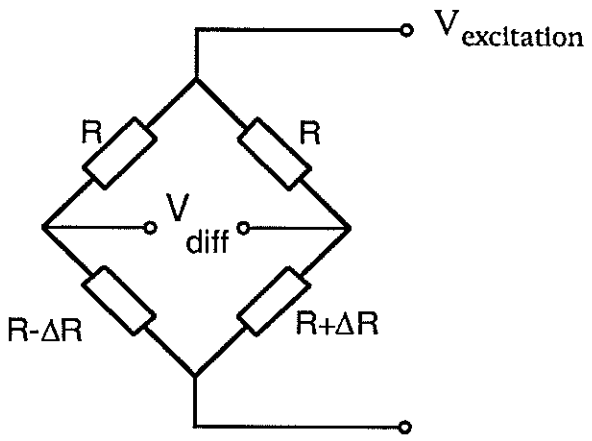
cos ξ = cos 31.2=0.86
sin ξ = sin 31.2 = 0.52

Alltså är felet hos utsignalen 100*(1-1/0.86) ≈ 16% vid en infallande kraft på armstödet i F_l-rikningen.

Då en försöksperson tar stöd mot armstödet borde kraften armstödet utsetts för, rimligtvis vara riktad utåt och nedåt. För att få en uppfattning om hur stora mätfel som uppstår, kan dessa vara av intresse att studera. Se Appendix 1.

Trådtöjningsgivare

För att kunna mäta ovan nämnda moment behövs en utrustning som känner av nedböjningen av armstödet. Nedböjningen kan detekteras i form av en spänningsförändring över trådtöjningsgivarna som är kopplade i en helbrygga enligt figur 9. Eftersom spänningen V_{diff} är mycket liten krävs det en förstärkning av signalen. Beräkningar och teori om trådtöjningsgivare och helbryggor finns att läsa i Appendix 2.



Figur 9. Trådtöjningsgivare (R) kopplade i en helbrygga.

Förstärkare

Förstärkaren som behövs för att kunna mäta signalen från armstöden är av typen instrumentförstärkare. Den består av ett antal OP-förstärkare som är kopplade så att den differentiella spänningen hos trådtöjningsgivaren förstärks *oberoende* av excitationsspänningen.

Av de olika förstärkaralternativ som fanns valde vi att bygga upp en förstärkare kring en krets, 1B31AN [1], som innehåller variabel förstärkare, exciteringsspänning och filter i en och samma kapsel. 1B31AN är en lågbrusande förstärkare med en variabel bandbredd mellan 10 Hz - 20000 Hz och en kontinuerlig, variabel excitationsspänning ner till 4 volt, vilket är önskvärt då termiska bruset ökar med högre spänningar. Olineäriteten är liten, endast 0.005% och förstärkningen sträcker sig upp till 5000 gånger.

Kretsen placeras på ett kretskort med tillhörande kringkomponenter. På kretskortet finns det utrymme får två stycken 1B31AN och alltså finns det möjlighet ta emot signal från två helbryggor. Instrumentlådan är förberett för ytterligare ett förstärkarkort.

I Appendix 4 finns beräkningar, schema, komponentförteckning och layout, både över förstärkardelen och spänningsmatningen till densamma.

Datorn och programvara

Datorn är en IBM-kompatibel persondator med ett tillhörande I/O-kort. Alla signaler in till och ut från datorn går via detta kort. Till hjälp vid försöken finns ett mätdatainsamlingsprogram, postcon3, skrivet i programspråket Modula2 (Logitech Modula2 V3.0 [10]). Dessutom används en realtidskärna utvecklad på Institutionen för

Modula2 är ett programspråk med instruktioner snarlika Pascal. Det som skiljer Modula2 från Pascal är i stort att olika progamavsnitt kan kompileras separat och sedan länkas ihop. Realtidskärnan gör det möjligt att exekvera flera processer samtidigt.

Postcon3 programmets uppbyggnad och funktion

Postcon3 (*post*ural *con*trol version3) är ett program som läser in signalerna från kraftplattan och armstödet samt styr ut signaler till givare beskrivna under avsnittet Försöksuppställning. Programmet visar signalerna på skärmen och de kan även lagras på en extern fil för att analyseras i matlab. Postcon3 är en vidareutveckling av postcon2 [3]. Skillnaden mellan de båda programmen är att det i postcon3 även är möjligt att behandla signaler från armstöden. Dessutom finns det i postcon3 en kalibreringsrutin som gör det lättare att nolljustera förstärkarna till kraftplattan och armstöden.

Postcon3 består av nio processer och fyra monitorer se figur 10. Varje monitor innehåller variabler som alla processer kan läsa och skriva i. Monitorvariablerna innehåller uppgifter om vilken status systemet befinner sig i. På så sätt samordnas de olika processernas aktiviteter då de läser av monitorvariablerna. Dessutom kommunicerar processerna Plotproc, Fileproc och Measure med varandra via brevlådor. De data som utbyts är mätvärden från kraftplattan och armstöden samt eventuell galvanisk och/eller vibrationsstimulering. Nedan följer en kort beskrivning av varje process och monitor. För en utförligare beskrivning finns en programutskrift i appendix 7.

Measure: Measure sköter inläsning av data från kraftplattan.

Gengalv: Gengalv lägger ut vald signal till I/O-kortet för galvanisk stimulering.

Genvib: Genvib lägger ut vald signal till I/O-kortet för stimulering med vibratorer.

Control: Control har hand om meny- och fönsterhanteringen .

Plotproc: Plotproc presenterar in- och utsignaler grafiskt på skärmen.

Fileproc: Fileprocess lagrar mätserierna på extern fil som direkt kan användas i matlab.

Helpprocess: Helpproc ger hjälputskrift om så önskas över de olika menyvalen.

Safe: Safe avbryter ett pågående försök omedelbart om så önskas.

Postcon: Postcon startar upp alla andra processer och avslutar även programmet.

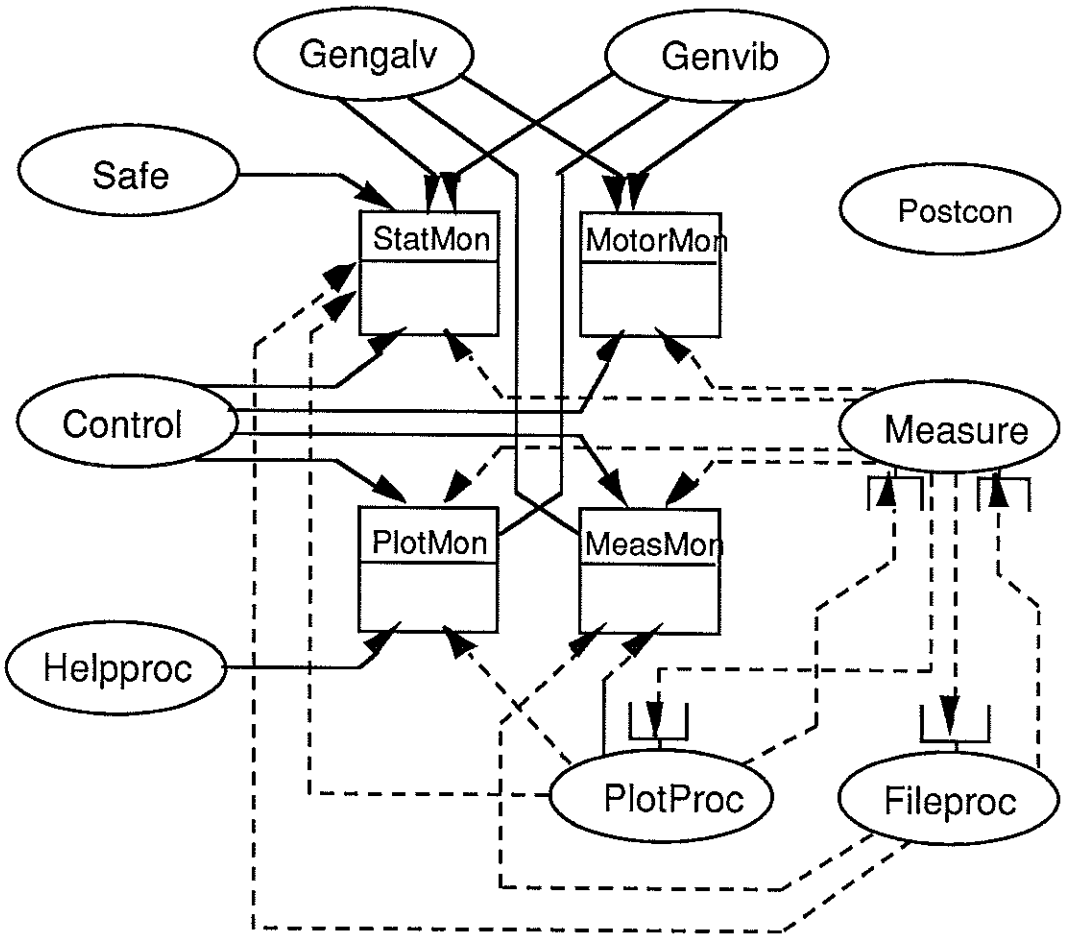
Whatmon: WhatMon innehåller variabler som styr plotfönstrets funktion och utseende.

StatMon: StatMon innehåller variabler för global hantering

som samplingstid, plottningshastighet, kraftplattans och armstödens mätsignaler samt emergency.

MeasMon: MeasMon innehåller variabler om pågående stimuleringar vilka datavärden som mäts, och övriga variabler som berör mät och stimuleringsfunktionen

MotorMon: MotorMon innehåller parametrar som styr stimuleringarnas utseende för alla testsekvenser samt testsekvensernas längd.



Figur 10 Processgraf Postcon3

Postcon3 handhavande av programmet

Programmet startas med att skriva postcon3. De olika val som finns presenteras i fönstermenyer där de presenterade alternativen väljs med mus. Menyerna är av två slag, logiska och numeriska menyer. I de logiska menyerna aktiveras ett alternativ med att föra musen mitt för alternativet och klicka. I de numeriska menyerna görs likadant men dessutom matas önskat siffervärde in följt av vagnretur. För beskrivning av de olika menyerna se appendix 6.

Försöksuppställning

Fem stycken olika mätserier utfördes på tolv personer, fem kvinnor och åtta män i åldrarna 23 till 59 år med medianålder 25 år. En kvinna utgick p g a medicinering som kunde påverka balansfunktionen.

Mätserierna kan indelas i två grupper där två olika typer av insignaler för att excitera systemet, som den sittande försökspersonen beskriver, användes. Försöks-personen hade antingen slutna eller öppna ögon, och två olika kroppsställningar provades. Utsignalerna från det system, som försökspersonen beskriver då hon försöker sitta upprätt, är de krafter och moment som mäts i kraftplattan och armstöden. Med hjälp av dessa kan det stabiliserande momentet beräknas.

Stimulering av nackmuskulaturen med vibratorer

Insignalen tillfördes systemet med en nackkrage som spänns fast runt halsen. På nackkragen fästes två vibratorer så att de låg an mot nackmuskulaturen men ej mot skallbenet. (Om vibratorerna ligger an mot skallbenet kan vibrationerna fortplantas till innerörat och påverka vestibularisapparaten.) När en spänning läggs över vibratorerna börjar de vibrera och stimulerar då nackmuskulaturen. Den pålagda vibrationen stör de tidigare beskrivna muskelspolarna och senorganen, som då börjar ge ut felaktiga nervsignaler. Resultatet blir att försökspersonens balansfunktion påverkas.

Galvanisk stimulering av balansnerven

Galvanisk stimulering tillförs systemet via två elektroder som fästs med hjälp av ledande pasta bakom öronen på mastoidbenet. Elektroderna är kopplade till en strömgenerator som kan ge varierande ström upp till 4mA. Vid de försök som gjordes användes ±0.8 mA. När man sänder en ström genom huvudet så tillförs de bägge balansnerverna en felsignal och försökspersonen får svårare att upprätthålla balansen.

Stimuleringssignal

Vid alla fem mätserierna användes utsignalen från ett åtta bitars binärt linjärt återkopplat skiftregister av fullängd som insignal till vibratorerna respektive elektroderna. Varje mätserie tog 235 sekunder och bestod av två delar. Först inleddes mätserien med 30 sekunder utan stimulering. Därefter sändes utsignalen från skiftregistret till vibratorerna, via ett spänningsaggregat, respektive elektroderna via en strömgenerator. Vid vibrationsstimulering

motsvarade en etta ut från skiftregistret att vibratorerna vibrerade med 60 Hz (effekt 850 mW, 1 mm:s amplitud) och en nolla motsvarade att vibratorerna var avstängda. Då elektroderna användes, motsvarade en etta, att 0.8 mA sändes från höger till vänster, och en nolla, att samma ström sändes åt andra hållet. Skiftregistersignalen skickades ut i 205 sekunder där uppräkningstiden för skiftregistret var 0.8 sekunder (0.8*28-1≈205). Signalerna från kraftplattan och armstöden samplades in med samplingsfrekvensen tio Hz.

Genomförande

Försökspersonerna ombads sitta upprätta, raka i ryggen och avslappnade. Vidare skulle benen hänga fritt icke korslagda och personerna placerades på plattan så att sätet var i bakkant med kraftplattan. Vid de mätserier där ögonen skulle vara öppna uppmanades försökspersonen att fixera en punkt, som fanns ungefär två meter framför ögonen i ögonhöjd. I fyra av mätserierna satt försökspersonerna med armarna i kors över bröstet och i en av mätserierna vilade armarna på stolens armstöd med armbågarna i bakkant av armstöden. För att inte distraheras av omgivningen fick försökspersonerna lyssna på musik av Mozart i hörlurar under alla fem mätserierna. De fem sorters mätserier som genomfördes var:

- Vibrationsstimulering med öppna ögon och armarna korslagda över bröstet
- Vibrationsstimulering med slutna ögon och armarna korslagda över bröstet
- Vibrationsstimulering med slutna ögon och armarna vilande på armstöden
- Galvanisk stimulering med slutna ögon och armarna korslagda över bröstet
- Galvanisk stimulering med öppna ögon och armarna korslagda över bröstet

Ordningen de fem olika mätserierna utfördes i byttes för varje försöksperson. Dock utfördes de tre vibrationsförsöken alltid före de två galvanikförsöken.

Identifieringsmetoder

Vid identifieringen användes både parametriska och icke-parametriska metoder, för att försöka verifiera den tidigare beskrivna modellen, eller hitta något annat samband. Till hjälp vid identifieringen användes programvarupaketet Matlab.

Parametrisk Identifiering

Maximumlikelihoodanpassning av en ARMAX-modell, för olika modellordningar, gjordes mellan insignalen och det balanserande momentet. Modellen försökte valideras genom simulering. Vidare undersöktes autokorrelationen av residualen och korskorrelationen mellan residualen och insignalen, för att utröna om modellen har fångat upp sambandet mellan in- och utsignal. Samtliga delar gjordes för både saggitalt och lateralt moment som utsignal.

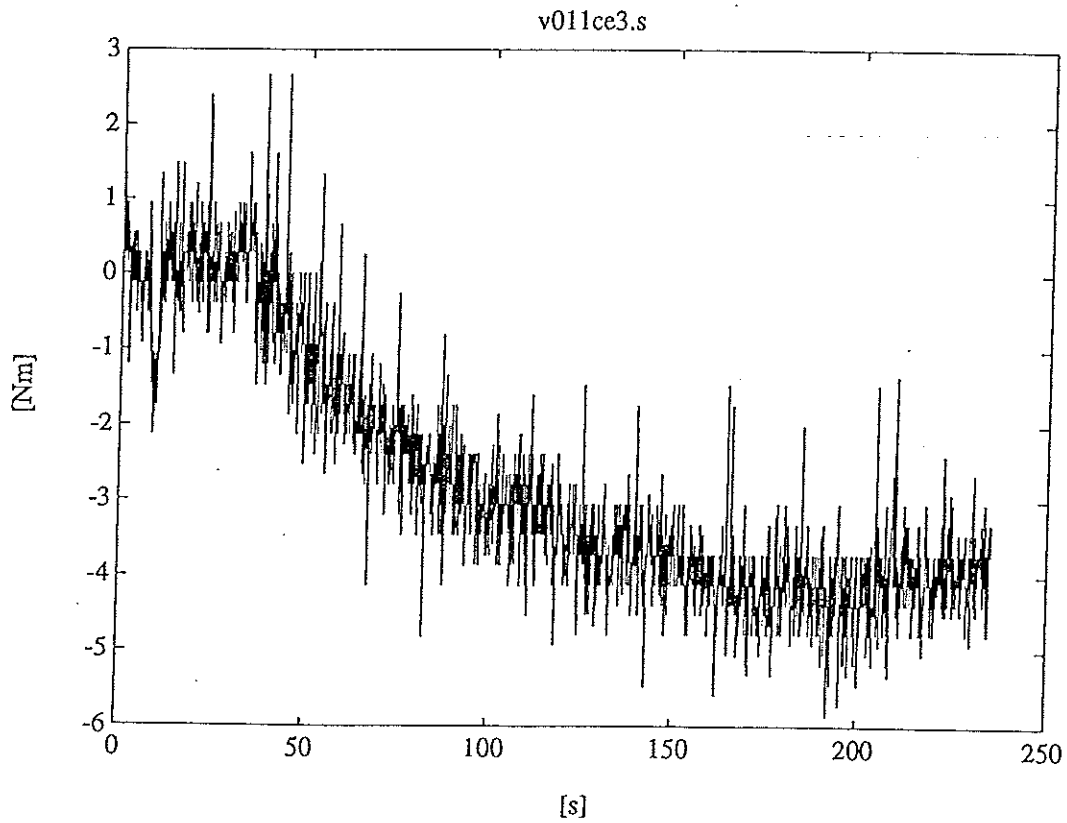
Ickeparametrisk Identifiering

Varje mätserie delades upp i fem tidsintervall, där första intervallet var de trettio inledande sekunderna utan stimulering, och variansen för varje intervall samt hela mätserien beräknades för att försöka se om stimuleringen påverkar amplituden hos försökspersonernas svaj. Vidare studerades insignalens och det balanserande momentets spektrum och överföringsfuntionen beräknades. Beräkningarna gjordes för både saggitalt och lateralt moment som utsignal.

Testresultat

Momentberäkningar och identifieringsförsök av mätserierna har gjorts med hjälp av programvarupaketet Matlab. I appendix 8 finns mätserierna presenterade. En person är ej medtagen vid galvanisk stimulering öppna ögon p g a att mätserien avbröts. Dessutom är en försöksperson helt borttagen p g a medicinering som kunde påverka balansfunktionen. Graferna visar hur försökspersonens sagittala och laterala moment varierar. Momenten är nolljusterade så att medelmomentet över hela mätperioden är noll. Fem olika moment är medtagna.

- Galvanikstimulering öppna ögon sagitallt moment armarna över bröstet
- Galvanikstimulering slutna ögon sagittalt moment armarna över bröstet
- Vibrationsstimulering öppna ögon sagittalt moment armarna över bröstet
- Vibrationsstimulering slutna ögon sagittalt moment armarna över bröstet
- Vibrationsstimulering slutna ögon lateralt moment underarmarna mot armstöden



figur 11. Ett exempel på hur en försöksperson successivt förflyttar sin tyngdpunkt bakåt. Grafen visar det sagittala momentet efter nolljustering vid vibrationsstimulering.

I mätserierna med vibrationsstimulering visar det sagittala momentet upp en tydlig trend (se figur 11 samt appendix 8). Momentet minskar systematiskt allteftersom en mätning fortlöper. Detta fenomen uppträder ej vid den galvaniska stimuleringen. Detta tyder på att försökspersonerna på något sätt påverkas av vibrationsstimuleringen. Tidigare forskning [13] tyder på att den typ av galvanikstimulering vi gör i huvudsak påverkar det laterala momentet.

Några säkra tecken, på att den påförda stimulin (vibration och galvanik) ökade svajet, kunde inte visas. Ur spektrum för in- och utsignalen kunde vi inte heller se några genomgående trender mellan försökspersonerna.

Försök gjordes även med att identifiera lateralt och sagittalt moment var för sig med en ARMAX-modell. Om den tidigare beskrivna mekaniska modellen och återkopplingen med en PID-regulator är giltig, så skulle en tredje ordningens ARMAX gå att validera. Denna och även högre modellordningar prövades men ingen gick att styrka. Med vibrationsstimulering och sagittalt moment gjordes även försök med att ta bort den systematiska förändringen av momentet och beräkna en modell för återstoden, dock med negativt resultat. En skillnad mellan de försök som gjorts här och de som tidigare gjorts för stående personer är storleken på det balanserande momentet. Momentutslaget för våra försök är avsevärt mindre än för stående personer.

Mätserierna med vibrationstimulering och armbågarna mot armstöden visar att en betydande del av momentet tas upp via dessa (se appendix 8). De armstödsmoment som presenteras i appendix 8 är de bägge armstödens moment omräknade till modellens stabiliserande moment och adderade till varandra. Momentet från armstöden måste betraktas med en viss reservation, då felen vid de approximationer som görs vid momentberäkningen ej är helt klarlagda.

Diskussion

Syfte

Vår uppgift var att konstruera och utvärdera en mätutrustning för att mäta balansen hos sittande personer. I vår konstruktion av en stol utrustad med en kraftplatta och kraftgivarförsedda armstöd återstår dock en del problem vad gäller armstödsmätningen och sittställningen att lösa.

Armstödsmätning

Ett problem vid mätningen av det balanserande momentet var att mäta den del som togs upp i armstöden. I den nuvarande konstruktionen mäts momentet från armstödet kring en vridpunkt under stolen. Detta moment antas sedan vara proportionellt mot den del av det moment som balanserar kroppen, då försökspersonen tar emot sig mot ena armstödet. Detta är sant, om den kraft som angriper i armstödet har en bestämd riktning. Om denna riktning dessutom är känd, så kan proportionalitetskonstanten beräknas.

För att förbättra armstödsmätningen kan man tänka sig en vidareutveckling på två olika sätt.

- Försöka bestämma hur kraftens riktning är beskaffad och eventuellt beror på armstödens höjd och försökspersonernas prestanda (längd, vikt och om försökspersonen redan i stationärt tillstånd belastar armstödet).

- Tillfoga ytterligare två trådtöjningsgivare till vardera armstöd. Dessa skulle t ex kunna fästas utmed armstödens vertikala balkar på ett sådant sätt att den horisontella kraften mot armstödet skulle kunna mätas. Då denna är känd kan den vertikala kraften beräknas med hjälp av de tidigare givarna. Ur dessa uppgifter kan den kraft som bidrar till det balanserande momentet exakt beräknas.

Att överhuvudtaget använda armstöd är ett problem då försökspersonen får ytterligare en referens (ny insignal) via hudens mekanoreceptorer då armstöden vidrörs. Den extra referensen kan påverka systemet. Ett känt exempel som styrker detta är då en person med balansstörning står upp och blundar och därmed tappar balansen. Hon återfår delvis denna genom att en ytterliggare referens tillkommer i form av en hand som läggs på axeln.

Sittställning och stimuli

Vi har i våra försök använt två olika sittställningar, en då försökspersonen satt med armarna i kors över bröstet och en då underarmarna vilade mot armstöden. För att få så lika villkor som möjligt och av beräkningstekniska skäl, ändrades ej armstödens höjd mellan de olika försökspersonerna. Ett problem med detta var att det visade sig att försökspersonerna fick olika långt ned till armstöden. Detta medverkade till att överkroppsställningen varierade mellan försökspersonerna och mätsituationen kom då att variera mellan försökspersonerna.

Vid försöken användes två olika stimuli. Dels vibratorer på nackmuskulaturen och dels galvanisk stimulering av balansnerven via elektroder placerade på mastoidbenet.

I mätserierna med vibrationsstimulering finns en klar trend. Allteftersom mätserien fortskrider förflyttas personens tyngdpunkt systematiskt bakåt. Det finns flera tänkbara orsaker till detta. En orsak skulle kunna vara att försökspersonen blir trött och sjunker ihop allteftersom mätserien fortlöper. En annan anledning skulle kunna vara att personen påverkas av vibrationsstimuleringen och förflyttar tyngdpunkten bakåt för att lättare sitta upprätt. Det går ej att finna någon systematisk tyngdpunktsförskjutning i sagittalled i försöken med galvanikstimulering. Vår slutsats är att vibrationsstimuleringen på något sätt påverkar försökspersonen så att överkroppens masscentrum lutas bakåt. För fortsatta försök med sittutrustningen måste det utredas vad som ligger bakom den systematiska förskjutningen av tyngdpunkten eller alternativt finna en sittställning eller stimulering där fenomenet ej uppträder. De vibratorer som användes i försöken vibrerar med relativt hög effekt. Vibratorer med lägre (120 mW, amplidud 0.4 mm och frekvens 60 Hz) effekt applicerade på vadmuskulaturen har framgångsrikt använts vid försök med stående personer [8]. Ett tredje sätt skulle kunna vara att försöka modellera denna långsamma trend separat.

Slutsats

- Utrustningen fungerar men kan förbättras. Armstöden bör kompletteras med ytterligare givare.

- Vi har visat att man kan studera balansfunktionen i sittande ställning, men för friska människor blir utslagen (påverkan av stimuli) små.

- Vi kunde ej statistiskt säkerställa systemidentifiering men påvisade en trend i sagittalled vid vibrations-stimulering.

Tillkännagivanden

Vi vill tacka våra handledare,
Rolf Johansson, LTH och Måns Magnusson, ÖNH-kliniken
för deras stöd, inspiration och goda råd.

Vi vill också tacka:
Per-Anders Fransson och Mikael Åkesson, ÖNH-kliniken, för hjälp med dator och programvara.
Zivorad Zivkovic, LTH för all hjälp med stolkonstruktion och klistring av trådtöjningsgivare.
Rolf Braun, LTH för hjälp med kretskortstillverkningen.
Lars Thomasson, MTA-Lund, för tips och råd gällande förstärkarkonstruktion.

Referenser

[1] Analog Devices, "Wide bandwidth strain gage signal conditioner".

[2] B Broberg "Teknisk mekanik för E", KF- Sigma 1990.

[3] P-A Fransson, Postcon 2.0, Användarhandledning.

[4] P-A Fransson, P Sundström, "Visual contribution in human postural control". Appendix 4. Examensarbete-5436, Reglerteknik, LTH, Lund.

[5] Grahm, Jubrink, Lauber, "Modern elektrisk mätteknik, 1". Bokförlaget Teknikinformation s.17- 31 1990.

[6] Grahm, Jubrink, Lauber, "Modern elektrisk mätteknik, Givare ". Bokförlaget Teknikinformation s.17- 32 1990.

[7] Guyton,"Textbook of medical physiology", seventh edition. W.B. Saunders company 1986, p. 608-614, 572-580 and 133.

[8] R Johansson, M Magnusson och M Åkesson, "Identification of human postural dynamics" IEEE Trans. Biomed. Eng., vol.35 p.858-869 no10 oct. 1988.

[9] R Johansson, M Magnusson ,"Human postural dynamics" CRC, Biomedical Engineering, vol.18 p413-437 1991.

[10] Logitech, Modula-2 version 3.0, Users manual.

[11] J LMeriam, L G Kraige, Mechanics, vol 1,2. Statics, John Wiely & sons. (1987).

[12] H Rundcrantz (red) Öron- näs- och halssjukdomar. Studentlitteratur 1982. s 120-128.

[13] J.Wiklund,"A study of the body sway induced in humans by galvanic stimulation of the vestibular nerve: The phenomen. A model. Parametric identification." Examensarbete-5392, Reglerteknik, LTH, Lund.

Appendix 1 - Armstödsmätning felkällor

Nedan studeras hur väl den sökta kraften mot armstöden mäts för olika riktningar hos den verkliga kraften. Tre fall tas upp. Använda beteckningar är definierade i figur 7 och 8. Följande värden på konstanter används:

$\beta = \arctan(57/12) \approx 78°$

$\alpha = \arctan(30/30) \approx 45°$

$\xi = \beta - \alpha = 33°$

I fall 1 antar vi att den verkliga kraften, F är riktad rakt utåt (åt höger i figuren). Den sökta kraften som bidrar till modellens balanserande moment, F_l och den kraft vi verkligen mäter, F_k blir då:

$F_l = F * \cos(90 - \alpha) \approx 0.707$

$F_k = F * \cos(90 - \alpha - \xi) = F * \cos(90 - \beta) \approx 0.978$

$F_k / F_l = 0.978 / 0.707 \approx 1.38$

Den kraft som mäts blir alltså 38% större än den som söks.

I fall 2 antar vi att den verkliga kraften, F är riktad rakt nedåt. Den sökta kraften som bidrar till modellens balanserande moment, F_l och den kraft vi verkligen mäter, F_k blir då:

$F_l = F * \cos(\alpha) \approx 0.707$

$F_k = F * \cos(\alpha + \xi) = F * \cos(\beta) \approx 0.208$

$F_k / F_l = 0.208 / 0.707 \approx 0.29$

Den kraft som mäts blir alltså 29% av den som söks.

I fall 3 antar vi att den verkliga kraften, F är riktad 45grader? från horisontalled och nedåt. Den sökta kraften som bidrar till modellens balanserande moment, F_l och den kraft vi verkligen mäter, F_k blir då:

$\sqrt{2} * F_l = F * (\cos(90 - \alpha) + \cos(\alpha)) \approx 1$

$\sqrt{2} * F_k = F * (\cos(90 - \alpha - \xi) + \cos(\alpha + \xi)) = F * (\cos(90 - \beta) + \cos(\beta)) \approx 1.19$

$F_k / F_l = 1.19 / 1 \approx 1.19$

Den kraft som mäts blir alltså 19% större än den som söks.

Ovanstående tre fall illustrerar hur olika riktningar hos den angripande kraften bidrar med olika stora mätfel då kompensering för de olika hävarmarna är gjorda. Om kraftens riktning är känd kan felet naturligtvis kompenseras fullt ut. Är kraften ungefär känd borde felet till stor del kunna kompenseras bort.

Appendix 2 - Trådtöjningsgivare kopplad i helbrygga

Trådtöjningsgivarekvationer

För trådtöjningsgivare gäller enligt [6] allmänt:

$$k = \frac{\frac{\Delta R}{R}}{\frac{\Delta L}{L}} \qquad \text{.....(1)}$$

där R är resistansen, ΔR är resistansförändringen och $\frac{\Delta L}{L}$ är töjningen ε. k anses vara ungefär 2.

Givarna fastlimmas på utvalt material , maskinstål 1650, där $\sigma_{töj}$ = 32 kp/mm^2 samt elastisitetsmodulen E = 20 * 10^{10} N/m^3 gäller.

Hookes lag ger töjningen ε i materialet:

$$\sigma = \varepsilon * E \Rightarrow \varepsilon = \frac{\sigma}{E} = 10^{-3}$$

vid ytan av stålet.

Då $\frac{\Delta L}{L} = \varepsilon$ samt $k = 2 \quad \Rightarrow \quad \frac{\Delta R}{R} = 2*10^{-3}$

Då detta är känt kan en uppskattning av utsignalen från bryggan göras.

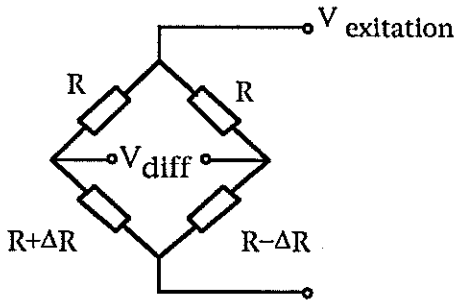
Bryggkoppling

För trådtöjningsgivare kopplad i helbrygga enligt figur 1 gäller följande beräkningar.

$$V_{diff} = V_- - V_+$$

$$V_- = \frac{R}{R + (R + \Delta R)} * V = \frac{V}{2 + \frac{\Delta R}{R}} = \frac{V}{2.002}$$

$$V_+ = \frac{R}{R + (R - \Delta R)} * V = \frac{V}{2 - \frac{\Delta R}{R}} = \frac{V}{1.998}$$



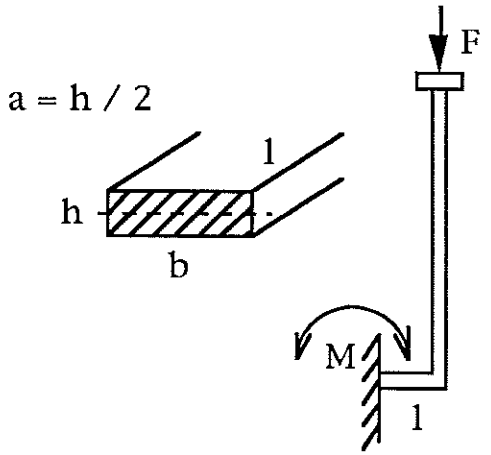
Figur 1. Trådtöjningsgivare kopplade i helbrygga.

$V_{diff} = V_- - V_+ = 0.001 V_{excitation}$
Med en förstärkning på t ex 2000 ggr och exciteringsspänning på 5 V

$\Rightarrow$

$V_{diff} = 10$ volt.

En beräkning av utsignalen från instrumentförstärkaren kan enkelt göras genom att först anta att det finns en kraft F på armstödet riktad t ex i vertikalled, se figur 2.



Figur 2. En kraft på armstödet ger upphov till en nedböjning, vilket i sin tur ger en resistansändring i givaren och en spänningsökning från förstärkaren.

Då kraften F och armlängden l är givna kan momentet, M och böjmotståndet,W bestämmas och därur kan spänningen σ beräknas. Ty,

$\sigma = M/W$ där

W = I/a , I är böjstyvheten och a är medellinjen hos balken.
Böjstyvheten I är för en stång med rektangulärt snitt är

$I = 12a/bh^3$

Alltså är

$$\sigma = \frac{M}{W} = \frac{F\,l}{\frac{I}{a}} = \frac{6\,F\,l}{b\,h^2} = 1.70\ \mathrm{N/mm^2}$$

Via formeln (1), kan spänningen σ överföras till en resitansändring och vidare till en spänningsökning U och med samma värden på excitationspänning och förstärkning som ovan ger det :

$$\frac{\Delta R}{R} = \varepsilon * k = \frac{\sigma}{E} * k = 1.7*10^{-5} \Rightarrow U = 0.17 \text{ volt}$$

Appendix 3 - Nedböjning av armstöd

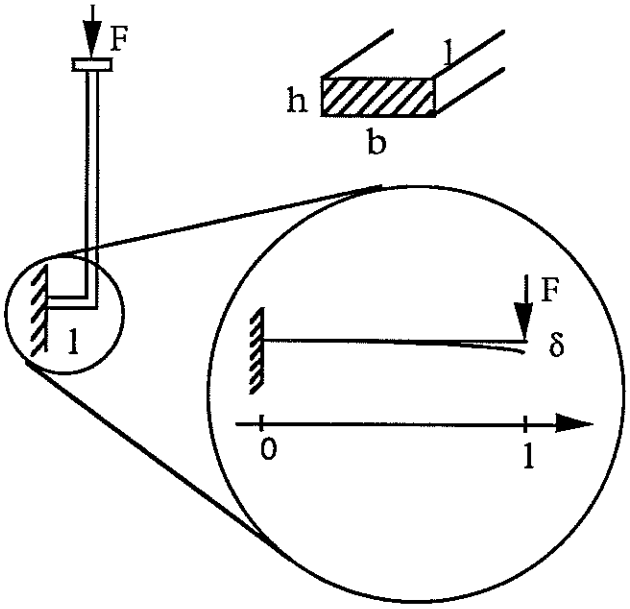


Bild 1. Kraft på ett armstöd samt en detaljbild av nedböjningen, δ.

Då en balk utsetts för en kraft enligt bild 1 böjs den ned en sträcka $\delta = v$.

Den bestäms [2] av

$$v'' = -\frac{M}{EI}$$

$$v' = -\frac{M}{EI}x + C$$

$$v = -\frac{M}{EI}\frac{x^2}{2} + C\,x + D$$

Där balken är infäst sker ingen nedböjning eller förändring. Alltså gäller följande randvillkor:

Randvillkor $v(0) = 0$

$$v'(0) = 0$$

$\Rightarrow$

$$v = -\frac{M}{EI}\frac{x^2}{2} \;;\; I = \frac{bh^3}{12}$$

Då $x = l$ och $v = \delta \Rightarrow$

$$\delta(F) = \frac{Fkl^2}{2EI} = \frac{510*F}{2*20*10^4*\frac{35*11}{12}} * 121^2 \Rightarrow$$

Antag t ex kraften F = 300N respektive F = 500N, detta ger

$\delta(300) = 1.44$ mm
och
$\delta(500) = 2.40$ mm
vilket är ett rimligt värde på nedböjningen.

Appendix 4 - Förstärkaren

Vi valde att bygga upp en förstärkare kring en krets, 1B31AN, som innehåller variabel förstärkare,exciteringsspänning och filter i en och samma kapsel.

Förstärkning

Förstärkningen (G) som sträcker sig upp till 5000 gånger, bestäms till önskat värde genom att justera trimpotentiometern, R_G, vilken sitter anslutet mellan pin 2 och pin 3.

$$G = 2 + \frac{80\ k\Omega}{R_G}$$

Filter

Det finns ett tvåpoligt lågpassfiler att tillgå som har en intern gränsfrekvens (-3 dB punkt) vid 1000 Hz. Gränsfrekvensen går dock att minska genom att koppla in kapacitancer på pin 12 till jord (C_{f1}) respektive pin 13 till pin 14 (C_{f2}).

$$C_{f1} = 0.015\ \mu F\ [\ \frac{1kHz}{f_c} - 1\ ]$$

$$C_{f2} = 0.0022\ \mu F\ [\ \frac{1kHz}{f_c} - 1\ ]$$

För att filtrera bort eventuellt nätbrum bestämdes gränsfrekvensen till 20 Hz . Alltså valdes kondensatorerna C_{f1} och C_{f2} till

$C_{f1} = 68\ \mu F$
$C_{f2} = 100\ nF$

(Gränsfrekvensen går även att öka upp till 20 000 Hz [1])

Excitationspänning

Exciteringsspänningen, som är förinställd till 10 V, går att variera steglöst mellan 4 och 15 V. I det här fallet är det önskvärt att ha en så låg exciteringsspänning som möjligt då termiska bruset ökar med högre spänningar. Detta åstadkoms enklast genom att ansluta en trimpotentiometer på 20 kΩ mellan pin 19 och pin 20.

För övriga data se datablad [1] över kretsen.

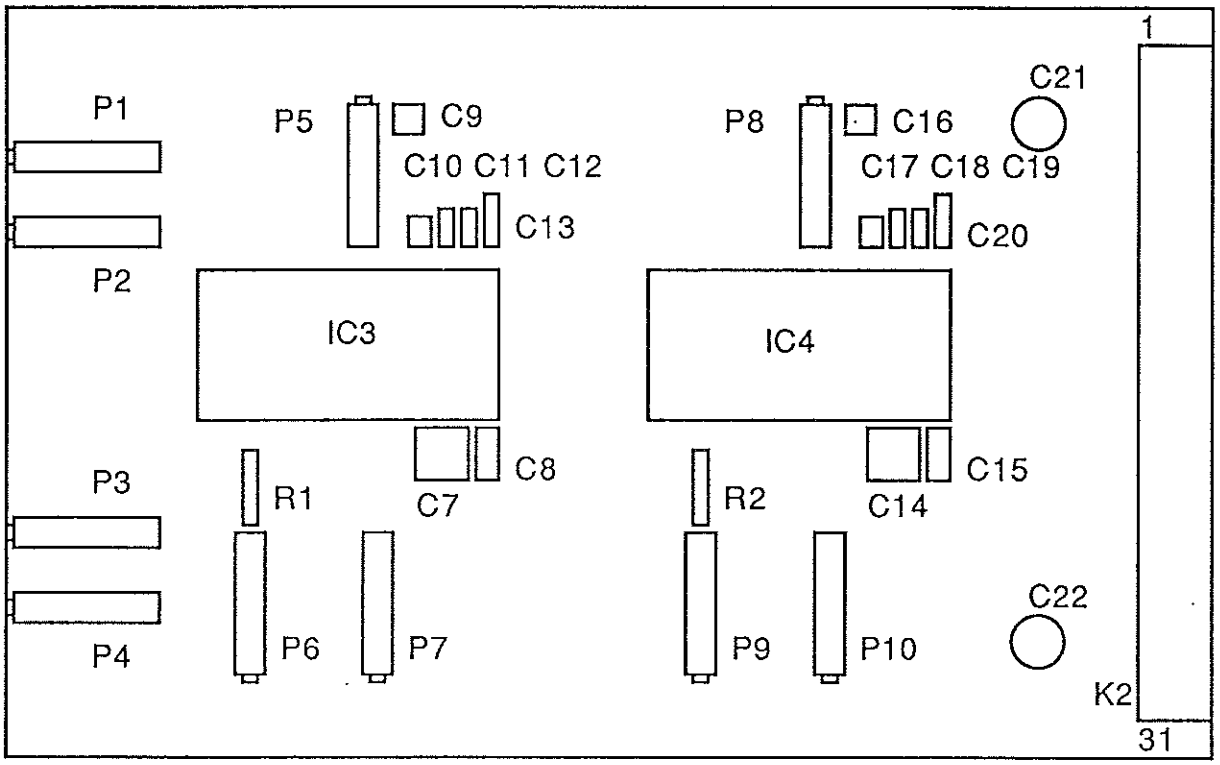
Nätaggregatet

Konstruktionen av nätaggregatet gjordes på traditionellt sätt genom att likrikta signalen från en transformator med hjälp av dioder. Med

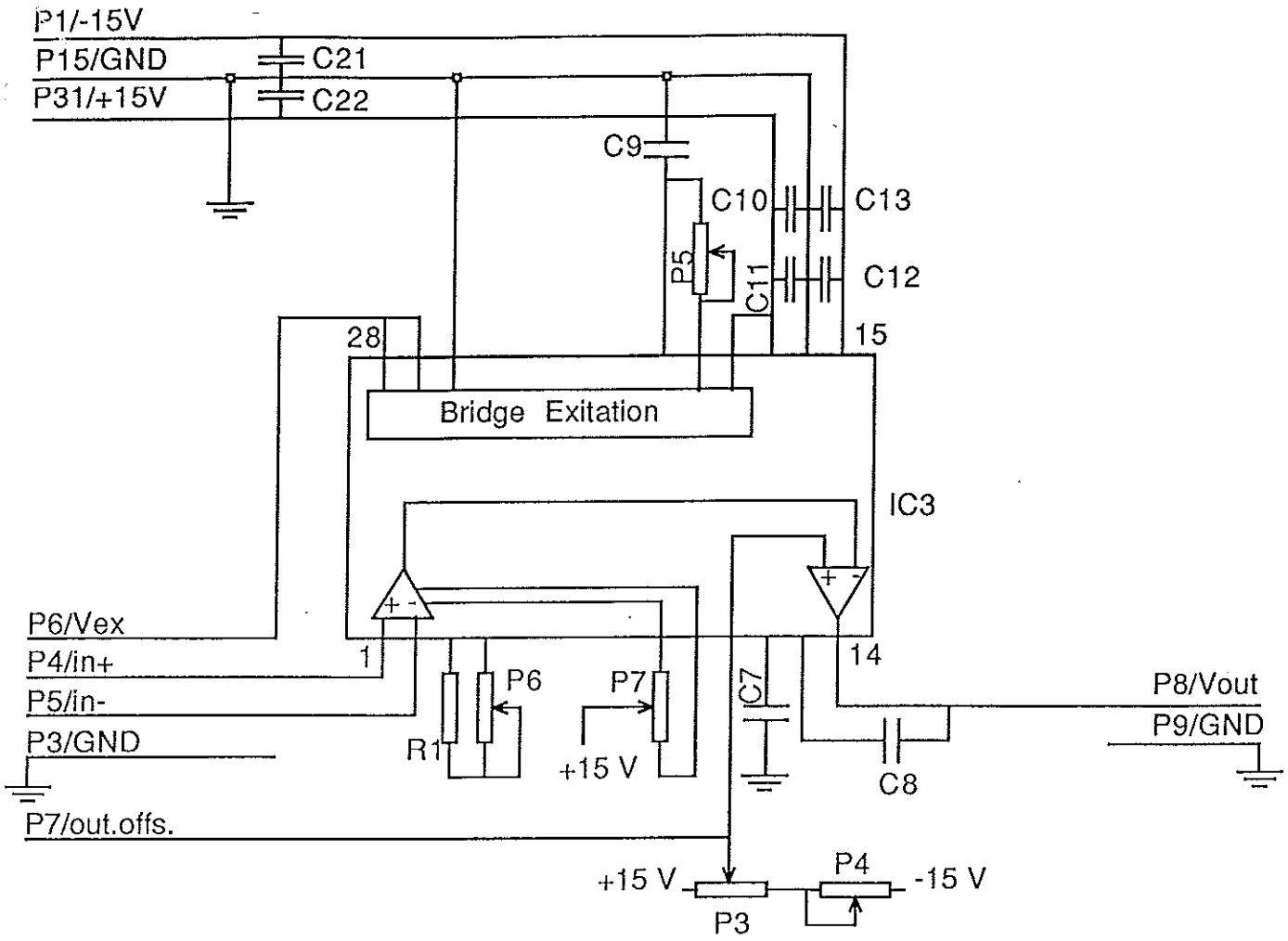
erfoderliga kondensatorer och spänningsstabilisatorer erhålls en matningsspänning till förstärkarkortet på ±15 V. Värden på kondensatorerna erhölls från datablad över spänningstabilisatorn.

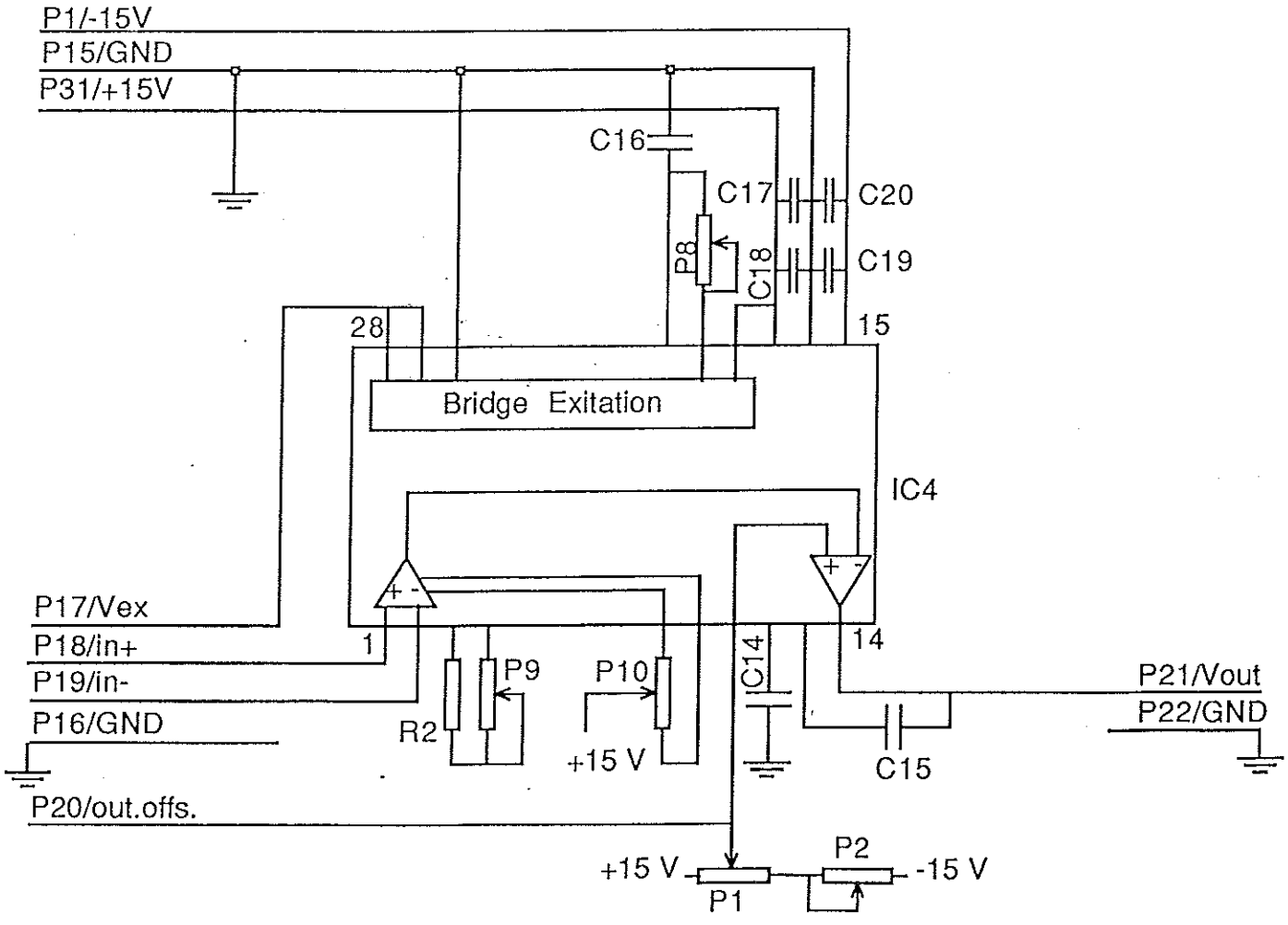
Nedan följer schema samt layout över förstärkare och spänningsaggregat.

Layout över förstärkaren

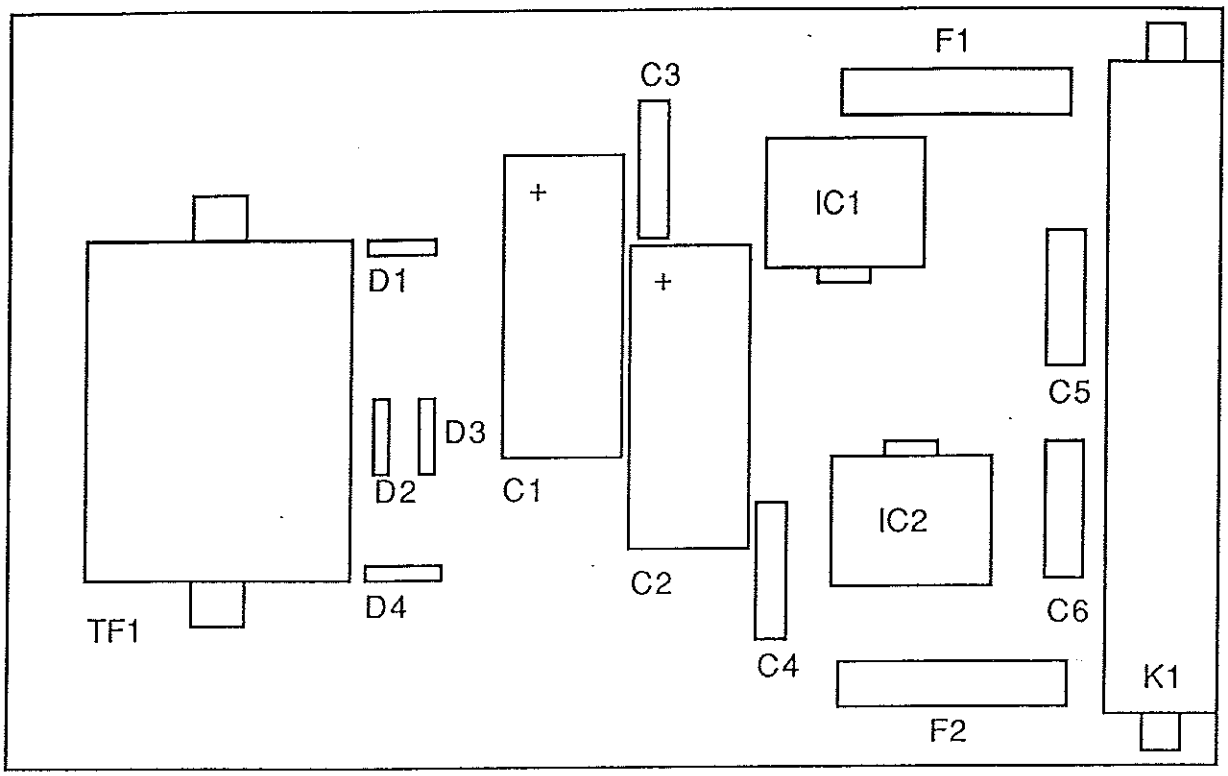


Schema över förstärkaren

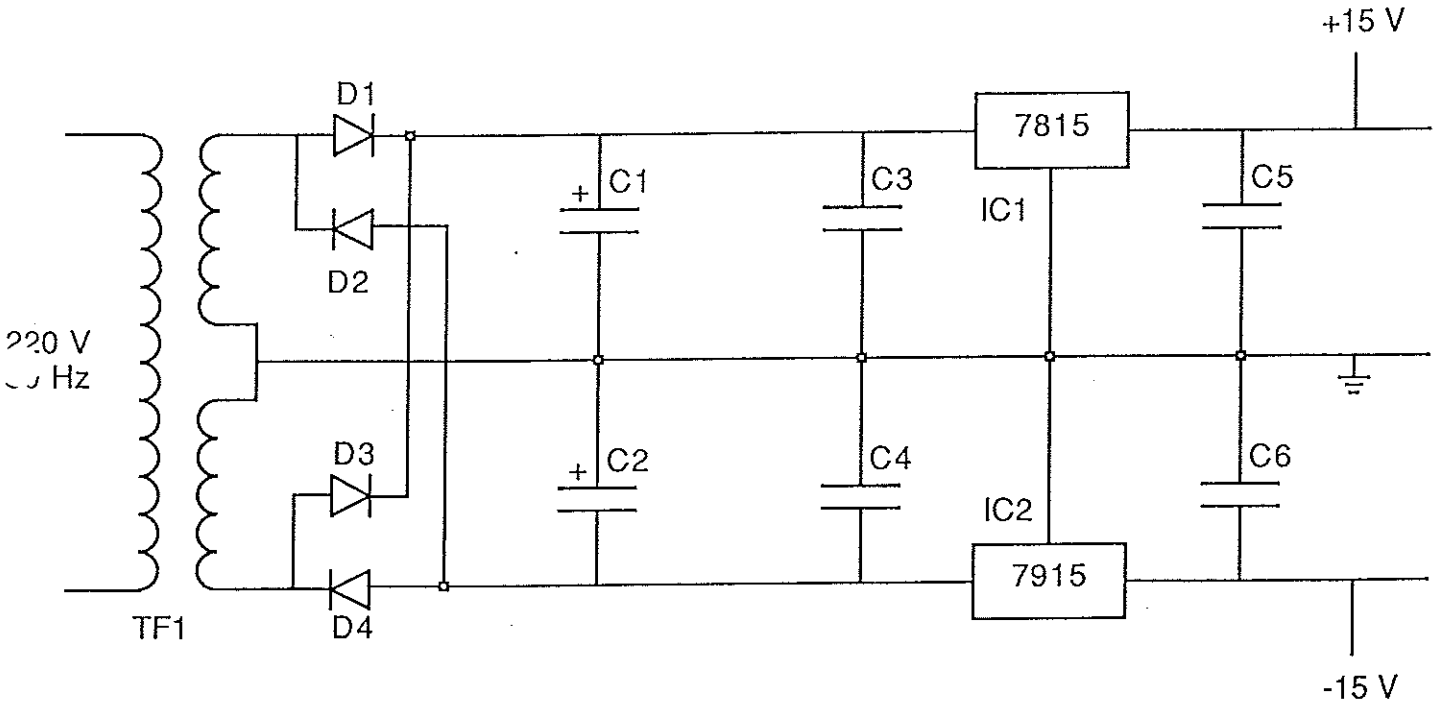




Layout över spänningsmatningen.



Schema över spänningsmatningen.



Komponentlista, kretskort

Spänningsmatning

Transformator	
TF1	4,5 VA
IC-kretsar	
IC1	7815 Spänningsstab. +15V
IC2	7915 Spänningsstab. -15V
Dioder	
D1-D5	1N4007
Kondensatorer	
C1,C2	4700uF
C3,C4	330nF
C6,C6	1uF
kontaker	
K1	Europakontakt

Instrumentförstärkare

IC-kretsar	
IC3,IC4	1B31AN
Potentiometrar	
P1,P3	50kΩ
P2,P4	1kΩ
P5,P8	20kΩ
P6,P9	50Ω
P7,P10	10kΩ
Kondensatorer	
C7,C14	68uF
C8,C15	100nF
C9,C16	4.7uF
C10,C13,C17,C20	1uF
C11,C12,C18,C19	1000pF
C21,C22	47uF
Motstånd	
R1,R2	15Ω
Kontakter	
K2	32-pol Europakontakt

Kontakter

31-pol europakontakt

1	-15V
2	NC
3	GND (vänster armstöd)
4	in+ IC3.1
5	in- IC3.2
6	Vex IC3.28
7	outputoffsrt adj. IC3.11π
8	Vout IC3.14
9	GND
10	Sense high
11	Sense low
12	NC
13	NC
14	NC
15	GND
16	GND
17	Vex IC4.28
18	in+ IC4.1 (höger armstöd)
19	in- IC4.2
20	outputoffsrt adj. IC4.11
21	Vout IC4.14
22	GND
23	Sense high
24	Sense low
25	NC
26	NC
27	NC
28	NC
29	NC
30	NC
31	+15 V

15-pol D-sub kontakt

1	GND
2	in+ IC3.1
3	in- IC3.2
4	Vex IC3.28
5	Sense high
6	Sense low
7	NC
8	NC
9	NC
10	GND
11	in+ IC3.1
12	in- IC3.2
13	Vex IC3.28
14	Sense high
15	Sense low

Appendix 5 - Kalibreringsprotokoll

Förstärkaren

Innan kalibrering skall utföras krävs en uppvärmningstid för förstärkaren på 5 min och för trådtöjningsgivarna på stolen 30 min.

Kalibrering av utrustningen sker i följande steg där *excitationsspänning* är den spänning som trådtöjningsgivarna matas med och *input/output* offset är kompensering för felnivå i in- resp. utgång. *Förstärkningen* kan varieras steglöst.

1 *Excitationsspänning*
a Justera potentiometer P5 *(P8)* så att önskad spänning erhålles på pin 28 ($4V < V_{exc} < 10V$).

2 *Input offset*
a. Anslut pin 1 och pin 2 till jord.
b. Justera potentiometer P7*(P10)* så att 0V mäts på pin 8.
c. Avlägsna kortslutningen.

3 *Förstärkning*
a. Anslut givaren till kontakt K1.
b. Anslut utgången, kontakt K2 *(K3)* till datorn.
c. Justera potentiometer P6 *(P9)* så att en belastning mitt på armstödet på 10 kg motsvarar 1.20 V på kontakt K2 *(K3)*.

4 *Output offset*
a Justera med en voltmeter, potentiometer P1*(P3)* (grovjustering) och P2 *(P4)* (finjustering) så att 0V erhålles på kontakt K2 *(K3)*.

Text inom parentes avser höger armstöd.

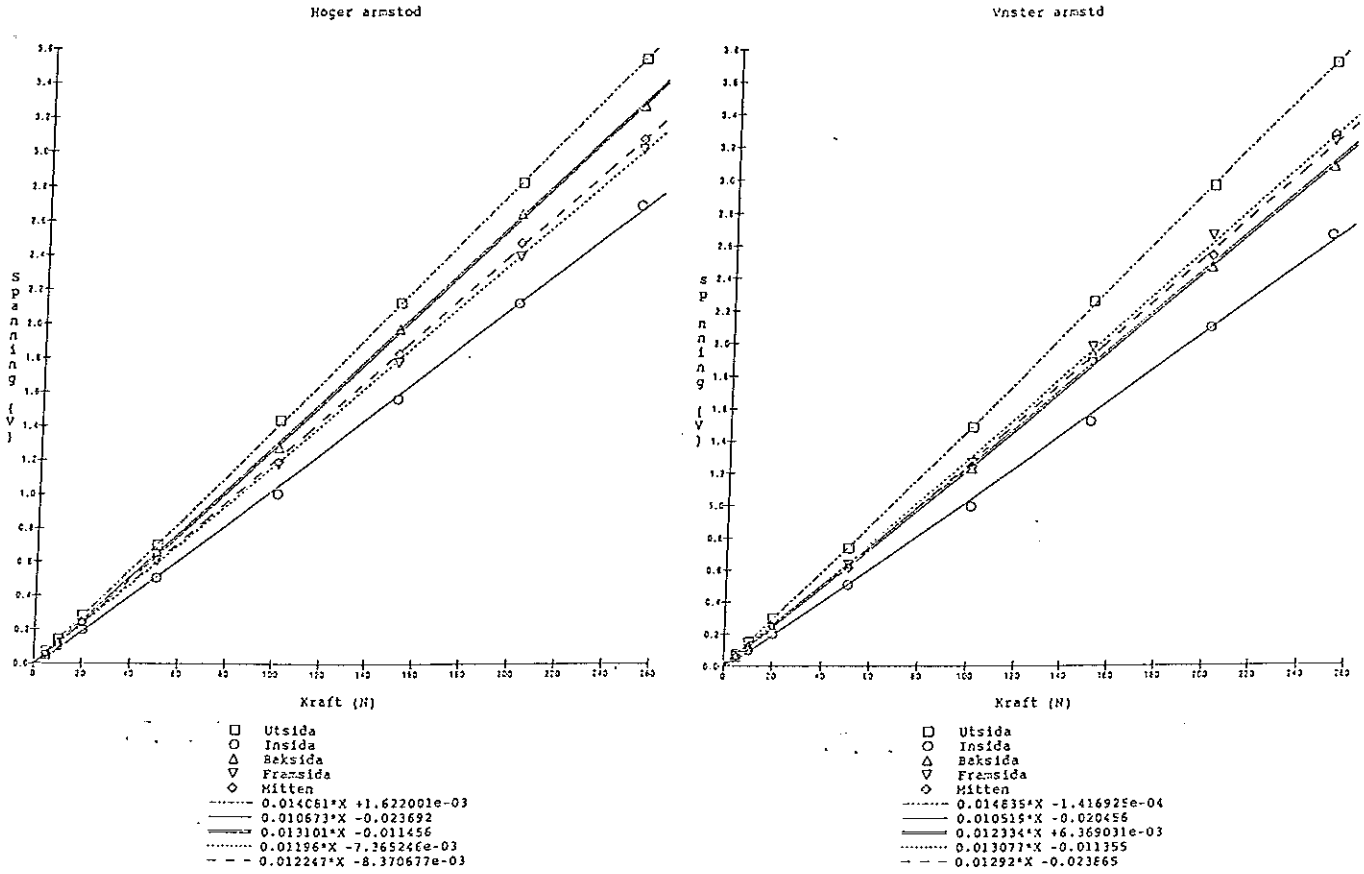
Linäritet

För att undersöka lineäriteten lades vikter om 0.5 kg till 25 kg på armstöden och utspänningen på kontakt K2 (K3) noterades. Se figur 1. Ytterligare en mätserie utfördes där vikterna placerades i omvänd ordning (25 kg - 0.5 kg).

Hysteres

Efter mätserierna konstaterades en hysteres som visade sig som en offsetdrift på 38 mV (0.5 - 25kg) respektive 20 mV (25 - 0.5kg) vilket motsvarar 100 - 200 gram.

Vid försöken uppstod också en svårförklarlig, icke systematisk hysteres där utspänningen kunde bli både positiv och negativ.



Figur 1. Kalibreringskurva över höger och vänster armstöd.

Datorinställningar

Följande datorinställningar i programmet Postcon3 användes vid försöken. (Indragningar i texten betyder ett steg ned i menyn i programmet.)

Systempar

Loadc.E 10Nm/V

Stimuli

vibration

PRBS

period	0.8
MaxTestTime	235s

Vibration

PRBS

period	0.8
centerspeed	0
Amplitide	8V
MaxTestTime	235s

Inputsignals

elbow&plate

Plot

sti&NPo&EPo

Appendix 6 - Användarhandledning program

Nedan kommer en beskrivning av de val och menyer som skiljer postcon2 och postcon3 åt samt en redogörelse för filformatet hos mätserier lagrade på externa filer. I övrigt hänvisas till Manual Postcon 2.0 [3] för en fullständig användarhandledning av postcon2.

Skillnader mellan postcon2 & postcon3

I huvudmenyn har en ny rubrik, input signals, tillkommit. Om denna väljs så kommer två alternativ att ges, plate och elbow&plate. Om plate väljs så kommer endast kraftplattans signaler att läsas in för vidare behandling och om elbow&plate väljs så kommer kraftplattans och armstödens signaler att läsas in.

Av de redan tidigare befintliga rubrikerna har förändringar gjorts i systempar och plot. I systempar har konstanten som i postcon2 kallas loadcoef döpts om till loadcoefp. en ny konstant loadcoefe har kommit till och är en omräknings faktor från V på AD-omvandlaren till Nm vid plottning på skärmen. I plot har elbow&xymoment tillkommit som gör det möjligt att plotta armstödsmomenten under mätningarna. I plot har även calibration tillkommit. Om calibration väljs så plottas AD-ingångarnas värden på skärmen för att underlätta nolljusteringen av förstärkarna.

Filformat för mätserier lagrade med postcon3

Den fil som genereras då en mätserie valts att lagras på en extern fil får extension .mat och kan direkt lagras in i matlab. I matlab läses filen in som en matris där första raden innehåller tre siffror som beskriver mätning och stimulering. Därefter fylls raden ut med nollor till det totala antalet mät- och stimuleringsvärden. De tre siffrorna står för antal stimuleringar, ett mätindex och ett stimuleringsindex. Mätindex och stimuleringsindex beskrivs i tabellen nedan.

Mätindex	Mätning
0	inga mätsignaler är lagrade
2	Av mätsignalerna så är kraftplattans sex signaler lagrade.
4	Av mätsignalerna så är kraftplattans sex signaler och de två armstödens moment lagrade.

Stimuleringsindex	Stimulering
0	Ingen stimulering
1	Galvanisk stimulering
2	Vibrationsstimulering
3	Vibrations och galvanisk stimulering

De nästföljande raderna innehåller kraftplattans och eventuellt armstödens sampelvärden. Varje rad motsvarar ett sampel och antalet kolumner beror på vilka mätsignaleroch och antal stimuleringar som lagras. Om maximalt antal mät och stimuleringssignaler lagras är ordningsföljden för ett sampel på en rad:

Fn1 Fn2 Fn3 Fx Fy1 Fy2 ElbowLeft ElbowRight Vib Galv

Om färre mät- och stimuleringssignaler är lagrade så stryks dessas kolumner.

Appendix 7 - Programlista

På följande sidor presenteras en utskrift av de programmoduler postcon3 består av.

```
 1 MODULE Postcon2;
 2
 3 (* Version 2.0b *)
 4 (* observ that this version is modified for use in compaq-386 *)
 5 (* with special connections to force plattform. The change is *)
 6 (* the input output connection and the loss of visual stimuli. *)
 7 (* Out Channel 0 vibration stimuli*)
 8 (* Out Channel 1 galvanik stimuli*)
 9 (* In Channel 0 fx-angle *)
10 (* In Channel 1 fy1-angle *)
11 (* In Channel 2 fy2-angle *)
12 (* In Channel 3 fn1-angle *)
13 (* In Channel 4 fn2-angle *)
14 (* In Channel 5 fn3-angle *)
15
16  IMPORT RTMouse,DebugPMD;
17
18  FROM Decl          IMPORT PBox,MBox,DBox,DRBox,ErrorBox,MailType,
19                            TheEnd,InitMonitors,OutBox,Error;
20
21  FROM Safe          IMPORT SafeProc;
22
23  FROM Measure       IMPORT DataMeasure;
24
25  FROM Control       IMPORT InitControl;
26
27  FROM HelpProcess   IMPORT HelpProc;
28
29  FROM PlotProcess   IMPORT Plot;
30
31  FROM FileProcess   IMPORT FileHandler;
32
33  FROM GenGalv       IMPORT InitGalv;
34
35  FROM GenVib        IMPORT InitVib;
36
37  FROM Graphics      IMPORT Shutdown;
38
39  FROM Messages      IMPORT InitMailBox,SendMessage,AcceptMessage;
40
41  FROM Kernel        IMPORT MaxPriority,CreateProcess,InitSem,Wait;
42
43  FROM AnalogIO      IMPORT DAOut;
44
45  FROM Storage       IMPORT ALLOCATE,DEALLOCATE;
46
47  CONST  PoolSize=10;
48
49  VAR                    i : CARDINAL;
50              PMail,DMail : MailType;
51
52  BEGIN
53     RTMouse.Init;
54     InitSem(TheEnd,0);
55     InitMonitors;
56     InitMailBox(DBox,2*PoolSize);
57     InitMailBox(DRBox,2*PoolSize);
58     InitMailBox(PBox,PoolSize);
59     InitMailBox(MBox,PoolSize);
60     InitMailBox(ErrorBox,10);
61     InitMailBox(OutBox,10);
62     FOR i:=1 TO PoolSize DO (* Create message pooles *)
63       NEW(PMail);
64       SendMessage(MBox,PMail);
65       NEW(DMail);
66       SendMessage(DBox,DMail);
67       NEW(DMail);
68       SendMessage(DRBox,DMail);
69     END;
70     InitControl;
71     CreateProcess(DataMeasure,5000);
72     CreateProcess(FileHandler,5000);
73     CreateProcess(SafeProc,1000);
74     CreateProcess(Plot,5000);
75     CreateProcess(HelpProc,1000);
76     InitVib;
77     InitGalv;
```



```
78     Wait(TheEnd);
79     DAOut(0,0.0);
80     DAOut(1,0.0);
81     Shutdown;
82  END Postcon2.
83
```

```
DEFINITION MODULE Control;

EXPORT QUALIFIED ControlProc,InitControl;

PROCEDURE ControlProc;

PROCEDURE InitControl;

END Control.
```



```
IMPLEMENTATION MODULE Control;

IMPORT RTMouse;

FROM Kernel    IMPORT CreateProcess,WaitUntil,WaitTime,IncTime,GetTime,
                      Time,SetPriority,Wait,Signal,Semaphore,InitSem;

FROM Graphics IMPORT handle,rectangle,color,point,SetWindow,SetViewPort,
                     VirtualScreen,SetFillColor,FillRectangle,
                     DrawRectangle,WaitMouseRectangle,HideCursor,
                     SetMouseRectangle,SetLineColor,SetTextColor,
                     WriteString,ReadString,ShowCursor;

FROM FileSystem IMPORT File,Lookup,done,notdone;

FROM Storage  IMPORT ALLOCATE,DEALLOCATE;

FROM MathLib  IMPORT float,round;

FROM Messages IMPORT ReceiveMessage,SendMessage;

FROM NumMenu  IMPORT MakeNumMenu,SetNumMenuEntry,ShowNumMenu,InitNumMenu,
                     GetNumMenuState,HideNumMenu,SetNumMenuColors,
                     GetNumMenuStateWait,NumMenuType,SetNumMenuState;

FROM LogMenu  IMPORT MakeLogMenu,SetLogMenuEntry,ShowLogMenu,InitLogMenu,
                     GetLogMenuState,HideLogMenu,SetLogMenuColors,
                     GetLogMenuStateWait,LogMenuType,SetLogMenuState;

FROM Decl     IMPORT TheEnd,ErrorBox,ErrorPtrType,ErrorType,Error,
                     OutBox,MotorType,
                     PutMotorData,ChangeWhat,CheckWhat,WhatType,
                     StartTest,SetEnabled,
                     GetMeasWork,SetMeasWork,MeasureType,PutEmerg
                     ,EndMeasure,TestText,StopTest,PutMomentData
                     ,SetFileConst,SetTStart,PutNr,TestType,StimType
                     ,MeasType,SetParTest,PutMeas,WaitOldReady,FileText,
                     GetNrStimuli,GetMotorData,PutPlotSort,
                     PlotType,GetMeasIndex,PutMeasIndex;

FROM ConvReal IMPORT RealToString;

CONST HMColor = blue;          HMTextColor = intensewhite;
      TMColor = green;         TMTextColor = black;
      TMAlertColor = red;
      PMColor = cyan;          PMTextColor = black;
      MEColor = magenta;       METextColor = black;
      METrue = lightgreen;     MEAlertColor = blue;
      QMColor = lightred;      QMTextColor = blue;
      SPMColor = white;        SPMTextColor = black;
      SPMAlertColor = red;
      HideColor = black;
      ErrorColor = lightred; ErrorTextColor = yellow;

TYPE  TextStr        = ARRAY[0..16] OF CHAR;
      RealVectorType = ARRAY[0..10] OF REAL;
      WindowType     = (Ord,Num,Log);

      CommandWindowType = POINTER TO RECORD
                             H : handle;
                             CASE Kind : WindowType OF
                                Ord  : WindowRectangle :rectangle|
                                Num  : NM : NumMenuType|
                                Log  : LM : LogMenuType|
                             END;
                          END;

      SysParType        =  RECORD
                             SampleTime   : REAL;
                             PlotConst    : INTEGER;
                             A,B,Gamma,Alfa,TStart : REAL;
                             FileConst    : INTEGER;
                          END;

      StimTestType      = RECORD
                             RandomT,SinusT,PRBSSinusT,
                             PRBST,PausT                 : BOOLEAN;
```

```
                        Stimuli                 : StimType;
                        Work                    : INTEGER;
                      END;

VAR         MeasPar             : MeasType;
            MotPar              : ARRAY[1..2],[1..4] OF MotorType;
            SysPar              : SysParType;
            Tabbe,Tabbe1,Fel,
            PlotOn,Miss,Locked  : BOOLEAN;
            PlotPar             : PlotType;
            TestPar             : StimTestType;
            Plotting            : ARRAY[1..6] OF BOOLEAN;
            Measuring           : ARRAY [0..1] OF BOOLEAN;
            PlotSort,MeasIndex  : INTEGER;
            SPStr               : ARRAY[1..8] OF TextStr;

PROCEDURE InitParameters;

VAR     i: INTEGER;

BEGIN

  FOR i:=1 TO 6 DO
    Plotting[i]:=FALSE;
  END;
  Measuring[0]:=FALSE;
  Measuring[1]:=TRUE;
  MeasIndex:=2;
  PutMeasIndex(MeasIndex);
  PlotOn:=FALSE;
  PlotSort:=2;
  WITH PlotPar DO
    NPower:=FALSE;
    XYMoment:=FALSE;
    ElbowPower:=FALSE;
    TestSeq:=FALSE;
    ADInCali:=FALSE;
  END;
  PutPlotSort(PlotPar);
  WITH MeasPar DO
    VibTest:=Paus;
    GalvTest:=Paus;
  END;
  PutMeas(MeasPar);
  WITH TestPar DO
    Stimuli:=None;
    RandomT:=FALSE;
    SinusT:=FALSE;
    PRBSSinusT:=FALSE;
    PRBST:=FALSE;
    PausT:=TRUE;
    Work:=4;
  END;
  FOR i :=1 TO 2 DO
    WITH MotPar[i,1] DO
      RandomTime:=10.0;
      RampTime:=0.0;
      WaitPaus:=0.0;
      BreakTime:=5.0;
      MaxSpeed:=1.0;
      RandomTime:=10.0;
      IF i=1 THEN
        tmax:=165.0;
      ELSE
        tmax:=240.0;
      END;
      Period:=30.0;
      RandomSpeed:=1.0;
    END;
    PutMotorData(MotPar[i,1],1,i);
    WITH MotPar[i,2] DO
      RandomTime:=0.0;
      RampTime:=0.0;
      WaitPaus:=0.0;
      BreakTime:=5.0;
      MaxSpeed:=0.0;
      RandomSpeed:=2.0;
```



```
      Period:=30.0;
      PRBSPeriod:=0.0;
      tmax:=240.0;
    END;
    PutMotorData(MotPar[i,2],2,i);
    WITH MotPar[i,3] DO
      RandomTime:=0.0;
      RampTime:=0.0;
      WaitPaus:=0.0;
      BreakTime:=5.0;
      MaxSpeed:=0.0;
      RandomSpeed:=2.0;
      Period:=30.0;
      PRBSPeriod:=3.0;
      tmax:=240.0;
    END;
    PutMotorData(MotPar[i,3],3,i);
    WITH MotPar[i,4] DO
      RandomTime:=0.0;
      RampTime:=0.0;
      WaitPaus:=0.0;
      BreakTime:=5.0;
      IF i=1 THEN
        MaxSpeed:=3.9;
        RandomSpeed:=3.9;
      ELSE
        MaxSpeed:=1.25;
        RandomSpeed:=1.25;
      END;
      Period:=30.0;
      PRBSPeriod:=1.0;
      tmax:=286.0;
    END;
    PutMotorData(MotPar[i,4],4,i);
  END;
  WITH SysPar DO
    SampleTime:=0.1;
    PlotConst:=5;
    A:=0.170;
    B:=0.170;
    Gamma:=200.0;
    Alfa:=20.0;
    FileConst:=5;
    TStart:=30.0;
    PutMomentData(PlotConst,SampleTime,A,B,Gamma,Alfa);
    SetFileConst(FileConst);
    SetTStart(TStart);
  END;
  END InitParameters;

(*****************************************************************************)

(******************** Procedures for screens *********************************)

PROCEDURE SetRectangle(VAR R : rectangle; Xlo,Ylo,Xhi,Yhi : REAL);

BEGIN
   R.xlo:=Xlo;
   R.ylo:=Ylo;
   R.xhi:=Xhi;
   R.yhi:=Yhi;
END SetRectangle;


PROCEDURE DefineMouseWindow(H : handle;y : INTEGER);

VAR     R : rectangle;

BEGIN
   SetRectangle(R, 0.0, float(y-1), 1.0, float(y));
   SetMouseRectangle(H,R,y);
END DefineMouseWindow;


PROCEDURE ShowText(H :handle; y :REAL; str :TextStr);
```

```
VAR      Textpoint : point;
         R         : rectangle;

BEGIN
   Textpoint.h:=0.05;
   Textpoint.v:=y+0.15;
   SetRectangle(R,0.0,y,1.0,y+1.0);
   HideCursor;
   DrawRectangle(H,R);
   WriteString(H,Textpoint,str);
   ShowCursor;
END ShowText;


PROCEDURE HideMenu(VAR Window : CommandWindowType);

BEGIN
   WITH Window^ DO
      SetFillColor(H, HideColor);
      HideCursor;
      FillRectangle(H,WindowRectangle);
      ShowCursor;
   END;
END HideMenu;


PROCEDURE NiceStartUpMessage;

VAR         Handtag      : handle;
            Fonster      : rectangle;
            Punkt        : point;
            Klar         : ARRAY[0..1]OF CHAR;
CONST       Fonsterfarg  = lightmagenta;
            Textfarg     = lightcyan;

BEGIN
  VirtualScreen(Handtag);
  SetRectangle(Fonster,0.05,0.05,1.45,0.95);
  SetFillColor(Handtag,Fonsterfarg);
  SetLineColor(Handtag,Textfarg);
  SetTextColor(Handtag,Textfarg);
  Punkt.h:=0.31;
  Punkt.v:=0.85;
  FillRectangle(Handtag,Fonster);
  DrawRectangle(Handtag,Fonster);
  WriteString(Handtag,Punkt,'           POSTCON v3.0a for 386');
  Punkt.v:=0.75;
  WriteString(Handtag,Punkt,'Control program for Postural stimulation equipment');
  Punkt.v:=0.65;
  WriteString(Handtag,Punkt,'      written by Per-Anders Fransson   ');
  Punkt.v:=0.60;
  WriteString(Handtag,Punkt,'            and modified by    ');
  Punkt.v:=0.55;
  WriteString(Handtag,Punkt,'     Johan Lidman and Gunilla H glund  ');
  Punkt.v:=0.50;
  WriteString(Handtag,Punkt,'                    for ');
  Punkt.h:=0.09;
  Punkt.v:=0.40;
  WriteString(Handtag,Punkt,'  the Department of Automatic Control, Lund Institute of Techno
logy');
  Punkt.h:=0.31;
  Punkt.v:=0.35;
  WriteString(Handtag,Punkt,'                    and         ');
  Punkt.h:=0.09;
  Punkt.v:=0.30;
  WriteString(Handtag,Punkt,'  the Department of OtoRhinoLaryngology, Lund University Hospit
al');
  WaitTime(1000);
  SetFillColor(Handtag,black);
  SetLineColor(Handtag,black);
  DrawRectangle(Handtag,Fonster);
  FillRectangle(Handtag,Fonster);
END NiceStartUpMessage;


(*************************************************************************)
```



```
(********************* Procedure for errorwindow ************************)


PROCEDURE InitErrorWindow(VAR errwindow : CommandWindowType);

VAR      errbox : rectangle;

BEGIN
   NEW(errwindow);
   WITH errwindow^ DO
      VirtualScreen(H);
      Kind:= Ord;
      SetRectangle(errbox,0.05,0.27,0.45,0.38);
      SetViewPort(H,errbox);
      SetRectangle(WindowRectangle,0.0,0.0,4.0,1.0);
      SetWindow(H,WindowRectangle);
      SetTextColor(H,ErrorTextColor);
      SetLineColor(H,ErrorTextColor);
      SetFillColor(H,ErrorColor);
   END;
END InitErrorWindow;


(*************************************************************************)

(********************* Procedures for HeadMenu **************************)


PROCEDURE InitHeadMenu(VAR HMwindow : CommandWindowType);

VAR       ViewPortRectangle : rectangle;
          i                 : INTEGER;

BEGIN
   NEW(HMwindow);
   WITH HMwindow^ DO
      VirtualScreen(H);
      Kind:= Ord;
      SetRectangle(ViewPortRectangle, 0.05, 0.40, 0.45, 1.0);
      SetViewPort(H,ViewPortRectangle);
      SetRectangle(WindowRectangle, 0.0, 0.0, 1.0, 7.0);
      SetWindow(H,WindowRectangle);
      SetTextColor(H, HMTextColor);
      SetLineColor(H, HMTextColor);
      SetFillColor(H, HMColor);
      FOR i:= 1 TO 7 DO
         DefineMouseWindow(H,i);
      END;
   END;
END InitHeadMenu;


PROCEDURE ShowHeadMenu(VAR HMwindow : CommandWindowType);
(*Shows headmenu after return from submenus*)

VAR       String : ARRAY [0..6] OF TextStr;
          i      : INTEGER;

BEGIN
   WITH HMwindow^ DO
      PutNr(1);
      SetFillColor(H,HMColor);
      HideCursor;
      FillRectangle(H,WindowRectangle);
      ShowCursor;
      String[0]:= "Quit program";
      String[1]:= "Stop test";
      String[2]:= "Start test";
      String[3]:= "Plot";
      String[4]:= "Stimuli";
      String[5]:= "Inputsignals";
      String[6]:= "System parameters";
      FOR i:= 0 TO 6 DO
         ShowText(H,float(i),String[i]);
      END;
   END;
```

```
END ShowHeadMenu;


(*****************************************************************************)

(******************** Procedures for ParmeterMenu ************************)


PROCEDURE InitSystemParMenu(VAR SPwindow : CommandWindowType);

VAR       Pos    : point;
          i      : CARDINAL;

BEGIN
  SPStr[1]:='Sample time[s]';   SPStr[2]:='Plot const';
  SPStr[3]:='File const';       SPStr[4]:='foot-back  [m]';
  SPStr[5]:='foot-front [m]';   SPStr[6]:='Loadc.P [N/V]';
  SPStr[7]:='Loadc.E [Nm/V]';  SPStr[8]:='StabTime   [s]';
  NEW(SPwindow);
  WITH SPwindow^ DO
    VirtualScreen(H);
    Kind:=Num;
    Pos.h:=0.0;
    Pos.v:=0.55;
    MakeNumMenu(NM,Pos,8);
    SetNumMenuColors(NM,SPMColor,SPMTextColor,SPMAlertColor,HideColor);
    WITH SysPar DO
      SetNumMenuEntry(NM,SPStr[1],SampleTime);
      SetNumMenuEntry(NM,SPStr[2],float(PlotConst));
      SetNumMenuEntry(NM,SPStr[3],float(FileConst));
      SetNumMenuEntry(NM,SPStr[4],A);
      SetNumMenuEntry(NM,SPStr[5],B);
      SetNumMenuEntry(NM,SPStr[6],Gamma);
      SetNumMenuEntry(NM,SPStr[7],Alfa);
      SetNumMenuEntry(NM,SPStr[8],TStart);
    END;
  END;
END InitSystemParMenu;


PROCEDURE CopySysPar(a : ARRAY OF REAL);

BEGIN
  WITH SysPar DO
    SampleTime:=a[0];
    PlotConst:=round(a[1]);
    FileConst:=round(a[2]);
    A:=a[3];
    B:=a[4];
    Gamma:=a[5];
    Alfa:=a[6];
    TStart:=a[7];
    PutMomentData(PlotConst,SampleTime,A,B,Gamma,Alfa);
    SetFileConst(FileConst);
    SetTStart(TStart);
  END;
END CopySysPar;


PROCEDURE SystemParRoutine(VAR HMwindow,SPwindow : CommandWindowType);

BEGIN
  HideMenu(HMwindow);
  PutNr(12);
  WITH SPwindow^ DO
    ShowNumMenu(NM);
    GetNumMenuStateWait(NM,CopySysPar);
    HideNumMenu(NM);
  END;
  ShowHeadMenu(HMwindow);
END SystemParRoutine;


(*****************************************************************************)

(********************** Procedures for StimuliMenu ***************************)
```



```
PROCEDURE InitStimuliMenu(VAR TMWindow : CommandWindowType);

VAR     ViewPortRectangle : rectangle;
        i                 : INTEGER;

BEGIN
  NEW(TMWindow);
  WITH TMWindow^ DO
    VirtualScreen(H);
    Kind:=Ord;
    SetRectangle(ViewPortRectangle,0.05,0.40,0.45,0.80);
    SetViewPort(H,ViewPortRectangle);
    SetRectangle(WindowRectangle,0.0,0.0,1.0,6.0);
    SetWindow(H,WindowRectangle);
    SetTextColor(H,TMTextColor);
    SetLineColor(H,TMTextColor);
    SetFillColor(H,TMColor);
    FOR i:= 1 TO 6 DO
      DefineMouseWindow(H,i);
    END;
  END;
END InitStimuliMenu;


PROCEDURE ShowStimuliMenu(VAR TMWindow : CommandWindowType);

VAR      String : ARRAY[0..5] OF TextStr;
         i      : INTEGER;

BEGIN
  WITH TMWindow^ DO
    SetFillColor(H,TMColor);
    HideCursor;
    FillRectangle(H,WindowRectangle);
    ShowCursor;
    String[5]:=' Vibration';
    String[4]:=' Galvanic';
    String[3]:=' Vib Par.';
    String[2]:=' Galv Par.';
    String[1]:=' RunFile';
    String[0]:=' Done';
    FOR i:=0 TO 5 DO
      ShowText(H,float(i),String[i]);
    END;
  END;
END ShowStimuliMenu;


PROCEDURE InitStimuliSubMenu(VAR TMSWindow : CommandWindowType);

VAR     Pos    : point;
        i      : CARDINAL;
        OOStr  : ARRAY[1..8] OF TextStr;
        a      : ARRAY[1..5] OF BOOLEAN;

BEGIN
  OOStr[1]:='Random';
  OOStr[2]:='Sinus';
  OOStr[3]:='PRBSSinus';
  OOStr[4]:='PRBS';
  OOStr[5]:='Paus';
  WITH TestPar DO
    NEW(TMSWindow);
    WITH TMSWindow^ DO
      VirtualScreen(H);
      Kind:=Log;
      Pos.h:=0.05;
      Pos.v:=0.55;
      MakeLogMenu(LM,Pos,5);
      SetLogMenuColors(LM,METrue,MEColor,METextColor,MEAlertColor,HideColor);
      SetLogMenuEntry(LM,OOStr[1],RandomT);
      SetLogMenuEntry(LM,OOStr[2],SinusT);
      SetLogMenuEntry(LM,OOStr[3],PRBSSinusT);
      SetLogMenuEntry(LM,OOStr[4],PRBST);
      SetLogMenuEntry(LM,OOStr[5],PausT);
      FOR i:=1 TO 4 DO
```

```
      a[i]:=FALSE;
    END;
    a[5]:=TRUE;
    SetLogMenuState(LM,a);
  END;
END;
END InitStimuliSubMenu;


PROCEDURE InitStimuliSubMenu1(VAR TMSWindow : CommandWindowType);

VAR     Pos    : point;
        i      : CARDINAL;
        OOStr  : ARRAY[1..8] OF TextStr;
        a      : ARRAY[1..5] OF BOOLEAN;

BEGIN
  OOStr[1]:='Finsk';
  OOStr[2]:='Sinus';
  OOStr[3]:='PRBSSinus';
  OOStr[4]:='PRBS';
  OOStr[5]:='Paus';
  WITH TestPar DO
    NEW(TMSWindow);
    WITH TMSWindow^ DO
      VirtualScreen(H);
      Kind:=Log;
      Pos.h:=0.05;
      Pos.v:=0.55;
      MakeLogMenu(LM,Pos,5);
      SetLogMenuColors(LM,METrue,MEColor,METextColor,MEAlertColor,HideColor);
      SetLogMenuEntry(LM,OOStr[1],RandomT);
      SetLogMenuEntry(LM,OOStr[2],SinusT);
      SetLogMenuEntry(LM,OOStr[3],PRBSSinusT);
      SetLogMenuEntry(LM,OOStr[4],PRBST);
      SetLogMenuEntry(LM,OOStr[5],PausT);
      FOR i:=1 TO 4 DO
        a[i]:=FALSE;
      END;
      a[5]:=TRUE;
      SetLogMenuState(LM,a);
    END;
  END;
END InitStimuliSubMenu1;


PROCEDURE InitTMPMenu(VAR TMPWindow : CommandWindowType);

VAR     ViewPortRectangle : rectangle;
        i                 : INTEGER;

BEGIN
  NEW(TMPWindow);
  WITH TMPWindow^ DO
    VirtualScreen(H);
    Kind:=Ord;
    SetRectangle(ViewPortRectangle,0.05,0.40,0.45,0.80);
    SetViewPort(H,ViewPortRectangle);
    SetRectangle(WindowRectangle,0.0,0.0,1.0,5.0);
    SetWindow(H,WindowRectangle);
    SetTextColor(H,TMTextColor);
    SetLineColor(H,TMTextColor);
    SetFillColor(H,TMColor);
    FOR i:= 1 TO 5 DO
      DefineMouseWindow(H,i);
    END;
  END;
END InitTMPMenu;


PROCEDURE InitTMPSubMenu(VAR PMW1,PMW2,PMW3,PMW4 : CommandWindowType);

VAR       Pos    : point;
          i      : CARDINAL;
          ParStr : ARRAY[1..8] OF TextStr;

BEGIN
```



```
  NEW(PMW1);
  WITH PMW1^ DO
    VirtualScreen(H);
    Kind:=Num;
    Pos.h:=0.00;
    Pos.v:=0.55;
    MakeNumMenu(NM,Pos,7);
    SetNumMenuColors(NM,TMColor,TMTextColor,TMAlertColor,HideColor);
    SetNumMenuEntry(NM,'Ramptime',MotPar[1,1].RampTime);
    SetNumMenuEntry(NM,'Breaktime',MotPar[1,1].BreakTime);
    SetNumMenuEntry(NM,'MinTime',MotPar[1,1].WaitPaus);
    SetNumMenuEntry(NM,'Center speed',MotPar[1,1].MaxSpeed);
    SetNumMenuEntry(NM,'RandomTime',MotPar[1,1].RandomTime);
    SetNumMenuEntry(NM,'RandomSpeed',MotPar[1,1].RandomSpeed);
    SetNumMenuEntry(NM,'Max testtime',MotPar[1,1].tmax);
  END;
  NEW(PMW2);
  WITH PMW2^ DO
    VirtualScreen(H);
    Kind:=Num;
    Pos.h:=0.00;
    Pos.v:=0.55;
    MakeNumMenu(NM,Pos,6);
    SetNumMenuColors(NM,TMColor,TMTextColor,TMAlertColor,HideColor);
    SetNumMenuEntry(NM,'Ramptime',MotPar[1,2].RampTime);
    SetNumMenuEntry(NM,'WaitPause',MotPar[1,2].WaitPaus);
    SetNumMenuEntry(NM,'Center speed',MotPar[1,2].MaxSpeed);
    SetNumMenuEntry(NM,'Period time',MotPar[1,2].Period);
    SetNumMenuEntry(NM,'Amplitude',MotPar[1,2].RandomSpeed);
    SetNumMenuEntry(NM,'Max testtime',MotPar[1,2].tmax);
  END;
  NEW(PMW3);
  WITH PMW3^ DO
    VirtualScreen(H);
    Kind:=Num;
    Pos.h:=0.00;
    Pos.v:=0.55;
    MakeNumMenu(NM,Pos,7);
    SetNumMenuColors(NM,TMColor,TMTextColor,TMAlertColor,HideColor);
    SetNumMenuEntry(NM,'RampTime',MotPar[1,3].RampTime);
    SetNumMenuEntry(NM,'WaitPause',MotPar[1,3].WaitPaus);
    SetNumMenuEntry(NM,'Center speed',MotPar[1,3].MaxSpeed);
    SetNumMenuEntry(NM,'Amplitude',MotPar[1,3].RandomSpeed);
    SetNumMenuEntry(NM,'Period',MotPar[1,3].Period);
    SetNumMenuEntry(NM,'PRBSPeriod',MotPar[1,3].PRBSPeriod);
    SetNumMenuEntry(NM,'Max testtime',MotPar[1,3].tmax);
  END;
  NEW(PMW4);
  WITH PMW4^ DO
    VirtualScreen(H);
    Kind:=Num;
    Pos.h:=0.00;
    Pos.v:=0.55;
    MakeNumMenu(NM,Pos,6);
    SetNumMenuColors(NM,TMColor,TMTextColor,TMAlertColor,HideColor);
    SetNumMenuEntry(NM,'Ramptime',MotPar[1,4].RampTime);
    SetNumMenuEntry(NM,'WaitPause',MotPar[1,4].WaitPaus);
    SetNumMenuEntry(NM,'Center speed',MotPar[1,4].MaxSpeed);
    SetNumMenuEntry(NM,'Amplitude',MotPar[1,4].RandomSpeed);
    SetNumMenuEntry(NM,'PRBSPeriod',MotPar[1,4].PRBSPeriod);
    SetNumMenuEntry(NM,'Max testtime',MotPar[1,4].tmax);
  END;
END InitTMPSubMenu;


PROCEDURE TMRoutine(HMWindow,TMWindow,TMVibWindow,TMGalvWindow,SPwindow,
                    QMWindow,TMPWindow,PMW1,PMW2,
                    PMW3,PMW4 : CommandWindowType);

VAR       Pos         : point;
          Name        : ARRAY[0..1] OF CHAR;
          SelectedBox : CARDINAL;
          Done        : BOOLEAN;

BEGIN
  Done:=FALSE;
  HideMenu(HMWindow);
```

```
  StopTest;
  PutNr(2);
  ShowStimuliMenu(TMWindow);
  REPEAT
    SelectedBox:=WaitMouseRectangle(TMWindow^.H);
    CASE SelectedBox OF
      6: TestPar.Stimuli:=Vibration;PutNr(3);TMSRoutine(TMWindow,TMVibWindow,
         QMWindow,SPwindow)|
      5: TestPar.Stimuli:=Galvanic;PutNr(4);TMSRoutine(TMWindow,TMGalvWindow,
         QMWindow,SPwindow)|
      4: TestPar.Stimuli:=Vibration;TMPRoutine(TMWindow,TMPWindow,QMWindow,
                  PMW1,PMW2,PMW3,PMW4)|
      3: TestPar.Stimuli:=Galvanic;TMPRoutine(TMWindow,TMPWindow,QMWindow,
                  PMW1,PMW2,PMW3,PMW4)|
      2: TMQRoutine(TMWindow,QMWindow);Done:=TRUE|
      1: Done:=TRUE;
    END;
  UNTIL Done;
  HideMenu(TMWindow);
  ShowHeadMenu(HMWindow);
END TMRoutine;


PROCEDURE TMSRoutine(TMWindow,TMSWindow,QMWindow,SPwindow : CommandWindowType);

VAR       Pos    : point;
          Name   : ARRAY[0..1] OF CHAR;
          Values : ARRAY[0..6] OF REAL;
          Nr,i   : CARDINAL;
          a      : ARRAY[1..5] OF BOOLEAN;

BEGIN
  HideMenu(TMWindow);
  WITH TMSWindow^ DO
    ShowLogMenu(LM);
    REPEAT;
      GetLogMenuStateWait(LM,CopyStimuliSubPar);
      IF Tabbe1 THEN
        FOR i:=1 TO 4 DO
          a[i]:=FALSE;
        END;
        a[5]:=TRUE;
        SetLogMenuState(LM,a);
      END;
      IF Tabbe THEN
        HideLogMenu(LM);
        HideCursor;
        WITH QMWindow^ DO
          SetFillColor(H,QMColor);
          FillRectangle(H,WindowRectangle);
          Pos.h:=0.05;
          Pos.v:=1.10;
          WriteString(H,Pos,'Only one test please');
          Pos.h:=0.05;
          Pos.v:=0.10;
          WriteString(H,Pos,'hit return');
          Pos.h:=1.55;
          ReadString(H,Pos,Name);
        END;
        HideMenu(QMWindow);
        ShowCursor;
        ShowLogMenu(LM);
      END;
    UNTIL NOT Tabbe;
    HideLogMenu(LM);
  END;
  ShowStimuliMenu(TMWindow);
END TMSRoutine;


PROCEDURE ShowTMPMenu(VAR TMPWindow : CommandWindowType);

VAR     String : ARRAY[0..4] OF TextStr;
        i      : INTEGER;

BEGIN
  PutNr(4);
```



```
  WITH TMPWindow^ DO
    SetFillColor(H,TMColor);
    HideCursor;
    FillRectangle(H,WindowRectangle);
    ShowCursor;
    String[4]:=' RampRandom';
    String[3]:=' Sinus';
    String[2]:=' PRBSSinus';
    String[1]:=' PRBS';
    String[0]:=' DONE';
    FOR i:=0 TO 4 DO
      ShowText(H,float(i),String[i]);
    END;
  END;
END ShowTMPMenu;


PROCEDURE ShowTMPMenu1(VAR TMPWindow : CommandWindowType);

VAR     String : ARRAY[0..4] OF TextStr;
        i      : INTEGER;

BEGIN
  PutNr(3);
  WITH TMPWindow^ DO
    SetFillColor(H,TMColor);
    HideCursor;
    FillRectangle(H,WindowRectangle);
    ShowCursor;
    String[4]:=' Finsk';
    String[3]:=' Sinus';
    String[2]:=' PRBSSinus';
    String[1]:=' PRBS';
    String[0]:=' DONE';
    FOR i:=0 TO 4 DO
      ShowText(H,float(i),String[i]);
    END;
  END;
END ShowTMPMenu1;


PROCEDURE TMPRoutine(VAR TMWindow,TMPWindow,QMWindow,PMW1,PMW2,
                     PMW3,PMW4 : CommandWindowType);

VAR     SelectedBox : CARDINAL;
        Done        : BOOLEAN;
BEGIN
  Done:=FALSE;
  HideMenu(TMWindow);
  REPEAT
    IF (TestPar.Stimuli=Vibration) THEN
      ShowTMPMenu1(TMPWindow);
    ELSE;
      ShowTMPMenu(TMPWindow);
    END;
    SelectedBox:=WaitMouseRectangle(TMPWindow^.H);
    CASE SelectedBox OF
      5 : IF (TestPar.Stimuli=Vibration) THEN
            Error('Cant be changed');
          ELSE;
            TMPSubRoutine(TMPWindow,PMW1,QMWindow,1);
          END|
      4 : TMPSubRoutine(TMPWindow,PMW2,QMWindow,2)|
      3 : TMPSubRoutine(TMPWindow,PMW3,QMWindow,3)|
      2 : TMPSubRoutine(TMPWindow,PMW4,QMWindow,4)|
      1 : Done:=TRUE;
    END;
  UNTIL Done=TRUE;
  ShowStimuliMenu(TMWindow);
END TMPRoutine;


PROCEDURE TMPSubRoutine(VAR TMWindow,SubMenu,QMWindow : CommandWindowType;
                        nr : INTEGER);

VAR       Pos : point;
          Name : ARRAY[0..1] OF CHAR;
```

```
        Motordata : MotorType;
        a : ARRAY[0..6] OF REAL;

BEGIN
  GetMotorData(Motordata,nr,TestPar.Stimuli);
  a[0]:=Motordata.RampTime;
  IF nr=1 THEN
    a[1]:=Motordata.BreakTime;
    a[2]:=Motordata.WaitPaus;
    a[3]:=Motordata.MaxSpeed;
    a[4]:=Motordata.RandomTime;
    a[5]:=Motordata.RandomSpeed;
    a[6]:=Motordata.tmax;
  ELSE
    a[1]:=Motordata.WaitPaus;
    a[2]:=Motordata.MaxSpeed;
  END;
  IF nr=2 THEN
    a[3]:=Motordata.Period;
    a[4]:=Motordata.RandomSpeed;
    a[5]:=Motordata.tmax;
  ELSIF nr=3 THEN
    a[3]:=Motordata.RandomSpeed;
    a[4]:=Motordata.Period;
    a[5]:=Motordata.PRBSPeriod;
    a[6]:=Motordata.tmax;
  ELSIF nr=4 THEN
    a[3]:=Motordata.RandomSpeed;
    a[4]:=Motordata.PRBSPeriod;
    a[5]:=Motordata.tmax;
  END;
  SetNumMenuState(SubMenu^.NM,a);
  HideMenu(TMWindow);
  WITH SubMenu^ DO
    ShowNumMenu(NM);
    CASE nr OF
      1:PutNr(8);GetNumMenuStateWait(NM,CopyPar1)|
      2:PutNr(9);GetNumMenuStateWait(NM,CopyPar2)|
      3:PutNr(10);GetNumMenuStateWait(NM,CopyPar3)|
      4:PutNr(11);GetNumMenuStateWait(NM,CopyPar4);
    END;
    HideNumMenu(NM);
  END;
END TMPSubRoutine;


PROCEDURE TMQRoutine(TMWindow,QMWindow : CommandWindowType);

VAR       Name              : TestText;

BEGIN
  ShowQuestionWindow(TMWindow,QMWindow,'File to load? ',Name);
  SetEnabled(TRUE);
  PutEmerg(FALSE);
  SetMeasWork(OldData);
  StartTest(Name);
  WaitOldReady;
  IF PlotOn THEN
    ChangeWhat(Data);
  END;
END TMQRoutine;


PROCEDURE CopyStimuliPar(a : ARRAY OF BOOLEAN);

VAR    i,check    : INTEGER;

BEGIN
  check:=0;
  WITH TestPar DO
    FOR i:=0 TO 4 DO
      IF a[i] THEN
        Work:=i;
        check:=check+1;
      END;
    END;
    Tabbe:=check>1;
```



```
    Tabbe1:=check=0;
    IF NOT Tabbe THEN
      CASE Work OF
        0: Stimuli:=Vibration|
        1: Stimuli:=Galvanic|
        2:|
        3:;
      END;
    END;
  END;
END CopyStimuliPar;


PROCEDURE CopyStimuliSubPar(a : ARRAY OF BOOLEAN);

VAR      i,check    : INTEGER;
         WorkSort : TestType;

BEGIN
  check:=0;
  WITH TestPar DO
    Work:=0;
    FOR i:=0 TO 4 DO
      IF a[i] THEN
        Work:=i;
        check:=check+1;
      END;
    END;
    Tabbe:=check>1;
    Tabbe1:=check=0;
    CASE Work OF
      0 : WorkSort:=Random|
      1 : WorkSort:=Sinus|
      2 : WorkSort:=PRBSSinus|
      3 : WorkSort:=PRBS|
      4 : WorkSort:=Paus;
    END;
    SetParTest(Stimuli,WorkSort);
    IF PlotOn THEN
      ChangeWhat(Data);
    END;
  END;
END CopyStimuliSubPar;


PROCEDURE CopyPar1(a : ARRAY OF REAL);

VAR Nr    : INTEGER;

BEGIN
  CASE TestPar.Stimuli OF
    Vibration : Nr:=1|
    Galvanic  : Nr:=2;
  END;
  WITH MotPar[Nr,1] DO
    RampTime:=a[0];
    BreakTime:=a[1];
    WaitPaus:=a[2];
    MaxSpeed:=a[3];
    RandomTime:=a[4];
    RandomSpeed:=a[5];
    tmax:=a[6];
  END;
  PutMotorData(MotPar[Nr,1],1,Nr);
END CopyPar1;


PROCEDURE CopyPar2(a : ARRAY OF REAL);

VAR Nr    : INTEGER;

BEGIN
  CASE TestPar.Stimuli OF
    Vibration : Nr:=1|
    Galvanic  : Nr:=2;
  END;
  WITH MotPar[Nr,2] DO
```

```
      RampTime:=a[0];
      WaitPaus:=a[1];
      MaxSpeed:=a[2];
      Period:=a[3];
      RandomSpeed:=a[4];
      tmax:=a[5];
    END;
    PutMotorData(MotPar[Nr,2],2,Nr);
END CopyPar2;


PROCEDURE CopyPar3(a : ARRAY OF REAL);

VAR Nr   : INTEGER;

BEGIN
  CASE TestPar.Stimuli OF
    Vibration : Nr:=1|
    Galvanic  : Nr:=2;
  END;
  WITH MotPar[Nr,3] DO
    RampTime:=a[0];
    WaitPaus:=a[1];
    MaxSpeed:=a[2];
    RandomSpeed:=a[3];
    Period:=a[4];
    PRBSPeriod:=a[5];
    tmax:=a[6];
  END;
  PutMotorData(MotPar[Nr,3],3,Nr);
END CopyPar3;


PROCEDURE CopyPar4(a : ARRAY OF REAL);

VAR Nr   : INTEGER;

BEGIN
  CASE TestPar.Stimuli OF
    Vibration : Nr:=1|
    Galvanic  : Nr:=2;
  END;
  WITH MotPar[Nr,4] DO
      RampTime:=a[0];
      WaitPaus:=a[1];
      MaxSpeed:=a[2];
      RandomSpeed:=a[3];
      PRBSPeriod:=a[4];
      tmax:=a[5];
    END;
    PutMotorData(MotPar[Nr,4],4,Nr);
END CopyPar4;


(*************************************************************************)

(********************** Procedures for ForceMenu ***************************)


PROCEDURE InitForceMenu(VAR OOW : CommandWindowType);

VAR     Pos    : point;
        OOStr : ARRAY[1..2] OF TextStr;

BEGIN
  OOStr[1]:='Elbow&Plate';
  OOStr[2]:='Plate';
  NEW(OOW);
  WITH OOW^ DO
    VirtualScreen(H);
    Kind:=Log;
    Pos.h:=0.00;
    Pos.v:=0.75;
    MakeLogMenu(LM,Pos,2);
    SetLogMenuColors(LM,METrue,MEColor,METextColor,MEAlertColor,HideColor);
    SetLogMenuEntry(LM,OOStr[1],Measuring[0]);
    SetLogMenuEntry(LM,OOStr[2],Measuring[1]);
```



```
  END;
END InitForceMenu;


PROCEDURE CopyMeasIndex(a : ARRAY OF BOOLEAN);

VAR  what   : WhatType;

BEGIN
  CheckWhat(what);
    Measuring[0]:=a[0];
    Measuring[1]:=a[1];
    MeasIndex:=0;
    IF a[0] THEN MeasIndex:=MeasIndex+4;
    END;
    IF a[1] THEN MeasIndex:=MeasIndex+2;
    END;
    IF (NOT EndMeasure()) AND (GetMeasIndex()<>MeasIndex)
      THEN Locked:=TRUE;
      ELSE Locked:=FALSE;
    END;
    IF (MeasIndex<>2) AND (MeasIndex<>4)
      THEN Fel:=TRUE;
      ELSE Fel:=FALSE;
    END;
    IF NOT Locked AND NOT Fel THEN PutMeasIndex(MeasIndex);
    END
END CopyMeasIndex;


PROCEDURE FMRoutine(HMWindow,FMWindow,PTMWindow,QMWindow : CommandWindowType);

VAR       Pos      : point;
          a        : ARRAY[0..1] OF BOOLEAN;
          Name : TestText;
BEGIN
  HideMenu(HMWindow);
  IF GetMeasIndex()=4 THEN a[0]:=TRUE;
    a[1]:=FALSE;
    ELSE a[0]:=FALSE;
         a[1]:=TRUE;
  END;
  WITH FMWindow^ DO
    SetLogMenuState(LM,a);
    ShowLogMenu(LM);
    PutNr(13);
    REPEAT
      GetLogMenuStateWait(LM,CopyMeasIndex);
      IF Fel OR Locked THEN
        HideLogMenu(LM);
        HideCursor;
        WITH QMWindow^ DO
          SetFillColor(H,QMColor);
          FillRectangle(H,WindowRectangle);
          Pos.h:=0.15;
          Pos.v:=1.10;
          IF Fel THEN WriteString(H,Pos,'Only one choice')
                 ELSE WriteString(H,Pos,"Can't change Input");
                 END;
          Pos.h:=0.32;
          Pos.v:=0.60;
          WriteString(H,Pos,'please');
          Pos.h:=0.22;
          Pos.v:=0.10;
          WriteString(H,Pos,'hit return');
          Pos.h:=1.55;
          ReadString(H,Pos,Name);
        END;
        HideMenu(QMWindow);
        ShowCursor;
        ShowLogMenu(LM);
      END;
    UNTIL NOT Fel AND NOT Locked;
    HideLogMenu(LM);
  END;
  ShowHeadMenu(HMWindow);
END FMRoutine;
```

```
(*****************************************************************************)

(********************** Procedures for PlotMenu ****************************)

PROCEDURE InitPlotMenu(VAR OOW : CommandWindowType);

VAR     Pos    : point;
        i      : CARDINAL;
        OOStr : ARRAY[1..8] OF TextStr;

BEGIN
  OOStr[1]:='Sti&NPo&XYM';
  OOStr[2]:='Sti&NPo';
  OOStr[3]:='Sti&XYM';
  OOStr[4]:='Sti&NPo&EPo';
  OOStr[5]:='Calibration';
  OOStr[6]:='TestSeq';
  NEW(OOW);
  WITH OOW^ DO
    VirtualScreen(H);
    Kind:=Log;
    Pos.h:=0.00;
    Pos.v:=0.55;
    MakeLogMenu(LM,Pos,6);
    SetLogMenuColors(LM,METrue,MEColor,METextColor,MEAlertColor,HideColor);
    SetLogMenuEntry(LM,OOStr[1],Plotting[1]);
    SetLogMenuEntry(LM,OOStr[2],Plotting[2]);
    SetLogMenuEntry(LM,OOStr[3],Plotting[3]);
    SetLogMenuEntry(LM,OOStr[4],Plotting[4]);
    SetLogMenuEntry(LM,OOStr[5],Plotting[5]);
    SetLogMenuEntry(LM,OOStr[6],Plotting[6]);
  END;
END InitPlotMenu;


PROCEDURE CopyPlotPar(a : ARRAY OF BOOLEAN);

VAR     i,check : INTEGER;

BEGIN
  check:=0;
  PlotSort:=6;
  WITH PlotPar DO
    FOR i:=0 TO 5 DO
      Plotting[i+1]:=a[i];
      IF a[i] THEN
        PlotSort:=i;
        check:=check+1;
      END;
    END;
    Tabbe:=check>1;
    IF NOT Tabbe THEN
      PlotOn:=TRUE;
      CASE PlotSort OF
        0: NPower:=TRUE;XYMoment:=TRUE;ElbowPower:=FALSE|
        1: NPower:=TRUE;XYMoment:=FALSE;ElbowPower:=FALSE|
        2: NPower:=FALSE;XYMoment:=TRUE;ElbowPower:=FALSE|
        3: NPower:=TRUE;XYMoment:=FALSE;ElbowPower:=TRUE|
        4: ADInCali:=TRUE;PlotOn:=FALSE|
        5: TestSeq:=TRUE;PlotOn:=FALSE|
        6: PlotOn:=FALSE;
      END;
      CASE PlotSort OF
        0..3      : ChangeWhat(Data)|
        4         : ChangeWhat(ADIn)|
        5         : ChangeWhat(Test)|
        6         : ChangeWhat(Oper);
      END;
    END;
  END;
  IF (GetMeasIndex()<>4) AND (a[3]=TRUE) THEN Miss:=TRUE;
    ELSE Miss:=FALSE;
    PutPlotSort(PlotPar);
  END;
```



```
END CopyPlotPar;


PROCEDURE PMRoutine(HMWindow,PLWindow,PTMWindow,QMWindow : CommandWindowType);

VAR       Pos          : point;
          Name         : TestText;
          a            : ARRAY[0..6] OF BOOLEAN;
          i, Number    : INTEGER;
          getmeaswork : MeasureType;

BEGIN
  GetMeasWork(getmeaswork);
  IF (getmeaswork=Sekv) OR (getmeaswork=Cali) THEN StopTest;
  END;
  HideMenu(HMWindow);
  WITH PLWindow^ DO
    ShowLogMenu(LM);
    PutNr(6);
    REPEAT
      GetLogMenuStateWait(LM,CopyPlotPar);
      IF Tabbe OR Miss THEN
        HideLogMenu(LM);
        HideCursor;
        WITH QMWindow^ DO
          SetFillColor(H,QMColor);
          FillRectangle(H,WindowRectangle);
          Pos.h:=0.05;
          Pos.v:=1.10;
          IF Tabbe THEN  WriteString(H,Pos,'Too many plotscreens');
                   ELSE  WriteString(H,Pos,"Can't plot elbowf.");
          END;
          Pos.h:=0.05;
          Pos.v:=0.10;
          WriteString(H,Pos,'hit return');
          Pos.h:=1.55;
          ReadString(H,Pos,Name);
        END;
        HideMenu(QMWindow);
        ShowCursor;
        ShowLogMenu(LM);
      END;
    UNTIL NOT Tabbe AND NOT Miss;
    IF PlotPar.TestSeq THEN
    GetNrStimuli(Number);
      IF Number=0 THEN Error('No Stimuli On');
        ELSE  SetMeasWork(Sekv);
              StartTest(Name);
      END;
      PlotPar.TestSeq:=FALSE;
      FOR i:=1 TO 6 DO
      a[i-1]:=Plotting[i];
      END;
      a[5]:=FALSE;
      Plotting[6]:=FALSE;
      SetLogMenuState(LM,a);
    END;
    IF PlotPar.ADInCali THEN
      SetMeasWork(Cali);
      StartTest(Name);
      PlotPar.ADInCali:=FALSE;
      Plotting[5]:=FALSE;
      FOR i:=1 TO 6 DO
        a[i-1]:=Plotting[i];
      END;
      SetLogMenuState(LM,a);
    END;
    HideLogMenu(LM);
  END;
  ShowHeadMenu(HMWindow);
END PMRoutine;

(*****************************************************************************)

(********************* Procedures for StartMenu ****************************)
```

```
PROCEDURE InitQuestionWindow(VAR QMWindow : CommandWindowType);

VAR       ViewPortRectangle : rectangle;

BEGIN
  NEW(QMWindow);
  WITH QMWindow^ DO
    VirtualScreen(H);
    Kind:=Ord;
    SetRectangle(ViewPortRectangle,0.05,0.40,0.45,0.60);
    SetViewPort(H,ViewPortRectangle);
    SetRectangle(WindowRectangle,0.0,0.0,1.0,2.0);
    SetWindow(H,WindowRectangle);
    SetTextColor(H,QMTextColor);
    SetLineColor(H,QMTextColor);
  END;
END InitQuestionWindow;


PROCEDURE ShowQuestionWindow(VAR HMWindow,QMWindow : CommandWindowType;
                                 Str : TextStr;VAR Name : TestText);

VAR       Pos      : point;
          Number,k : CARDINAL;
          Ch       : CHAR;

BEGIN
  HideMenu(HMWindow);
  PutNr(7);
  HideCursor;
  WITH QMWindow^ DO
    SetFillColor(H,QMColor);
    FillRectangle(H,WindowRectangle);
    Pos.h:=0.05;
    Pos.v:=1.10;
    WriteString(H,Pos,Str);
    IF Str[0]='S' THEN
      Pos.h:=0.05;
      Pos.v:=0.55;
      WriteString(H,Pos,'No name = No save');
    END;
    Pos.h:=0.05;
    Pos.v:=0.10;
    FOR k:=0 TO 7 DO
      Name[k]:=' ';
    END;
    ReadString(H,Pos,Name);
    FOR k:=0 TO 7 DO
      Ch:=Name[k];
      Number:= ORD(Ch);
      IF ((Number >=48) AND (Number<=57)) OR
         ((Number >=65) AND (Number<=90)) OR
         ((Number >=97) AND (Number<=122)) THEN
        Name[k]:=Ch;
      ELSE
        Name[k]:=' ';
      END;
    END;
    HideMenu(QMWindow);
  END;
  ShowCursor;
END ShowQuestionWindow;


PROCEDURE StartRoutine(HMWindow,QMWindow : CommandWindowType);

VAR     Name      : TestText;
        TestName  : FileText;
        Number,K  : INTEGER;
        Ok        : BOOLEAN;
        dataFile  : File;

BEGIN
  Ok:=FALSE;
  StopTest;
  SetMeasWork(Rest);
```



```
  GetNrStimuli(Number);
  IF (Number=0) OR (Measuring[1] AND PlotPar.ElbowPower)THEN
    IF Number=0 THEN Error('No stimuli on');
                ELSE Error('No match inputsignals');
    END;
  ELSE
    WHILE NOT Ok DO
      ShowQuestionWindow(HMWindow,QMWindow,'Save test as? ',Name);
      IF (Name[0]=' ') AND  (Name[1]=' ') THEN
        Ok:=TRUE;
      ELSE
        FOR K:=0 TO 7 DO
          TestName[K]:=Name[K];
        END;
        TestName[8]:='.';
        TestName[9]:='m';
        TestName[10]:='a';
        TestName[11]:='t';
        Lookup(dataFile,TestName,FALSE);  (* check memory *)
        IF dataFile.res=notdone THEN
          Ok:=TRUE;
        ELSE
          Error('used filename');
          Ok:=FALSE;
        END;
      END;
    END;
    ShowHeadMenu(HMWindow);
    PutEmerg(FALSE);
    SetEnabled(TRUE);
    SetMeasWork(Meas);
    CASE PlotSort OF
      0..3 : ChangeWhat(Data)|
      4,5  : |
      6    : ChangeWhat(Oper);
    END;
    StartTest(Name);
  END;
END StartRoutine;


(*****************************************************************************)

(******************** Procedure for Quit *************************************)


PROCEDURE Quit;

BEGIN
   Signal(TheEnd);
END Quit;


(*****************************************************************************)

(******************** Procedure for ErrorProc ********************************)


(*Process*)PROCEDURE ErrorProc;

VAR       ErrWindow : CommandWindowType;
          ErrorMess : ErrorPtrType;
          Textpoint : point;
          t         : Time;

BEGIN
   SetPriority(89);
   InitErrorWindow(ErrWindow);
   Textpoint.h:= 0.1;
   Textpoint.v:= 0.3;
   LOOP
      ReceiveMessage(ErrorBox,ErrorMess);
      HideCursor;
      WITH ErrWindow^ DO
         SetFillColor(H,ErrorColor);
         FillRectangle(H,WindowRectangle);
         DrawRectangle(H,WindowRectangle);
```

```
        WriteString(H,Textpoint,ErrorMess^.ErrStr);
      END;
      ShowCursor;
      DISPOSE(ErrorMess);
      GetTime(t);
      IncTime(t,5000);
      WaitUntil(t);
      HideMenu(ErrWindow);
    END;
END ErrorProc;


(*****************************************************************************)

(******************** Procedure for ControlProc ***************************)


(*Process*)PROCEDURE ControlProc;

VAR SelectedBox                                    : CARDINAL;
    HMWindow,TMVibWindow,TMGalvWindow,
    TMWindow,QMWindow,TMPWindow,
    PMWindow,P1Window,P2Window,P3Window,P4Window,
    MEWindow,PTMWindow,SPwindow,FMWindow           : CommandWindowType;


BEGIN
  SetPriority(90);
  InitParameters;
  InitLogMenu(90);
  InitNumMenu(90);
  InitHeadMenu(HMWindow);
  InitSystemParMenu(SPwindow);
  InitStimuliMenu(TMWindow);
  InitStimuliSubMenu1(TMVibWindow);
  InitStimuliSubMenu(TMGalvWindow);
  InitTMPMenu(TMPWindow);
  InitTMPSubMenu(P1Window,P2Window,P3Window,P4Window);
  InitPlotMenu(PMWindow);
  InitForceMenu(FMWindow);
  InitQuestionWindow(QMWindow);
  ShowHeadMenu(HMWindow);
  ShowCursor;
  LOOP
    SelectedBox:=WaitMouseRectangle(HMWindow^.H);
    CASE SelectedBox OF
      7 :SystemParRoutine(HMWindow,SPwindow)|
      6 :FMRoutine(HMWindow,FMWindow,PTMWindow,QMWindow)|
      5 :TMRoutine(HMWindow,TMWindow,TMVibWindow,TMGalvWindow,
                   SPwindow,QMWindow,TMPWindow,P1Window,P2Window,
                   P3Window,P4Window)|
      4 :PMRoutine(HMWindow,PMWindow,PTMWindow,QMWindow)|
      3 :StartRoutine(HMWindow,QMWindow)|
      2 :StopTest|
      1 :Quit;
    END;
  END;
END ControlProc;


(*****************************************************************************)

(******************** Procedure for Initation *****************************)

PROCEDURE InitControl;

BEGIN
  NiceStartUpMessage;
  CreateProcess(ControlProc,10000);
  CreateProcess(ErrorProc,1000);
END InitControl;

END Control.
```



```
DEFINITION MODULE Decl;
(* Changes to 386 system *)

FROM Messages IMPORT MailBox;

FROM Kernel IMPORT Semaphore;

FROM Graphics IMPORT handle,rectangle;

EXPORT QUALIFIED     WhatType,SpeedType,TestText,MeasData,
                     FileText,MeasureType,MotorType,MeasType,
                     ErrorPtrType,ErrorType,MailType,PlotWindowType,
                     GetNr,PutNr,TestType,StimType,
                     PBox,MBox,OutBox,ErrorBox,TheEnd,
                     InitMonitors,DBox,DRBox,
                     Write,Error,
                     PutEmerg,
                     SetFileConst,GetFileConst,GetNrStimuli,PausMeas,
                     CheckWhat,PlotData,PlotSekv,PlotADIn,ChangeWhat,
                     CheckPlotWhat,GetEmerg,PlotType,
                     PausTest,StartTest,GetTest,GetFileName,
                     SetMeasWork,GetMeasWork,HOldData,HSekv,HCali,HMeas,
                     StopMeasure,StopTest,GetMomentData,PausRest,
                     PutMomentData,EndMeasure,GetSample,GetCounter,
                     SetEnabled,GetSpeed,PutVibSpeed,
                     PutGalvSpeed,GetMotorPar,SetParTest,PutMeas,
                     GetMotorData,PutMotorData,GetTStart,SetTStart,
                     StoreData,WaitForData,SendOldData,PausFile,
                     GetStimIndex,PutStimIndex,GetOldData,
                     WaitFull,CauseEmpty,SetOldReady,GetStimSort,
                     WaitOldReady,PutNrStimuli,GetMotorTime,
                     GetPlotSort,PutPlotSort,GetMeasIndex,PutMeasIndex;

TYPE    MeasData    =ARRAY[1..6] OF REAL;
        WhatType    =(Test,ADIn,Data,Oper);
        SpeedType   =ARRAY[1..2] OF REAL;
        TestText    =ARRAY[0..8] OF CHAR;
        FileText    =ARRAY[0..11] OF CHAR;
        MeasureType =(OldData,Sekv,Cali,Meas,Rest);
        TestType    =(Random,Sinus,PRBSSinus,PRBS,Paus);
        StimType    =(Vibration,Galvanic,None);
        ErrorType   =ARRAY[0..20] OF CHAR;

        MeasType    =RECORD
                         VibTest,GalvTest : TestType;
                     END;

        PlotType    =RECORD
                         NPower,XYMoment,ElbowPower,TestSeq,ADInCali : BOOLEAN;
                     END;

        MotorType   =RECORD
                         RampTime,WaitPaus,BreakTime,
                         Period,MaxSpeed,RandomTime,
                         RandomSpeed,tmax,PRBSPeriod  : REAL;
                     END;

        MotorStatus =RECORD
                         MaxSpeed,RandomTime,
                         RampTime,WaitPaus,
                         BreakTime,tmax,Period,
                         PRBSPeriod,RandomSpeed : ARRAY[1..4] OF REAL;
                     END;

        ErrorPtrType=POINTER TO RECORD
                         ErrStr:ErrorType;
                     END;

        MailType    =POINTER TO RECORD
                         MForce                 : MeasData;
                         ElbowForce,Stimulus    : SpeedType;
                     END;

        PlotWindowType = POINTER TO RECORD
                             XMin,YMin,XMax,YMax : REAL;
                             WindowRectangle     : rectangle;
                             H                   : handle;
```

```
                    END;

 VAR                PBox,MBox,DBox,DRBox,
                    ErrorBox,OutBox       : MailBox;
                    TheEnd                : Semaphore;

(* Initiation *)
PROCEDURE InitMonitors;
(* Errormonitor *)
PROCEDURE Error(ErrStr:ErrorType);
PROCEDURE Write(ErrStr:ErrorType);
(* Plotmonitor *)
PROCEDURE CheckWhat(VAR WhatKind:WhatType);
PROCEDURE PlotData():BOOLEAN;
PROCEDURE PlotSekv():BOOLEAN;
PROCEDURE PlotADIn():BOOLEAN;
PROCEDURE PutNr( Nr : INTEGER );
PROCEDURE GetNr(VAR Nr : INTEGER );
PROCEDURE ChangeWhat(WhatKind:WhatType);
PROCEDURE CheckPlotWhat(VAR WhatKind:WhatType;VAR Change : BOOLEAN);
PROCEDURE GetPlotSort(VAR PlotSort:PlotType);
PROCEDURE PutPlotSort(PlotSort:PlotType);
(* MeasureMonitor *)
PROCEDURE PausFile;
PROCEDURE PausMeas;
PROCEDURE PausRest;
PROCEDURE PausTest(Stim : StimType);
PROCEDURE StartTest(Name : TestText);
PROCEDURE GetStimSort(VAR VibRun1,GalvRun1 : TestType);
PROCEDURE SetOldReady;
PROCEDURE WaitOldReady;
PROCEDURE GetTest(VAR Test1 : TestType;Stim : StimType);
PROCEDURE GetFileName(VAR Name : FileText);
PROCEDURE GetNrStimuli(VAR Number1 : INTEGER);
PROCEDURE PutNrStimuli(Number1 : INTEGER);
PROCEDURE SetMeasWork(MWork : MeasureType);
PROCEDURE GetMeasWork(VAR MWork : MeasureType);
PROCEDURE StopMeasure(Stim : StimType);
PROCEDURE HOldData():BOOLEAN;
PROCEDURE HSekv():BOOLEAN;
PROCEDURE HCali():BOOLEAN;
PROCEDURE HMeas():BOOLEAN;
PROCEDURE EndMeasure():BOOLEAN;
PROCEDURE StopTest;
PROCEDURE PutMeas(MeasPar : MeasType);
PROCEDURE GetStimIndex(VAR StimIndex1 : INTEGER);
PROCEDURE PutStimIndex(StimIndex1 : INTEGER);
PROCEDURE SetParTest(Stimuli : StimType;WorkSort : TestType);
PROCEDURE GetMeasIndex() : INTEGER;
PROCEDURE PutMeasIndex(MeasIndex1 : INTEGER);
(* StatusMonitor *)
PROCEDURE GetSample(VAR h1 :REAL);
PROCEDURE GetCounter(VAR c:INTEGER);
PROCEDURE WaitFull;
PROCEDURE CauseEmpty;
PROCEDURE StoreData(Ready : BOOLEAN);
PROCEDURE WaitForData(VAR Samples: INTEGER);
PROCEDURE SendOldData(Ready : BOOLEAN;Samples : INTEGER);
PROCEDURE GetOldData(VAR Ready : BOOLEAN;VAR Samples : INTEGER);
PROCEDURE GetMomentData(VAR a1,b1,gamma1,alfa1 : REAL);
PROCEDURE PutMomentData(c: INTEGER;h1,a1,b1,gamma1,alfa1 : REAL);
PROCEDURE SetFileConst(Number : CARDINAL);
PROCEDURE GetFileConst(VAR Number : CARDINAL);
PROCEDURE PutEmerg(e:BOOLEAN);
PROCEDURE GetEmerg(VAR e:BOOLEAN);
(* MotorMonitor *)
PROCEDURE SetEnabled(Enabled1 : BOOLEAN);
PROCEDURE SetTStart(TStart1 : REAL);
PROCEDURE GetTStart(VAR TStart1 : REAL);
PROCEDURE GetSpeed(VAR CurrentSpeed:SpeedType);
PROCEDURE PutVibSpeed(CurrentSpeed,t : REAL;Test : INTEGER;
                      VAR TestEnd : BOOLEAN);
PROCEDURE PutGalvSpeed(CurrentSpeed,t : REAL;Test : INTEGER;
                       VAR TestEnd : BOOLEAN);
PROCEDURE GetMotorPar(VAR Enabled1 : BOOLEAN;VAR Speed1:SpeedType);
PROCEDURE PutMotorData(MData : MotorType;Nr,Index :INTEGER);
PROCEDURE GetMotorData(VAR MData : MotorType;Nr :INTEGER;Stim:StimType);
```



```
PROCEDURE GetMotorTime(VAR Time,RunTime : REAL; Nr : INTEGER;
                                          Stim : StimType);
END Decl.
```

```
IMPLEMENTATION MODULE Decl;
(* Changes to 386 system *)

FROM Messages IMPORT MailBox,SendMessage,ReceiveMessage,AcceptMessage,
                     InitMailBox;

FROM Kernel   IMPORT Wait,Signal,Semaphore,Event,InitEvent,Await,Cause,
                     InitSem;

FROM Storage  IMPORT ALLOCATE,DEALLOCATE;

FROM Graphics IMPORT rectangle,handle,WriteString,ReadString,HideCursor,
                     ShowCursor;


TYPE    PlotMonType   =RECORD
                         WhatSem   : Semaphore;
                         Change    : BOOLEAN;
                         What      : WhatType;
                         HelpNr    : INTEGER;
                         PlotSort  : PlotType;
                        END;

        MeasMonType   =RECORD
                         MeasSem                      : Semaphore;
                         MeasureWork                  : MeasureType;
                         Stop,NewName,OldReady        : BOOLEAN;
                         StimIndex,Number,MeasIndex   : INTEGER;
                         VibTest,GalvTest,VibRun,
                         GalvRun                      : TestType;
                         TestName                     : FileText;
                         Change                       : Event;
                        END;

        StatMonType   =RECORD
                         StatSem             : Semaphore;
                         FileConst,FileIndex : CARDINAL;
                         Emerg,FileReady,
                         FileEmpty           : BOOLEAN;
                         Counter             : INTEGER;
                         h,a,b,gamma,alfa    : REAL;
                         Change,Full,Empty   : Event;
                        END;
                MotorMonType  =RECORD
                         MotSem              : Semaphore;
                         Enabled             : BOOLEAN;
                         Speed               : SpeedType;
                         TStart,TestTime     : REAL;
                         TestView            : ARRAY[1..2] OF MotorStatus;
                         Change              : Event;
                        END;

VAR             WhatMon    : PlotMonType;
                StatMon    : StatMonType;
                MotorMon   : MotorMonType;
                MeasMon    : MeasMonType;
                MVErr      : BOOLEAN;

(*********************************************************************)

(*****************    initial  procedure        *********************)

PROCEDURE InitMonitors;

BEGIN
   InitWhatMon;
   InitStatusMonitor;
   InitMotorMonitor;
   InitMeasMonitor;
   MVErr:=FALSE;
END InitMonitors;

(*********************************************************************)
```



```
(*****************    plotmonitor  procedure    **********************)


(*monitor*)PROCEDURE InitWhatMon;

BEGIN
   WITH WhatMon DO
     InitSem(WhatSem,1);
     What:=Oper;
     Change:=FALSE;
     HelpNr:=0;
     PlotSort.NPower:=FALSE;
     PlotSort.XYMoment:=FALSE;
     PlotSort.TestSeq:=FALSE;
   END;
END InitWhatMon;


(*Monitor*) PROCEDURE PlotData():BOOLEAN;

VAR  ret:BOOLEAN;

BEGIN
   Wait(WhatMon.WhatSem);
   IF WhatMon.What=Data THEN
     ret:=TRUE;
   ELSE
     ret:=FALSE;
   END;
   Signal(WhatMon.WhatSem);
   RETURN ret;
END PlotData;


(*Monitor*) PROCEDURE PlotSekv():BOOLEAN;

VAR  ret:BOOLEAN;

BEGIN
   Wait(WhatMon.WhatSem);
   IF WhatMon.What=Test THEN
     ret:=TRUE;
   ELSE
     ret:=FALSE;
   END;
   Signal(WhatMon.WhatSem);
   RETURN ret;
END PlotSekv;


(*Monitor*) PROCEDURE PlotADIn():BOOLEAN;

VAR  ret:BOOLEAN;

BEGIN
   Wait(WhatMon.WhatSem);
   IF WhatMon.What=ADIn THEN
     ret:=TRUE;
   ELSE
     ret:=FALSE;
   END;
   Signal(WhatMon.WhatSem);
   RETURN ret;
END PlotADIn;


(*Monitor*) PROCEDURE ChangeWhat(WhatKind:WhatType);

BEGIN
   Wait(WhatMon.WhatSem);
   WhatMon.What:=WhatKind;
   WhatMon.Change:=TRUE;
   Signal(WhatMon.WhatSem);
END ChangeWhat;


(*Monitor*) PROCEDURE GetPlotSort(VAR PlotSort: PlotType);
```

```
BEGIN
   Wait(WhatMon.WhatSem);
   PlotSort:=WhatMon.PlotSort;
   Signal(WhatMon.WhatSem);
END GetPlotSort;


(*Monitor*) PROCEDURE PutPlotSort(PlotSort: PlotType);

BEGIN
   Wait(WhatMon.WhatSem);
   WhatMon.PlotSort:=PlotSort;
   Signal(WhatMon.WhatSem);
END PutPlotSort;


(*Monitor*) PROCEDURE PutNr( Nr : INTEGER );

BEGIN
   Wait(WhatMon.WhatSem);
   WhatMon.HelpNr:=Nr;
   Signal(WhatMon.WhatSem);
END PutNr;


(*Monitor*) PROCEDURE GetNr(VAR Nr : INTEGER );

BEGIN
   Wait(WhatMon.WhatSem);
   Nr:=WhatMon.HelpNr;
   Signal(WhatMon.WhatSem);
END GetNr;


(*Monitor*) PROCEDURE CheckWhat(VAR WhatKind:WhatType);

BEGIN
   Wait(WhatMon.WhatSem);
   WhatKind:=WhatMon.What;
   Signal(WhatMon.WhatSem);
END CheckWhat;


(*Monitor*) PROCEDURE CheckPlotWhat(VAR WhatKind:WhatType;VAR Change : BOOLEAN);

BEGIN
   Wait(WhatMon.WhatSem);
   WhatKind:=WhatMon.What;
   Change:=WhatMon.Change;
   WhatMon.Change:=FALSE;
   Signal(WhatMon.WhatSem);
END CheckPlotWhat;


(*************************************************************************)

(*****************    errormonitor  procedure   ********************)


PROCEDURE Error(ErrStr:ErrorType);

VAR      ErrorPtr : ErrorPtrType;

BEGIN
   NEW(ErrorPtr);
   ErrorPtr^.ErrStr:=ErrStr;
   SendMessage(ErrorBox,ErrorPtr);
END Error;


PROCEDURE Write(ErrStr:ErrorType);

VAR      ErrorPtr : ErrorPtrType;

BEGIN
   NEW(ErrorPtr);
```



```
   ErrorPtr^.ErrStr:=ErrStr;
   SendMessage(OutBox,ErrorPtr);
END Write;


(*************************************************************************)

(*****************   Measuremonitor  procedure   ********************)


(*monitor*)PROCEDURE InitMeasMonitor;

BEGIN
  WITH MeasMon DO
    InitSem(MeasSem,1);
    InitEvent(Change,MeasSem);
    MeasureWork:=Rest;
    Stop:=TRUE;
    Number:=0;
    OldReady:=FALSE;
    StimIndex:=0;
    VibTest:=Paus;
    GalvTest:=Paus;
    VibRun:=Paus;
    GalvRun:=Paus;
    NewName:=FALSE;
  END;
END InitMeasMonitor;


(*monitor*)PROCEDURE PausFile;

BEGIN
  WITH MeasMon DO
    Wait(MeasSem);
    IF (MeasureWork<>Meas) OR (MeasureWork<>OldData) THEN
      Await(Change);
    END;
    Signal(MeasSem);
  END;
END PausFile;


(*monitor*)PROCEDURE PausMeas;

BEGIN
  WITH MeasMon DO
    Wait(MeasSem);
    WHILE MeasureWork=Rest DO
      Await(Change);
    END;
    Signal(MeasSem);
  END;
END PausMeas;


(*monitor*)PROCEDURE PausRest;

BEGIN
  WITH MeasMon DO
    Wait(MeasSem);
    WHILE MeasureWork=Meas DO
      Await(Change);
    END;
    Signal(MeasSem);
  END;
END PausRest;


(*monitor*)PROCEDURE PausTest(Stim : StimType);

BEGIN
  WITH MeasMon DO
    Wait(MeasSem);
    CASE Stim OF
      Vibration : WHILE VibRun=Paus DO
                    Await(Change);
```

```
                END|
      Galvanic  : WHILE GalvRun=Paus DO
                    Await(Change);
                END;
    END;
    Signal(MeasSem);
  END;
END PausTest;


(*monitor*)PROCEDURE StartTest(Name : TestText);

VAR          K : INTEGER;

BEGIN
  WITH MeasMon DO
    Wait(MeasSem);
    FOR K:=0 TO 7 DO
      TestName[K]:=Name[K];
    END;
    TestName[8]:='.';
    TestName[9]:='m';
    TestName[10]:='a';
    TestName[11]:='t';
    VibRun:=VibTest;
    GalvRun:=GalvTest;
    Stop:=FALSE;
    NewName:=TRUE;
    Cause(Change);
    Signal(MeasSem);
  END;
END StartTest;


(*monitor*)PROCEDURE GetTest(VAR Test1 : TestType; Stim : StimType);

BEGIN
  WITH MeasMon DO
    Wait(MeasSem);
    IF MeasureWork=OldData THEN
      Await(Change);
    END;
    CASE Stim OF
      Vibration : Test1:=VibRun|
      Galvanic  : Test1:=GalvRun;
    END;
    Signal(MeasSem);
  END;
END GetTest;


(*monitor*)PROCEDURE GetNrStimuli(VAR Number1 : INTEGER);

BEGIN
  WITH MeasMon DO
    Wait(MeasSem);
    IF MeasureWork=OldData THEN
      Number1:=Number;
    ELSE
      Number1:=0;
      IF VibTest<>Paus THEN
        Number1:=Number1+1;
      END;
      IF GalvTest<>Paus THEN
        Number1:=Number1+1;
      END;
    END;
    Signal(MeasSem);
  END;
END GetNrStimuli;


(*monitor*)PROCEDURE PutNrStimuli(Number1 : INTEGER);

BEGIN
  WITH MeasMon DO
    Wait(MeasSem);
```



```
    Number:=Number1;
    Signal(MeasSem);
  END;
END PutNrStimuli;


(*monitor*)PROCEDURE SetParTest(Stimuli : StimType;WorkSort : TestType);

BEGIN
  WITH MeasMon DO
    Wait(MeasSem);
    CASE Stimuli OF
      Vibration : VibTest:=WorkSort|
      Galvanic  : GalvTest:=WorkSort;
    END;
    StimIndex:=0;
    IF VibTest<>Paus THEN
      StimIndex:=StimIndex+2;
    END;
    IF GalvTest<>Paus THEN
      StimIndex:=StimIndex+1;
    END;
    Signal(MeasSem);
  END;
END SetParTest;

(*monitor*)PROCEDURE GetFileName(VAR Name : FileText);

BEGIN
  WITH MeasMon DO
    Wait(MeasSem);
    WHILE NOT NewName AND (MeasureWork=OldData) DO
      Await(Change);
    END;
    Name:=TestName;
    NewName:=FALSE;
    Signal(MeasSem);
  END;
END GetFileName;


(*monitor*)PROCEDURE SetMeasWork(MWork : MeasureType);

BEGIN
  WITH MeasMon DO
    Wait(MeasSem);
    MeasureWork:=MWork;
    Cause(Change);
    Signal(MeasSem);
  END;
END SetMeasWork;


(*monitor*)PROCEDURE GetMeasWork( VAR MWork : MeasureType);

BEGIN
  WITH MeasMon DO
    Wait(MeasSem);
    MWork:=MeasureWork;
    Signal(MeasSem);
  END;
END GetMeasWork;


(*monitor*)PROCEDURE SetOldReady;

BEGIN
  WITH MeasMon DO
    Wait(MeasSem);
    OldReady:=TRUE;
    Cause(Change);
    Signal(MeasSem);
  END;
END SetOldReady;


(*monitor*)PROCEDURE GetStimSort(VAR VibRun1,GalvRun1 : TestType);
```

```
VAR          K : INTEGER;

BEGIN
  WITH MeasMon DO
    Wait(MeasSem);
    VibRun1:=VibTest;
    GalvRun1:=GalvTest;
    Signal(MeasSem);
  END;
END GetStimSort;


(*monitor*)PROCEDURE WaitOldReady;

BEGIN
  WITH MeasMon DO
    Wait(MeasSem);
    WHILE NOT OldReady DO
      Await(Change);
    END;
    Signal(MeasSem);
  END;
END WaitOldReady;


(*monitor*)PROCEDURE StopMeasure(Stim : StimType);

BEGIN
  WITH MeasMon DO
    Wait(MeasSem);
    CASE Stim OF
      Vibration : VibRun:=Paus|
      Galvanic  : GalvRun:=Paus;
    END;
    Cause(Change);
    WHILE (VibRun<>Paus) OR (GalvRun<>Paus) DO
      Await(Change);
    END;
    Stop:=TRUE;
    MeasureWork:=Rest;
    Signal(MeasSem);
  END;
END StopMeasure;


(*monitor*)PROCEDURE StopTest;

BEGIN
  Wait(MeasMon.MeasSem);
  MeasMon.Stop:=TRUE;
  MeasMon.OldReady:=FALSE;
  Cause(MeasMon.Change);
  Signal(MeasMon.MeasSem);
END StopTest;


(*monitor*)PROCEDURE PutMeas(MeasPar : MeasType);

BEGIN
  WITH MeasMon DO
    Wait(MeasSem);
    VibTest:=MeasPar.VibTest;
    GalvTest:=MeasPar.GalvTest;
    StimIndex:=0;
    IF VibTest<>Paus THEN
      StimIndex:=StimIndex+2;
    END;
    IF GalvTest<>Paus THEN
      StimIndex:=StimIndex+1;
    END;
    Signal(MeasSem);
  END;
END PutMeas;


(*monitor*)PROCEDURE GetMeasIndex() : INTEGER;
```



```
VAR    MeasIndex1 : INTEGER;

BEGIN
  Wait(MeasMon.MeasSem);
  MeasIndex1:=MeasMon.MeasIndex;
  Signal(MeasMon.MeasSem);
  RETURN MeasIndex1;
END GetMeasIndex;


(*monitor*)PROCEDURE PutMeasIndex(MeasIndex1 : INTEGER);

BEGIN
  Wait(MeasMon.MeasSem);
  MeasMon.MeasIndex:=MeasIndex1;
  Signal(MeasMon.MeasSem);
END PutMeasIndex;


(*monitor*)PROCEDURE GetStimIndex(VAR StimIndex1 : INTEGER);

BEGIN
  Wait(MeasMon.MeasSem);
  StimIndex1:=MeasMon.StimIndex;
  Signal(MeasMon.MeasSem);
END GetStimIndex;


(*monitor*)PROCEDURE PutStimIndex(StimIndex1 : INTEGER);

BEGIN
  Wait(MeasMon.MeasSem);
  MeasMon.StimIndex:=StimIndex1;
  Signal(MeasMon.MeasSem);
END PutStimIndex;


(*monitor*)PROCEDURE EndMeasure():BOOLEAN;

VAR       Running : BOOLEAN;
BEGIN
  Wait(MeasMon.MeasSem);
  Running:= MeasMon.Stop;
  Signal(MeasMon.MeasSem);
  RETURN Running;
END EndMeasure;


(*monitor*)PROCEDURE HOldData():BOOLEAN;

VAR     Running : BOOLEAN;

BEGIN
   WITH MeasMon DO
     Wait(MeasMon.MeasSem);
     WHILE Stop DO
       Await(Change);
     END;
     IF MeasMon.MeasureWork=OldData THEN
       Running:=TRUE
     ELSE
       Running:=FALSE;
     END;
     Signal(MeasMon.MeasSem);
   END;
   RETURN Running;
END HOldData;


(*monitor*)PROCEDURE HSekv():BOOLEAN;

VAR     Running : BOOLEAN;

BEGIN
   WITH MeasMon DO
     Wait(MeasMon.MeasSem);
```

```
      IF Stop THEN
        Await(Change);
      END;
      IF MeasMon.MeasureWork=Sekv THEN
        Running:=TRUE
      ELSE
        Running:=FALSE;
      END;
      Signal(MeasMon.MeasSem);
    END;
    RETURN Running;
END HSekv; (*monitor*)


(*monitor*)PROCEDURE HCali():BOOLEAN;

VAR     Running : BOOLEAN;

BEGIN
    WITH MeasMon DO
      Wait(MeasMon.MeasSem);
      IF Stop THEN
        Await(Change);
      END;
      IF MeasMon.MeasureWork=Cali THEN
        Running:=TRUE
      ELSE
        Running:=FALSE;
      END;
      Signal(MeasMon.MeasSem);
    END;
    RETURN Running;
END HCali;(*monitor*)



(*monitor*)PROCEDURE HMeas():BOOLEAN;

VAR     Running : BOOLEAN;

BEGIN
    WITH MeasMon DO
      Wait(MeasMon.MeasSem);
      IF MeasMon.MeasureWork=Meas THEN
        Running:=TRUE
      ELSE
        Running:=FALSE;
      END;
      Signal(MeasMon.MeasSem);
    END;
    RETURN Running;
END HMeas;


(*****************************************************************)

(*****************   statusmonitor  procedure   ******************)


(*monitor*)PROCEDURE InitStatusMonitor;

BEGIN
  WITH StatMon DO
    InitSem(StatSem,1);
    InitEvent(Change,StatSem);
    InitEvent(Full,StatSem);
    InitEvent(Empty,StatSem);
    Emerg:=FALSE;
    FileReady:=FALSE;
    FileEmpty:=FALSE;
    Counter:=10;
    FileIndex:=0;
    h:=0.1;
    a:=1.0;
    b:=1.0;
    gamma:=1.0;
    alfa:=1.0;
```



```
  END;
END InitStatusMonitor;


(*monitor*)PROCEDURE SetFileConst(Number : CARDINAL);

BEGIN
  WITH StatMon DO
    Wait(StatSem);
    FileConst:=Number;
    Cause(Change);
    Signal(StatSem);
  END;
END SetFileConst;


(*monitor*)PROCEDURE GetFileConst(VAR Number : CARDINAL);

BEGIN
  WITH StatMon DO
    Wait(StatSem);
    Number:=FileConst;
    Signal(StatSem);
  END;
END GetFileConst;


(*Monitor*)PROCEDURE PutEmerg(e :BOOLEAN);

BEGIN
  WITH StatMon DO
     Wait (StatSem);
     Emerg:= e;
     Signal(StatSem);
  END;
END PutEmerg;


(*Monitor*)PROCEDURE GetEmerg(VAR e :BOOLEAN);

BEGIN
  WITH StatMon DO
     Wait (StatSem);
     e:=Emerg;
     Signal(StatSem);
  END;
END GetEmerg;


(*Monitor*)PROCEDURE WaitFull;

BEGIN
  WITH StatMon DO
    Wait(StatSem);
     WHILE NOT FileEmpty DO
       Await(Empty);
     END;
    Signal(StatSem);
  END;
END WaitFull;


(*Monitor*)PROCEDURE CauseEmpty;

BEGIN
  WITH StatMon DO
    Wait(StatSem);
      FileReady:=FALSE;
      FileEmpty:=TRUE;
      Cause(Empty);
    Signal(StatSem);
  END;
END CauseEmpty;


(*Monitor*)PROCEDURE StoreData(Ready : BOOLEAN);
```

```
 BEGIN
   WITH StatMon DO
     Wait (StatSem);
     FileIndex:=FileIndex+1;
     IF (FileIndex=FileConst) OR Ready THEN
       IF Ready THEN
         FileIndex:=FileIndex-1;
       END;
       FileReady:=TRUE;
       Cause(Full);
       WHILE FileReady DO
         Await(Empty);
       END;
     END;
     Signal(StatSem);
   END;
 END StoreData;


(*Monitor*)PROCEDURE WaitForData(VAR Samples: INTEGER);

 BEGIN
   WITH StatMon DO
     Wait(StatSem);
     WHILE NOT FileReady DO
       Await(Full);
     END;
     FileReady:=FALSE;
     Samples:=FileIndex;
     FileIndex:=0;
     Cause(Empty);
     Signal(StatSem);
   END;
 END WaitForData;


(*Monitor*)PROCEDURE SendOldData(Ready : BOOLEAN;Samples : INTEGER);

 BEGIN
   WITH StatMon DO
     Wait (StatSem);
     FileIndex:=Samples;
     IF (FileIndex=FileConst) OR NOT Ready THEN
       IF NOT Ready THEN
         FileIndex:=FileIndex-1;
       END;
       FileReady:=TRUE;
       Cause(Full);
       WHILE FileReady DO
         Await(Empty);
       END;
     END;
     Signal(StatSem);
   END;
 END SendOldData;


(*Monitor*)PROCEDURE GetOldData(VAR Ready : BOOLEAN;VAR Samples : INTEGER);

 BEGIN
   Ready:=TRUE;
   WITH StatMon DO
     Wait (StatSem);
     WHILE NOT FileReady DO
       Await(Full);
     END;
     Samples:=FileIndex;
     IF (FileIndex<>FileConst) THEN
       Ready:=FALSE;
     END;
     FileIndex:=0;
     FileReady:=FALSE;
     Cause(Empty);
     Signal(StatSem);
   END;
 END GetOldData;
```



```
(*Monitor*)PROCEDURE GetSample(VAR h1 :REAL);

 BEGIN
   WITH StatMon DO
     Wait (StatSem);
     h1:=h;
     Signal(StatSem);
   END;
 END GetSample;


(*Monitor*)PROCEDURE GetCounter(VAR c :INTEGER);

 BEGIN
   WITH StatMon DO
     Wait (StatSem);
     c:=Counter;
     Signal(StatSem);
   END;
 END GetCounter;


(*Monitor*)PROCEDURE GetMomentData(VAR a1,b1,gamma1,alfa1 : REAL);

 BEGIN
   WITH StatMon DO
     Wait (StatSem);
     a1:=a;
     b1:=b;
     gamma1:=gamma;
     alfa1:=alfa;
     Signal(StatSem);
   END;
 END GetMomentData;


(*Monitor*)PROCEDURE PutMomentData(c : INTEGER; h1,a1,b1,gamma1,alfa1 : REAL);

 BEGIN
   WITH StatMon DO
     Wait (StatSem);
     Counter:=c;
     h:=h1;
     a:=a1;
     b:=b1;
     gamma:=gamma1;
     alfa:=alfa1;
     Signal(StatSem);
   END;
 END PutMomentData;


(*********************************************************************)

(*****************    motormonitor  procedure   *********************)

 (*monitor*)PROCEDURE InitMotorMonitor;

 VAR          K,L : INTEGER;

 BEGIN
   WITH MotorMon DO
     InitSem(MotSem,1);
     InitEvent(Change,MotSem);
     Enabled:=FALSE;
     Speed[1]:=0.0;
     Speed[2]:=0.0;
     TStart:=30.0;
     TestTime:=0.0;
     FOR L:=1 TO 2 DO
       WITH TestView[L] DO
         FOR K:=1 TO 4 DO
           PRBSPeriod[K]:=2.0;
           RandomSpeed[K]:=2.0;
           MaxSpeed[K]:=5.0;
```

```
          RampTime[K]:=5.0;
          WaitPaus[K]:=10.0;
          RandomTime[K]:=10.0;
          BreakTime[K]:=10.0;
          Period[K]:=10.0;
          tmax[K]:=50.0;
        END;
      END;
    END;
  END;
END InitMotorMonitor;


(*monitor*)PROCEDURE SetTStart(TStart1 : REAL);

BEGIN
  WITH MotorMon DO
    Wait(MotSem);
    TStart:=TStart1;
    Signal(MotSem);
  END;
END SetTStart;


(*monitor*)PROCEDURE GetTStart(VAR TStart1 : REAL);

BEGIN
  WITH MotorMon DO
    Wait(MotSem);
    TStart1:=TStart;
    Signal(MotSem);
  END;
END GetTStart;



(*monitor*)PROCEDURE SetEnabled(Enabled1 : BOOLEAN);

BEGIN
  WITH MotorMon DO
    Wait(MotSem);
    Enabled:=Enabled1;
    TestTime:=0.0;
    Cause(Change);
    Signal(MotSem);
  END;
END SetEnabled;


(*monitor*)PROCEDURE GetSpeed(VAR CurrentSpeed:SpeedType);

BEGIN
   Wait(MotorMon.MotSem);
   CurrentSpeed:=MotorMon.Speed;
   Signal(MotorMon.MotSem);
END GetSpeed;


(*monitor*)PROCEDURE PutVibSpeed(CurrentSpeed,t:REAL;Test : INTEGER;
                                 VAR TestEnd : BOOLEAN);

BEGIN
   TestEnd:=FALSE;
   Wait(MotorMon.MotSem);
   MotorMon.Speed[1]:=CurrentSpeed;
   IF t>MotorMon.TestView[1].tmax[Test] THEN
      TestEnd:=TRUE;
   END;
   MotorMon.TestTime:=t;
   Signal(MotorMon.MotSem);
END PutVibSpeed;

(*monitor*)PROCEDURE PutGalvSpeed(CurrentSpeed,t:REAL;Test : INTEGER;
                                 VAR TestEnd : BOOLEAN);

BEGIN
   TestEnd:=FALSE;
```



```
   Wait(MotorMon.MotSem);
   MotorMon.Speed[2]:=CurrentSpeed;
   IF t>MotorMon.TestView[2].tmax[Test] THEN
      TestEnd:=TRUE;
   END;
   MotorMon.TestTime:=t;
   Signal(MotorMon.MotSem);
END PutGalvSpeed;


(*monitor*)PROCEDURE GetMotorPar(VAR Enabled1 : BOOLEAN;
                                 VAR Speed1:SpeedType);

BEGIN
   WITH MotorMon DO
     Wait(MotSem);
     Enabled1:=Enabled;
     Speed1:=Speed;
     Signal(MotSem);
   END;
END GetMotorPar;


(*monitor*)PROCEDURE PutMotorData(MData : MotorType; Nr,Index : INTEGER);

BEGIN
   Wait(MotorMon.MotSem);
   WITH MotorMon.TestView[Index] DO
     RampTime[Nr]:=MData.RampTime;
     WaitPaus[Nr]:=MData.WaitPaus;
     RandomTime[Nr]:=MData.RandomTime;
     BreakTime[Nr]:=MData.BreakTime;
     Period[Nr]:=MData.Period;
     PRBSPeriod[Nr]:=MData.PRBSPeriod;
     MaxSpeed[Nr]:=MData.MaxSpeed;
     RandomSpeed[Nr]:=MData.RandomSpeed;
     tmax[Nr]:=MData.tmax;
   END;
   Signal(MotorMon.MotSem);
END PutMotorData;


(*monitor*)PROCEDURE GetMotorData(VAR MData : MotorType; Nr : INTEGER;
                                  Stim : StimType);

VAR       Index : INTEGER;

BEGIN
   CASE Stim OF
     Vibration : Index:=1|
     Galvanic  : Index:=2;
   END;
   Wait(MotorMon.MotSem);
   WITH MotorMon.TestView[Index] DO
     MData.RampTime:=RampTime[Nr];
     MData.WaitPaus:=WaitPaus[Nr];
     MData.RandomTime:=RandomTime[Nr];
     MData.BreakTime:=BreakTime[Nr];
     MData.MaxSpeed:=MaxSpeed[Nr];
     MData.Period:=Period[Nr];
     MData.PRBSPeriod:=PRBSPeriod[Nr];
     MData.RandomSpeed:=RandomSpeed[Nr];
     MData.tmax:=tmax[Nr];
   END;
   Signal(MotorMon.MotSem);
END GetMotorData;

(*monitor*)PROCEDURE GetMotorTime(VAR Time,RunTime : REAL; Nr : INTEGER;
                                  Stim : StimType);

VAR       Index : INTEGER;

BEGIN
   CASE Stim OF
     Vibration : Index:=1|
     Galvanic  : Index:=2;
   END;
```

```
    Wait(MotorMon.MotSem);
    WITH MotorMon.TestView[Index] DO
      Time:=tmax[Nr];
    END;
    RunTime:=MotorMon.TestTime;
    Signal(MotorMon.MotSem);
 END GetMotorTime;

END Decl.
```



```
DEFINITION MODULE FileProcess;

EXPORT QUALIFIED FileHandler;

PROCEDURE FileHandler;

END FileProcess.
```

```
IMPLEMENTATION MODULE FileProcess;
(* Changes to 386 system *)

 FROM Kernel      IMPORT CreateProcess,SetPriority,Signal;

 FROM Decl        IMPORT Error,DBox,DRBox,SpeedType,FileText,
                         MeasData,EndMeasure,GetMeasWork,PausFile,
                         GetMotorPar,HOldData,MailType,SendOldData,
                         HMeas,HSekv,GetFileName,WaitForData,
                         StopMeasure,StopTest,MeasureType,StimType,
                         GetFileConst,GetNrStimuli,PausRest,
                         GetStimIndex,PutStimIndex,
                         WaitFull,CauseEmpty,SetOldReady,
                         PutNrStimuli,SetMeasWork,GetMeasIndex,
                         PutMeasIndex;

 FROM Messages    IMPORT AcceptMessage,ReceiveMessage,SendMessage;

 FROM Strings     IMPORT Assign;

 FROM ConvReal    IMPORT RealToString,StringToReal;

 FROM FileSystem IMPORT File,Lookup,Delete,WriteChar,ReadChar,Close,
                         Again,done,notdone;

 FROM MathLib     IMPORT round,float;

 CONST               EOL       = 36C;
                     SPACE     = ' ';


VAR                    dataFile : File;
                      Number,StimIndex,
                      NrOffData,measindex : INTEGER;
                      FileConst : CARDINAL;
                          DMail : MailType;
                           Work : MeasureType;
                              y : MeasData;
                          Speed : SpeedType;
               Allowed,NewStart : BOOLEAN;
                           Name : FileText;
                          Empty : BOOLEAN;

(*************************************************************************)

(*****************        file procedure        ********************)


 PROCEDURE CharOk(Ch : CHAR) : BOOLEAN;

 VAR       Number : CARDINAL;

 BEGIN
    Number:= ORD(Ch);
    RETURN (Number >= 33) AND (Number <= 126);
 END CharOk;


 PROCEDURE SkipAsciiBlanks(VAR f : File;VAR FirstNonBlank : CHAR);

 VAR         Ch : CHAR;
             ok : BOOLEAN;

 BEGIN
    REPEAT
      ReadChar(f,Ch);            (* remove blanks in line *)
      ok:= f.res = done;
    UNTIL ((Ch <> ' ') AND CharOk(Ch)) OR f.eof;
    FirstNonBlank:= Ch;
 END SkipAsciiBlanks;


 PROCEDURE ReadString(VAR f:File; VAR s : ARRAY OF CHAR);

 CONST       maxi = 40;

 VAR         Ch       : CHAR;
```



```
             Temps    : ARRAY [0..maxi] OF CHAR;
             i        : CARDINAL;
             Stop,Ok  : BOOLEAN;

BEGIN
   SkipAsciiBlanks(f,Temps[0]);   (* reads string from file *)
   i:= 1;
   Stop:= (NOT CharOk(Temps[0])) OR f.eof;
   Ok:= f.res = done;
   WHILE Ok AND (NOT Stop) DO
     ReadChar(f,Ch);
     Ok:= (f.res = done) AND (i <= maxi);
     Stop:= (NOT CharOk(Ch)) OR (Ch = ' ') OR f.eof;
     IF NOT Stop THEN
       Temps[i]:= Ch;
       INC(i);
     END;
   END;
   Again(f);
   IF (NOT Ok) AND (NOT f.eof) THEN
     Error('error reading');
   END;
   Temps[i]:= 0C;
   Assign(Temps,s);
END ReadString;


PROCEDURE ReadReal(VAR f:File; VAR Value : REAL;VAR Last : BOOLEAN);

VAR        Text : ARRAY [0..20] OF CHAR;
           Pos  : CARDINAL;

BEGIN
   ReadString(f,Text);     (* reads real from file *)
   IF f.eof THEN
     Last:=TRUE;
   ELSE
     Last:=FALSE;
   END;
   Pos := 0;
   StringToReal(Text,Pos,Value);
   IF (Pos = 0) AND (NOT f.eof) THEN
     Error('error reading');
   END;
END ReadReal;


PROCEDURE CloseWrite(VAR f : File);

BEGIN
   WriteString(f,0C);
   Close(f);
END CloseWrite;


PROCEDURE CloseRead(VAR f : File);

BEGIN
   Close(f);
END CloseRead;


PROCEDURE WriteString(VAR f : File; s : ARRAY OF CHAR);

VAR
   i, maxi : CARDINAL;
        ok : BOOLEAN;
BEGIN
   i := 0;
   maxi := HIGH(s);       (* writes real on string *)
   ok := TRUE;
   WHILE (i <= maxi) AND (s[i] <> 0C) AND ok DO
     WriteChar(f,s[i]);
     INC(i);
     ok := f.res = done;
   END;
   IF NOT ok THEN
```

```
     Error('error writing');
   END;
 END WriteString;


 PROCEDURE WriteReal(VAR f : File; Value : REAL);

 VAR Text : ARRAY [0..20] OF CHAR;

 BEGIN
    RealToString(Value,Text,8);  (* writes real on file *)
    WriteString(f,Text);
 END WriteReal;


(*******************************************************************)

(*****************    filehandle procedure      *********************)


 PROCEDURE GetFirstData(VAR Ok :BOOLEAN);

 VAR           FileEnd           : BOOLEAN;
               Num,Meas,Stim     : REAL;
               i,GetMeasSlask    : INTEGER;

 BEGIN
    Ok:=TRUE;
    ReadReal(dataFile,Num,FileEnd);
    ReadReal(dataFile,Meas,FileEnd);
    ReadReal(dataFile,Stim,FileEnd);
    Number:=round(Num);
    PutMeasIndex(round(Meas));
    StimIndex:=round(Stim);
    PutNrStimuli(Number);
    PutStimIndex(StimIndex);
    IF (measindex<>1) AND
       (measindex<>2) AND
       (measindex<>4)  THEN
      Error('wrong datafiles');
    END;
    FOR i:=1 TO 3 DO
      ReadReal(dataFile,Meas,FileEnd);
    END;
    FOR i:=1 TO Number DO
      ReadReal(dataFile,Meas,FileEnd);
    END;
    WaitFull;
    SetOldReady;
 END GetFirstData;


 PROCEDURE ReadData(VAR Ok :BOOLEAN;VAR Samples : INTEGER);

 VAR           i,k        : CARDINAL;
               FileEnd    : BOOLEAN;

 BEGIN
    Ok:=TRUE;
    FileEnd:=FALSE;
    Samples:=0;
    WHILE NOT FileEnd AND (Samples<NrOfData) DO
      AcceptMessage(DBox,DMail);
      IF DMail<> NIL THEN
        WITH DMail^ DO
           FOR i:=1 TO 6 DO
              ReadReal(dataFile,MForce[i],FileEnd);
           END;
           IF measindex=4 THEN
             FOR i:=1 TO 2 DO
               ReadReal(dataFile,ElbowForce[i],FileEnd);
             END;
           END;
           CASE StimIndex OF
             3 : ReadReal(dataFile,Stimulus[1],FileEnd);
                 ReadReal(dataFile,Stimulus[2],FileEnd)|
             2 : ReadReal(dataFile,Stimulus[1],FileEnd)|
```



```
             1 : ReadReal(dataFile,Stimulus[2],FileEnd)|
             0 :;
           END;
         END;
       END;              (* memory                          *)
       SendMessage(DRBox,DMail);
       Samples:=Samples+1;
    END;
    IF FileEnd THEN
      Ok:=FALSE;
    END;
 END ReadData;


 PROCEDURE StoreFirstData;

 VAR          Length,i  : INTEGER;

 BEGIN
    WriteReal(dataFile,float(Number));
    WriteChar(dataFile, SPACE);
    CASE measindex OF
      1,2  : WriteReal(dataFile,2.0)|
      4    : WriteReal(dataFile,4.0);
    END;
    WriteChar(dataFile, SPACE);
    WriteReal(dataFile,float(StimIndex));
    WriteChar(dataFile, SPACE);
    FOR i:=1 TO 3 DO
      WriteReal(dataFile,0.0);
      WriteChar(dataFile, SPACE);
    END;
    FOR i:=1 TO Number-1 DO
      WriteReal(dataFile,0.0);
      WriteChar(dataFile, SPACE);
    END;
    (* !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!! *)
    (* !!                                  !! *)
    (* !! lagg till tv  0.0 vid measindex=4 !! *)
    (* !!                                  !! *)
    (* !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!! *)
    IF measindex=4 THEN
      FOR i:=1 TO 2 DO
        WriteReal(dataFile,0.0);
        WriteChar(dataFile, SPACE);
      END;
    END;
    WriteReal(dataFile,0.0);
    WriteChar(dataFile, EOL);
    Empty:=FALSE;
    WHILE NOT Empty DO
      AcceptMessage(DBox,DMail);
      IF DMail=NIL THEN
        Empty:=TRUE;
      ELSE
        SendMessage(DRBox,DMail);
      END;
    END;
    CauseEmpty;
 END StoreFirstData;


 PROCEDURE WriteData(Samples : INTEGER);

 VAR          i,K       : INTEGER;

 BEGIN
   FOR K:=1 TO Samples DO
     ReceiveMessage(DBox,DMail);
     IF DMail<> NIL THEN
       WITH DMail^ DO
         Speed:=Stimulus;
         FOR i := 1 TO 6 DO
           WriteReal(dataFile,MForce[i]);
           WriteChar(dataFile, SPACE);
         END;
         IF measindex=4 THEN
```

```
          FOR i := 1 TO 2 DO
            WriteReal(dataFile,ElbowForce[i]);
            WriteChar(dataFile, SPACE);
          END;
        END;
        CASE StimIndex OF
          3 : WriteReal(dataFile,Speed[1]);
              WriteChar(dataFile, SPACE);
              WriteReal(dataFile,Speed[2])|
          2 : WriteReal(dataFile,Speed[1])|
          1 : WriteReal(dataFile,Speed[2])|
          0 :;
        END;
      END;
      WriteChar(dataFile, EOL);
      SendMessage(DRBox,DMail);
      Allowed:= TRUE;
    ELSIF Allowed THEN
      Error('Plot data ovfl (Meas)');
      Allowed:= FALSE;
    END;
  END;
END WriteData;


(*********************************************************************)

(*****************    datahandle  procedure     *********************)

PROCEDURE HandleOldData;

VAR               Ok    : BOOLEAN;
                  Samples : INTEGER;

BEGIN
  WHILE HOldData() DO              (* procedure for fetching old *)
    NewStart:=TRUE;
    GetFileConst(FileConst);       (* data stored on external    *)
    measindex:=GetMeasIndex();
    NrOffData:=FileConst;
    GetFileName(Name);
    IF (Name[1]<>' ') THEN
      Ok:=TRUE;
      Lookup(dataFile,Name,FALSE);
      IF dataFile.res=notdone THEN
        Error('no such file');
        Ok:=FALSE;
        SetOldReady;
        SetMeasWork(Rest);
        SendOldData(FALSE,1);
      ELSE
        GetFirstData(Ok);
      END;
      WHILE NOT EndMeasure() AND Ok DO
        ReadData(Ok,Samples);
        SendOldData(Ok,Samples);
        IF NOT Ok THEN
          StopTest;
        END;
      END;
      CloseRead(dataFile);
    ELSE
      SetOldReady;
      SetMeasWork(Rest);
      SendOldData(FALSE,1);
    END;
  END;
  SetOldReady;
END HandleOldData;


PROCEDURE HandleMeasure;

VAR       Samples : INTEGER;
          Save    : BOOLEAN;
```



```
BEGIN
  Save:=TRUE;
  GetStimIndex(StimIndex);
  NewStart:=TRUE;
  GetNrStimuli(Number);
  GetFileName(Name);           (* makes a new measure and     *)
  Delete(Name,dataFile);       (* store it on external memory *)
  IF (Name[0]=' ') AND  (Name[1]=' ') THEN
    Save:=FALSE;
  ELSE
    Lookup(dataFile,Name,TRUE);
  END;
  GetFileConst(FileConst);
  measindex:=GetMeasIndex();
  IF Save THEN
    StoreFirstData;
  END;
  WHILE HMeas() DO
    IF Save THEN
      WaitForData(Samples);
      WriteData(Samples);
    ELSE
      PausRest;
    END;
  END;
  IF Save THEN
    CloseWrite(dataFile);
  END;
END HandleMeasure;


(*PROCESS*)PROCEDURE FileHandler;

BEGIN
  SetPriority(100);
  Allowed:=TRUE;       (* to measured data          *)
  LOOP
    GetMeasWork(Work);
    CASE Work OF
      OldData : HandleOldData|
      Sekv    : PausFile|
      Cali    : PausFile|
      Meas    : HandleMeasure|
      Rest    : PausFile;
    END;
  END;
END FileHandler;

END FileProcess.
```

```
DEFINITION MODULE GenGalv;

EXPORT QUALIFIED GalvManager,InitGalv;

PROCEDURE InitGalv;
PROCEDURE GalvManager;

END GenGalv.
```



```
IMPLEMENTATION MODULE GenGalv;

FROM Kernel   IMPORT CreateProcess,SetPriority,WaitUntil,GetTime,
                     IncTime,Time;

FROM Decl     IMPORT PausTest,GetMotorData,MotorType,GetSample,
                     SetEnabled,PutGalvSpeed,EndMeasure,GetTest,
                     StopMeasure,Error,MeasureType,
                     GetMeasWork,GetTStart,TestType,StimType;

FROM Random   IMPORT RandomReal;

FROM MathLib IMPORT round,float,sin;

FROM FloatingUtilities IMPORT Trunc;

 CONST               MakePointTime = 0.1;
                     PI            = 3.1416;
                     Dangle        = 0.6580;         (* radian *)

 VAR                 t1,t2,t3,tdelta,
                     hTime,Speed,TStart : REAL;
                     t                  : Time;
                     Stim               : StimType;
                     MotorData          : MotorType;
                     h                  : INTEGER;
                     Test               : TestType;
                     TestEnd            : BOOLEAN;
                     Work               : MeasureType;
                     Sequence           : ARRAY[1..10] OF INTEGER;

(*********************************************************************)

(*****************  initiation  procedure       *********************)

 PROCEDURE InitGalv;

 BEGIN
    CreateProcess(GalvManager,3000);
    Stim:=Galvanic;
 END InitGalv;


 PROCEDURE InitPar(Number : INTEGER);

 VAR            i : INTEGER;

 BEGIN
    GetTStart(TStart);
    hTime:=0.0;
    TestEnd:=FALSE;
    FOR i := 2 TO 10 DO
      Sequence[i]:=0;
    END;
    Sequence[1]:=1;
    GetMotorData(MotorData,Number,Stim);
    tdelta:=MotorData.RandomTime*RandomReal();
    t1:=MotorData.RampTime+TStart;
    IF Number<3 THEN
      t2:=tdelta+MotorData.WaitPaus+t1;
      t3:=MotorData.BreakTime+t2;
    ELSE
      t2:=MotorData.WaitPaus+t1;
      t3:=MotorData.BreakTime+t2;
    END;
   GetTime(t);
 END InitPar;

(*********************************************************************)

(*****************       PRBS    procedure       *********************)

 PROCEDURE PRBSSequencer( VAR PRBSOut : INTEGER);
```

```
VAR          i,PRBS : INTEGER;

BEGIN
   FOR i := 0 TO 6 DO              (* makes sequence PRBS *)(*8*)
     Sequence[8-i]:=Sequence[7-i];
   END;                            (* special ndring f r anpassning *)
   PRBS:=1;                        (* till gammal vibrationstest *)
   IF PRBS<>Sequence[1] THEN       (* normalt s tt 3 10 i st llet *)
     PRBS:=1;                      (* 1 2 7 8 anpassat till 256 s 1 Hz*)
   ELSE
     PRBS:=0;
   END;
   IF PRBS<>Sequence[2] THEN
     PRBS:=1;
   ELSE
     PRBS:=0;
   END;
   IF PRBS<>Sequence[7] THEN
     PRBS:=1;
   ELSE
     PRBS:=0;
   END;
   IF PRBS<>Sequence[8] THEN
     Sequence[1]:=1;
   ELSE
     Sequence[1]:=0;
   END;
   PRBSOut:=Sequence[8];
END PRBSSequencer;


(*********************************************************************)

(*****************   testsequence procedure      *********************)


PROCEDURE PutOutSpeed(Speed,hTime : REAL;Test : INTEGER;
                      VAR TestEnd : BOOLEAN);

BEGIN
   IF Speed>10.0 THEN
     Speed:=10.0
   END;
   IF Speed<-10.0 THEN
     Speed:=-10.0
   END;
   IF ABS(Speed)<0.001 THEN
     Speed:=0.0;
   END;
   PutGalvSpeed(Speed,hTime,Test,TestEnd);
END PutOutSpeed;


PROCEDURE RandomTest;

VAR          RSpeed,OldSpeed,
             TestSpeed,hTime1 : REAL;
             RStart           : BOOLEAN;

BEGIN
   InitPar(1);
   RStart:=TRUE;
   OldSpeed:=MotorData.MaxSpeed;
   WHILE NOT EndMeasure() AND RStart DO
     h:=round(MakePointTime*1000.0);
     WaitUntil(t);
     IncTime(t,h);
     hTime:=hTime+MakePointTime;
     WITH MotorData DO
       IF hTime<=TStart THEN
         Speed:=0.0
       ELSIF hTime<=t1 THEN
         Speed:=OldSpeed*(hTime-TStart)/RampTime
       ELSE
         RStart:=FALSE
       END;
```



```
       PutOutSpeed(Speed,hTime,1,TestEnd);
     END;
   END;
   t1:=MotorData.RampTime;
   t2:=tdelta+MotorData.WaitPaus+t1;
   t3:=MotorData.BreakTime+t2;
   hTime1:=t2+t3;
   WHILE NOT EndMeasure() AND NOT TestEnd DO
     h:=round(MakePointTime*1000.0);
     WaitUntil(t);
     IncTime(t,h);
     hTime:=hTime+MakePointTime;
     hTime1:=hTime1+MakePointTime;
     WITH MotorData DO
       IF hTime1>t3 THEN
         OldSpeed:=Speed;
         tdelta:=RandomTime*RandomReal();
         RSpeed:=RandomReal();
         IF RSpeed<0.5 THEN
           RSpeed:=-RandomSpeed
         ELSE
           RSpeed:=RandomSpeed
         END;
         TestSpeed:=MaxSpeed+RSpeed;
         IF TestSpeed>10.0 THEN
           RSpeed:=-RSpeed;
         END;
         IF TestSpeed<-10.0 THEN
           RSpeed:=-RSpeed;
         END;
         tdelta:=MotorData.RandomTime*RandomReal();
         hTime1:=0.0;
         t2:=WaitPaus+tdelta+t1;
         t3:=BreakTime+t2;
       END;
       IF hTime1<=WaitPaus THEN
         Speed:=OldSpeed
       ELSIF hTime1<=(t1+WaitPaus) THEN
         Speed:=OldSpeed-RSpeed*(hTime1-WaitPaus)/RampTime
       ELSIF hTime1<=t2 THEN
         Speed:=OldSpeed-RSpeed
       ELSIF hTime1<=t3 THEN
         Speed:=OldSpeed-RSpeed*(t3-hTime1)/BreakTime
       END;
     END;
     PutOutSpeed(Speed,hTime,1,TestEnd);
   END;
   PutOutSpeed(0.0,hTime,1,TestEnd);
   StopMeasure(Stim);
   SetEnabled(FALSE);
END RandomTest;


PROCEDURE SinusTest;

VAR            Rest : BOOLEAN;
             hTime1 : REAL;

BEGIN
   InitPar(2);
   Rest:=TRUE;
   WHILE NOT EndMeasure() AND Rest DO
     h:=round(MakePointTime*1000.0);
     WaitUntil(t);
     IncTime(t,h);
     hTime:=hTime+MakePointTime;
     WITH MotorData DO
       IF hTime<=TStart THEN
         Speed:=0.0
       ELSIF hTime<=t1 THEN
         Speed:=MaxSpeed*(hTime-TStart)/RampTime;
       ELSIF hTime<=t2 THEN
         Speed:=MaxSpeed;
       ELSE
         Rest:=FALSE;
       END;
       PutOutSpeed(Speed,hTime,2,TestEnd);
```

```
      END;
    END;
    hTime1:=0.0;
    WHILE NOT EndMeasure() AND NOT TestEnd DO
      h:=round(MakePointTime*1000.0);
      WaitUntil(t);
      IncTime(t,h);
      WITH MotorData DO
        hTime:=hTime+MakePointTime;
        hTime1:=hTime1+MakePointTime;
        Speed:=MaxSpeed+RandomSpeed*sin(hTime1*2.0*PI/Period);
      END;
      PutOutSpeed(Speed,hTime,2,TestEnd);
    END;
    PutOutSpeed(0.0,hTime,2,TestEnd);
    StopMeasure(Stim);
    SetEnabled(FALSE);
END SinusTest;


PROCEDURE PRBSSinusTest;

VAR           Rest                              : BOOLEAN;
              PRBSRandom,RoundTime,PRBSTime     : INTEGER;
              hTime1                            : REAL;

BEGIN
    InitPar(3);
    Rest:=TRUE;
    PRBSTime:=round(10.0*MotorData.PRBSPeriod);
    WHILE NOT EndMeasure() AND Rest DO
      h:=round(MakePointTime*1000.0);
      WaitUntil(t);
      IncTime(t,h);
      hTime:=hTime+MakePointTime;
      WITH MotorData DO
        IF hTime<=TStart THEN
          Speed:=0.0
        ELSIF hTime<=t1 THEN
          Speed:=MaxSpeed*(hTime-TStart)/RampTime;
        ELSIF hTime<=t2 THEN
          Speed:=MaxSpeed;
        ELSE
          Rest:=FALSE;
        END;
        PutOutSpeed(Speed,hTime,3,TestEnd);
      END;
    END;
    hTime1:=0.0;
    WHILE NOT EndMeasure() AND NOT TestEnd DO
      h:=round(MakePointTime*1000.0);
      WaitUntil(t);
      IncTime(t,h);
      WITH MotorData DO
        hTime:=hTime+MakePointTime;
        hTime1:=hTime1+MakePointTime;
        RoundTime:=round(10.0*hTime);
        Speed:=MaxSpeed+RandomSpeed*sin(hTime1*2.0*PI/Period);
        IF (RoundTime MOD PRBSTime) = 0 THEN
          PRBSSequencer(PRBSRandom);
          IF PRBSRandom=1 THEN
            IncTime(t,2*h);
            hTime1:=hTime1+Dangle*Period/2.0*PI;
          END;
        END;
      END;
      PutOutSpeed(Speed,hTime,3,TestEnd);
    END;
    PutOutSpeed(0.0,hTime,3,TestEnd);
    StopMeasure(Stim);
    SetEnabled(FALSE);
END PRBSSinusTest;


PROCEDURE PRBSTest;

VAR           Rest,up                           : BOOLEAN;
```



```
              PRBSRandom,RoundTime,PRBSTime     : INTEGER;

BEGIN
    InitPar(4);
    Rest:=TRUE;
    PRBSTime:=round(10.0*MotorData.PRBSPeriod);
    WHILE NOT EndMeasure() AND Rest DO
      h:=round(MakePointTime*1000.0);
      WaitUntil(t);
      IncTime(t,h);
      hTime:=hTime+MakePointTime;
      WITH MotorData DO
        IF hTime<=TStart THEN
          Speed:=0.0;
        ELSIF hTime<=t2 THEN
          Speed:=0.0;
        ELSE
          Rest:=FALSE;
        END;
        PutOutSpeed(Speed,hTime,4,TestEnd);
      END;
    END;
    up:=TRUE;
    WHILE NOT EndMeasure() AND NOT TestEnd DO
      h:=round(MakePointTime*1000.0);
      WaitUntil(t);
      IncTime(t,h);
      WITH MotorData DO
        hTime:=hTime+MakePointTime;
        RoundTime:=round(10.0*hTime);
        IF (RoundTime MOD PRBSTime) = 0 THEN
          PRBSSequencer(PRBSRandom);
          IF PRBSRandom=1 THEN
            Speed:=RandomSpeed+MaxSpeed;
          ELSE
            Speed:=-RandomSpeed+MaxSpeed;
          END;
        END;
      END;
      PutOutSpeed(Speed,hTime,4,TestEnd);
    END;
    PutOutSpeed(0.0,hTime,4,TestEnd);
    StopMeasure(Stim);
    SetEnabled(FALSE);
END PRBSTest;


(*process*) PROCEDURE GalvManager;

BEGIN
    SetPriority(95);
    LOOP
      GetTest(Test,Stim);          (* selects and run test *)
      GetMeasWork(Work);                (* from the menu        *)
      IF Work<>OldData THEN
        CASE Test OF
          Paus       : PausTest(Stim)|
          Random     : RandomTest|
          Sinus      : SinusTest|
          PRBSSinus  : PRBSSinusTest|
          PRBS       : PRBSTest;
        END;
      END;
    END;
END GalvManager;

END GenGalv.
```

```
DEFINITION MODULE GenVib;

EXPORT QUALIFIED VibManager,InitVib;

PROCEDURE VibManager;
PROCEDURE InitVib;

END GenVib.
```



```
IMPLEMENTATION MODULE GenVib;
(* Changes to 386 system *)

FROM Kernel    IMPORT   CreateProcess,SetPriority,WaitUntil,GetTime,
                        IncTime,Time;

FROM Decl      IMPORT   PausTest,GetMotorData,MotorType,GetSample,
                        SetEnabled,PutVibSpeed,EndMeasure,GetTest,
                        StopMeasure,Error,MeasureType,
                        GetMeasWork,GetTStart,TestType,StimType;

FROM Random    IMPORT   RandomReal;

FROM MathLib   IMPORT   round,float,sin;

 CONST              MakePointTime = 0.1;
                    PI            = 3.1416;
                    Dangle        = 0.6580;         (* radian *)

 VAR                t1,t2,t3,tdelta,
                    hTime,Speed,TStart : REAL;
                    t                  : Time;
                    MotorData          : MotorType;
                    h                  : INTEGER;
                    Test               : TestType;
                    Stim               : StimType;
                    TestEnd            : BOOLEAN;
                    Work               : MeasureType;
                    Sequence           : ARRAY[1..10] OF INTEGER;

(*********************************************************************)

(*****************  initiation  procedure       **********************)

 PROCEDURE InitVib;

 BEGIN
    CreateProcess(VibManager,3000);
    Stim:=Vibration;
 END InitVib;


 PROCEDURE InitPar(Number : INTEGER);

 VAR            i : INTEGER;

 BEGIN
    GetTStart(TStart);
    hTime:=0.0;
    TestEnd:=FALSE;
    FOR i := 2 TO 10 DO
      Sequence[i]:=0;
    END;
    Sequence[1]:=1;
    GetMotorData(MotorData,Number,Stim);
    tdelta:=MotorData.RandomTime*RandomReal();
    t1:=MotorData.RampTime+TStart;
    IF Number<3 THEN
      t2:=tdelta+MotorData.WaitPaus+t1;
      t3:=MotorData.BreakTime+t2;
    ELSE
      t2:=MotorData.WaitPaus+t1;
      t3:=MotorData.BreakTime+t2;
    END;
    GetTime(t);
 END InitPar;


(*********************************************************************)

(*****************       PRBS    procedure       **********************)

 PROCEDURE PRBSSequencer( VAR PRBSOut : INTEGER);
```

```
 VAR          i,PRBS : INTEGER;

 BEGIN
    FOR i := 0 TO 6 DO              (* makes sequence PRBS *)(*8*)
      Sequence[8-i]:=Sequence[7-i];
    END;                            (* special ndring f r anpassning *)
    PRBS:=1;                        (* till gammal vibrationstest *)
    IF PRBS<>Sequence[1] THEN       (* normalt s tt 3 10 i st llet *)
      PRBS:=1;                      (* 1 2 7 8 anpassat till 256 s 1 Hz*)
    ELSE
      PRBS:=0;
    END;
    IF PRBS<>Sequence[2] THEN
      PRBS:=1;
    ELSE
      PRBS:=0;
    END;
    IF PRBS<>Sequence[7] THEN
      PRBS:=1;
    ELSE
      PRBS:=0;
    END;
    IF PRBS<>Sequence[8] THEN
      Sequence[1]:=1;
    ELSE
      Sequence[1]:=0;
    END;
    PRBSOut:=Sequence[8];
 END PRBSSequencer;


(*********************************************************************)

(*****************   testsequence procedure     ********************)


 PROCEDURE PutOutSpeed(Speed,hTime : REAL;Test : INTEGER;
                       VAR TestEnd : BOOLEAN);

 BEGIN
    IF Speed>15.0 THEN
      Speed:=15.0
    END;
    IF Speed<-15.0 THEN
      Speed:=-15.0
    END;
    IF ABS(Speed)<0.001 THEN
      Speed:=0.0;
    END;
    PutVibSpeed(Speed,hTime,Test,TestEnd);
 END PutOutSpeed;


 PROCEDURE RandomTest;

 VAR          RSpeed,OldSpeed,
              TestSpeed,hTime1 : REAL;
              RStart           : BOOLEAN;

 BEGIN
    InitPar(1);
    RStart:=TRUE;
    OldSpeed:=MotorData.MaxSpeed;
    WHILE NOT EndMeasure() AND RStart DO
      h:=round(MakePointTime*1000.0);
      WaitUntil(t);
      IncTime(t,h);
      hTime:=hTime+MakePointTime;
      WITH MotorData DO
        IF hTime<=TStart THEN
          Speed:=0.0
        ELSIF hTime<=t1 THEN
          Speed:=OldSpeed*(hTime-TStart)/RampTime
        ELSE
          RStart:=FALSE
        END;
        PutOutSpeed(Speed,hTime,1,TestEnd);
```



```
        END;
      END;
      t1:=MotorData.RampTime;
      t2:=tdelta+MotorData.WaitPaus+t1;
      t3:=MotorData.BreakTime+t2;
      hTime1:=t2+t3;
      WHILE NOT EndMeasure() AND NOT TestEnd DO
        h:=round(MakePointTime*1000.0);
        WaitUntil(t);
        IncTime(t,h);
        hTime:=hTime+MakePointTime;
        hTime1:=hTime1+MakePointTime;
        WITH MotorData DO
          IF hTime1>t3 THEN
            OldSpeed:=Speed;
            tdelta:=RandomTime*RandomReal();
            RSpeed:=RandomReal();
            IF RSpeed<0.5 THEN
              RSpeed:=-RandomSpeed
            ELSE
              RSpeed:=RandomSpeed
            END;
            TestSpeed:=MaxSpeed+RSpeed;
            IF TestSpeed>10.0 THEN
              RSpeed:=-RSpeed;
            END;
            IF TestSpeed<-10.0 THEN
              RSpeed:=-RSpeed;
            END;
            tdelta:=MotorData.RandomTime*RandomReal();
            hTime1:=0.0;
            t2:=WaitPaus+tdelta+t1;
            t3:=BreakTime+t2;
          END;
          IF hTime1<=WaitPaus THEN
            Speed:=OldSpeed
          ELSIF hTime1<=(t1+WaitPaus) THEN
            Speed:=OldSpeed-RSpeed*(hTime1-WaitPaus)/RampTime
          ELSIF hTime1<=t2 THEN
            Speed:=OldSpeed-RSpeed
          ELSIF hTime1<=t3 THEN
            Speed:=OldSpeed-RSpeed*(t3-hTime1)/BreakTime
          END;
        END;
        PutOutSpeed(Speed,hTime,1,TestEnd);
      END;
      PutOutSpeed(0.0,hTime,1,TestEnd);
      StopMeasure(Stim);
      SetEnabled(FALSE);
 END RandomTest;


 PROCEDURE FinskTest;

 VAR          RSpeed,OldSpeed,
              TestSpeed,hTime1 : REAL;
              RStart           : BOOLEAN;

        (* This test should be run with 33 Hz sampling *)
        (* frequency *)
 BEGIN
    InitPar(1);
    RStart:=TRUE;
    WHILE NOT EndMeasure() AND RStart DO
      h:=round(MakePointTime*1000.0);
      WaitUntil(t);
      IncTime(t,h);
      hTime:=hTime+MakePointTime;
      RStart:=FALSE
    END;
    hTime1:=0.0;
    WHILE NOT EndMeasure() AND NOT TestEnd DO
      h:=round(MakePointTime*1000.0);
      WaitUntil(t);
      IncTime(t,h);
      hTime:=hTime+MakePointTime;
      hTime1:=hTime1+MakePointTime;
```

```
      WITH MotorData DO
        IF (hTime1>40.0) AND (hTime1<53.4) THEN
          Speed:=4.81;
        ELSIF (hTime1>63.2) AND (hTime1<72.5) THEN
          Speed:=11.47;
        ELSIF (hTime1>92.1) AND (hTime1<106.1) THEN
          Speed:=2.06;
        ELSIF (hTime1>115.4) AND (hTime1<125.5) THEN
          Speed:=7.46;
        ELSIF (hTime1>135.3) AND (hTime1<150.0) THEN
          Speed:=12.54;
        ELSE
          Speed:=0.0;
        END;
      END;
      PutOutSpeed(Speed,hTime,1,TestEnd);
    END;
    PutOutSpeed(0.0,hTime,1,TestEnd);
    StopMeasure(Stim);
    SetEnabled(FALSE);
END FinskTest;


PROCEDURE SinusTest;

VAR           Rest,Puls : BOOLEAN;
            hTime1,Pulstime : REAL;

BEGIN
   InitPar(2);
   Rest:=TRUE;
   Puls:=FALSE;
   WHILE NOT EndMeasure() AND Rest DO
     h:=round(MakePointTime*1000.0);
     WaitUntil(t);
     IncTime(t,h);
     hTime:=hTime+MakePointTime;
     WITH MotorData DO
       IF hTime<=TStart THEN
         Speed:=0.0
       ELSIF hTime<=t1 THEN
         Speed:=MaxSpeed*(hTime-TStart)/RampTime;
       ELSIF hTime<=t2 THEN
         Speed:=MaxSpeed;
       ELSE
         Rest:=FALSE;
       END;
       PutOutSpeed(Speed,hTime,2,TestEnd);
     END;
   END;
   hTime1:=0.0;
   WHILE NOT EndMeasure() AND NOT TestEnd DO
     h:=round(MakePointTime*1000.0);
     WaitUntil(t);
     IncTime(t,h);
     WITH MotorData DO
       hTime:=hTime+MakePointTime;
       hTime1:=hTime1+MakePointTime;
       Speed:=MaxSpeed+RandomSpeed*sin(hTime1*2.0*PI/Period);
     END;
     PutOutSpeed(Speed,hTime,2,TestEnd);
   END;
   PutOutSpeed(0.0,hTime,2,TestEnd);
   StopMeasure(Stim);
   SetEnabled(FALSE);
END SinusTest;


PROCEDURE PRBSSinusTest;

VAR           Rest                              : BOOLEAN;
              PRBSRandom,RoundTime,PRBSTime     : INTEGER;
              hTime1                            : REAL;

BEGIN
   InitPar(3);
   Rest:=TRUE;
```



```
   PRBSTime:=round(10.0*MotorData.PRBSPeriod);
   WHILE NOT EndMeasure() AND Rest DO
     h:=round(MakePointTime*1000.0);
     WaitUntil(t);
     IncTime(t,h);
     hTime:=hTime+MakePointTime;
     WITH MotorData DO
       IF hTime<=TStart THEN
         Speed:=0.0
       ELSIF hTime<=t1 THEN
         Speed:=MaxSpeed*(hTime-TStart)/RampTime;
       ELSIF hTime<=t2 THEN
         Speed:=MaxSpeed;
       ELSE
         Rest:=FALSE;
       END;
       PutOutSpeed(Speed,hTime,3,TestEnd);
     END;
   END;
   hTime1:=0.0;
   WHILE NOT EndMeasure() AND NOT TestEnd DO
     h:=round(MakePointTime*1000.0);
     WaitUntil(t);
     IncTime(t,h);
     WITH MotorData DO
       hTime:=hTime+MakePointTime;
       hTime1:=hTime1+MakePointTime;
       RoundTime:=round(10.0*hTime);
       Speed:=MaxSpeed+RandomSpeed*sin(hTime1*2.0*PI/Period);
       IF (RoundTime MOD PRBSTime) = 0 THEN
         PRBSSequencer(PRBSRandom);
         IF PRBSRandom=1 THEN
           IncTime(t,2*h);
           hTime1:=hTime1+Dangle*Period/2.0*PI;
         END;
       END;
     END;
     PutOutSpeed(Speed,hTime,3,TestEnd);
   END;
   PutOutSpeed(0.0,hTime,3,TestEnd);
   StopMeasure(Stim);
   SetEnabled(FALSE);
END PRBSSinusTest;


PROCEDURE PRBSTest;

VAR           Rest                              : BOOLEAN;
              PRBSRandom,RoundTime,PRBSTime     : INTEGER;

BEGIN
   InitPar(4);
   Rest:=TRUE;
   PRBSTime:=round(10.0*MotorData.PRBSPeriod);
   WHILE NOT EndMeasure() AND Rest DO
     h:=round(MakePointTime*1000.0);
     WaitUntil(t);
     IncTime(t,h);
     hTime:=hTime+MakePointTime;
     WITH MotorData DO
       IF hTime<=TStart THEN
         Speed:=0.0;
       ELSIF hTime<=t2 THEN
         Speed:=0.0;
       ELSE
         Rest:=FALSE;
       END;
       PutOutSpeed(Speed,hTime,4,TestEnd);
     END;
   END;
   WHILE NOT EndMeasure() AND NOT TestEnd DO
     h:=round(MakePointTime*1000.0);
     WaitUntil(t);
     IncTime(t,h);
     WITH MotorData DO
       hTime:=hTime+MakePointTime;
       RoundTime:=round(10.0*hTime);
```

```
        IF (RoundTime MOD PRBSTime) = 0 THEN
          PRBSSequencer(PRBSRandom);
          IF PRBSRandom=1 THEN
            Speed:=RandomSpeed+MaxSpeed;
          ELSE
            Speed:=-RandomSpeed+MaxSpeed;
          END;
        END;
      END;
      PutOutSpeed(Speed,hTime,4,TestEnd);
    END;
    PutOutSpeed(0.0,hTime,4,TestEnd);
    StopMeasure(Stim);
    SetEnabled(FALSE);
 END PRBSTest;


(*process*) PROCEDURE VibManager;

 BEGIN
    SetPriority(95);
    LOOP
      GetTest(Test,Stim);          (* selects and run test *)
      GetMeasWork(Work);                (* from the menu        *)
      IF Work<>OldData THEN
        CASE Test OF
          Paus       : PausTest(Stim)|
          Random     : FinskTest|(*RandomTest*)
          Sinus      : SinusTest|
          PRBSSinus  : PRBSSinusTest|
          PRBS       : PRBSTest;
        END;
      END;
    END;
 END VibManager;

END GenVib.
```



```
DEFINITION MODULE HelpProcess;

EXPORT QUALIFIED HelpProc;

PROCEDURE HelpProc;

END HelpProcess.
```

```
IMPLEMENTATION MODULE HelpProcess;

IMPORT RTMouse;

FROM Kernel   IMPORT CreateProcess,SetPriority;

FROM Graphics IMPORT handle,rectangle,color,point,SetWindow,SetViewPort,
                     VirtualScreen,SetFillColor,FillRectangle,
                     DrawRectangle,WaitMouseRectangle,HideCursor,
                     SetMouseRectangle,SetLineColor,SetTextColor,
                     WriteString,ReadString,ShowCursor,buttonset,
                     GetMouseRectangle,WaitForMouse;

FROM Storage  IMPORT ALLOCATE,DEALLOCATE;

FROM MathLib  IMPORT float;

FROM NumMenu  IMPORT MakeNumMenu,SetNumMenuEntry,ShowNumMenu,InitNumMenu,
                     GetNumMenuState,HideNumMenu,SetNumMenuColors,
                     GetNumMenuStateWait,NumMenuType;

FROM LogMenu  IMPORT MakeLogMenu,SetLogMenuEntry,ShowLogMenu,InitLogMenu,
                     GetLogMenuState,HideLogMenu,SetLogMenuColors,
                     GetLogMenuStateWait,LogMenuType;

FROM Decl     IMPORT ChangeWhat,WhatType,Error,CheckPlotWhat,GetNr;

CONST HideColor = black;
      HelpColor = grey;       HelpTextColor = yellow;



TYPE  TextStr        = ARRAY[0..16] OF CHAR;
      RealVectorType = ARRAY[0..10] OF REAL;
      WindowType     = (Ord,Num,Log);

      CommandWindowType = POINTER TO RECORD
                             H : handle;
                             CASE Kind : WindowType OF
                                Ord  : WindowRectangle :rectangle|
                                Num  : NM : NumMenuType|
                                Log  : LM : LogMenuType|
                             END;
                          END;


PROCEDURE SetRectangle(VAR R : rectangle; Xlo,Ylo,Xhi,Yhi : REAL);

BEGIN
   R.xlo:=Xlo;
   R.ylo:=Ylo;
   R.xhi:=Xhi;
   R.yhi:=Yhi;
END SetRectangle;


PROCEDURE DefineMouseWindow(H : handle;y : INTEGER);

VAR     R : rectangle;

BEGIN
   SetRectangle(R, 0.0, float(y-1), 1.0, float(y));
   SetMouseRectangle(H,R,y);
END DefineMouseWindow;


PROCEDURE ShowText(H :handle; y :REAL; str :TextStr);

VAR     Textpoint : point;
        R         : rectangle;

BEGIN
   Textpoint.h:=0.05;
   Textpoint.v:=y+0.15;
```



```
   SetRectangle(R,0.0,y,1.0,y+1.0);
   HideCursor;
   DrawRectangle(H,R);
   WriteString(H,Textpoint,str);
   ShowCursor;
END ShowText;


PROCEDURE HideMenu(VAR Window : CommandWindowType);

BEGIN
   WITH Window^ DO
      SetFillColor(H, HideColor);
      HideCursor;
      FillRectangle(H,WindowRectangle);
      ShowCursor;
   END;
END HideMenu;


(*****************************************************************************)

(******************** Procedures for helpmenu ********************************)


PROCEDURE InitHelpButton(VAR HelpButton : CommandWindowType);
(* makes the helpbutton *)

VAR     HelpBox : rectangle;
        TextPos : point;

BEGIN
  NEW(HelpButton);
  WITH HelpButton^ DO
    VirtualScreen(H);
    Kind:=Ord;
    SetRectangle(HelpBox,0.05,0.17,0.45,0.25);
    SetViewPort(H,HelpBox);
    SetRectangle(WindowRectangle,0.0,0.0,1.0,1.0);
    SetWindow(H,WindowRectangle);
    SetTextColor(H,HelpTextColor);
    SetLineColor(H,HelpTextColor);
    SetFillColor(H,HelpColor);
    DefineMouseWindow(H,1);
    TextPos.h:=0.30;
    TextPos.v:=0.20;
    FillRectangle(H,WindowRectangle);
    DrawRectangle(H,WindowRectangle);
    WriteString(H,TextPos,'Help');
  END;
END InitHelpButton;


PROCEDURE InitHelpTextWindow(VAR HelpWindow : CommandWindowType);
(* initiates the helpwindow *)

VAR     HelpBox : rectangle;

BEGIN
  NEW(HelpWindow);
  WITH HelpWindow^ DO
    VirtualScreen(H);
    Kind:=Ord;
    SetRectangle(HelpBox,0.60,0.05,1.50,1.0);
    SetViewPort(H,HelpBox);
    SetRectangle(WindowRectangle,0.0,0.0,1.0,1.0);
    SetWindow(H,WindowRectangle);
    SetTextColor(H,HelpTextColor);
    SetLineColor(H,HelpTextColor);
    SetFillColor(H,HelpColor);
  END;
END InitHelpTextWindow;


PROCEDURE ShowHelpTextWindow(VAR HelpWindow : CommandWindowType);
(* shows the window where the helptexts are written *)
```

```
BEGIN
  WITH HelpWindow^ DO
    HideCursor;
    SetFillColor(H,HelpColor);
    FillRectangle(H,WindowRectangle);
    DrawRectangle(H,WindowRectangle);
    ShowCursor;
  END;
END ShowHelpTextWindow;


PROCEDURE HelpText1(VAR TextWindow : CommandWindowType);
(* headmenu help *)

VAR      Pos,var : point;
         butt    : buttonset;

BEGIN
   WITH TextWindow^ DO
     Pos.h:=0.01;
     Pos.v:=0.95;
     WriteString(H,Pos,'- System parameters -');
     Pos.v:=Pos.v - 0.04;
     WriteString(H,Pos,' Change global parameters.');
     Pos.v:=Pos.v - 0.06;
     WriteString(H,Pos,'- Inputsignals -');
     Pos.v:=Pos.v - 0.04;
     WriteString(H,Pos,' Choose Inputsignals to Postcon.');

     Pos.v:=Pos.v - 0.06;
     WriteString(H,Pos,'- Stimuli -');
     Pos.v:=Pos.v - 0.04;
     WriteString(H,Pos,' Choose Stimulies to run or change.');
     Pos.v:=Pos.v - 0.06;
     WriteString(H,Pos,'- Plot -');
     Pos.v:=Pos.v - 0.04;
     WriteString(H,Pos,' Choose what to plot.');
     Pos.v:=Pos.v - 0.06;
     WriteString(H,Pos,'- Start test -');
     Pos.v:=Pos.v - 0.04;
     WriteString(H,Pos,' Start up new test with selected');
     Pos.v:=Pos.v - 0.04;
     WriteString(H,Pos,' stimuli and configuration.');
     Pos.v:=Pos.v - 0.06;
     WriteString(H,Pos,'- Stop test -');
     Pos.v:=Pos.v - 0.04;
     WriteString(H,Pos,' Cancel test in progress.');
     Pos.v:=Pos.v - 0.06;
     WriteString(H,Pos,'- Quit program -');
     Pos.v:=Pos.v - 0.06;
     WriteString(H,Pos,' Leave POSTCON.');
     Pos.v:=0.04;
     WriteString(H,Pos,'Click mouse to continue');
     WaitForMouse(H,var,butt);
   END;
END HelpText1;


PROCEDURE HelpText2(VAR TextWindow : CommandWindowType);
(* testmenu help *)

VAR      Pos,var : point;
         butt    : buttonset;

BEGIN
  WITH TextWindow^ DO
    Pos.h:=0.01;
    Pos.v:=0.95;
    WriteString(H,Pos,'- Vibration -');
    Pos.v:=Pos.v - 0.04;
    WriteString(H,Pos,' Selecet shape off the vibration stimuli.');
    Pos.v:=Pos.v - 0.06;
    WriteString(H,Pos,'- Galvanic -');
    Pos.v:=Pos.v - 0.04;
    WriteString(H,Pos,' Selecet shape off the galvanic stimuli.');
    Pos.v:=Pos.v - 0.06;
    WriteString(H,Pos,'- Par vib -');
```



```
    Pos.v:=Pos.v - 0.04;
    WriteString(H,Pos,' Change parameters of vibration tests.');
    Pos.v:=Pos.v - 0.06;
    WriteString(H,Pos,'- Par galv -');
    Pos.v:=Pos.v - 0.04;
    WriteString(H,Pos,' Change parameters of galvanic tests.');
    Pos.v:=Pos.v - 0.06;
    WriteString(H,Pos,'- RunFile -');
    Pos.v:=Pos.v - 0.04;
    WriteString(H,Pos,' Load and run an old test.');
    Pos.v:=Pos.v - 0.06;
    WriteString(H,Pos,'- DONE -');
    Pos.v:=Pos.v - 0.04;
    WriteString(H,Pos,' Return to the headmenu.');
    Pos.v:=0.04;
    WriteString(H,Pos,'Click mouse to continue.');
    WaitForMouse(H,var,butt);
  END;
END HelpText2;


PROCEDURE HelpText3(VAR TextWindow : CommandWindowType);
(* testmenu help *)

VAR      Pos,var : point;
         butt    : buttonset;

BEGIN
  WITH TextWindow^ DO
    Pos.h:=0.01;
    Pos.v:=0.95;
    WriteString(H,Pos,'- Finsk -');
    Pos.v:=Pos.v - 0.04;
    WriteString(H,Pos,' Pulses with changing amplitud after finsk ');
    Pos.v:=Pos.v - 0.04;
    WriteString(H,Pos,' model.');
    Pos.v:=Pos.v - 0.06;
    WriteString(H,Pos,'- Sinus -');
    Pos.v:=Pos.v - 0.04;
    WriteString(H,Pos,' The speed varies as a sinus signal.');
    Pos.v:=Pos.v - 0.06;
    WriteString(H,Pos,'- PRBSSinus -');
    Pos.v:=Pos.v - 0.04;
    WriteString(H,Pos,' A sinus with psuedorandom phaseshifts.');
    Pos.v:=Pos.v - 0.06;
    WriteString(H,Pos,'- PRBS -');
    Pos.v:=Pos.v - 0.04;
    WriteString(H,Pos,' Runs forwards and backwards with pseudo-');
    Pos.v:=Pos.v - 0.04;
    WriteString(H,Pos,' random shifts of direction.');
    Pos.v:=Pos.v - 0.06;
    WriteString(H,Pos,'- DONE -');
    Pos.v:=Pos.v - 0.04;
    WriteString(H,Pos,' Return to the headmenu.');
    Pos.v:=0.04;
    WriteString(H,Pos,'Click mouse to continue');
    WaitForMouse(H,var,butt);
  END;
END HelpText3;


PROCEDURE HelpText4(VAR TextWindow : CommandWindowType);
(* testmenu help *)

VAR      Pos,var : point;
         butt    : buttonset;

BEGIN
  WITH TextWindow^ DO
    Pos.h:=0.01;
    Pos.v:=0.95;
    WriteString(H,Pos,'- RampRandom -');
    Pos.v:=Pos.v - 0.04;
    WriteString(H,Pos,' Ramps up and dowm with random intervalls.');
    Pos.v:=Pos.v - 0.06;
    WriteString(H,Pos,'- Sinus -');
    Pos.v:=Pos.v - 0.04;
```

```
    WriteString(H,Pos,' The speed varies as a sinus signal.');
    Pos.v:=Pos.v - 0.06;
    WriteString(H,Pos,'- PRBSSinus -');
    Pos.v:=Pos.v - 0.04;
    WriteString(H,Pos,' A sinus with psuedorandom phaseshifts.');
    Pos.v:=Pos.v - 0.06;
    WriteString(H,Pos,'- PRBS -');
    Pos.v:=Pos.v - 0.04;
    WriteString(H,Pos,' Runs forwards and backwards with pseudo-');
    Pos.v:=Pos.v - 0.04;
    WriteString(H,Pos,' random shifts of direction.');
    Pos.v:=Pos.v - 0.06;
    WriteString(H,Pos,'- DONE -');
    Pos.v:=Pos.v - 0.04;
    WriteString(H,Pos,' Return to the headmenu.');
    Pos.v:=0.04;
    WriteString(H,Pos,'Click mouse to continue');
    WaitForMouse(H,var,butt);
  END;
END HelpText4;


PROCEDURE HelpText6(VAR TextWindow:CommandWindowType);
(* plotmenu help *)

VAR       Pos,var : point;
          butt    : buttonset;

BEGIN
  WITH TextWindow^ DO
    Pos.h:=0.01;
    Pos.v:=0.95;
    WriteString(H,Pos,'- Sti&NPo&XYM -');
    Pos.v:=Pos.v - 0.04;
    WriteString(H,Pos,' Plots stimuli, normal power and momentum');
    Pos.v:=Pos.v - 0.04;
    WriteString(H,Pos,' measured by the platform.');
    Pos.v:=Pos.v - 0.06;
    WriteString(H,Pos,'- Sti&NPo -');
    Pos.v:=Pos.v - 0.04;
    WriteString(H,Pos,' Plots stimuli and normal power.');
    Pos.v:=Pos.v - 0.06;
    WriteString(H,Pos,'- Sti&XYM -');
    Pos.v:=Pos.v - 0.04;
    WriteString(H,Pos,' Plots stimuli and momentum.');
    Pos.v:=Pos.v - 0.06;
    WriteString(H,Pos,'- Sti&NPo&EPo -');
    Pos.v:=Pos.v - 0.04;
    WriteString(H,Pos,' Plots stimuli, normal power and elbow power.');
    Pos.v:=Pos.v - 0.06;
    WriteString(H,Pos,'- Calibration -');
    Pos.v:=Pos.v - 0.04;
    WriteString(H,Pos,' Plot the AD inputs');
    Pos.v:=Pos.v - 0.06;
    WriteString(H,Pos,'- TestSeq -');
    Pos.v:=Pos.v - 0.04;
    WriteString(H,Pos,' Plot the test sequences, without the motor');
    Pos.v:=Pos.v - 0.04;
    WriteString(H,Pos,' running.');
    Pos.v:=Pos.v-0.06;
    WriteString(H,Pos,'- DONE -');
    Pos.v:=0.04;
    WriteString(H,Pos,'Click mouse to continue');
    WaitForMouse(H,var,butt);
  END;
END HelpText6;


PROCEDURE HelpText7(VAR TextWindow:CommandWindowType);
(* questionwindow help *)

VAR     Pos,var : point;
        butt    : buttonset;

BEGIN
  WITH TextWindow^ DO
    Pos.h:=0.01;
```



```
    Pos.v:=0.95;
    WriteString(H,Pos,' Write the name of the file you want to');
    Pos.v:=Pos.v - 0.04;
    WriteString(H,Pos,' save ( or load ). If no data is to be saved');
    Pos.v:=Pos.v - 0.04;
    WriteString(H,Pos,' ( or loaded ) just press return.');
    Pos.v:=0.04;
    WriteString(H,Pos,'Click mouse to continue');
    WaitForMouse(H,var,butt);
  END;
END HelpText7;


PROCEDURE HelpText8(VAR TextWindow:CommandWindowType);
(* testpar1 help *)

VAR     Pos,var : point;
        butt    : buttonset;

BEGIN
  WITH TextWindow^ DO
    Pos.h:=0.01;
    Pos.v:=0.95;
    WriteString(H,Pos,' RampTime, the acceleration time.');
    Pos.v:=Pos.v - 0.06;
    WriteString(H,Pos,' BreakTime, the retardation time.');
    Pos.v:=Pos.v - 0.06;
    WriteString(H,Pos,' MinTime, the minimum time of constant speed.');
    Pos.v:=Pos.v - 0.06;
    WriteString(H,Pos,' Center speed, the speed around which the ');
    Pos.v:=Pos.v - 0.04;
    WriteString(H,Pos,' changes in speed are centered.');
    Pos.v:=Pos.v - 0.06;
    WriteString(H,Pos,' RandomTime, the maximum time with added');
    Pos.v:=Pos.v - 0.04;
    WriteString(H,Pos,' random speed.');
    Pos.v:=Pos.v - 0.06;
    WriteString(H,Pos,' RandomSpeed, the value of the changes');
    Pos.v:=Pos.v - 0.04;
    WriteString(H,Pos,' in speed.');
    Pos.v:=Pos.v - 0.06;
    WriteString(H,Pos,' Max testtime, the total time of the test.');
    Pos.v:=Pos.v - 0.08;
    WriteString(H,Pos,' Center speed + RandomSpeed must not be');
    Pos.v:=Pos.v - 0.04;
    WriteString(H,Pos,' larger than 10.');
    Pos.v:=0.04;
    WriteString(H,Pos,'Click mouse to continue');
    WaitForMouse(H,var,butt);
  END;
END HelpText8;


PROCEDURE HelpText9(VAR TextWindow:CommandWindowType);
(* testpar2 help*)

VAR     Pos,var : point;
        butt    : buttonset;

BEGIN
  WITH TextWindow^ DO
    Pos.h:=0.01;
    Pos.v:=0.95;
    WriteString(H,Pos,' RampTime, the time to reach centerspeed.');
    Pos.v:=Pos.v - 0.06;
    WriteString(H,Pos,' WaitPause, the time with centerspeed before');
    Pos.v:=Pos.v - 0.04;
    WriteString(H,Pos,' the sinewave starts.');
    Pos.v:=Pos.v - 0.06;
    WriteString(H,Pos,' Center speed, the speed around which the ');
    Pos.v:=Pos.v - 0.04;
    WriteString(H,Pos,' changes are centered.');
    Pos.v:=Pos.v - 0.06;
    WriteString(H,Pos,' Period Time, the period of the sinewave.');
    Pos.v:=Pos.v - 0.06;
    WriteString(H,Pos,' Amplitude, the amplitude of the changes.');
    Pos.v:=Pos.v - 0.06;
```

```
    WriteString(H,Pos,' Max testtime, the total time of the test');
    Pos.v:=Pos.v - 0.08;
    WriteString(H,Pos,' Center speed + RandomSpeed must not be');
    Pos.v:=Pos.v - 0.04;
    WriteString(H,Pos,' larger than 10.');
    Pos.v:=0.04;
    WriteString(H,Pos,'Click mouse to continue');
    WaitForMouse(H,var,butt);
  END;
END HelpText9;


PROCEDURE HelpText10(VAR TextWindow:CommandWindowType);
(* testpar3 help *)

VAR       Pos,var : point;
          butt    : buttonset;

BEGIN
  WITH TextWindow^ DO
    Pos.h:=0.01;
    Pos.v:=0.95;
    WriteString(H,Pos,' RampTime, the time to reach centerspeed.');
    Pos.v:=Pos.v - 0.06;
    WriteString(H,Pos,' WaitPause, the time with centerspeed before');
    Pos.v:=Pos.v - 0.04;
    WriteString(H,Pos,' the sinewave starts.');
    Pos.v:=Pos.v - 0.06;
    WriteString(H,Pos,' Center speed, the speed around which the ');
    Pos.v:=Pos.v - 0.04;
    WriteString(H,Pos,' changes are centered.');
    Pos.v:=Pos.v - 0.06;
    WriteString(H,Pos,' Amplitude, the amplitude of the changes.');
    Pos.v:=Pos.v - 0.06;
    WriteString(H,Pos,' Period Time, the period of the sinewave.');
    Pos.v:=Pos.v - 0.06;
    WriteString(H,Pos,' PRBSPeriod, the period of the PRBS sequence.');
    Pos.v:=Pos.v - 0.06;
    WriteString(H,Pos,' Max testtime, the total time of the test');
    Pos.v:=Pos.v - 0.08;
    WriteString(H,Pos,' Center speed + RandomSpeed must not be');
    Pos.v:=Pos.v - 0.04;
    WriteString(H,Pos,' larger than 10.');
    Pos.v:=0.04;
    WriteString(H,Pos,'Click mouse to continue');
    WaitForMouse(H,var,butt);
  END;
END HelpText10;


PROCEDURE HelpText11(VAR TextWindow:CommandWindowType);
(* testpar4 help *)

VAR       Pos,var : point;
          butt    : buttonset;

BEGIN
  WITH TextWindow^ DO
    Pos.h:=0.01;
    Pos.v:=0.95;
    WriteString(H,Pos,' RampTime, the time to reach centerspeed.');
    Pos.v:=Pos.v - 0.06;
    WriteString(H,Pos,' WaitPause, the time with centerspeed before');
    Pos.v:=Pos.v - 0.04;
    WriteString(H,Pos,' the sequence starts.');
    Pos.v:=Pos.v - 0.06;
    WriteString(H,Pos,' Center speed, the speed around which the ');
    Pos.v:=Pos.v - 0.04;
    WriteString(H,Pos,' changes are centered.');
    Pos.v:=Pos.v - 0.06;
    WriteString(H,Pos,' Amplitude, the amplitude of the changes.');
    Pos.v:=Pos.v - 0.06;
    WriteString(H,Pos,' PRBSPeriod, the period of the PRBS sequence.');
    Pos.v:=Pos.v - 0.06;
    WriteString(H,Pos,' Max testtime, the total time of the test');
    Pos.v:=Pos.v - 0.08;
    WriteString(H,Pos,' Center speed + RandomSpeed must not be');
```



```
    Pos.v:=Pos.v - 0.04;
    WriteString(H,Pos,' larger than 10');
    Pos.v:=0.04;
    WriteString(H,Pos,'Click mouse to continue');
    WaitForMouse(H,var,butt);
  END;
END HelpText11;


PROCEDURE HelpText12(VAR TextWindow:CommandWindowType);
(* Systemparameters help *)

VAR       Pos,var : point;
          butt    : buttonset;

BEGIN
  WITH TextWindow^ DO
    Pos.h:=0.01;
    Pos.v:=0.95;
    WriteString(H,Pos,' SampleTime, how often the program collects');
    Pos.v:=Pos.v - 0.04;
    WriteString(H,Pos,' data.');
    Pos.v:=Pos.v - 0.06;
    WriteString(H,Pos,' Plot const, how often the program sends');
    Pos.v:=Pos.v - 0.04;
    WriteString(H,Pos,' data to plot.');
    Pos.v:=Pos.v - 0.06;
    WriteString(H,Pos,' File const, how many sets of data the ');
    Pos.v:=Pos.v - 0.04;
    WriteString(H,Pos,' program collects before writing to disk.');
    Pos.v:=Pos.v - 0.06;
    WriteString(H,Pos,' Foot-front, the distance between the foot');
    Pos.v:=Pos.v - 0.04;
    WriteString(H,Pos,' and the front edge of the platform.');
    Pos.v:=Pos.v - 0.06;
    WriteString(H,Pos,' Foot-back, the distance between the foot');
    Pos.v:=Pos.v - 0.04;
    WriteString(H,Pos,' and the rear edge of the platform.');
    Pos.v:=Pos.v - 0.06;
    WriteString(H,Pos,' Load koeff, the relationship between the');
    Pos.v:=Pos.v - 0.04;
    WriteString(H,Pos,' load on the platform and the measured voltage');
    Pos.v:=Pos.v - 0.06;
    WriteString(H,Pos,' StabTime, The stabilizing time at the');
    Pos.v:=Pos.v - 0.04;
    WriteString(H,Pos,' beginning of each test.');
    Pos.v:=0.04;
    WriteString(H,Pos,'Click mouse to continue');
    WaitForMouse(H,var,butt);
  END;
END HelpText12;

PROCEDURE HelpText13(VAR TextWindow:CommandWindowType);
(* inputsignalmenu help *)

VAR       Pos,var : point;
          butt    : buttonset;

BEGIN
  WITH TextWindow^ DO
    Pos.h:=0.01;
    Pos.v:=0.95;
    WriteString(H,Pos,'- Elbow&Plate -');
    Pos.v:=Pos.v - 0.04;
    WriteString(H,Pos,' Postcon read elbowforces and platform.');
    Pos.v:=Pos.v - 0.06;
    WriteString(H,Pos,'- Plate -');
    Pos.v:=Pos.v - 0.04;
    WriteString(H,Pos,' Postcon read platform.');
    Pos.v:=Pos.v-0.06;
    WriteString(H,Pos,'- DONE -');
    Pos.v:=0.04;
    WriteString(H,Pos,'Click mouse to continue');
    WaitForMouse(H,var,butt);
  END;
END HelpText13;
```

```
(*****************************************************************************)

(******************** Procedure for HelpProc *****************************)


(*Process*)PROCEDURE HelpProc;

VAR         HelpButt,HelpWindow : CommandWindowType;
            n                   : CARDINAL;
            Ret                 : ARRAY[0..1]OF CHAR;
            Plotting            : WhatType;
            Change              : BOOLEAN;
            Helpnr              : INTEGER;

BEGIN
  SetPriority(89);
  InitHelpButton(HelpButt);
  InitHelpTextWindow(HelpWindow);
  WITH HelpWindow^ DO
    LOOP
      n:=WaitMouseRectangle(HelpButt^.H);
      GetNr(Helpnr);
      IF (Helpnr>0) THEN
        CheckPlotWhat(Plotting,Change);
        ChangeWhat(Oper);
        CASE Helpnr OF
          1:  ShowHelpTextWindow(HelpWindow);
              HelpText1(HelpWindow);
              HideMenu(HelpWindow)|
          2:  ShowHelpTextWindow(HelpWindow);
              HelpText2(HelpWindow);
              HideMenu(HelpWindow)|
          3:  ShowHelpTextWindow(HelpWindow);
              HelpText3(HelpWindow);
              HideMenu(HelpWindow)|
          4:  ShowHelpTextWindow(HelpWindow);
              HelpText4(HelpWindow);
              HideMenu(HelpWindow)|
          6:  ShowHelpTextWindow(HelpWindow);
              HelpText6(HelpWindow);
              HideMenu(HelpWindow)|
          7:  ShowHelpTextWindow(HelpWindow);
              HelpText7(HelpWindow);
              HideMenu(HelpWindow)|
          8:  ShowHelpTextWindow(HelpWindow);
              HelpText8(HelpWindow);
              HideMenu(HelpWindow)|
          9:  ShowHelpTextWindow(HelpWindow);
              HelpText9(HelpWindow);
              HideMenu(HelpWindow)|
          10: ShowHelpTextWindow(HelpWindow);
              HelpText10(HelpWindow);
              HideMenu(HelpWindow)|
          11: ShowHelpTextWindow(HelpWindow);
              HelpText11(HelpWindow);
              HideMenu(HelpWindow)|
          12: ShowHelpTextWindow(HelpWindow);
              HelpText12(HelpWindow);
              HideMenu(HelpWindow)|
          13: ShowHelpTextWindow(HelpWindow);
              HelpText13(HelpWindow);
              HideMenu(HelpWindow)|
        END;
        IF Helpnr <> 5 THEN
          ChangeWhat(Plotting);
        END;
      END;
    END;
  END;
END HelpProc;


END HelpProcess.
```



```
DEFINITION MODULE Measure;

EXPORT QUALIFIED DataMeasure;


PROCEDURE DataMeasure;
```

```
IMPLEMENTATION MODULE Measure;
(* Changes to 386 system *)
 FROM AnalogIO    IMPORT DAOut,ADIn;

 FROM Kernel      IMPORT WaitUntil,GetTime,IncTime,SetPriority,Time;

 FROM Decl        IMPORT MailType,MBox,PBox,GetCounter,
                         PlotData,PlotSekv,PlotADIn,Error,DBox,DRBox,
                         SpeedType,MeasureType,FileText,MeasData,
                         EndMeasure,GetEmerg,GetMeasWork,
                         GetMotorPar,HOldData,
                         HMeas,HSekv,GetMomentData,GetSample,
                         StopMeasure,StopTest,StimType,GetFileName,
                         PausMeas,GetNrStimuli,StoreData,
                         GetStimIndex,GetOldData,
                         WaitFull,CauseEmpty,WaitOldReady,GetMeasIndex,
                         PutMeasIndex,HCali;

 FROM Messages    IMPORT AcceptMessage,ReceiveMessage,SendMessage;

 FROM Strings     IMPORT Assign;

 FROM MathLib     IMPORT round,float;


 VAR                PlotCount,StimIndex,
                    Number                : INTEGER;
                    t                     : Time;
                    Allowed,Emerg         : BOOLEAN;
                    Work                  : MeasureType;
                    a,b,gamma,alfa        : REAL;
                    Speed                 : SpeedType;
                    y                     : MeasData;
                    Empty                 : BOOLEAN;
                    PMail,DMail           : MailType;
                    x                     : SpeedType;

(*********************************************************************)

(*****************      in/out procedure        *********************)


 PROCEDURE PlMeas(VAR y : MeasData);

 BEGIN
    y[1]:= 10.0*ADIn(15);  (* fz1 *)   (* m tdata fr n plattan *)
    y[2]:= 10.0*ADIn(16);  (* fz2 *)
    y[3]:= 10.0*ADIn(17);  (* fz3 *)
    y[4]:= 10.0*ADIn(12);  (* fx *)
    y[5]:= 10.0*ADIn(13);  (* fy1 *)
    y[6]:= 10.0*ADIn(14);  (* fy2 *)
 END PlMeas;


 PROCEDURE ElbowMeas(VAR x : SpeedType);

 BEGIN
    x[1]:=10.0*ADIn(28);  (* left elbowforce *) (* m tdata fr n stolen *)
    x[2]:=10.0*ADIn(29);  (* right elbowforce *)
 END ElbowMeas;


 PROCEDURE MotorCntrol;

 VAR           Enabled : BOOLEAN;

 BEGIN
    GetMotorPar(Enabled,Speed);
    IF Enabled
    THEN
      CASE StimIndex OF
        3 : DAOut(8,Speed[1]*0.1);DAOut(9,Speed[2]*0.1)|
        2 : IF Speed[1]>10.0 THEN
              DAOut(8,1.0);
              DAOut(9,((Speed[1]-10.0)*0.1));
            ELSE
              DAOut(8,(Speed[1]*0.1));
```



```
              DAOut(9,0.0);
            END|
        1 : DAOut(8,Speed[2]*0.1)|
        0 :;
      END;
    END;
 END MotorControl;


 PROCEDURE EmergencyLoop;

 VAR              h1,Speed1       : REAL;
                  h,K             : INTEGER;
                  Enabled,TestEnd : BOOLEAN;

 BEGIN
    DAOut(8,0.0);
    DAOut(9,0.0);
    GetMotorPar(Enabled,Speed); (* controls output after *)
    Speed[1]:=0.0;
    Speed[2]:=0.0;
    FOR K:=1 TO 5 DO            (* emergency signal      *)
      WaitUntil(t);
      IncTime(t,100);
      PlotDataRoutine;
    END;
    StopMeasure(Vibration);
    StopMeasure(Galvanic);
    GetSample(h1);
    h:=round(h1*1000.0);
    GetTime(t);
    WHILE Emerg DO      (* emergency rest *)
      GetEmerg(Emerg);
      WaitUntil(t);
      IncTime(t,h);
      GetSample(h1);
      h:=round(h1*1000.0);
    END;
 END EmergencyLoop;


(*********************************************************************)

(*****************   plotmessage  procedure     *********************)


 PROCEDURE MPForce(VAR PMail : MailType; y: MeasData);

 BEGIN
    PMail^.MForce[1]:=gamma*y[1];
    PMail^.MForce[2]:=gamma*y[2];
    PMail^.MForce[3]:=gamma*y[3];
    PMail^.MForce[4]:=gamma*(b*(y[1]+y[2])-a*y[3]);
    PMail^.MForce[5]:=gamma*y[4];
    PMail^.MForce[6]:=gamma*(y[6]+y[5]);
 END MPForce;


PROCEDURE MPForceCali(VAR PMail : MailType; y: MeasData);

 BEGIN
    PMail^.MForce[1]:=gamma*y[1];
    PMail^.MForce[2]:=gamma*y[2];
    PMail^.MForce[3]:=gamma*y[3];
    PMail^.MForce[4]:=gamma*y[4];
    PMail^.MForce[5]:=gamma*y[5];
    PMail^.MForce[6]:=gamma*y[6];
 END MPForceCali;


 PROCEDURE EForce(VAR PMail : MailType; x: SpeedType);

 BEGIN
    PMail^.ElbowForce[1]:=alfa*x[1];
    PMail^.ElbowForce[2]:=alfa*x[2];
 END EForce;
```

```
PROCEDURE PlotDataRoutine;

VAR           c      : INTEGER;

BEGIN
   GetCounter(c);
   IF c<1 THEN
      c:=1;
   END;
   PlotCount:=(PlotCount+1) MOD c;
   IF PlotData() AND (PlotCount=0)
   THEN
     AcceptMessage(MBox,PMail);
     IF PMail<> NIL THEN
       PlMeas(y);
       MPForce(PMail,y);
       IF (GetMeasIndex()=4) THEN
         ElbowMeas(x);
         EForce(PMail,x);
       END;
       PMail^.Stimulus:=Speed;
       SendMessage(PBox,PMail);
       Allowed:= TRUE;
     ELSIF Allowed THEN
       Error('Plot data ovfl (Meas)');
       Allowed:= FALSE;
     END;
   END;
END PlotDataRoutine;


PROCEDURE PlotOldDataRoutine;

VAR           c      : INTEGER;

BEGIN
   GetCounter(c);
   IF c<1 THEN
      c:=1;
   END;
   PlotCount:=(PlotCount+1) MOD c;
   IF PlotData() AND (PlotCount=0) THEN
     AcceptMessage(MBox,PMail);
     IF PMail<> NIL THEN
       MPForce(PMail,y);
       IF (GetMeasIndex()=4) THEN
         EForce(PMail,x);
       END;
       PMail^.Stimulus:=Speed;
       SendMessage(PBox,PMail);
       Allowed:= TRUE;
     ELSIF Allowed THEN
       Error('Plot data ovfl (Meas)');
       Allowed:= FALSE;
     END;
   END;
END PlotOldDataRoutine;


PROCEDURE PlotSekvRoutine;

VAR           c      : INTEGER;

BEGIN
   GetCounter(c);
   IF c<1 THEN
      c:=1;
   END;
   PlotCount:=(PlotCount+1) MOD c;
   IF PlotSekv() AND (PlotCount=0) THEN
     AcceptMessage(MBox,PMail);
     IF PMail<> NIL THEN
       PMail^.Stimulus:=Speed;
```



```
       SendMessage(PBox,PMail);
       Allowed:= TRUE;
     ELSIF Allowed THEN
       Error('Plot data ovfl (Meas)');
       Allowed:= FALSE;
     END;
   END;
END PlotSekvRoutine;


PROCEDURE PlotCaliRoutine;

VAR   c    : INTEGER;

BEGIN
  GetCounter(c);
  IF c<1 THEN c:=1
  END;
  PlotCount:=(PlotCount+1) MOD c;
  IF PlotADIn() AND (PlotCount=0) THEN
      AcceptMessage(MBox,PMail);
      IF PMail<>NIL THEN
          PlMeas(y);
          MPForceCali(PMail,y);
          IF GetMeasIndex()=4 THEN
              ElbowMeas(x);
              EForce(PMail,x);
          END;
              SendMessage(PBox,PMail);
          Allowed:= TRUE;
      ELSIF Allowed THEN
          Error('Plot data ovfl (Meas)');
          Allowed:= FALSE;
      END;
  END;
END PlotCaliRoutine;

(*****************************************************************)

(*****************    datahandle  procedure     ********************)


PROCEDURE HandleOldData;

VAR                Ok         : BOOLEAN;
                   h1         : REAL;
                   h          : CARDINAL;
                   K,Samples  : INTEGER;

BEGIN
     Empty:=FALSE;
     WHILE NOT Empty DO
       AcceptMessage(DRBox,DMail);
       IF DMail=NIL THEN
         Empty:=TRUE;
       ELSE
         SendMessage(DBox,DMail);
       END;
     END;
     CauseEmpty;
   WHILE HOldData() DO          (* procedure for fetching old *)
     WaitOldReady;
     GetMomentData(a,b,gamma,alfa);      (* memory *)
     Ok:=TRUE;
     GetTime(t);
     WHILE NOT EndMeasure() AND Ok DO
       GetOldData(Ok,Samples);
       FOR K:=1 TO Samples DO
         ReceiveMessage(DRBox,DMail);
         IF DMail<> NIL THEN
           WITH DMail^ DO
             y:=MForce;
             IF (GetMeasIndex()=4) THEN
               x:=ElbowForce;
             END;
             Speed:=Stimulus;
           END;
```

```
          SendMessage(DBox,DMail);
          Allowed:= TRUE;
        ELSIF Allowed THEN
          Error('Plot data ovfl (Meas)');
          Allowed:= FALSE;
        END;
        IF Ok THEN
          GetSample(h1);
          h:=round(h1*1000.0);
          WaitUntil(t);
          IncTime(t,h);
          PlotOldDataRoutine;
        END;
      END;
    END;
    CauseEmpty;
  END;
END HandleOldData;


PROCEDURE HandleSekv;

VAR               h1 : REAL;
                  h  : CARDINAL;
            Enabled  : BOOLEAN;

BEGIN
  GetTime(t);
  WHILE HSekv() DO
    GetSample(h1);
    h:= round(h1*1000.0);
    WaitUntil(t);                (* sends control signal for *)
    IncTime(t,h);                (* "visual speed" to plot   *)
    GetMotorPar(Enabled,Speed);
    PlotSekvRoutine;
  END;
END HandleSekv;


PROCEDURE HandleCali;

VAR  h1  : REAL;
  h  : CARDINAL;

BEGIN
  GetMomentData(a,b,gamma,alfa);
  GetTime(t);
  WHILE HCali() DO
      GetSample(h1);
      h:=round(h1*1000.0);
      WaitUntil(t);      (* sends AD-input signals for *)
      IncTime(t,h);
      PlotCaliRoutine;
  END;
END HandleCali;


PROCEDURE InitNewMeasure(VAR Save : BOOLEAN);

VAR         Name : FileText;

BEGIN
  GetFileName(Name);             (* makes a new measure and     *)
  IF (Name[0]=' ') AND  (Name[1]=' ') THEN
    Save:=FALSE;
  ELSE
    Save:=TRUE;
    WaitFull;
  END;
  GetNrStimuli(Number);
  GetMomentData(a,b,gamma,alfa);
  GetStimIndex(StimIndex);
END InitNewMeasure;


PROCEDURE NewMeasure;
```



```
VAR               h     : INTEGER;
        h1,speed2       : REAL;
        Save,Enabled    : BOOLEAN;

BEGIN
  InitNewMeasure(Save);
  GetTime(t);
  GetSample(h1);
  h:=round(h1*1000.0);
  WHILE HMeas() DO
    GetEmerg(Emerg);
    IF Emerg THEN
      EmergencyLoop;
    END;
    WaitUntil(t);
    IncTime(t,h);
    MotorControl;
    IF Save THEN
      AcceptMessage(DRBox,DMail);
      IF DMail<> NIL THEN
        GetMotorPar(Enabled,Speed);
        WITH DMail^ DO
          PlMeas(MForce);
          IF (GetMeasIndex()=4) THEN
            ElbowMeas(ElbowForce);
          END;
          Stimulus:=Speed;
        END;
        SendMessage(DBox,DMail);
        Allowed:= TRUE;
        StoreData(FALSE);
      ELSIF Allowed THEN
        Error('Store ovfl (Meas)');
        Allowed:= FALSE;
      END;
    END;
    PlotDataRoutine;
  END;
  IF Save THEN
    StoreData(TRUE);
  END;
END NewMeasure;


PROCEDURE HandlePaus;

BEGIN
  DAOut(8,0.0);
  DAOut(9,0.0);
  PausMeas;
END HandlePaus;


(*PROCESS*)PROCEDURE DataMeasure;

BEGIN
  SetPriority(100);
  DAOut(8,0.0);
  DAOut(9,0.0);
  PlotCount:=0;          (* selects wanted operation *)
  Allowed:=TRUE;         (* to measured data         *)
  LOOP
    GetMeasWork(Work);
    CASE Work OF
      OldData : HandleOldData|
      Sekv    : HandleSekv|
      Cali    : HandleCali|
      Meas    : NewMeasure|
      Rest    : HandlePaus;
    END;
  END;
END DataMeasure;

END Measure.
```

```
DEFINITION MODULE PlotProcess;

EXPORT QUALIFIED Plot;

 PROCEDURE Plot;

END PlotProcess.
```



```
IMPLEMENTATION MODULE PlotProcess;

FROM Kernel      IMPORT SetPriority,Wait,Signal;

FROM Storage     IMPORT ALLOCATE,DEALLOCATE;

FROM Graphics    IMPORT rectangle,VirtualScreen,SetViewPort,SetWindow,
                        SetLineColor,SetFillColor,PolyLine,point,color,
                        SetTextColor,handle,CharacterSize,FillRectangle,
                        DrawRectangle,WriteString,HideCursor,ShowCursor;

FROM Messages    IMPORT ReceiveMessage,AcceptMessage,SendMessage;

FROM Decl        IMPORT WhatType,MailType,CheckPlotWhat,PBox,MBox,
                        Error,PlotWindowType,GetPlotSort,
                        GetNrStimuli,PlotType,MeasData,GetStimIndex,
                        WaitOldReady,MeasureType,GetMeasWork,
                        GetStimSort,GetMotorTime,GetCounter,GetSample,
                        StimType,TestType,GetMeasIndex;

FROM MathLib     IMPORT exp,float,entier,ln;

FROM ConvReal    IMPORT RealToString;

CONST          MaxWindow = 15;
               UpScale   = 10;
               DownScale = 10;
               Delta     = 0.01;
               Size      = 95;

TYPE           PointerType = ARRAY[1..15] OF point;
               TextStr     = ARRAY[0..15] OF CHAR;
               WinType     = ARRAY[1..4] OF REAL;

VAR            PLWindow           : ARRAY[1..MaxWindow] OF PlotWindowType;
               upA,downA,Ax       : ARRAY[1..3] OF INTEGER;
               Start,Lines,Win,
               Nr,NumWin,PlotIndex,
               Number,StimIndex,
               PLines,Index,AxAD  : INTEGER;
               PlotMess           : MailType;
               Work               : MeasureType;
               NewPoint,OldPoint  : PointerType;
               WhatKind           : WhatType;
               x,step,RunTime     : REAL;
               Change,NewStart    : BOOLEAN;
               ColorStr           : ARRAY[0..5] OF color;
               TextWin            : PlotWindowType;

(*************************************************************************)

(*****************   grafical  procedure        ***********************)

 PROCEDURE InitColor;

 BEGIN
    ColorStr[0]:=lightblue;
    ColorStr[1]:=lightred;
    ColorStr[2]:=green;
    ColorStr[3]:=brown;
    ColorStr[4]:=black;
    ColorStr[5]:=grey;
 END InitColor;


 PROCEDURE SetPoint(VAR p : point; x,y : REAL);

 BEGIN                        (* place plotpoint *)
    p.h:=x;
    p.v:=y;
 END SetPoint;


 PROCEDURE PlotLine(Window : PlotWindowType; p1,p2 : point);
```

```
VAR         Line : ARRAY[0..1] OF point;

BEGIN
   Line[0]:=p1;                (* makes line *)
   Line[1]:=p2;
   PolyLine(Window^.H,Line,2);
END PlotLine;


PROCEDURE SetRectangle(VAR R : rectangle ;Size : WinType);

BEGIN
   R.xlo:=Size[1];                    (* defines plotarea *)
   R.ylo:=Size[2];
   R.xhi:=Size[3];
   R.yhi:=Size[4];
END SetRectangle;


PROCEDURE WriteReal(H:handle;X,Y,Val : REAL);

VAR            P : point;
              Str : ARRAY[1..8] OF CHAR;

BEGIN
   SetPoint(P,X,Y);
   IF ABS(Val)<0.0001 THEN
     Val:=0.0;
   END;
   RealToString(Val,Str,6);
   WriteString(H,P,Str);
END WriteReal;


PROCEDURE ShowText(H :handle; x,y,dx :REAL; str :TextStr;paint : color);

VAR Textpoint      : point;
    R              : rectangle;
    Wind           : WinType;

BEGIN
   Textpoint.h:= x+0.2*dx;  (* makes text in plotwindow *)
   Textpoint.v:= y+ 0.1;
   IF Win=3 THEN
     Textpoint.v:= y+ 0.1;
   ELSE
     Textpoint.v:= y+ 0.3;
   END;
   SetFillColor(H,paint);
   Wind[1]:=x;
   Wind[2]:=y;
   Wind[3]:=x+dx;
   Wind[4]:=y+1.0;
   SetRectangle(R,Wind);
   DrawRectangle(H,R);
   FillRectangle(H,R);
   WriteString(H,Textpoint,str);
END ShowText;


PROCEDURE InitPlotWindow(VAR Window : PlotWindowType;
                         MeasSize,WinSize :WinType; paint:color);

VAR       ViewPortRectangle : rectangle;

BEGIN
   NEW(Window);
   Window^.XMin:=MeasSize[1];    (* initiate windowsize on the sceen   *)
   Window^.YMin:=MeasSize[2];    (* and the size on the signals to be  *)
   Window^.XMax:=MeasSize[3];    (* plotted                            *)
   Window^.YMax:=MeasSize[4];
   WITH  Window^ DO
    VirtualScreen(H);
    SetRectangle(ViewPortRectangle,WinSize);
    SetViewPort(H,ViewPortRectangle);
    SetRectangle(WindowRectangle,MeasSize);
    SetWindow(H,WindowRectangle);
```



```
    SetLineColor(H,paint);
    IF paint=black THEN
      SetTextColor(H,black);
    ELSE
      SetTextColor(H,intensewhite);
    END;
    SetFillColor(H,intensewhite);
   END;
END InitPlotWindow;


PROCEDURE InitPlotWindow1(VAR Window : PlotWindowType;
                          MeasSize,WinSize :WinType; paint:color);

VAR       ViewPortRectangle : rectangle;

BEGIN
   NEW(Window);
   Window^.XMin:=MeasSize[1];    (* initiate windowsize on the sceen   *)
   Window^.YMin:=MeasSize[2];    (* and the size on the signals to be  *)
   Window^.XMax:=MeasSize[3];    (* plotted                            *)
   Window^.YMax:=MeasSize[4];
   WITH  Window^ DO
    VirtualScreen(H);
    SetRectangle(ViewPortRectangle,WinSize);
    SetViewPort(H,ViewPortRectangle);
    SetRectangle(WindowRectangle,MeasSize);
    SetWindow(H,WindowRectangle);
    SetLineColor(H,paint);
    SetFillColor(H,blue);
   END;
END InitPlotWindow1;


PROCEDURE ClearWindow;

VAR                      K : INTEGER;
         MeasSize,WinSize : WinType;

BEGIN
   MeasSize[1]:=-1.0;MeasSize[2]:=-1.0;
   MeasSize[3]:=1.0;MeasSize[4]:=1.0;
   WinSize[1]:=0.6;WinSize[2]:=0.0;
   WinSize[3]:=1.5;WinSize[4]:=1.0;
   InitPlotWindow(PLWindow[1],MeasSize,WinSize,lightblue);
   SetFillColor(PLWindow[1]^.H,black);
   FillRectangle(PLWindow[1]^.H,PLWindow[1]^.WindowRectangle);
   FOR K:= 1 TO NumWin DO
     SetFillColor(PLWindow[K]^.H,black);
     FillRectangle(PLWindow[K]^.H,PLWindow[K]^.WindowRectangle);
     DISPOSE(PLWindow[K]);          (* removes used windows *)
   END;
   IF NumWin>1 THEN
     DISPOSE(TextWin);
   END;
END ClearWindow;


(*************************************************************************)

(*****************   calculation  procedure     ********************)


PROCEDURE Lg(X : REAL):REAL;

BEGIN
   RETURN (ln(X)/10.0);          (* 10 logaritm *)
END Lg;


PROCEDURE ExpTen(X : REAL): REAL;

BEGIN
   RETURN (exp(X*ln(10.0)));
END ExpTen;
```

```
PROCEDURE ScalePoint(X : REAL;N : INTEGER): REAL;

VAR           Divisor : REAL;

BEGIN
   IF X<0.0 THEN
     Divisor:=ExpTen(float(entier(Lg(-X))+N-1));
     RETURN -(Divisor*float(entier((-X)/Divisor)));
   ELSIF X>0.0 THEN
     Divisor:=ExpTen(float(entier(Lg(X))+N-1));
     RETURN Divisor*float(entier((X)/Divisor));
   ELSE
     RETURN 0.0;
   END;
END ScalePoint;


PROCEDURE ComputeScalePoints(X,Y : REAL; VAR NewX,NewY : REAL);

VAR      N : CARDINAL;

BEGIN
   N:=1;
   REPEAT
     NewX:=ScalePoint(X,N);       (* rescale axelmarks *)
     NewY:=ScalePoint(Y,N);
     N:=N+1;
   UNTIL (NewX<>NewY) OR(N>4);
END ComputeScalePoints;


PROCEDURE StartUp;

VAR   Plotting : PlotType;
      MessUp   : INTEGER;
BEGIN
   ClearWindow;
   IF Work=OldData THEN
     WaitOldReady;
   END;
   GetPlotSort(Plotting);
   MessUp:=0;
   WITH Plotting DO
     IF NPower THEN MessUp:=MessUp+1;
     END;
     IF XYMoment THEN MessUp:=MessUp+2;
     END;
     IF ElbowPower THEN MessUp:=MessUp+4;
     END;
       CASE MessUp OF
         3 : DataWindow(1)|
         1 : DataWindow(2)|
         2 : DataWindow(3)|
         5 : DataWindow(4);
       END;
  END;
  NewStart:=TRUE;
END StartUp;


(*************************************************************************)

(*****************    windowtext  procedure     **********************)


PROCEDURE DataText(TextSort: INTEGER);

VAR               String : ARRAY [0..9] OF TextStr;
                  i      : INTEGER;
                  dx,dy  : REAL;
      MeasSize1,WinSize1,
      MeasSize2,WinSize2 : WinType;


BEGIN
  MeasSize1[1]:=0.0;MeasSize1[2]:=0.0;
  MeasSize1[3]:=9.0;MeasSize1[4]:=21.0;
```



```
  WinSize1[1]:=0.6;WinSize1[2]:=0.0;
  WinSize1[3]:=1.5;WinSize1[4]:=0.84;
  MeasSize2[1]:=0.0;MeasSize2[2]:=0.0;
  MeasSize2[3]:=9.0;MeasSize2[4]:=6.0;
  WinSize2[1]:=0.6;WinSize2[2]:=0.0;
  WinSize2[3]:=1.5;WinSize2[4]:=0.6;
  String[0]:= "Fn1Force";
  String[1]:= "Fn2Force";
  String[2]:= "Fn3Force";
  String[3]:= "XMoment";
  String[4]:= "FxForce";
  String[5]:= "FyForce";
  String[6]:= "Vibration";
  String[7]:= "Galvanic";
  String[8]:= "LeftMoment";
  String[9]:= "RightMoment";
  GetNrStimuli(Number);
  IF (TextSort=2) OR (TextSort=3) THEN
    InitPlotWindow(TextWin,MeasSize2,WinSize2,lightblue);
  ELSE
    InitPlotWindow(TextWin,MeasSize1,WinSize1,lightblue);
  END;
  WITH TextWin^ DO
    IF TextSort=1 THEN
      dx:=3.0;
      FOR i:= 0 TO 2 DO
         ShowText(H,3.0*float(i),10.0,dx,String[i],ColorStr[i]);
         ShowText(H,3.0*float(i),0.0,dx,String[i+3],ColorStr[i]);
      END;
      dy:=20.0;
    ELSIF TextSort=2 THEN
      dx:=3.0;
      FOR i:= 0 TO 2 DO
         ShowText(H,3.0*float(i),0.0,dx,String[i],ColorStr[i]);
      END;
      dy:=5.0;
    ELSIF TextSort=3 THEN
      dx:=3.0;
      FOR i:= 0 TO 2 DO
         ShowText(H,3.0*float(i),0.0,dx,String[i+3],ColorStr[i]);
      END;
      dy:=5.0;
    ELSIF TextSort=4 THEN
      dx:=3.0;
      FOR i:=0 TO 2 DO
         ShowText(H,3.0*float(i),10.0,dx,String[i],ColorStr[i]);
      END;
         ShowText(H,3.0*float(0),0.0,dx,String[3],ColorStr[0]);
         ShowText(H,3.0*float(1),0.0,dx,String[8],ColorStr[1]);
         ShowText(H,3.0*float(2),0.0,dx,String[9],ColorStr[2]);
      dy:=20.0;
    END;
    dx:=9.0/float(Number);
    GetStimIndex(StimIndex);
    CASE StimIndex OF
      3: ShowText(H,0.0,dy,dx,String[6],ColorStr[1]);
         ShowText(H,4.5,dy,dx,String[7],ColorStr[2]);PLines:=3|
      2: ShowText(H,0.0,dy,dx,String[6],ColorStr[1]);PLines:=2|
      1: ShowText(H,0.0,dy,dx,String[7],ColorStr[2]);PLines:=2|
      0: PLines:=1;
    END;
  END;
END DataText;


PROCEDURE ADInText;

VAR   String          : ARRAY [0..8] OF TextStr;
  i           : INTEGER;
  MeasSize,WinSize    : WinType;

BEGIN
  WinSize[1]:=0.6;WinSize[2]:=0.0;
  WinSize[3]:=1.5;WinSize[4]:=0.72;
  MeasSize[1]:=0.0;MeasSize[2]:=0.0;
  MeasSize[3]:=9.0;MeasSize[4]:=24.0;
  String[0]:="ADIn15";
```

```
   String[1]:="ADIn16";
   String[2]:="ADIn17";
   String[3]:="ADIn12";
   String[4]:="ADIn13";
   String[5]:="ADIn14";
   String[6]:="ADIn28";
   String[7]:="ADIn29";
   String[8]:=" ";
   InitPlotWindow(TextWin,MeasSize,WinSize,lightblue);
   WITH TextWin^ DO
       FOR i:=0 TO 2 DO
           ShowText(H,3.0*float(i),23.0,3.0,String[8],ColorStr[i]);
           ShowText(H,3.0*float(i),22.0,3.0,String[i],ColorStr[i]);
           ShowText(H,3.0*float(i),12.0,3.0,String[8],ColorStr[i]);
           ShowText(H,3.0*float(i),11.0,3.0,String[i+3],ColorStr[i]);
       END;
       IF GetMeasIndex()=4 THEN
           ShowText(H,0.0,1.0,4.5,String[8],ColorStr[0]);
           ShowText(H,0.0,0.0,4.5,String[6],ColorStr[0]);
           ShowText(H,4.5,1.0,4.5,String[8],ColorStr[1]);
           ShowText(H,4.5,0.0,4.5,String[7],ColorStr[1]);
       END;
  END;
END ADInText;


PROCEDURE TestText;

VAR             String : ARRAY [0..1] OF TextStr;
                i      : INTEGER;
                dx     : REAL;
      MeasSize,WinSize : WinType;


BEGIN
  MeasSize[1]:=0.0;MeasSize[2]:=0.0;
  MeasSize[3]:=12.0;MeasSize[4]:=1.0;
  WinSize[1]:=0.6;WinSize[2]:=0.5;
  WinSize[3]:=1.5;WinSize[4]:=0.6;
  InitPlotWindow(TextWin,MeasSize,WinSize,lightblue);
  WITH TextWin^ DO
    String[0]:= "Vibration";
    String[1]:= "Galvanic";
    GetNrStimuli(Number);
    dx:=12.0/float(Number);
    GetStimIndex(StimIndex);
    CASE StimIndex OF
      3: ShowText(H,0.0,0.0,dx,String[0],ColorStr[1]);
         ShowText(H,6.0,0.0,dx,String[1],ColorStr[2])|
      2: ShowText(H,0.0,0.0,dx,String[0],ColorStr[1])|
      1: ShowText(H,0.0,0.0,dx,String[1],ColorStr[2])|
      0: ;
    END;
  END;
END TestText;           (* control signal text *)


(*************************************************************************)

(*****************   plotwindow  procedure      *********************)


PROCEDURE DataWindow(WindowSort: INTEGER);

BEGIN
   CASE WindowSort OF
     1: SetWind(1);DataText(1);PlotIndex:=1|
     2: SetWind(2);DataText(2);PlotIndex:=2|
     3: SetWind(3);DataText(3);PlotIndex:=3|
     4: SetWind(4);DataText(4);PlotIndex:=4;
   END;
END DataWindow;


PROCEDURE SetWind(WinNr : INTEGER);

VAR                     K : INTEGER;
```



```
      MeasSize1,MeasSize2,
      MeasSize3,
      WinSize1,WinSize2,
      WinSize3,WinSize4,
      WinSize5,MeasSize,
      WinSize             : WinType;


BEGIN
   MeasSize1[1]:=0.0;MeasSize1[2]:=-15.0;
   MeasSize1[3]:=1.0;MeasSize1[4]:=15.0;
   WinSize1[1]:=0.6;WinSize1[2]:=0.84;
   WinSize1[3]:=1.5;WinSize1[4]:=1.0;
   MeasSize2[1]:=0.0;MeasSize2[2]:=-1048.0;
   MeasSize2[3]:=1.0;MeasSize2[4]:=1048.0;
   WinSize2[1]:=0.6;WinSize2[2]:=0.6;
   WinSize2[3]:=1.5;WinSize2[4]:=1.0;
   MeasSize3[1]:=0.0;MeasSize3[2]:=-64.0;
   MeasSize3[3]:=1.0;MeasSize3[4]:=64.0;
   WinSize3[1]:=0.6;WinSize3[2]:=0.44;
   WinSize3[3]:=1.5;WinSize3[4]:=0.8;
   WinSize4[1]:=0.6;WinSize4[2]:=0.04;
   WinSize4[3]:=1.5;WinSize4[4]:=0.4;
   WinSize5[1]:=0.6;WinSize5[2]:=0.1;
   WinSize5[3]:=1.5;WinSize5[4]:=0.5;
   GetNrStimuli(Number);
   IF (WinNr=2) OR (WinNr=3) THEN
      MeasSize:=MeasSize1;
      WinSize:=WinSize2;
   ELSE
      MeasSize:=MeasSize1;
      WinSize:=WinSize1;
   END;
   GetStimIndex(StimIndex);
   CASE StimIndex OF
     3: InitPlotWindow(PLWindow[1],MeasSize,WinSize,ColorStr[1]);
        InitPlotWindow(PLWindow[2],MeasSize,WinSize,ColorStr[2])|
     2: InitPlotWindow(PLWindow[1],MeasSize,WinSize,ColorStr[1])|
     1: InitPlotWindow(PLWindow[1],MeasSize,WinSize,ColorStr[2])|
     0: ;
   END;
   IF (WinNr=1) OR (WinNr=4) THEN
     FOR K:=1 TO 3 DO
      InitPlotWindow(PLWindow[Number+K],MeasSize2,WinSize3,ColorStr[K-1]);
      InitPlotWindow(PLWindow[Number+K+3],MeasSize3,WinSize4,ColorStr[K-1]);
     END;
     InitPlotWindow(PLWindow[Number+7],MeasSize1,WinSize1,black);
     InitPlotWindow(PLWindow[Number+8],MeasSize2,WinSize3,black);
     InitPlotWindow(PLWindow[Number+9],MeasSize3,WinSize4,black);
     NumWin:=Number+9;
     Lines:=Number+6;
     Start:=Number+7;
     Win:=3;
     Ax[1]:=Number+8;
     Ax[2]:=Number+9;
   ELSIF WinNr=2 THEN
     FOR K:=1 TO 3 DO
       InitPlotWindow(PLWindow[Number+K],MeasSize2,WinSize5,ColorStr[K-1]);
     END;
     InitPlotWindow(PLWindow[Number+4],MeasSize1,WinSize2,black);
     InitPlotWindow(PLWindow[Number+5],MeasSize2,WinSize5,black);
     NumWin:=Number+5;
     Lines:=Number+3;
     Start:=Number+4;
     Win:=2;
     Ax[1]:=Number+5;
   ELSIF WinNr=3 THEN
     FOR K:=1 TO 3 DO
       InitPlotWindow(PLWindow[Number+K],MeasSize3,WinSize5,ColorStr[K-1]);
     END;
     InitPlotWindow(PLWindow[Number+4],MeasSize1,WinSize2,black);
     InitPlotWindow(PLWindow[Number+5],MeasSize3,WinSize5,black);
     NumWin:=Number+5;
     Lines:=Number+3;
     Start:=Number+4;
     Win:=2;
     Ax[2]:=Number+5;
```

```
    END;
    Nr:=0;
    x:=0.05;          (* windowsize on data window *)
    upA[1]:=0;
    downA[1]:=0;
    upA[2]:=0;
    downA[2]:=0;
    IF Work=Meas THEN
      ScaleWindow;
    END;
END SetWind;


PROCEDURE ADInWindow;

VAR   K                    :   INTEGER;
  MeasSize,WinSize1,WinSize2,WinSize3 :   WinType;

BEGIN
  ClearWindow;
  MeasSize[1]:=0.0;   MeasSize[3]:=1.0;
  MeasSize[2]:=-2000.0;   MeasSize[4]:=2000.0;
  WinSize1[1]:=0.6;   WinSize1[3]:=1.5;
  WinSize1[2]:=0.04;  WinSize1[4]:=0.33;
  WinSize2[1]:=0.6;   WinSize2[3]:=1.5;
  WinSize2[2]:=0.37;  WinSize2[4]:=0.66;
  WinSize3[1]:=0.6;   WinSize3[3]:=1.5;
  WinSize3[2]:=0.70;  WinSize3[4]:=1.0;
  FOR K:=1 TO 3 DO
      InitPlotWindow(PLWindow[K],MeasSize,WinSize3,ColorStr[K-1]);
      InitPlotWindow(PLWindow[K+3],MeasSize,WinSize2,ColorStr[K-1]);
  END;
  IF GetMeasIndex()=4 THEN
      InitPlotWindow(PLWindow[7],MeasSize,WinSize1,ColorStr[0]);
      InitPlotWindow(PLWindow[8],MeasSize,WinSize1,ColorStr[1]);
      InitPlotWindow(PLWindow[9],MeasSize,WinSize3,ColorStr[4]);
      InitPlotWindow(PLWindow[10],MeasSize,WinSize2,ColorStr[4]);
      InitPlotWindow(PLWindow[11],MeasSize,WinSize1,ColorStr[4]);
      NumWin:=11;
      Lines:=8;
  ELSE
      InitPlotWindow(PLWindow[7],MeasSize,WinSize3,ColorStr[4]);
      InitPlotWindow(PLWindow[8],MeasSize,WinSize2,ColorStr[4]);
      NumWin:=8;
      Lines:=6;
  END;
  Win:=4;
  Nr:=0;
  x:=0.05;
  upA[1]:=0;
  downA[1]:=0;
  upA[2]:=0;
  downA[2]:=0;
  upA[3]:=0;
  downA[3]:=0;
  ADInText;
END ADInWindow;



PROCEDURE TestWindow;

VAR                    K : INTEGER;
        MeasSize,WinSize : WinType;

BEGIN
   MeasSize[1]:=0.0;MeasSize[2]:=-15.0;
   MeasSize[3]:=1.0;MeasSize[4]:=15.0;
   WinSize[1]:=0.6;WinSize[2]:=0.6;
   WinSize[3]:=1.5;WinSize[4]:=1.0;
   GetStimIndex(StimIndex);
   CASE StimIndex OF
     3: InitPlotWindow(PLWindow[1],MeasSize,WinSize,ColorStr[1]);
        InitPlotWindow(PLWindow[2],MeasSize,WinSize,ColorStr[2])|
     2: InitPlotWindow(PLWindow[1],MeasSize,WinSize,ColorStr[1])|
```



```
     1: InitPlotWindow(PLWindow[1],MeasSize,WinSize,ColorStr[2])|
     0: ;
   END;
   GetNrStimuli(Number);
   InitPlotWindow(PLWindow[Number+1],MeasSize,WinSize,black);
   x:=0.05;
   Nr:=0;
   Win:=1;
   TestText;
   NumWin:=Number+1;
   Lines:=Number;
   Start:=Number+1;
   ScaleWindow;
END TestWindow;


PROCEDURE ScaleWindow;

VAR                        K,t : INTEGER;
              MeasSize,WinSize : WinType;
              VibTest,GalvTest : TestType;
            TMax,Time1,Time2,h : REAL;

BEGIN
   GetStimSort(VibTest,GalvTest);
   CASE VibTest OF
     Random    : GetMotorTime(Time1,RunTime,1,Vibration)|
     Sinus     : GetMotorTime(Time1,RunTime,2,Vibration)|
     PRBSSinus : GetMotorTime(Time1,RunTime,3,Vibration)|
     PRBS      : GetMotorTime(Time1,RunTime,4,Vibration)|
     Paus      : Time1:=0.0;
   END;
   CASE GalvTest OF
     Random    : GetMotorTime(Time2,RunTime,1,Galvanic)|
     Sinus     : GetMotorTime(Time2,RunTime,2,Galvanic)|
     PRBSSinus : GetMotorTime(Time2,RunTime,3,Galvanic)|
     PRBS      : GetMotorTime(Time2,RunTime,4,Galvanic)|
     Paus      : Time2:=0.0;
   END;
   IF Time1>=Time2 THEN
     TMax:=Time1;
   ELSE
     TMax:=Time2;
   END;
   GetCounter(t);
   GetSample(h);
   step:=float(t)*h;
   MeasSize[1]:=0.0;MeasSize[2]:=0.0;
   MeasSize[3]:=TMax;MeasSize[4]:=1.0;
   WinSize[1]:=0.05;WinSize[2]:=0.27;
   WinSize[3]:=0.45;WinSize[4]:=0.35;
   InitPlotWindow(PLWindow[NumWin+1],MeasSize,WinSize,blue);
   NumWin:=NumWin+1;
   Index:=0;
   MeasSize[1]:=0.0;MeasSize[2]:=0.0;
   MeasSize[3]:=TMax;MeasSize[4]:=1.0;
   WinSize[1]:=0.05;WinSize[2]:=0.27;
   WinSize[3]:=(0.4*RunTime/TMax)+0.05;WinSize[4]:=0.35;
   InitPlotWindow1(PLWindow[NumWin+1],MeasSize,WinSize,blue);
   NumWin:=NumWin+1;
END ScaleWindow;

(*********************************************************************)

(*****************   plotaxes  procedure        *********************)


PROCEDURE Axes(Window : PlotWindowType;textmax : REAL);

VAR        Line  : ARRAY [0..1] OF point;
           Lo,Hi : REAL;
           CW,CH : REAL;                (* Character sizes    *)

BEGIN
   HideCursor;
   WITH Window^ DO
```

```
    CharacterSize(H, CH, CW);
    FillRectangle(H, WindowRectangle);      (* Clear window       *)
    DrawRectangle(H, WindowRectangle);      (* Draw rims          *)
    SetPoint(Line[0], 0.05, 0.0);
    SetPoint(Line[1], 1.0, 0.0);
    PolyLine(H, Line, 2);                   (* Draw X-axis        *)
    SetPoint(Line[0], 0.05, YMin);
    SetPoint(Line[1], 0.05, YMax);
    PolyLine(H, Line, 2);                   (* Draw Y-axis        *)
    ComputeScalePoints(YMin,YMax,Lo,Hi);    (* Draw Y-axis  Scale     *)
    SetPoint(Line[0],Lo,0.0);
    SetPoint(Line[1],Hi,0.0);
    WriteReal(H,0.1,0.93*YMin,Lo);          (* Lo *)
    WriteReal(H,0.1,textmax*YMax,Hi);       (* Hi *)
  END;
  ShowCursor;
END Axes;


(*********************************************************************)

(*****************   rescale   procedure        *********************)


PROCEDURE Ashow(A1,A2,A3,A4 : REAL;M,Lo,Hi,Ax : INTEGER);
(* Automatic scaling of plot axes *)

VAR         Change              : BOOLEAN;
            K,L                 : INTEGER;
            Ymax,Xmax,Xmin,max  : REAL;
            A                   : ARRAY[1..4]OF REAL;
            Win                 : WinType;

BEGIN
  A[1]:=A1;             (* automatical rescaling of the windows  *)
  A[2]:=A2;             (* signallimits after the size on A1..4  *)
  A[3]:=A3;
  A[4]:=A4;
  Ymax:=PLWindow[Lo]^.YMax;
  Change:=FALSE;
  max:=0.0;
  FOR K:=1 TO 4 DO
    IF (ABS(A[K])>max) THEN
      max:=ABS(A[K]);
    END;
  END;
  IF max>=0.8*Ymax THEN
     upA[M]:=upA[M]+1;
  END;
  IF max<=0.2*Ymax THEN
    downA[M]:=downA[M]+ 1;
  ELSIF downA[M]>0 THEN
    downA[M]:=downA[M]-1;
  END;
  IF upA[M]>=UpScale THEN
    Change:=TRUE;
    upA[M]:=0;
    WHILE Ymax<=max*1.5 DO
      Ymax:=Ymax*2.0;
    END;
  END;
  IF downA[M]>=DownScale THEN
    Change:=TRUE;
    downA[M]:=0;
    WHILE Ymax>=3.0*max DO
      Ymax:=Ymax*0.5;
    END;
  END;
  IF Change THEN
    FOR K:=Lo TO Hi DO
      WITH PLWindow[K]^ DO
        YMax:=Ymax;
        YMin:=-Ymax;
        Win[1]:=XMin;
        Win[2]:=YMin;
        Win[3]:=XMax;
        Win[4]:=YMax;
```



```
        SetRectangle(WindowRectangle,Win);
        SetWindow(H,WindowRectangle);
      END;
    END;
    PLWindow[Ax]^.YMax:=Ymax;
    PLWindow[Ax]^.YMin:=-Ymax;
    WITH PLWindow[Ax]^ DO
      SetRectangle(WindowRectangle,Win);
      SetWindow(H,WindowRectangle);
    END;
    x:=0.05;
    Nr:=0;
  END;
END Ashow;


(*********************************************************************)

(*****************     plotdata  procedure      *********************)


PROCEDURE MPForce(VAR PlotMess : MailType;PLines :INTEGER);

BEGIN
  WITH PlotMess^ DO
    SetPoint(NewPoint[PLines],x,MForce[1]);
    SetPoint(NewPoint[PLines+1],x,MForce[2]);
    SetPoint(NewPoint[PLines+2],x,MForce[3]);
  END;
END MPForce;


PROCEDURE MTForce(VAR PlotMess : MailType;PLines :INTEGER);

BEGIN
  WITH PlotMess^ DO
    SetPoint(NewPoint[PLines],x,MForce[4]);
    SetPoint(NewPoint[PLines+1],x,MForce[5]);
    SetPoint(NewPoint[PLines+2],x,MForce[6]);
  END;
END MTForce;


PROCEDURE MFForce(VAR PlotMess : MailType;PLines : INTEGER);

BEGIN
  WITH PlotMess^ DO
    SetPoint(NewPoint[PLines],x,MForce[4]);
    SetPoint(NewPoint[PLines+1],x,ElbowForce[1]);
    SetPoint(NewPoint[PLines+2],x,ElbowForce[2]);
  END;
END MFForce;


PROCEDURE DMailHandler(VAR NewPoint : PointerType);

BEGIN
   WITH PlotMess^ DO
     CASE PlotIndex OF
       1: Ashow(MForce[1],MForce[2],MForce[3],MForce[1],
                1,PLines,PLines+2,Ax[1]);
          Ashow(MForce[4],MForce[5],MForce[6],MForce[4],
                2,PLines+3,PLines+5,Ax[2])|
       2: Ashow(MForce[1],MForce[2],MForce[3],MForce[1],
                1,PLines,PLines+2,Ax[1])|
       3: Ashow(MForce[4],MForce[5],MForce[6],MForce[4],
                2,PLines,PLines+2,Ax[2])|
       4: Ashow(MForce[1],MForce[2],MForce[3],MForce[1],
                1,PLines,PLines+2,Ax[1]);
          Ashow(MForce[4],ElbowForce[1],ElbowForce[2],
                MForce[4],2,PLines+3,PLines+5,Ax[2]);
     END;
     CASE StimIndex OF
       3: SetPoint(NewPoint[1],x,Stimulus[1]);
          SetPoint(NewPoint[2],x,Stimulus[2])|
       2: SetPoint(NewPoint[1],x,Stimulus[1])|
       1: SetPoint(NewPoint[1],x,Stimulus[2])|
```

```
      0: ;
    END;
    CASE PlotIndex OF
      1: MPForce(PlotMess,PLines);
         MTForce(PlotMess,PLines+3);
      2: MPForce(PlotMess,PLines);
      3: MTForce(PlotMess,PLines);
      4: MPForce(PlotMess,PLines);
         MFForce(PlotMess,PLines+3);
    END;
  END;
  SendMessage(MBox,PlotMess);
END DMailHandler;


PROCEDURE AMailHandler(VAR NewPoint : PointerType);

VAR   i   :   INTEGER;
BEGIN
  WITH PlotMess^ DO
      Ashow(MForce[1],MForce[2],MForce[3],MForce[1],1,1,3,Lines+1);
      Ashow(MForce[4],MForce[5],MForce[6],MForce[4],2,4,6,Lines+2);
      FOR i:=1 TO 6 DO
          SetPoint(NewPoint[i],x,MForce[i]);
      END;
      IF GetMeasIndex()=4 THEN
          Ashow(ElbowForce[1],ElbowForce[2],ElbowForce[1],
                ElbowForce[2],3,7,8,Lines+3);
          SetPoint(NewPoint[7],x,ElbowForce[1]);
          SetPoint(NewPoint[8],x,ElbowForce[2]);
      END;
  END;
  SendMessage(MBox,PlotMess);
END AMailHandler;


PROCEDURE SMailHandler(VAR NewPoint : PointerType);

VAR PLines : INTEGER;

BEGIN
   WITH PlotMess^ DO
     CASE StimIndex OF
       3: SetPoint(NewPoint[1],x,Stimulus[1]);
          SetPoint(NewPoint[2],x,Stimulus[2]);
       2: SetPoint(NewPoint[1],x,Stimulus[1]);
       1: SetPoint(NewPoint[1],x,Stimulus[2]);
       0: ;
     END;
   END;
   SendMessage(MBox,PlotMess);
END SMailHandler;


(*********************************************************************)

(*****************   windowplot  procedure      *********************)


PROCEDURE PlotCurve;

VAR         K : INTEGER;

BEGIN
   PlotScale;
   IF (Nr = 0) THEN                  (* Makes a x vs time plot *)
     IF Win=3 THEN
       Axes(PLWindow[Start],0.6);
       FOR K :=1 TO (Win-1) DO
         Axes(PLWindow[K+Start],0.8);
       END;
     ELSIF Win<>4 THEN
       FOR K :=0 TO (Win-1) DO
         Axes(PLWindow[K+Start],0.8);
       END;
     END;
     IF Win=4 THEN
```



```
       FOR K:=Lines+1 TO NumWin DO
       Axes(PLWindow[K],0.8);
       END;
     END;
     x:=x+Delta;            (* plots the values in a timediagram *)
     Nr:=Nr+1;
     FOR K:=1 TO Lines DO
       OldPoint[K]:=NewPoint[K];
     END;
   ELSE
     FOR K:= 1 TO Lines DO
       HideCursor;
       PlotLine(PLWindow[K],OldPoint[K],NewPoint[K]);
       ShowCursor;
       OldPoint[K]:=NewPoint[K];
     END;
     x:=x+Delta;
     Nr:=Nr+1;
     IF (Nr>=Size) THEN
       x:=0.05;
       Nr:=0;
     END;
   END;
END PlotCurve;


PROCEDURE PlotCurve1;

VAR         K : INTEGER;

BEGIN
   IF (Nr = 0) THEN                  (* Makes a x vs time plot *)
     IF Win=3 THEN
       Axes(PLWindow[Start],0.6);
       FOR K :=1 TO (Win-1) DO
         Axes(PLWindow[K+Start],0.8);
       END;
     ELSE
       FOR K :=0 TO (Win-1) DO
         Axes(PLWindow[K+Start],0.8);
       END;
     END;
     x:=x+Delta;            (* plots the values in a timediagram *)
     Nr:=Nr+1;
     FOR K:=1 TO Lines DO
       OldPoint[K]:=NewPoint[K];
     END;
   ELSE
     FOR K:= 1 TO Lines DO
       HideCursor;
       PlotLine(PLWindow[K],OldPoint[K],NewPoint[K]);
       ShowCursor;
       OldPoint[K]:=NewPoint[K];
     END;
     x:=x+Delta;
     Nr:=Nr+1;
     IF (Nr>=Size) THEN
       x:=0.05;
       Nr:=0;
     END;
   END;
END PlotCurve1;


PROCEDURE PlotScale;

  VAR Point1,Point2 : point;

BEGIN
   SetPoint(Point1,step*float(Index)+RunTime,0.0);
   SetPoint(Point2,step*float(Index)+RunTime,1.0);
   IF Index=0 THEN
     WITH PLWindow[NumWin-1]^ DO
       FillRectangle(H, WindowRectangle);      (* Clear window      *)
       DrawRectangle(H, WindowRectangle);      (* Draw rims         *)
     END;
     WITH PLWindow[NumWin]^ DO
```

```
        FillRectangle(H, WindowRectangle);      (* Clear window       *)
        DrawRectangle(H, WindowRectangle);      (* Draw rims          *)
      END;
    END;
    HideCursor;
    PlotLine(PLWindow[NumWin-1],Point1,Point2);
    ShowCursor;
    Index:=Index+1;
 END PlotScale;


(*Process*) PROCEDURE Plot;

 BEGIN
    SetPriority(98);
    NumWin:=1;
    InitColor;
    GetMeasWork(Work);
    LOOP;
      ReceiveMessage(PBox,PlotMess);
      CheckPlotWhat(WhatKind,Change);
      IF WhatKind=Oper THEN
        SendMessage(MBox,PlotMess);
      END;
      IF Change THEN              (* selects plotobjekt  *)
        GetMeasWork(Work);
        CASE WhatKind OF
          Data : StartUp|
          Test : ClearWindow;TestWindow|
          ADIn : ADInWindow|
          Oper : ClearWindow;
        END;
      END;
      IF (WhatKind = Data) AND (Work=OldData) THEN
        DMailHandler(NewPoint);       (* data value plotting *)
        PlotCurve1;
      ELSIF WhatKind = Data THEN
        DMailHandler(NewPoint);       (* data value plotting *)
        PlotCurve;
      ELSIF WhatKind = Test THEN        (* control plotting *)
        SMailHandler(NewPoint);
        PlotCurve;
      ELSIF WhatKind = ADIn THEN        (* calibration plotting *)
        AMailHandler(NewPoint);
        PlotCurve;
      END;
    END;
 END Plot;
END PlotProcess.
```



```
DEFINITION MODULE Safe;


EXPORT QUALIFIED SafeProc;


PROCEDURE SafeProc;
(*Creates the emergencywindow and stops the regulator when the emergencybutton
  is pressed.*)


END Safe.
```

```
IMPLEMENTATION MODULE Safe;

 IMPORT RTMouse;

 FROM Kernel    IMPORT SetPriority;

 FROM Graphics IMPORT handle,rectangle,color,point,SetWindow,
                      SetViewPort,VirtualScreen,SetFillColor,
                      FillRectangle,DrawRectangle,WaitMouseRectangle,
                      HideCursor,SetMouseRectangle,SetLineColor,
                      SetTextColor,WriteString,ShowCursor;

 FROM Decl      IMPORT PutEmerg,TheEnd;

 CONST      EmergencyColor     = red;
            EmergencyTextColor = yellow;

 PROCEDURE SetRectangle(VAR R :rectangle; Xlo,Ylo,Xhi,Yhi :REAL);

 BEGIN
    R.xlo:= Xlo;
    R.ylo:= Ylo;
    R.xhi:= Xhi;
    R.yhi:= Yhi;
 END SetRectangle;


 PROCEDURE InitEmergencyWindow(VAR H : handle);

 VAR ViewPortRectangle,WindowRectangle : rectangle;
     TextPoint                         : point;

 BEGIN                    (* initiate emergensy window *)
    VirtualScreen(H);
    SetRectangle(ViewPortRectangle, 0.05, 0.05, 0.45, 0.15);
    SetViewPort(H,ViewPortRectangle);
    SetRectangle(WindowRectangle, 0.0, 0.0, 1.0, 1.0);
    SetWindow(H,WindowRectangle);
    SetTextColor(H, EmergencyTextColor);
    SetLineColor(H, EmergencyTextColor);
    SetFillColor(H, EmergencyColor);
    HideCursor;
    FillRectangle(H,WindowRectangle);
    TextPoint.h:= 0.3;
    TextPoint.v:= 0.3;
    DrawRectangle(H,WindowRectangle);
    WriteString(H, TextPoint, "Emergency");
    SetMouseRectangle(H,WindowRectangle,1);
    ShowCursor;
 END InitEmergencyWindow;

(*Process*)PROCEDURE SafeProc;

 VAR        SelectedBox : CARDINAL;
            H           : handle;

 BEGIN
    SetPriority(89);
    InitEmergencyWindow(H);
    LOOP
      SelectedBox:=WaitMouseRectangle(H);
      PutEmerg(TRUE);
    END;
 END SafeProc;

END Safe.
```

Appendix 8 - Mätdata

Detta kapitel är indelat i två delar där första delen visar grafer på hur momentet varierar i tiden, och den andra visar mätseriernas stabilogram. (Hur momentet i sagittalled varierar som funktion av momentet i lateralled.)

1

1.1 Galvanisk stimulering och öppna ögon. (sagittalled)

1.2 Galvanisk stimulering och slutna ögon. (sagittalled)

1.3 Vibrationsstimulering och öppna ögon. (sagittalled)

1.4 Vibrationsstimulering och slutna ögon. (sagittalled)

1.5 Vibrationsstimulering och slutna ögon. (armstöd, lateralled)

2

2.1 Galvanisk stimulering och öppna ögon. (stabilogram)

2.2 Galvanisk stimulering och slutna ögon. (stabilogram)

2.3 Vibrationsstimulering och öppna ögon. (stabilogram)

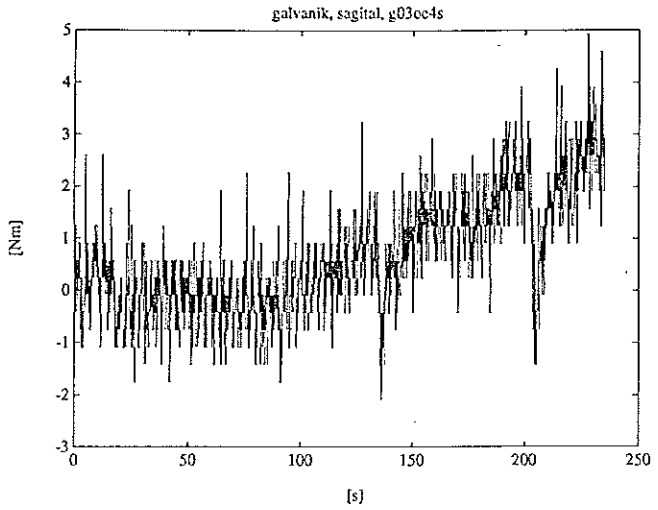
2.4 Vibrationsstimulering och slutna ögon. (stabilogram)

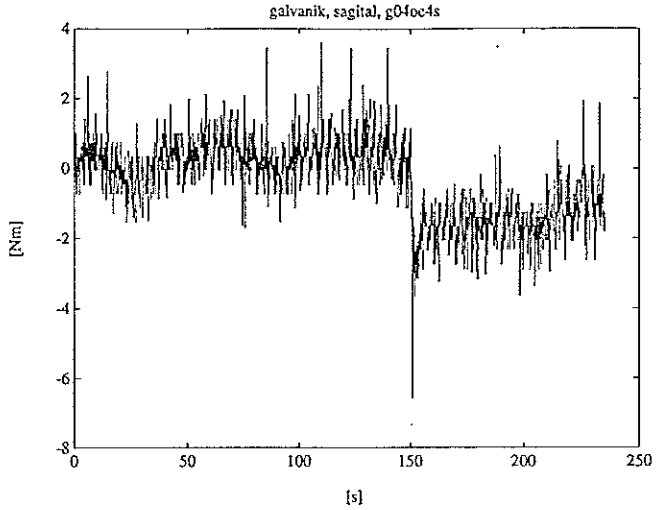
2.4 Vibrationsstimulering och slutna ögon. (armstöd, stabilogram)

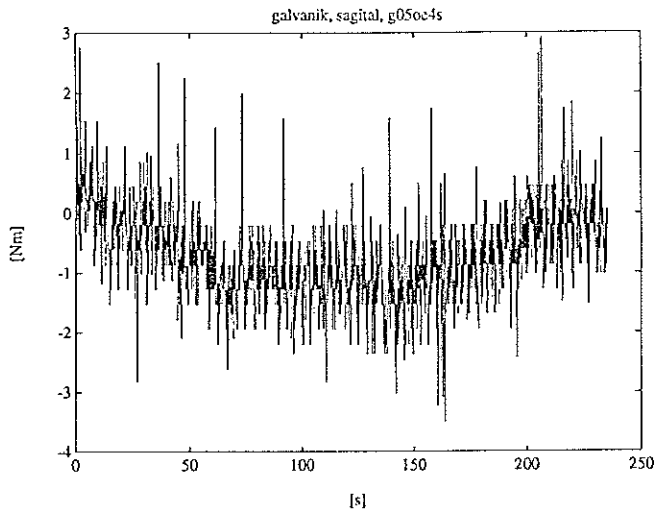
Försökspersonerna är redovisade i samma ordningsföljd vid samtliga försök.

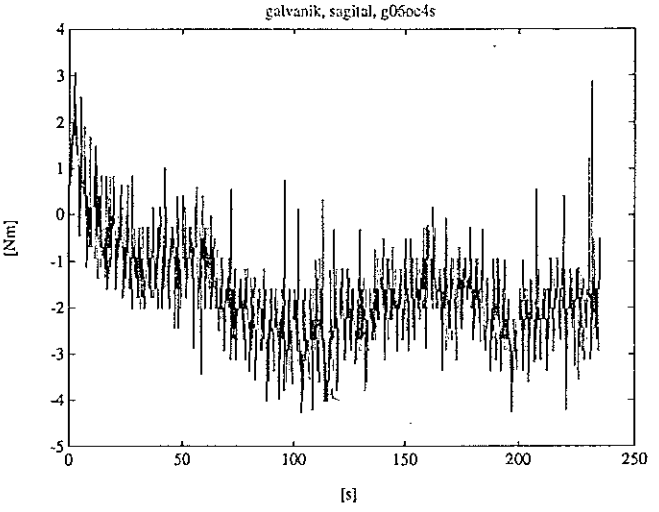
1.1 Mätdata vid galvanisk stimulering

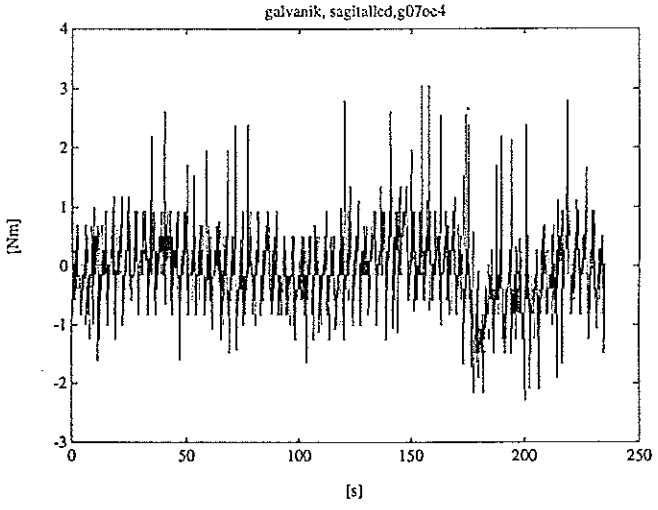
Följande 11 bilder visar utsignalen från kraftplattan i sagittalled vid försök med galvanisk stimulering och öppna ögon. Positivt respektive negativt moment innebär tyngdpunktsförflyttning framåt respektive bakåt.

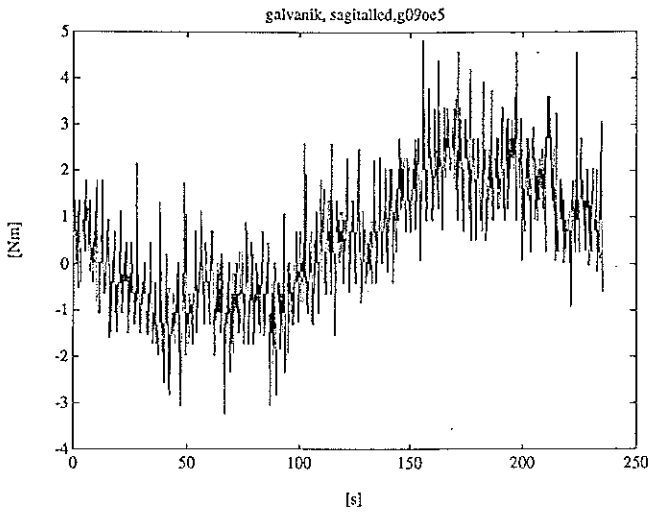


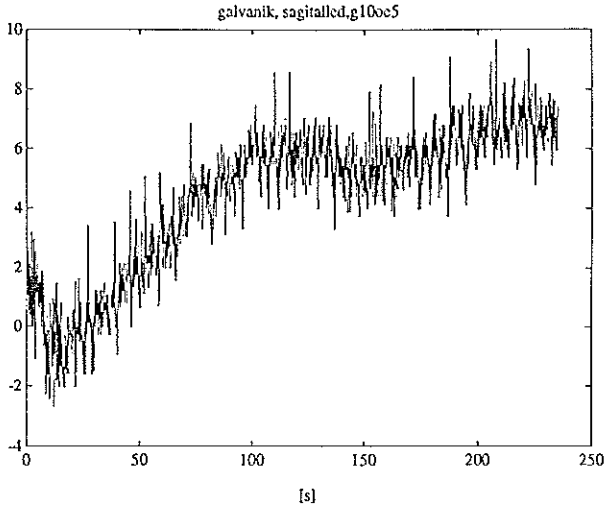


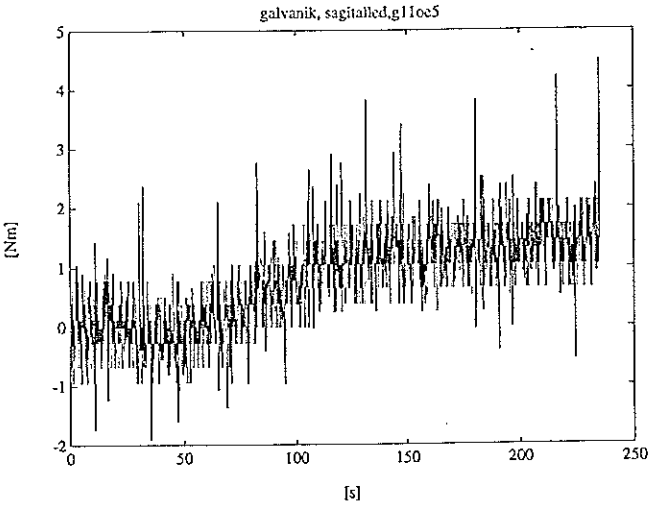


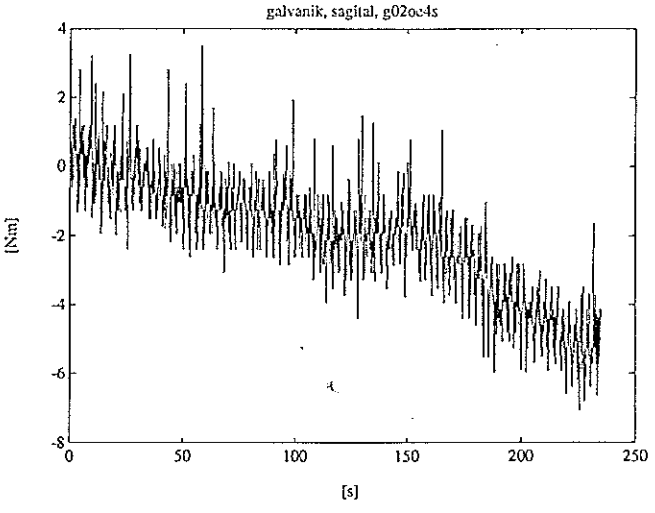


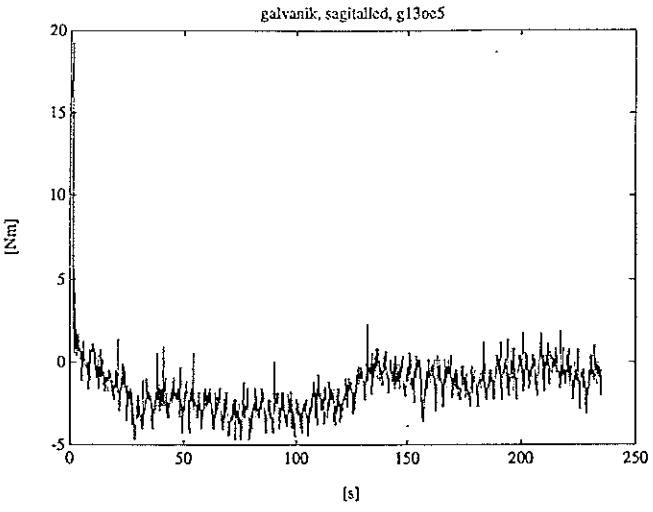


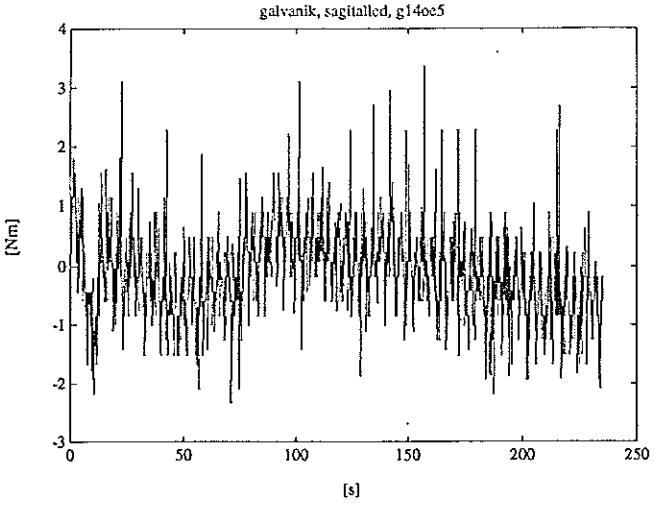

galvanik, sagitalled,g09oe5
[Nm]
[s]


galvanik, sagitalled,g10oe5
[s]


galvanik, sagitalled,g11oe5
[Nm]
[s]

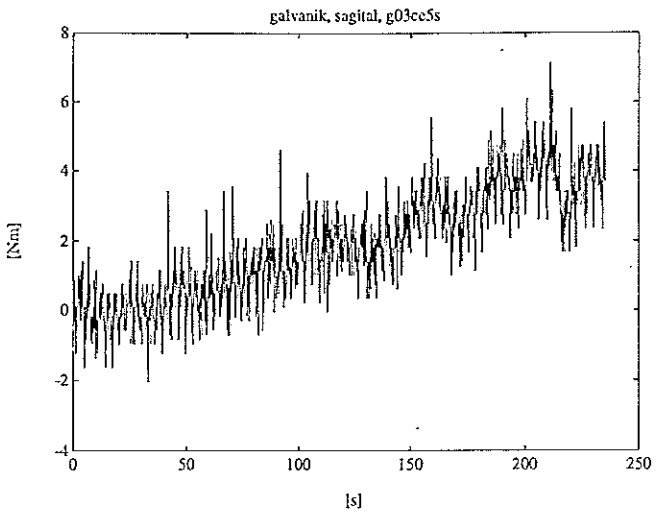

galvanik, sagital, g02oe4s
[Nm]
[s]

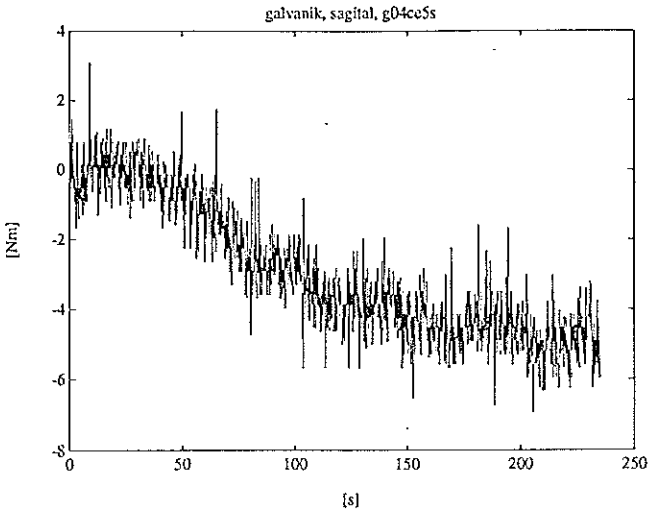

galvanik, sagitalled, g13oe5
[Nm]
[s]

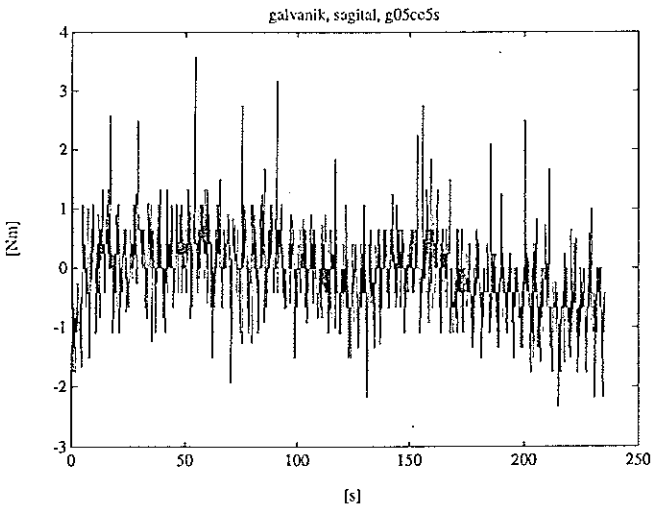

galvanik, sagitalled, g14oe5
[Nm]
[s]

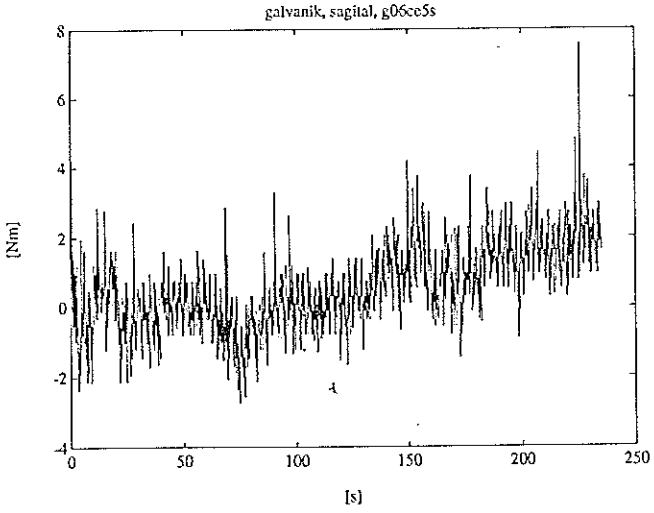
1.2

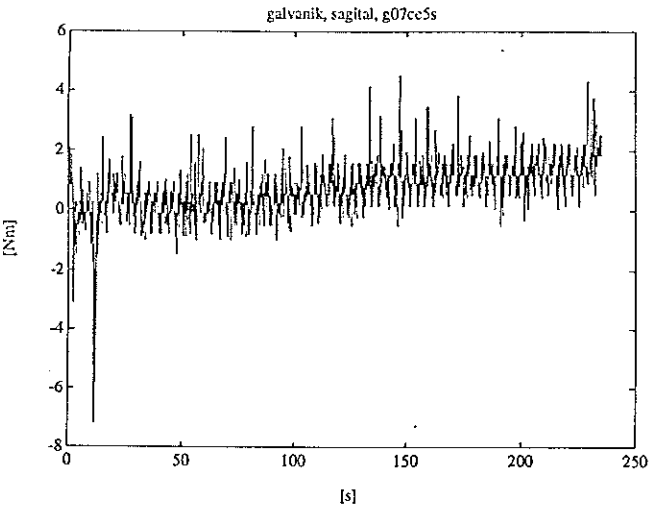
Följande 12 bilder visar utsignalen från kraftplattan i sagittalled vid försök med galvanisk stimulering och slutna ögon. Positivt respektive negativt moment innebär tyngdpunktsförflyttning framåt respektive bakåt.

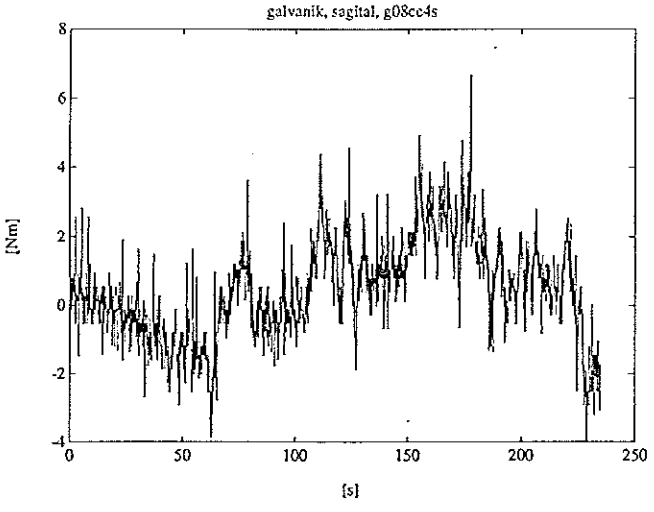


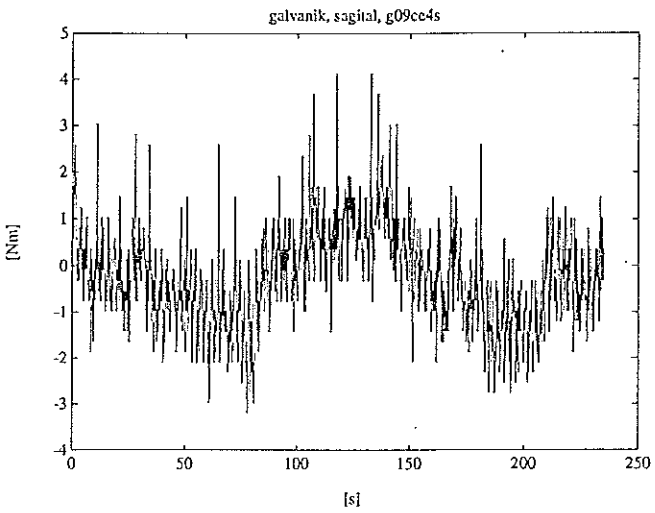


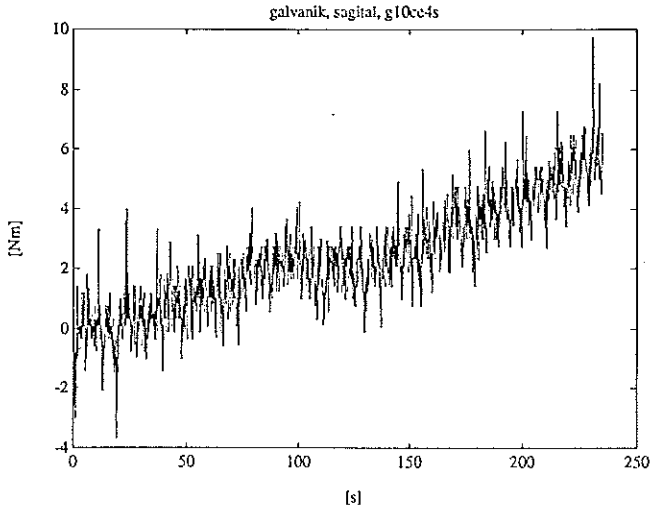


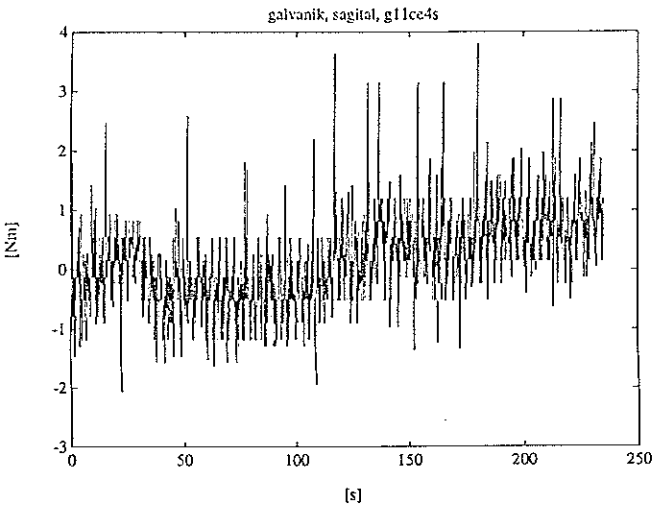


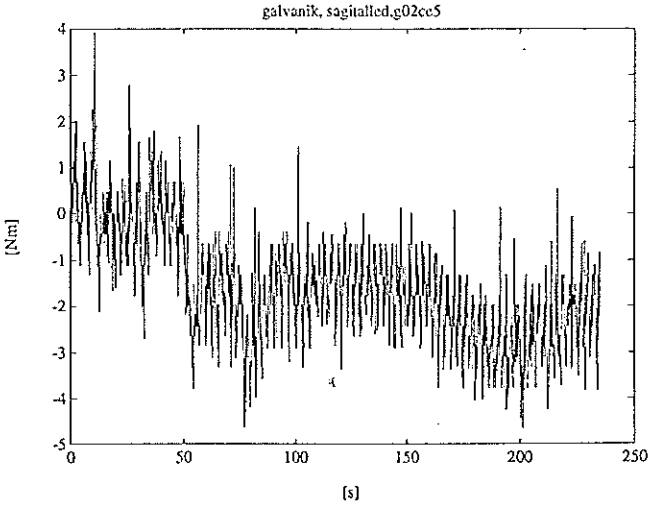


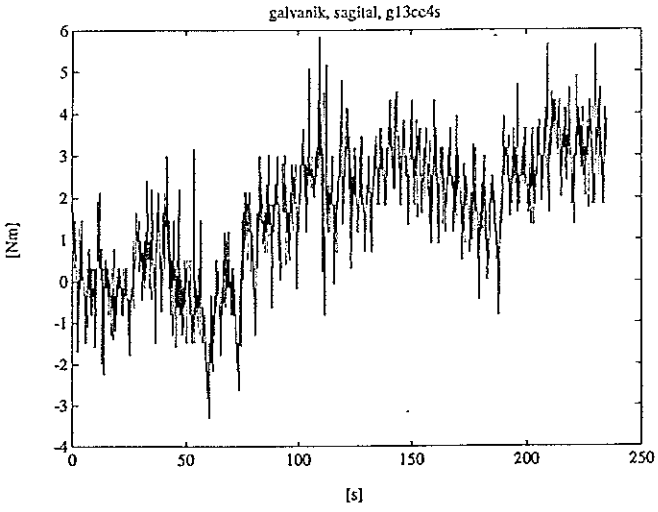


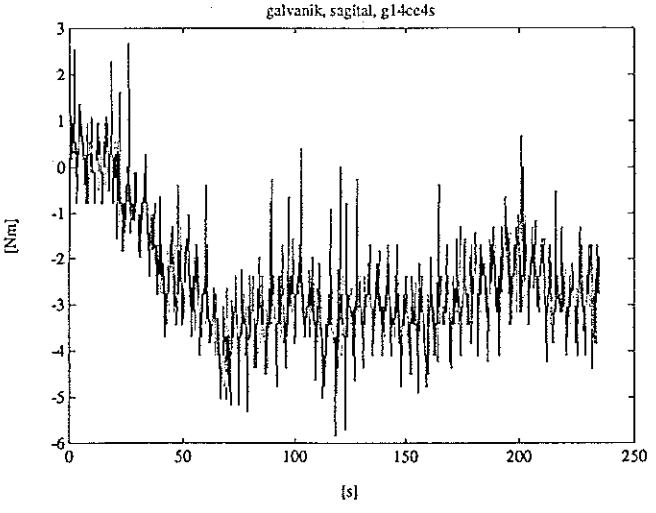

galvanik, sagital, g09ce4s
[Nm]
[s]


galvanik, sagital, g10ce4s
[Nm]
[s]


galvanik, sagital, g11ce4s
[Nm]
[s]

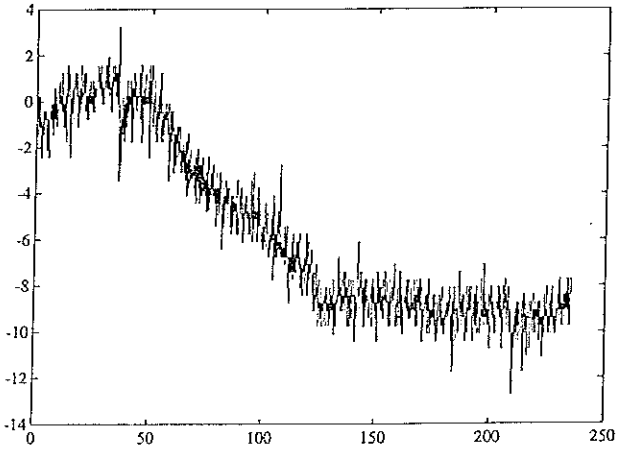

galvanik, sagitalled,g02ce5
[Nm]
[s]

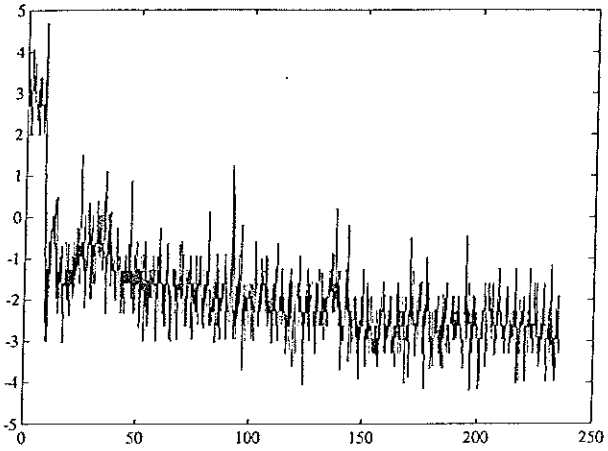

galvanik, sagital, g13ce4s
[Nm]
[s]

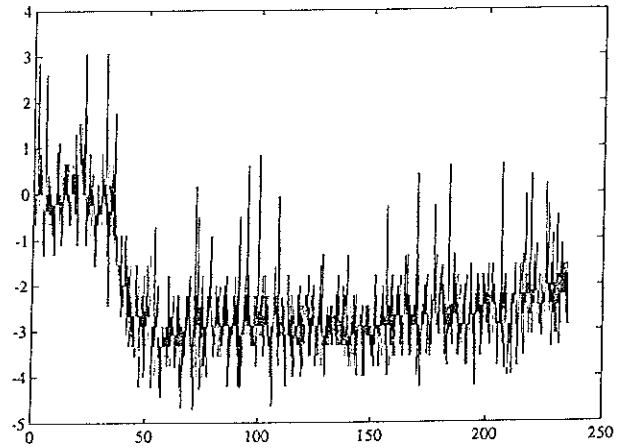

galvanik, sagital, g14ce4s
[Nm]
[s]

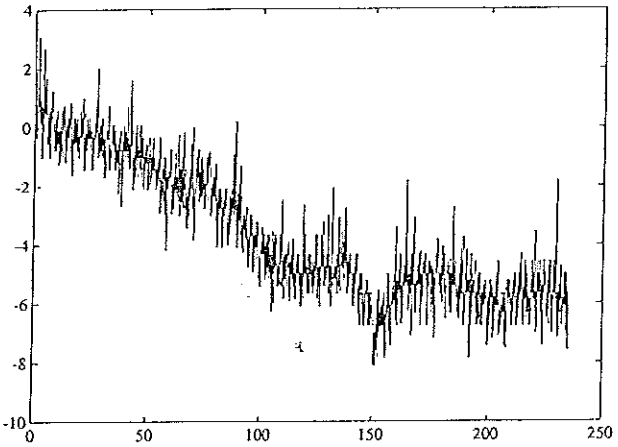
1.3 Mätdata vid vibrationsstimulering

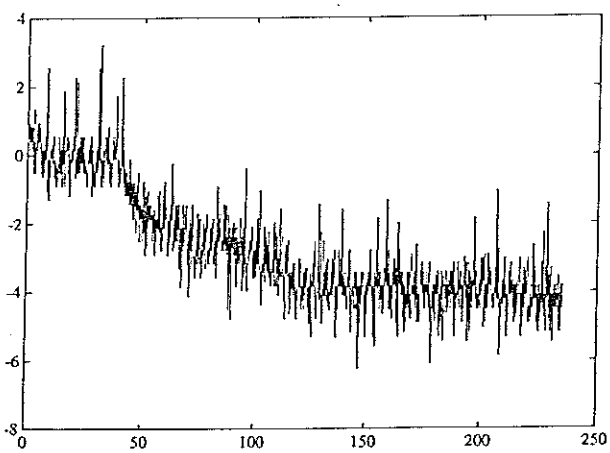
Följande 12 bilder visar utsignalen från kraftplattan i sagittalled vid försök med vibrationsstimulering och öppna ögon. Positivt respektive negativt moment innebär tyngdpunktsförflyttning framåt respektive bakåt.

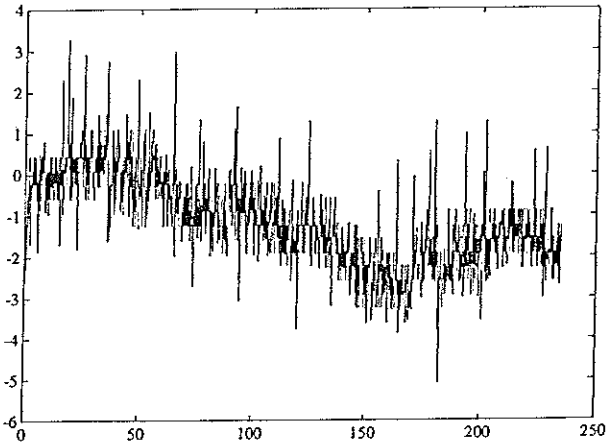


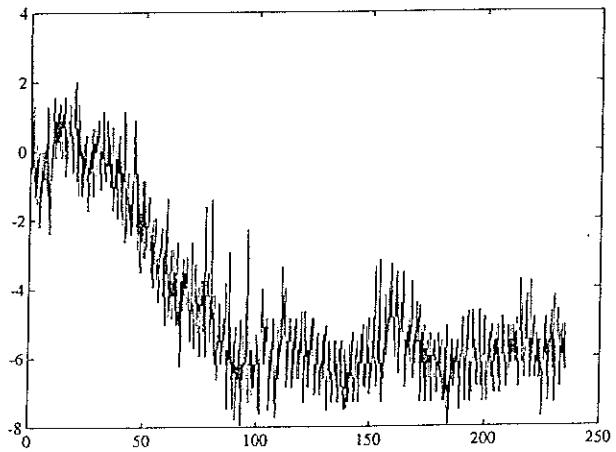


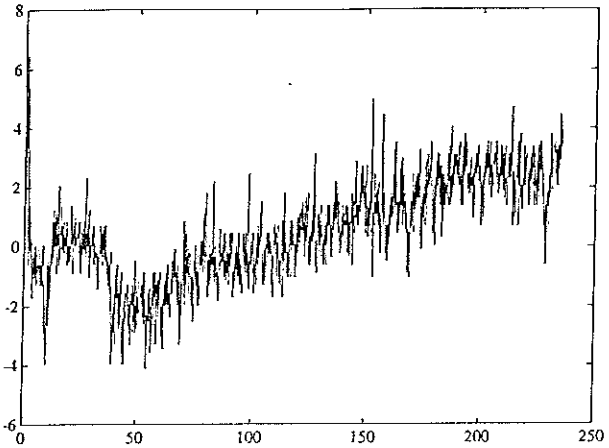


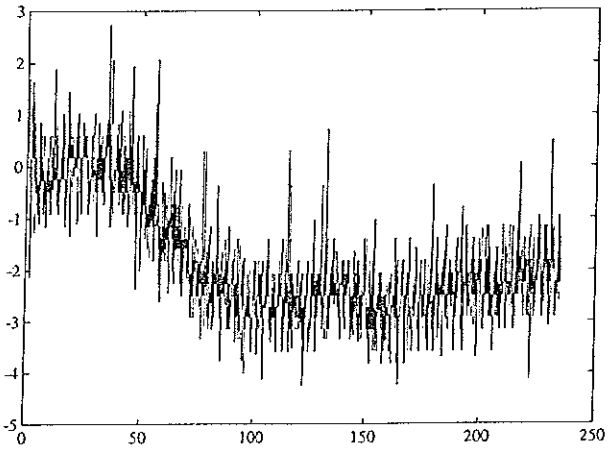


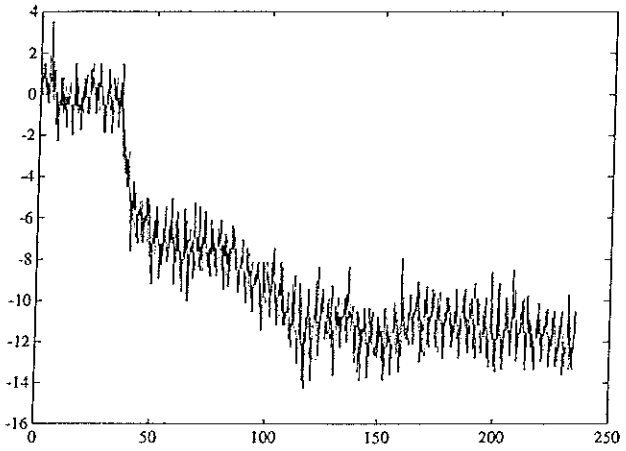


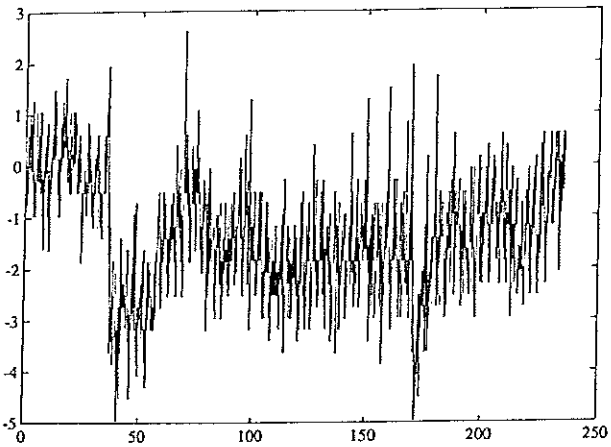


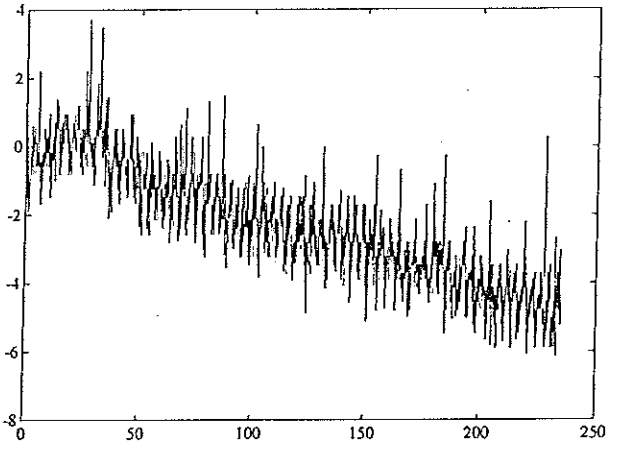

4
2
0
-2
-4
-6
-8
0
50
100
150
200
250


8
6
4
2
0
-2
-4
-6
0
50
100
150
200
250


3
2
1
0
-1
-2
-3
-4
-5
0
50
100
150
200
250

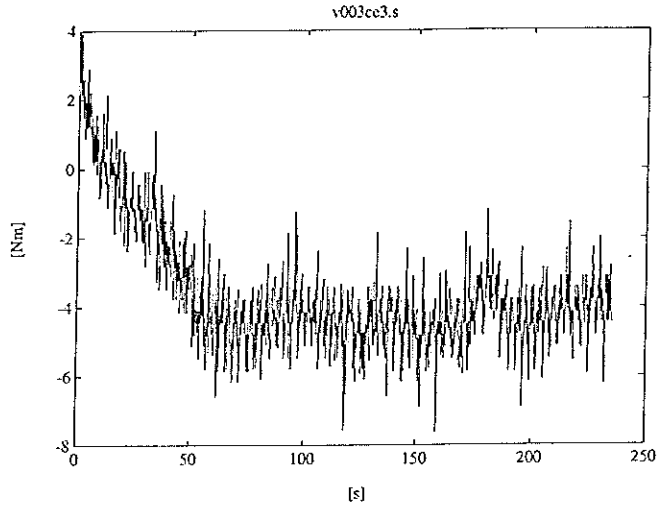

4
2
0
-2
-4
-6
-8
-10
-12
-14
-16
0
50
100
150
200
250

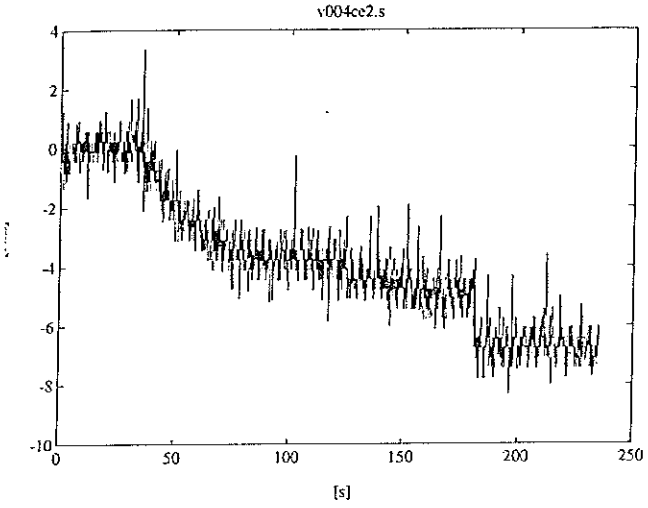

3
2
1
0
-1
-2
-3
-4
-5
0
50
100
150
200
250

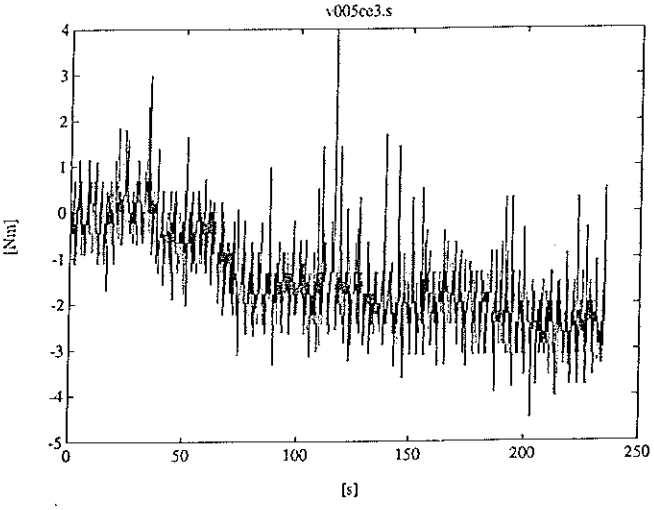

4
2
0
-2
-4
-6
-8
0
50
100
150
200
250

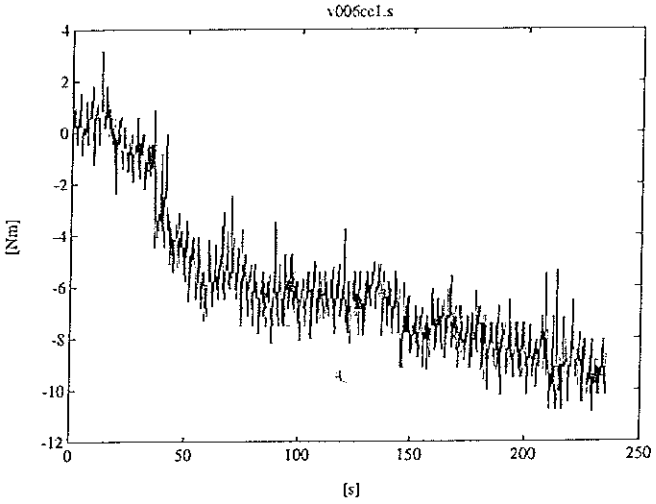
1.4

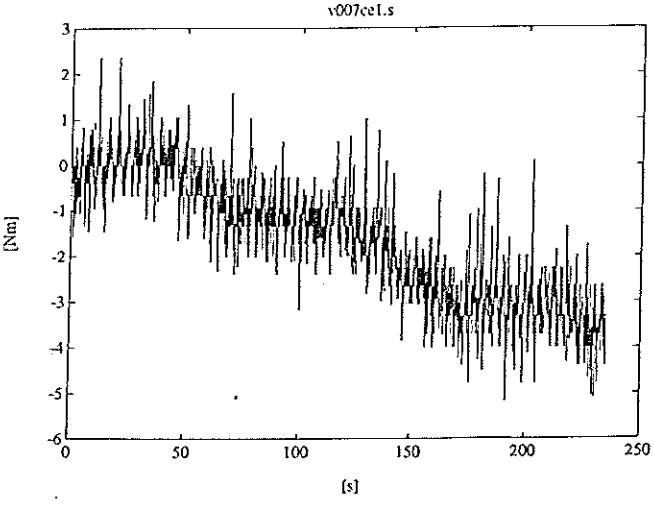
Följande 12 bilder visar utsignalen från kraftplattan i sagittalled vid försök med vibrationsstimulering och slutna ögon. Positivt respektive negativt moment innebär tyngdpunktsförflyttning framåt respektive bakåt.

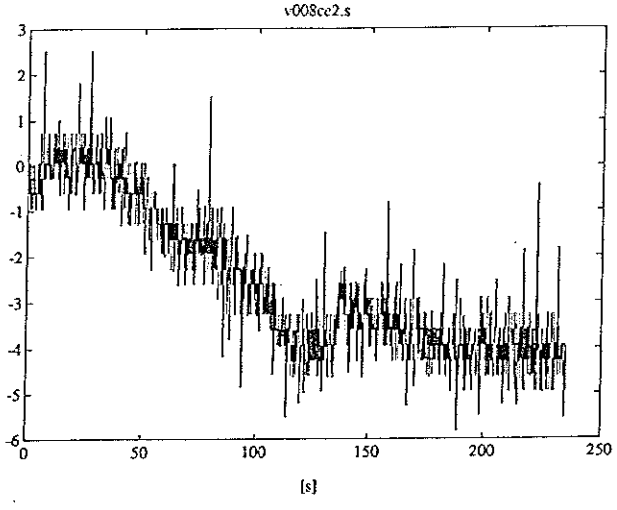


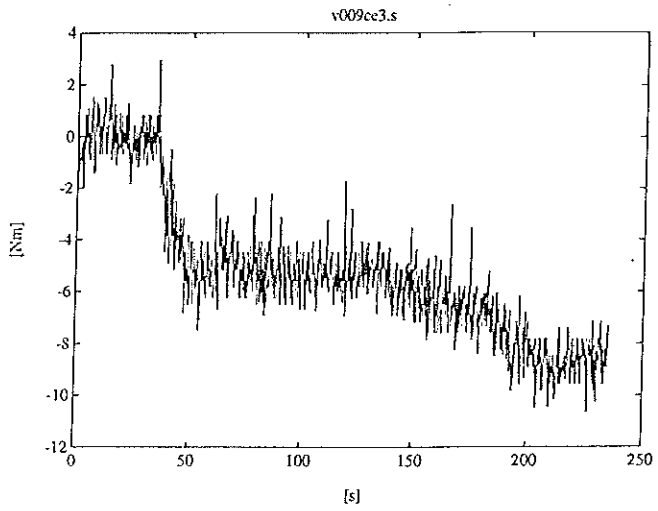


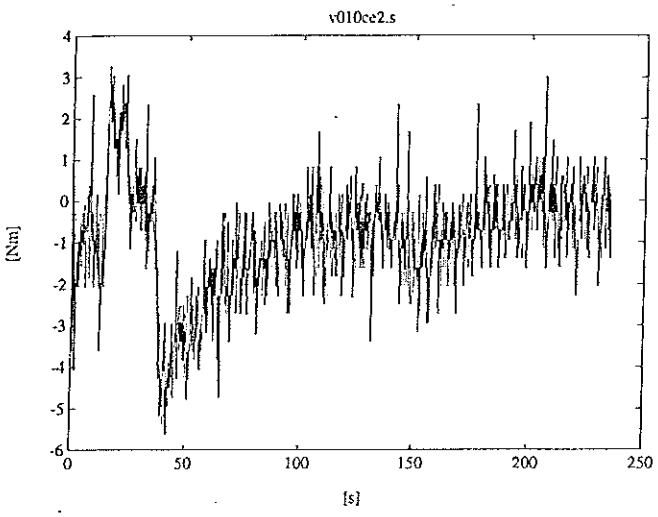


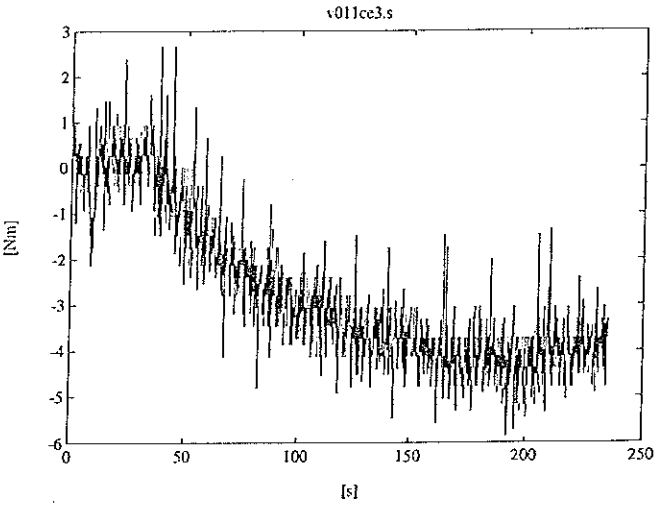


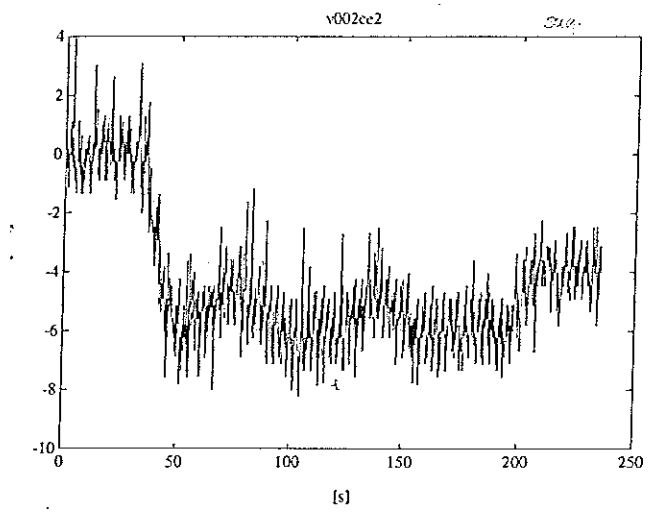


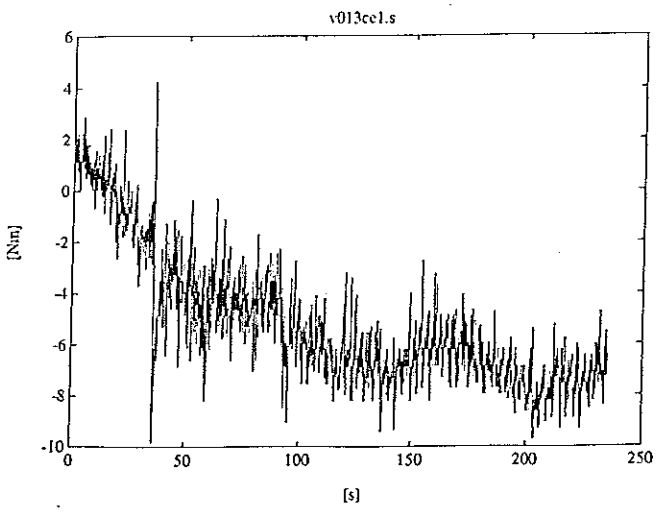


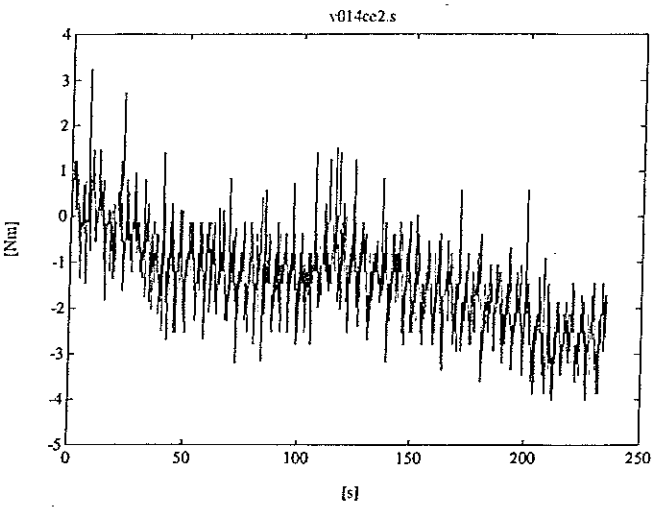

v009ce3.s
[Nm]
[s]


v010ce2.s
[Nm]
[s]


v011ce3.s
[Nm]
[s]

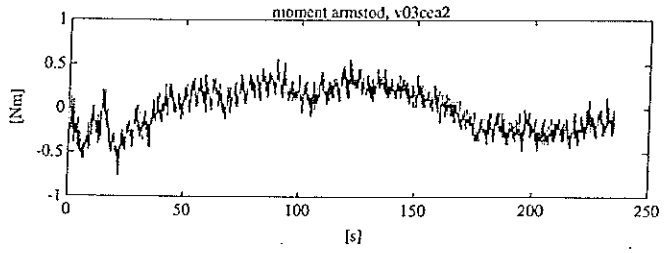

v002ce2
[s]

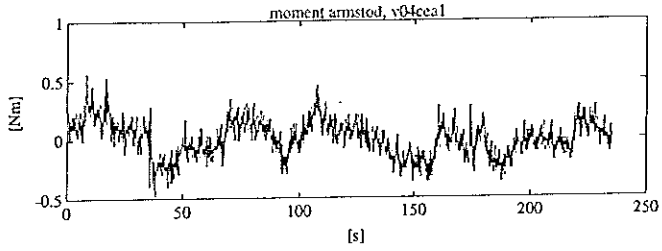

v013ce1.s
[Nm]
[s]

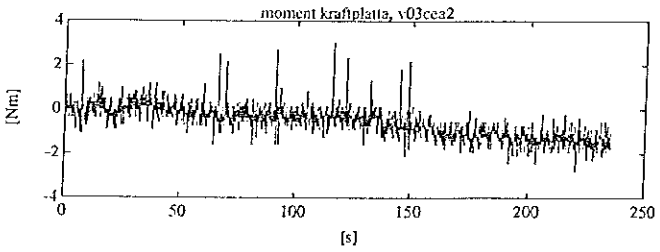

v014ce2.s
[Nm]
[s]

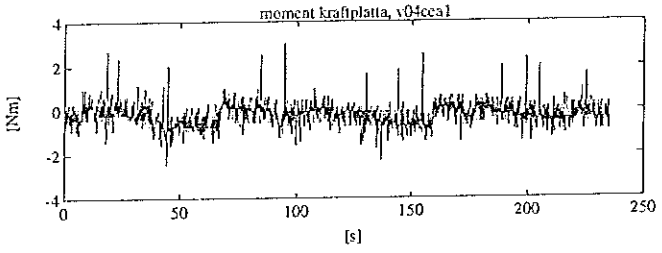
Mätdata vid vibrationsstimulering
1.5

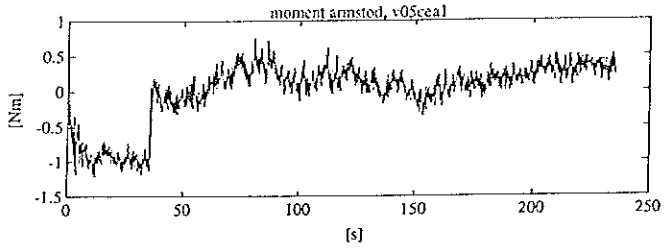
Följande 12 bilder visar utsignalen från *både* kraftplattan och armstöden i lateralled vid försök med vibrationsstimulering och slutna ögon. Positivt respektive negativt moment innebär tyngdpunktsförflyttning framåt respektive bakåt.

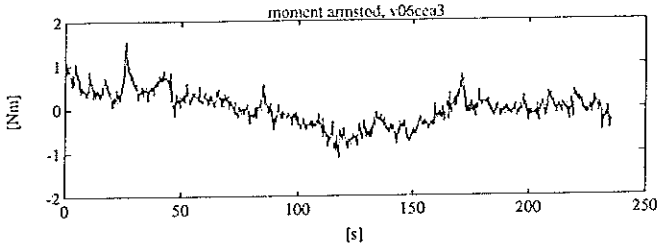


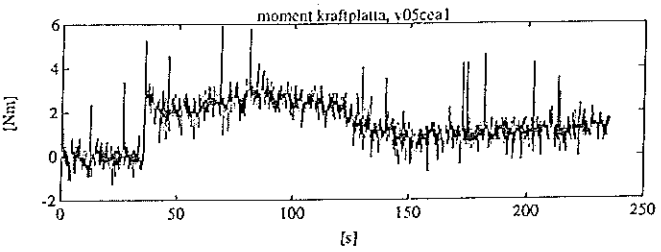


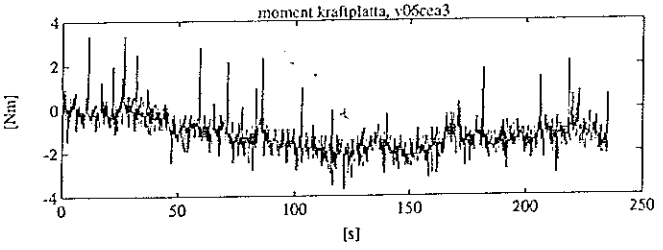


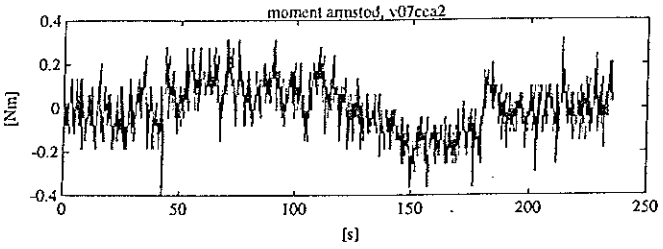


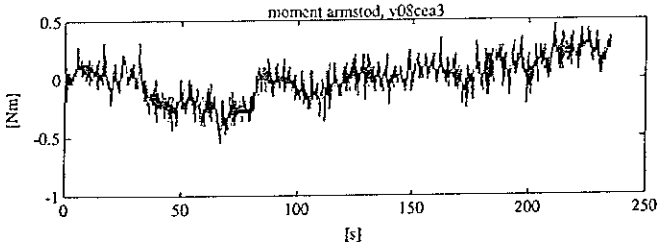


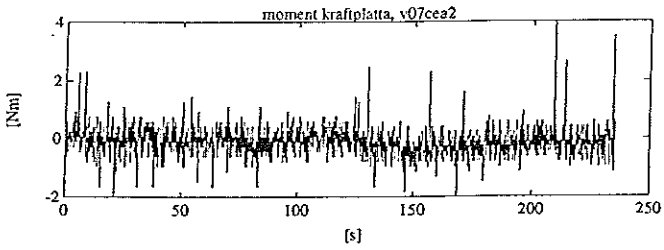


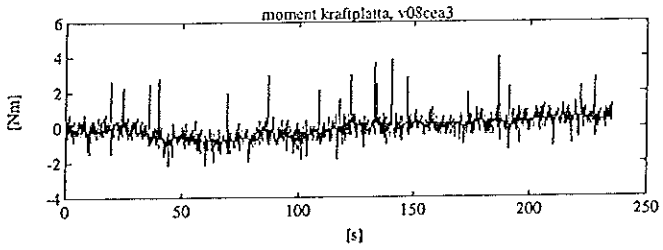


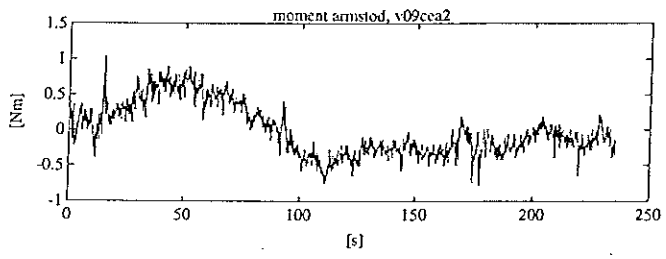


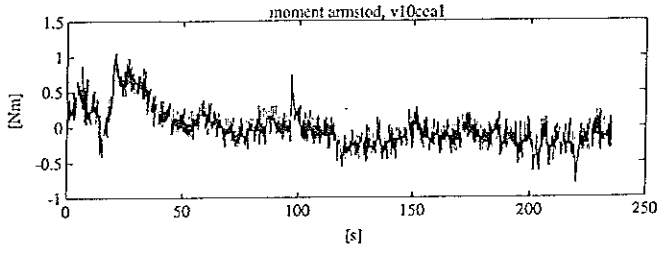


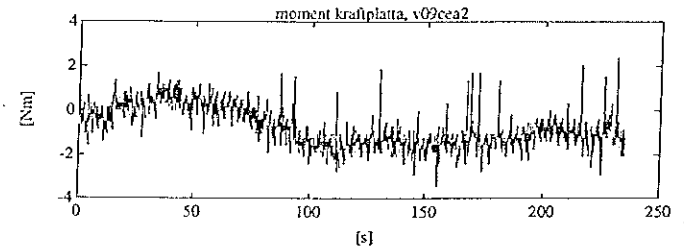


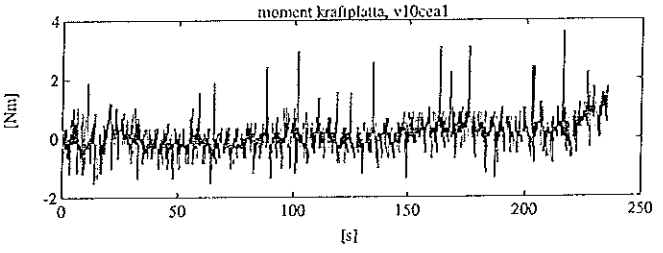


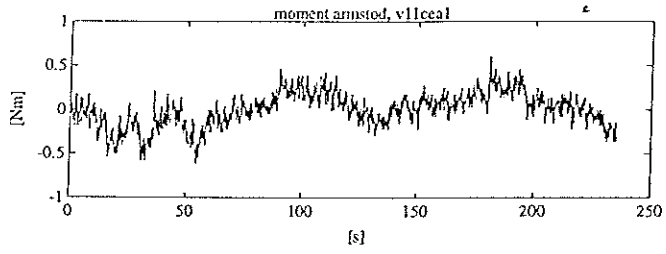


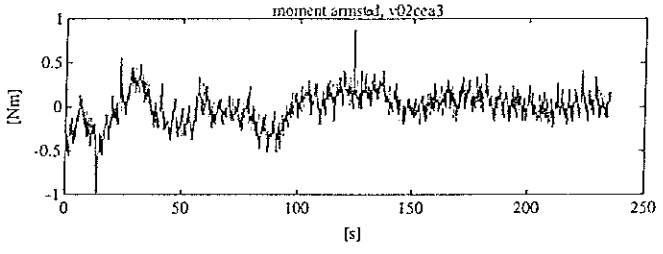

moment armstod, v09cea2
[Nm]
[s]

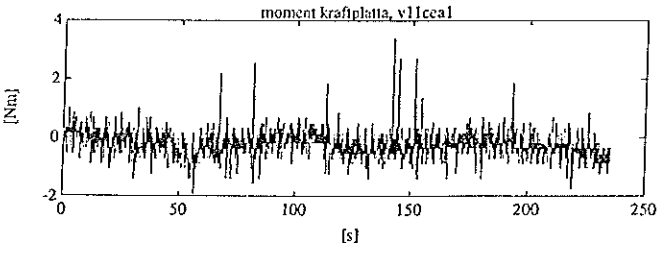

moment armstod, v10cea1
[Nm]
[s]

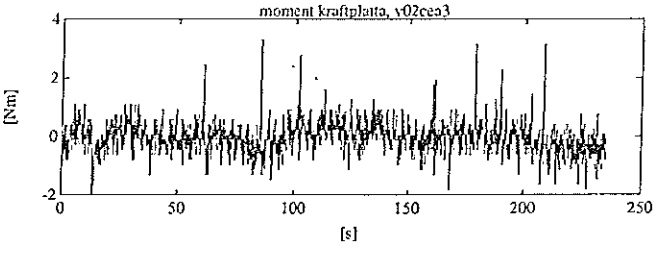

moment kraftplatta, v09cea2
[Nm]
[s]

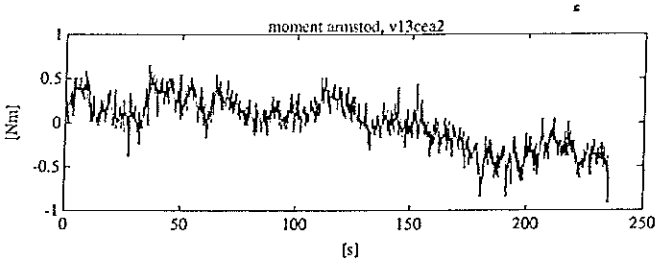

moment kraftplatta, v10cea1
[Nm]
[s]

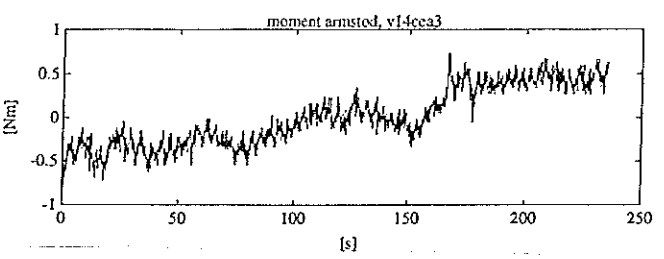

moment armstod, v11cea1
[Nm]
[s]

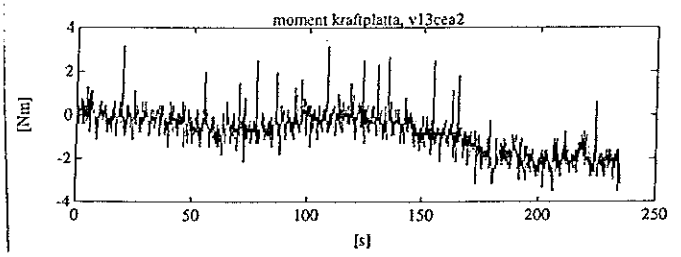

moment armstad, v02cea3
[Nm]
[s]

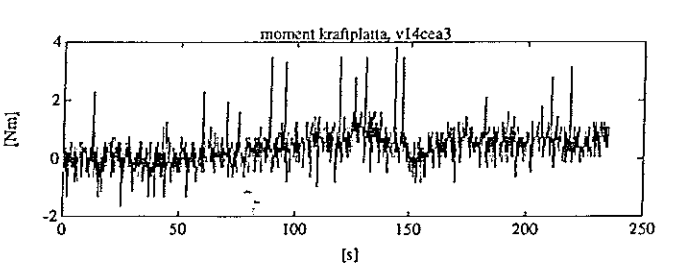

moment kraftplatta, v11cea1
[Nm]
[s]


moment kraftplatta, v02cea3
[Nm]
[s]


moment armstod, v13cea2
[Nm]
[s]

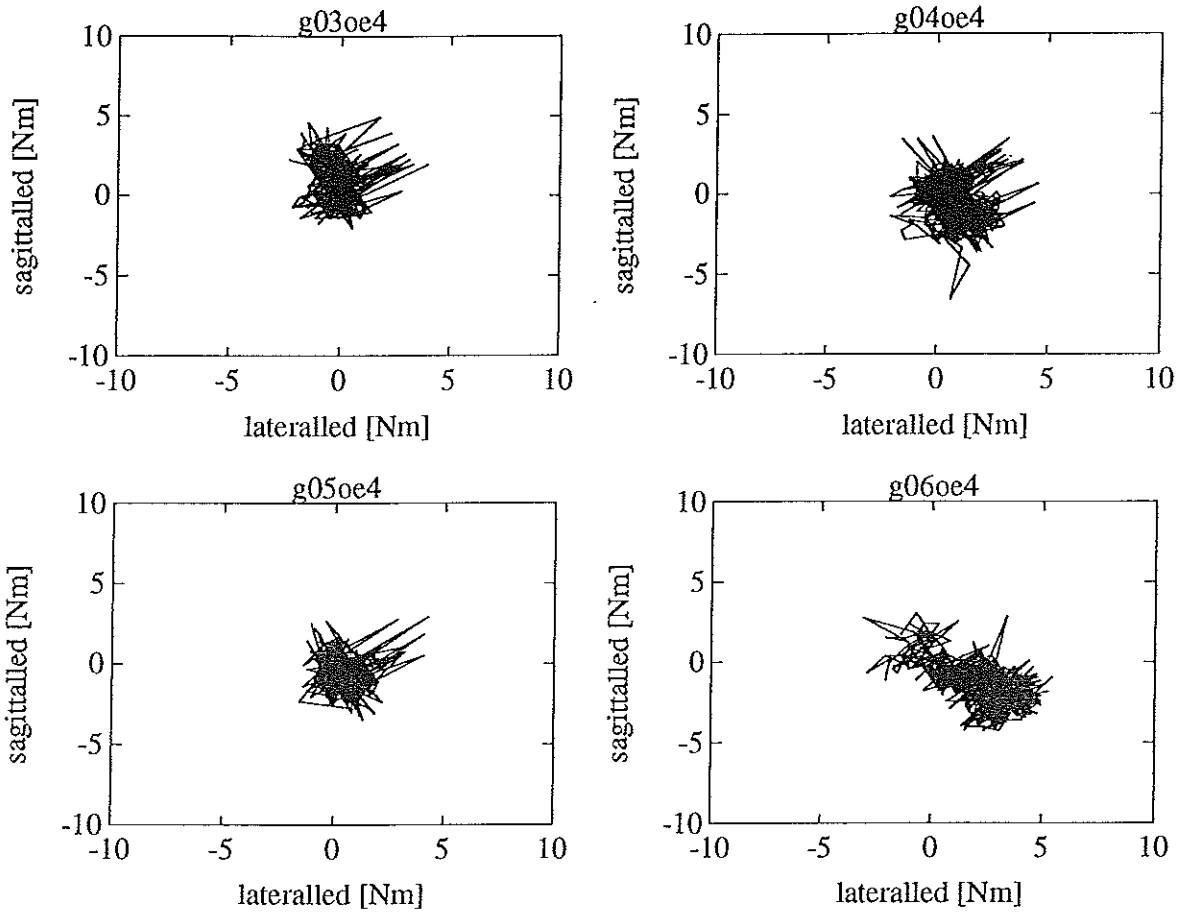

moment armstod, v14cea3
[Nm]
[s]

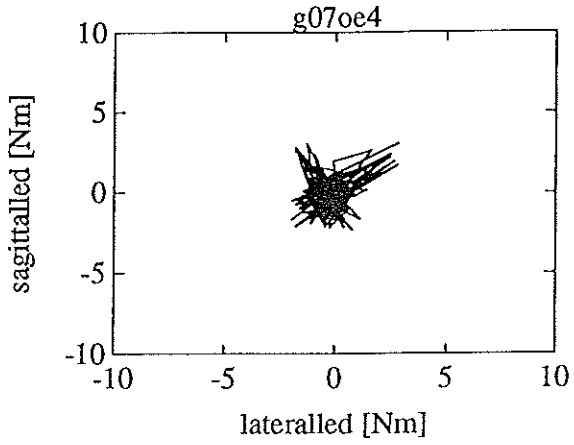

moment kraftplatta, v13cea2
[Nm]
[s]

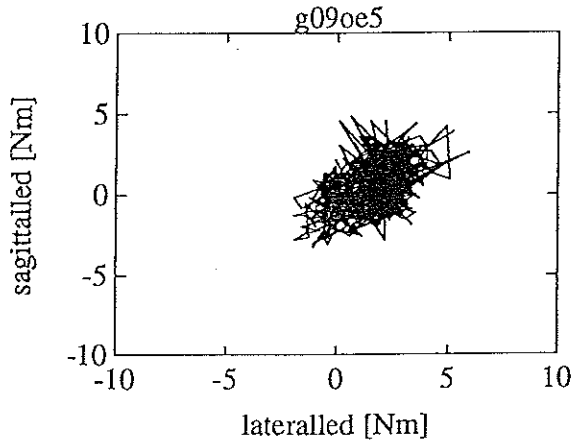

moment kraftplatta, v14cea3
[Nm]
[s]

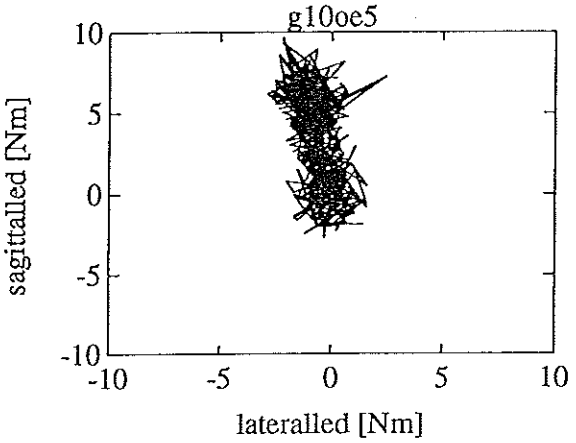
2.1 Mätdata vid galvanisk stimulering

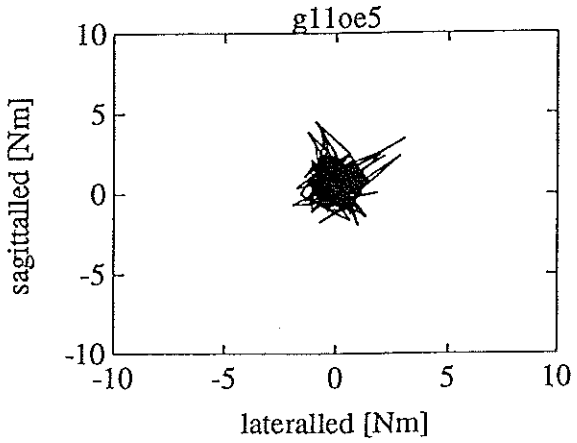
Följande 11 bilder visar utsignalen från kraftplattan vid försök med galvanisk stimulering och öppna ögon. Bilderna är ritade med lateralt moment i x-led och sagittalt moment i y-led. Bilderna visar alltså kroppens rörelse på kraftplattan. Försökspersonerna är redovisade i samma ordningsföljd vid samtliga försök.

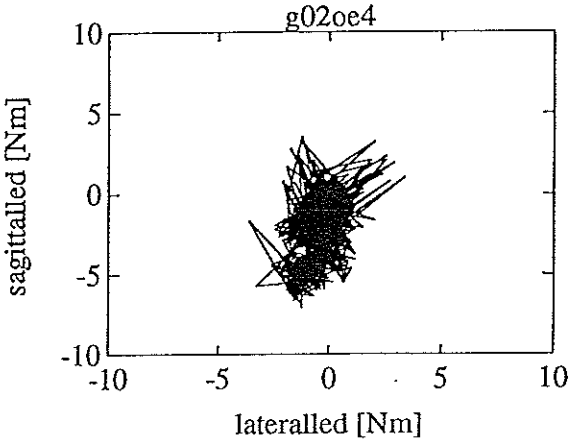


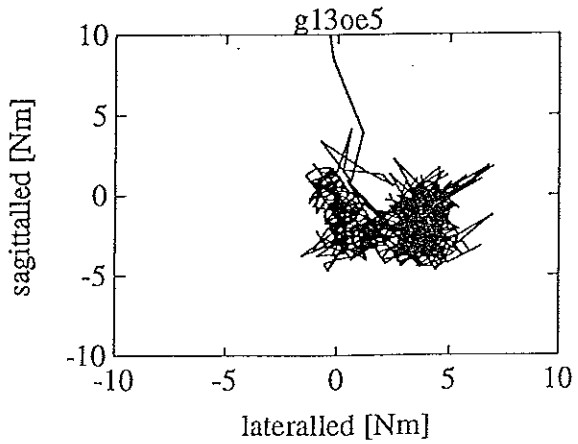

g07oe4
sagittalled [Nm]
lateralled [Nm]
10
5
0
-5
-10

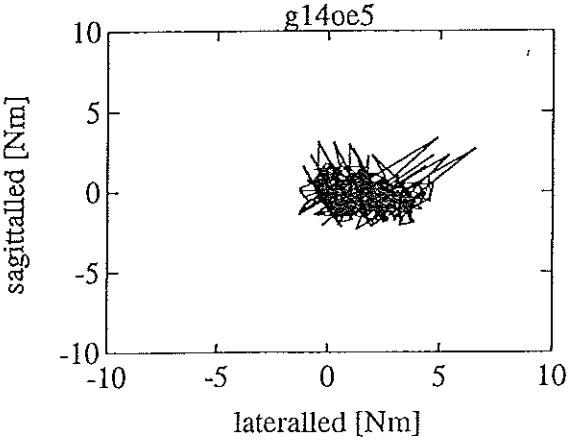

g09oe5
sagittalled [Nm]
lateralled [Nm]
10
5
0
-5
-10


g10oe5
sagittalled [Nm]
lateralled [Nm]
10
5
0
-5
-10


g11oe5
sagittalled [Nm]
lateralled [Nm]
10
5
0
-5
-10

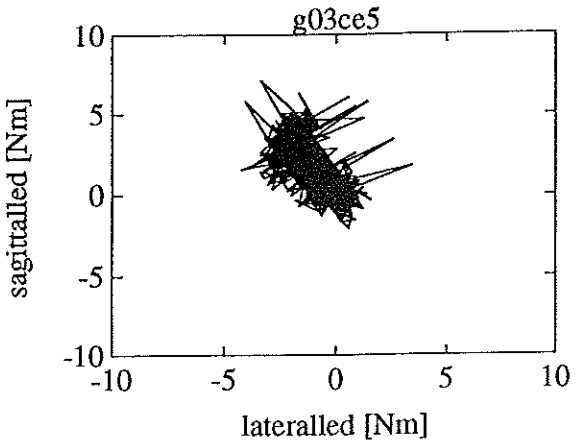

g02oe4
sagittalled [Nm]
lateralled [Nm]
10
5
0
-5
-10

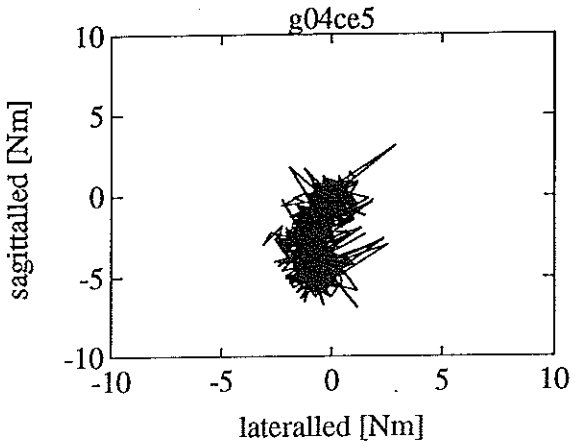

g13oe5
sagittalled [Nm]
lateralled [Nm]
10
5
0
-5
-10

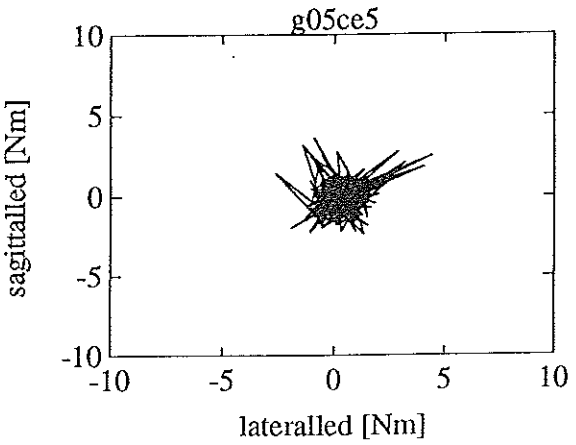

g14oe5
sagittalled [Nm]
lateralled [Nm]
10
5
0
-5
-10

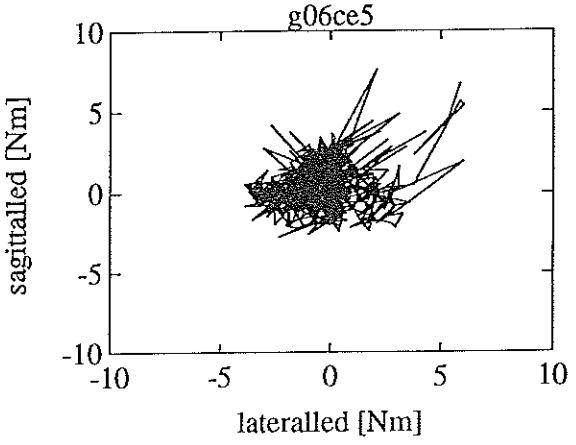
2.2

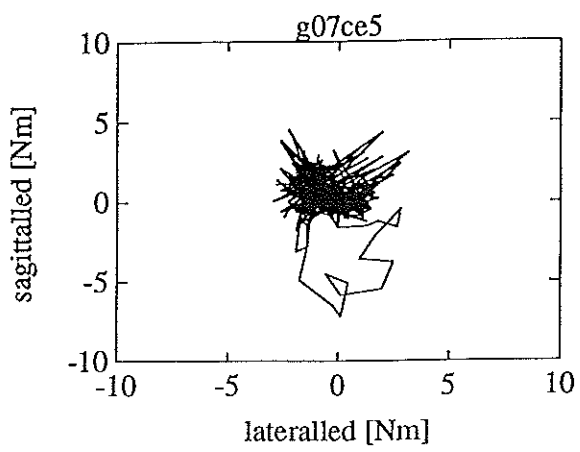
Följande 12 bilder visar utsignalen från kraftplattan vid försök med galvanisk stimulering och slutna ögon.Bilderna är ritade med lateralt moment i x-led och sagittalt moment i y-led. Bilderna visar alltså kroppens rörelse på kraftplattan.

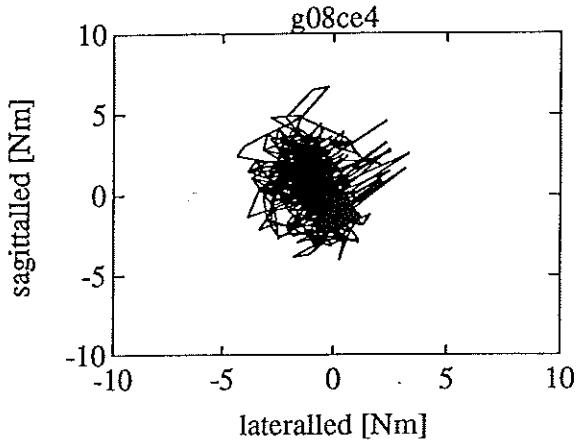


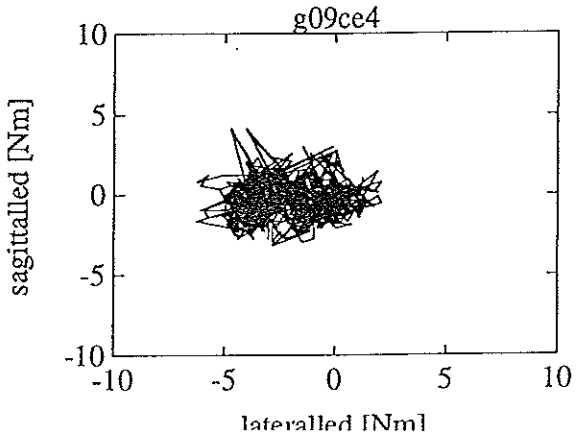


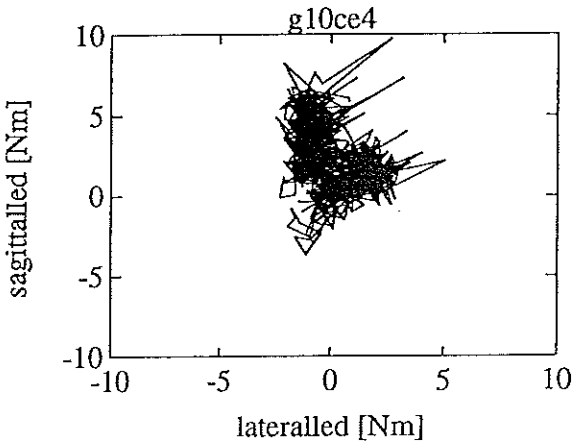


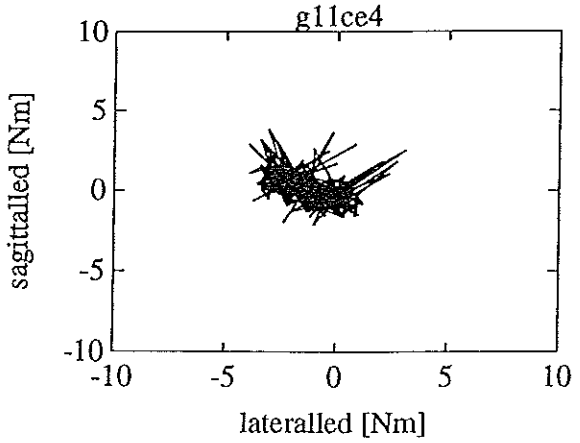


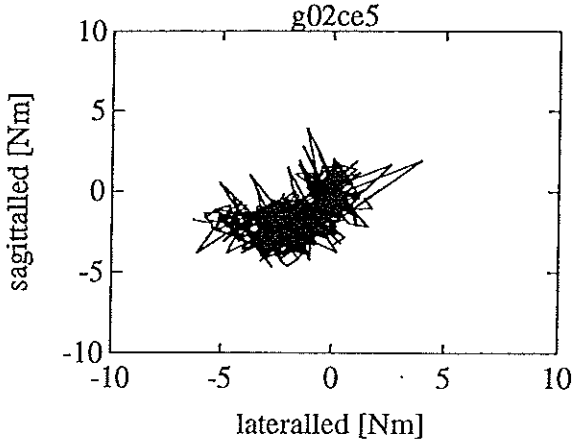

g07ce5
sagittalled [Nm]
lateralled [Nm]
10
5
0
-5
-10

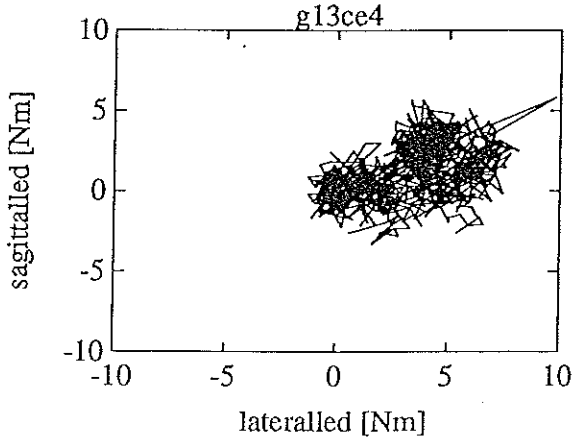

g08ce4
sagittalled [Nm]
lateralled [Nm]
10
5
0
-5
-10

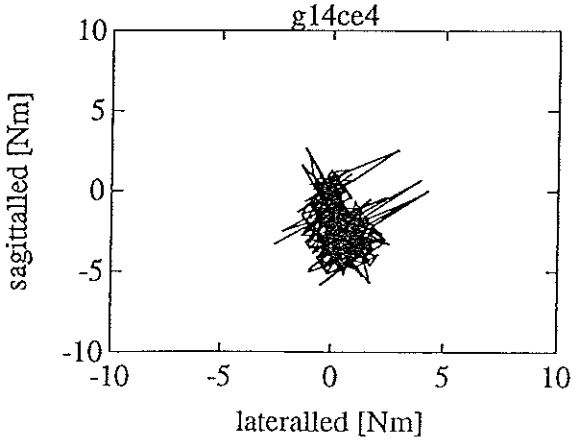

g09ce4
sagittalled [Nm]
10
5
0
-5
-10


g10ce4
sagittalled [Nm]
lateralled [Nm]
10
5
0
-5
-10


g11ce4
sagittalled [Nm]
lateralled [Nm]
10
5
0
-5
-10

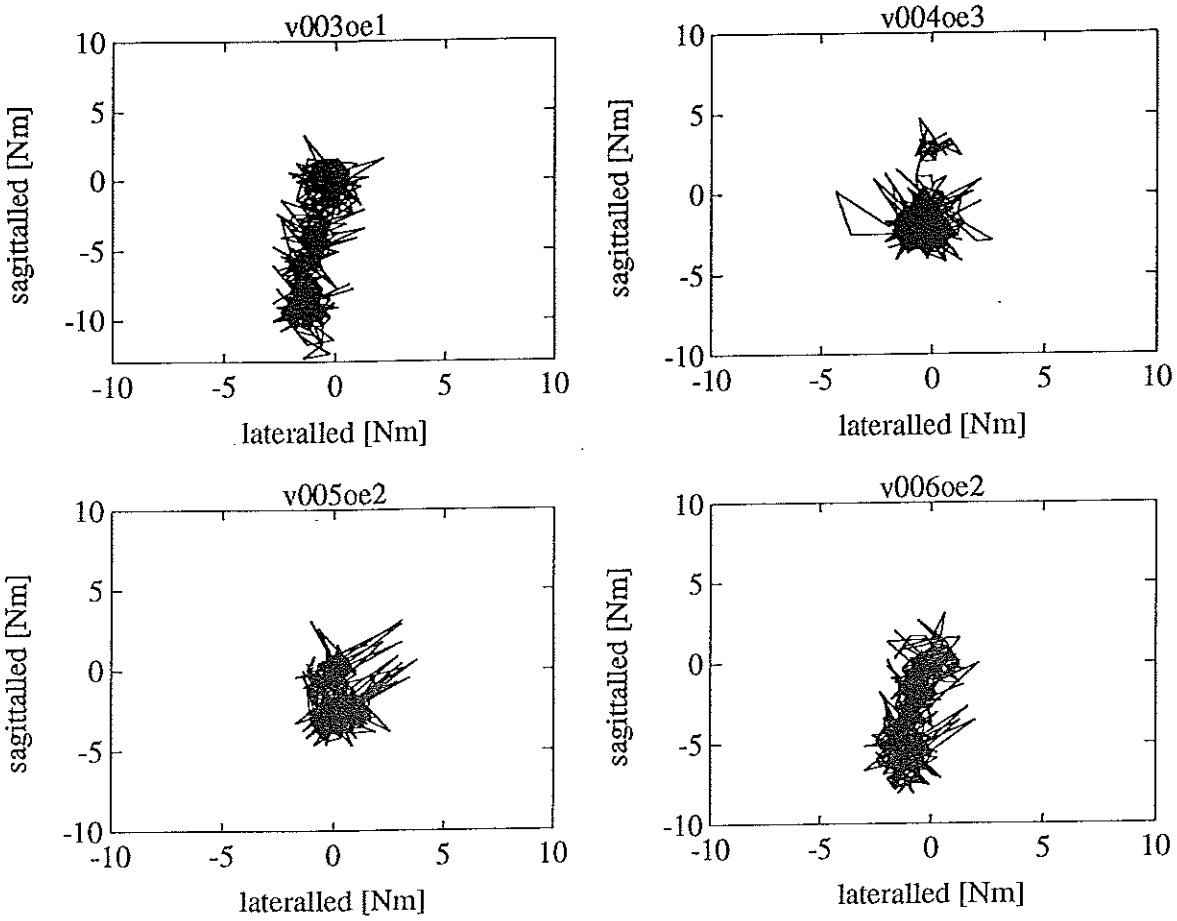

g02ce5
sagittalled [Nm]
lateralled [Nm]
10
5
0
-5
-10

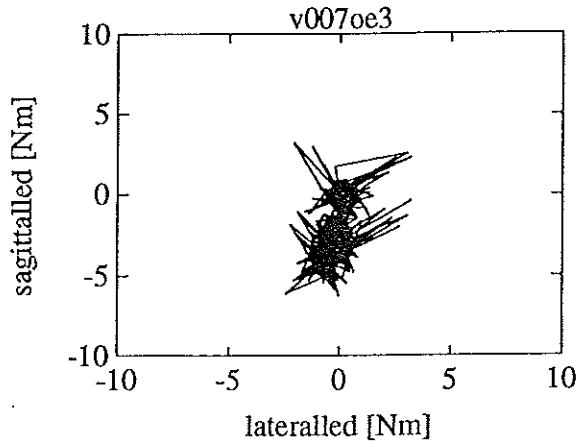

g13ce4
sagittalled [Nm]
lateralled [Nm]
10
5
0
-5
-10

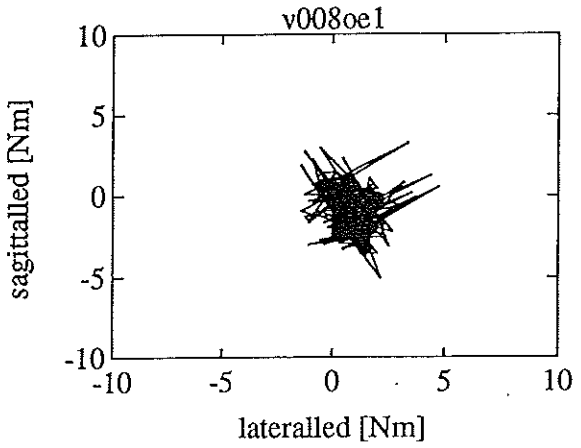

g14ce4
sagittalled [Nm]
lateralled [Nm]
10
5
0
-5
-10

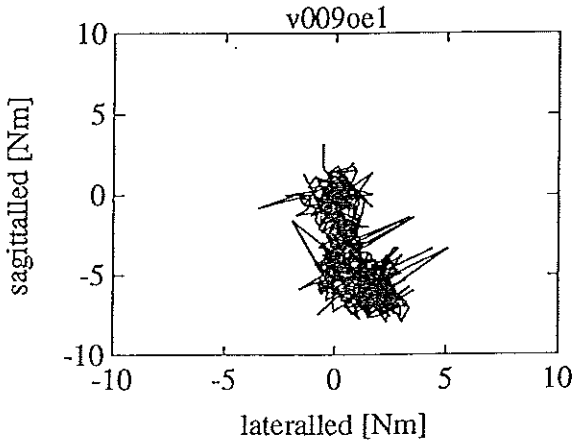
2.3 Mätdata vid vibrationsstimulering

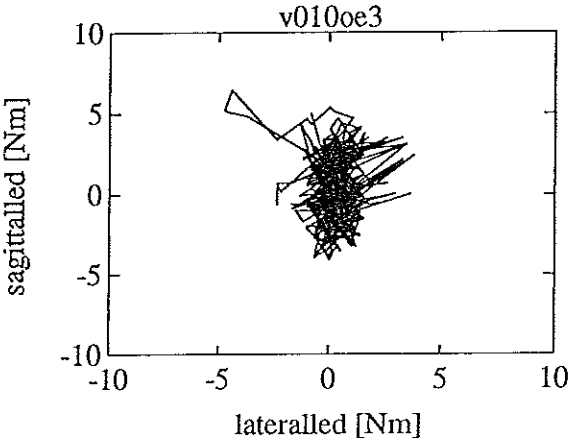
Följande 12 bilder visar utsignalen från kraftplattan vid försök med vibrationsstimulering och öppna ögon. Bilderna är ritade med lateralt moment i x-led och sagittalt moment i y-led. Bilderna visar alltså kroppens rörelse på kraftplattan.

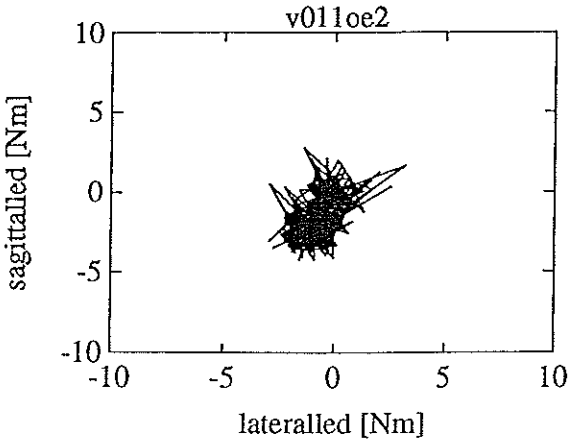


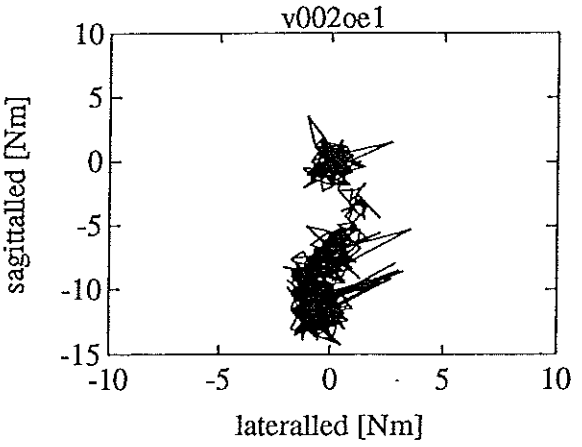

v007oe3
sagittalled [Nm]
lateralled [Nm]
10
5
0
-5
-10
-10
-5
0
5
10

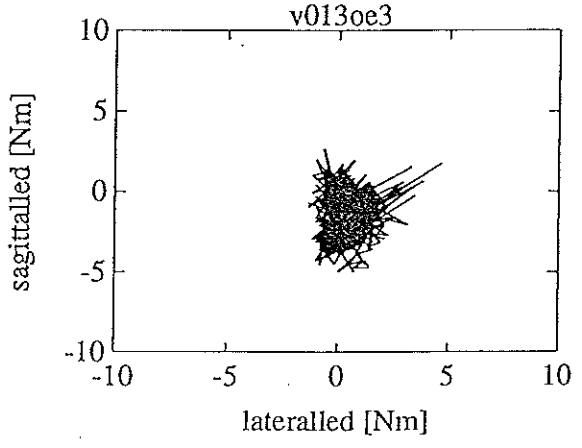

v008oe1
sagittalled [Nm]
lateralled [Nm]
10
5
0
-5
-10
-10
-5
0
5
10

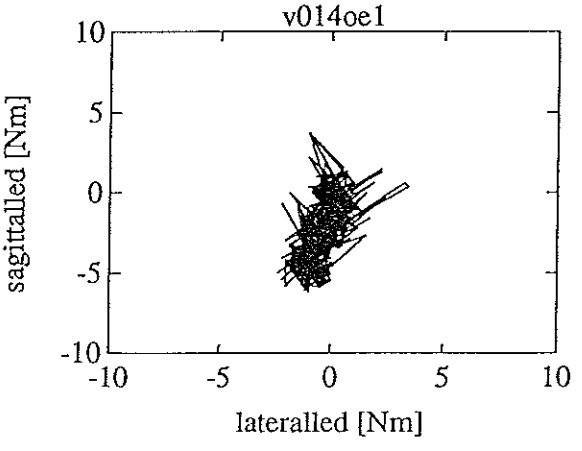

v009oe1
sagittalled [Nm]
lateralled [Nm]
10
5
0
-5
-10
-10
-5
0
5
10


v010oe3
sagittalled [Nm]
lateralled [Nm]
10
5
0
-5
-10
-10
-5
0
5
10


v011oe2
sagittalled [Nm]
lateralled [Nm]
10
5
0
-5
-10
-10
-5
0
5
10

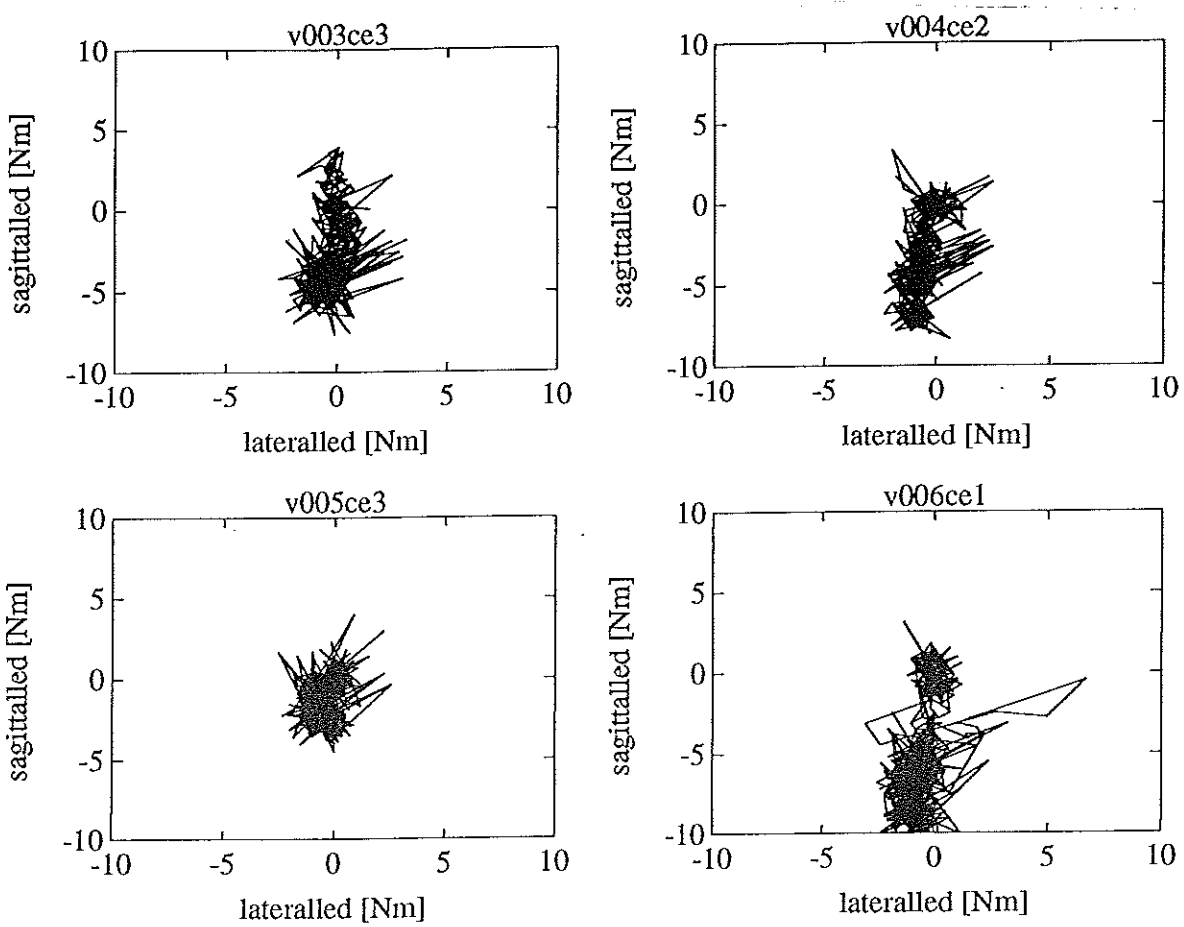

v002oe1
sagittalled [Nm]
lateralled [Nm]
10
5
0
-5
-10
-15
-10
-5
0
5
10

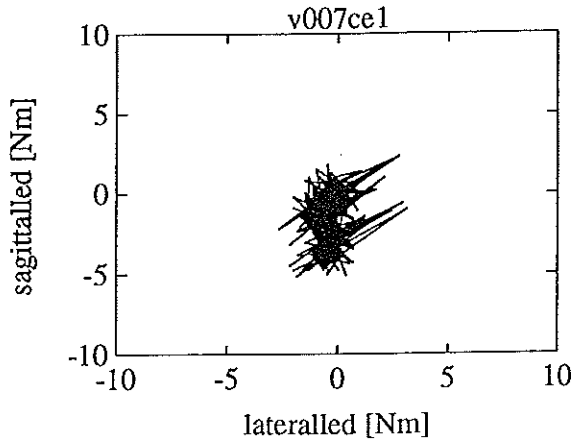

v013oe3
sagittalled [Nm]
lateralled [Nm]
10
5
0
-5
-10
-10
-5
0
5
10

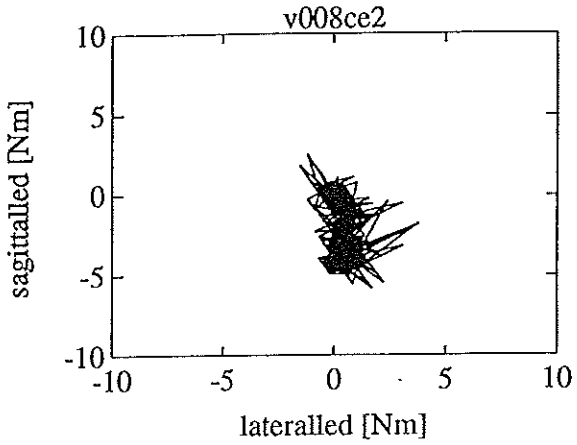

v014oe1
sagittalled [Nm]
lateralled [Nm]
10
5
0
-5
-10
-10
-5
0
5
10

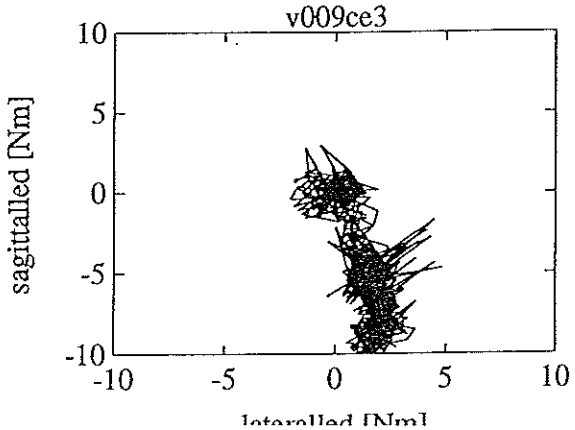
2.4

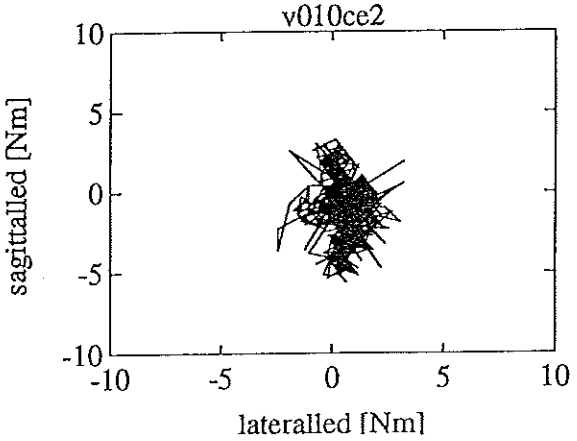
Följande 12 bilder visar utsignalen från kraftplattan vid försök med vibrationsstimulering och slutna ögon.Bilderna är ritade med lateralt moment i x-led och sagittalt moment i y-led. Bilderna visar alltså kroppens rörelse på kraftplattan.

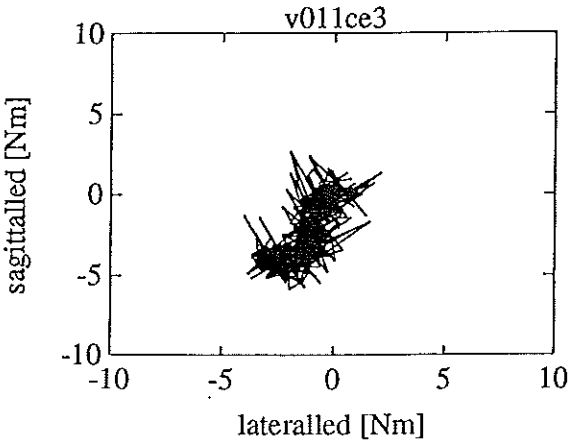


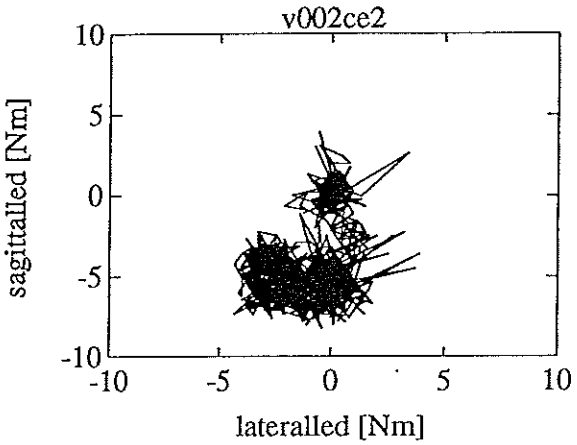

v007ce1
sagittalled [Nm]
lateralled [Nm]
10
5
0
-5
-10
-10
-5
0
5
10

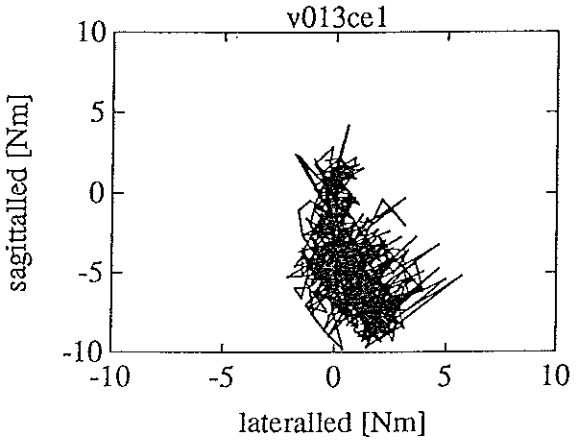

v008ce2
sagittalled [Nm]
lateralled [Nm]
10
5
0
-5
-10
-10
-5
0
5
10

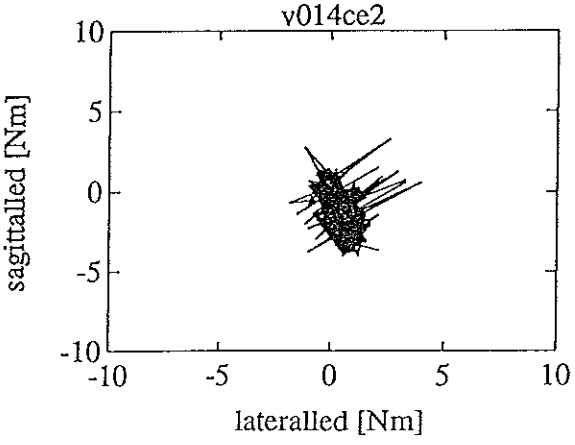

v009ce3
sagittalled [Nm]
10
5
0
-5
-10
-10
-5
0
5
10


v010ce2
sagittalled [Nm]
lateralled [Nm]
10
5
0
-5
-10
-10
-5
0
5
10


v011ce3
sagittalled [Nm]
lateralled [Nm]
10
5
0
-5
-10
-10
-5
0
5
10

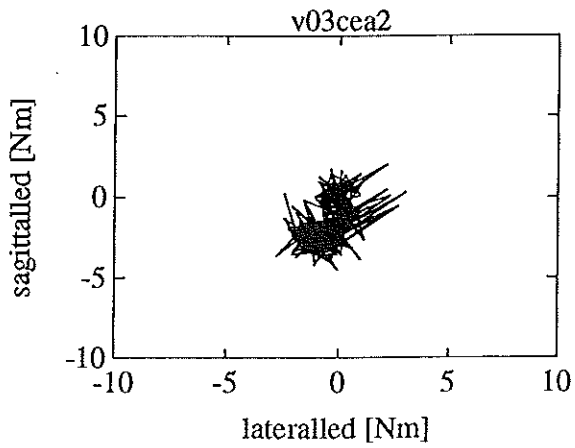

v002ce2
sagittalled [Nm]
lateralled [Nm]
10
5
0
-5
-10
-10
-5
0
5
10

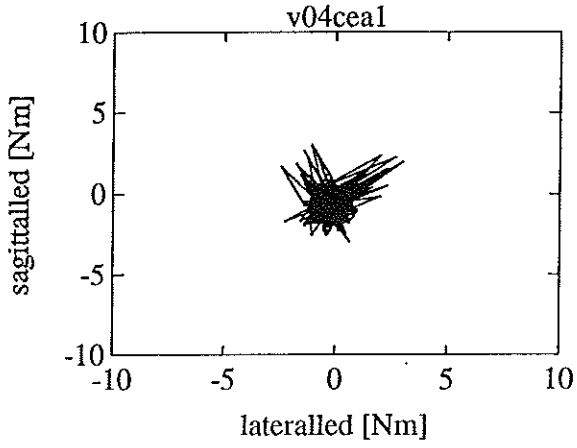

v013ce1
sagittalled [Nm]
lateralled [Nm]
10
5
0
-5
-10
-10
-5
0
5
10

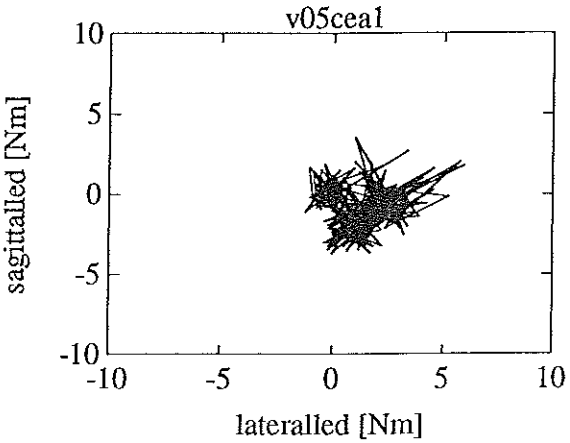

v014ce2
sagittalled [Nm]
lateralled [Nm]
10
5
0
-5
-10
-10
-5
0
5
10

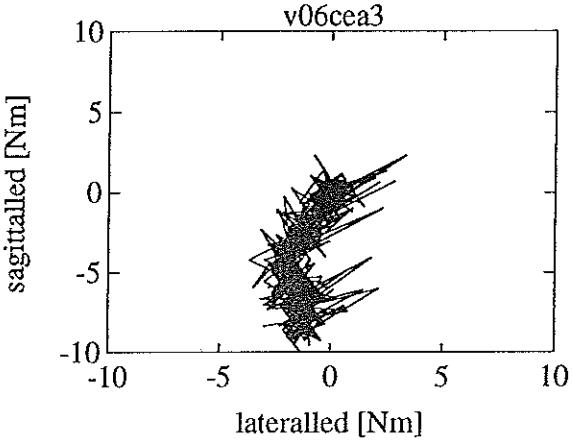
2.5

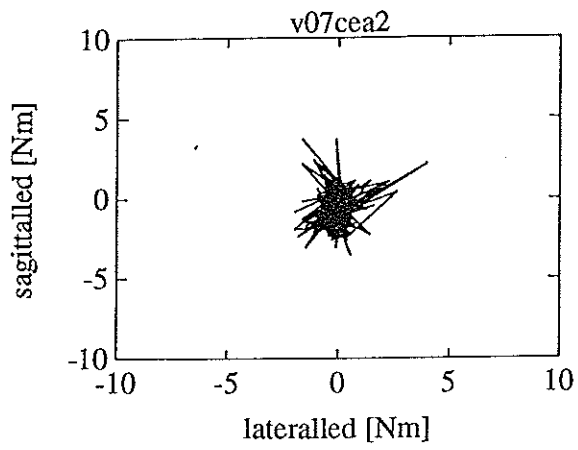
Följande 12 bilder visar utsignalen från kraftplattan, då armstöden utnyttjas, vid försök med vibrationsstimulering och slutna ögon. Bilderna är ritade med lateralt moment i x-led och sagittalt moment i y-led. Bilderna visar alltså kroppens rörelse på kraftplattan.

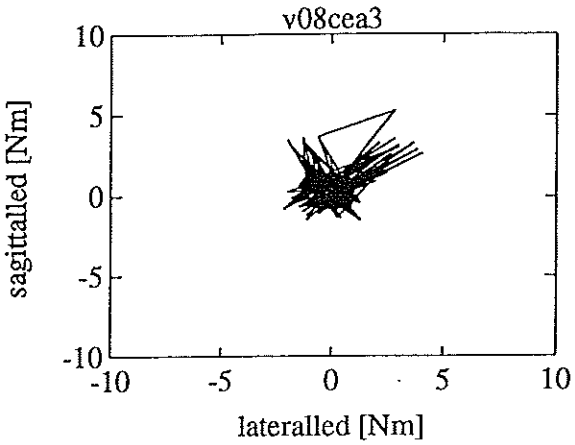


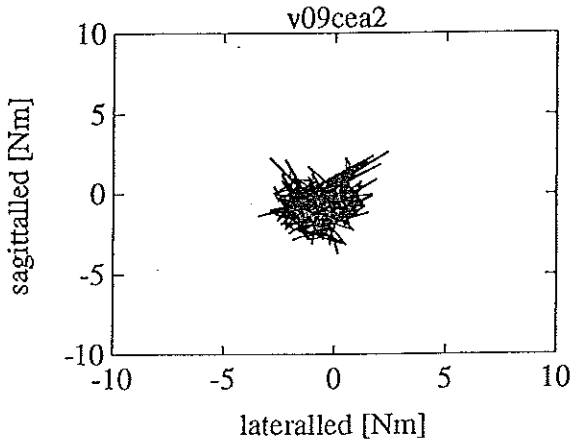


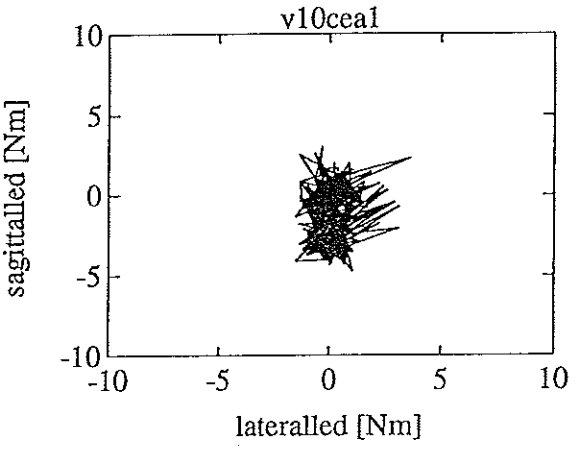


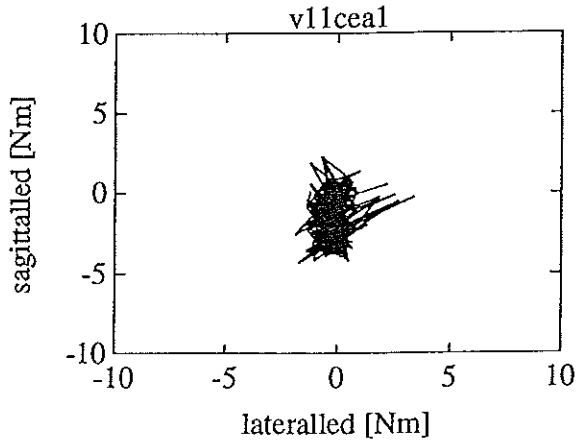


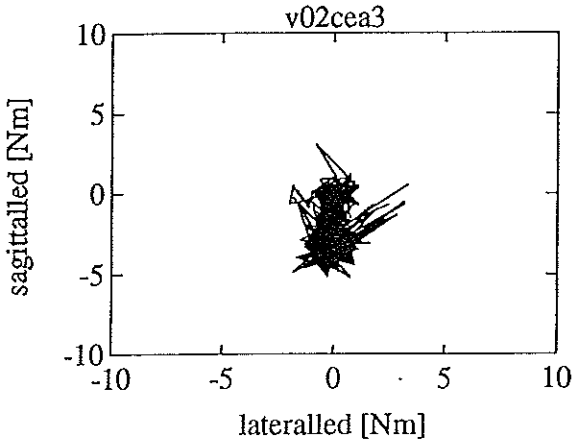

v07cea2
sagittalled [Nm]
lateralled [Nm]
10
5
0
-5
-10

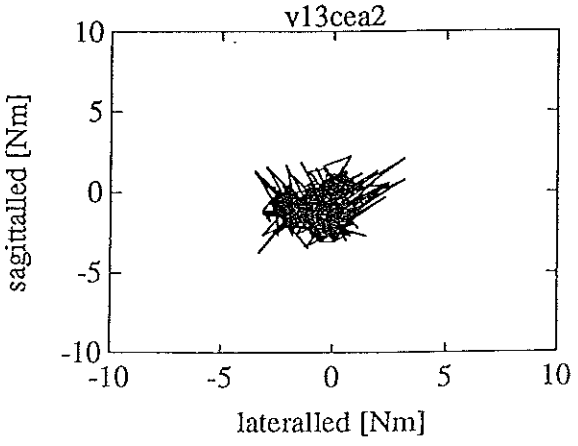

v08cea3
sagittalled [Nm]
lateralled [Nm]
10
5
0
-5
-10

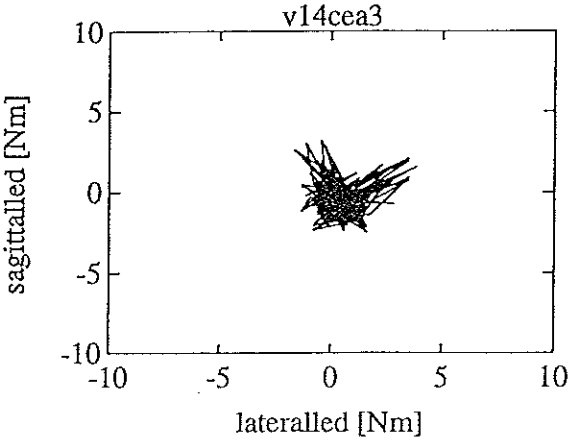

v09cea2
sagittalled [Nm]
lateralled [Nm]
10
5
0
-5
-10

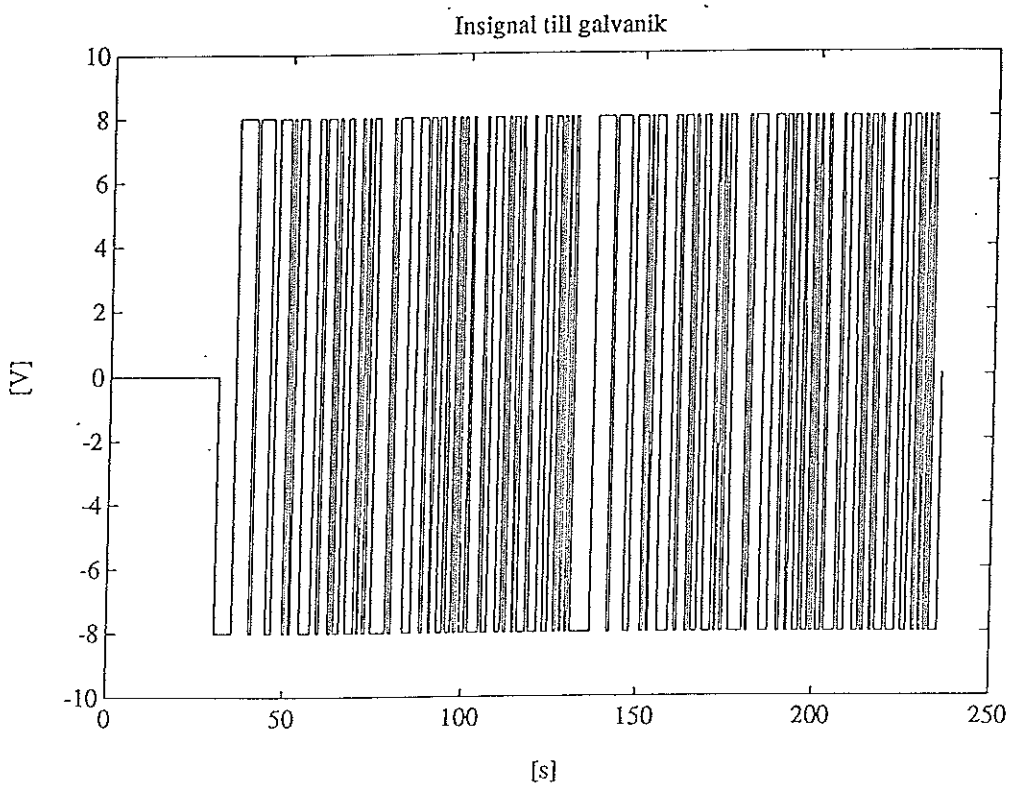

v10cea1
sagittalled [Nm]
lateralled [Nm]
10
5
0
-5
-10

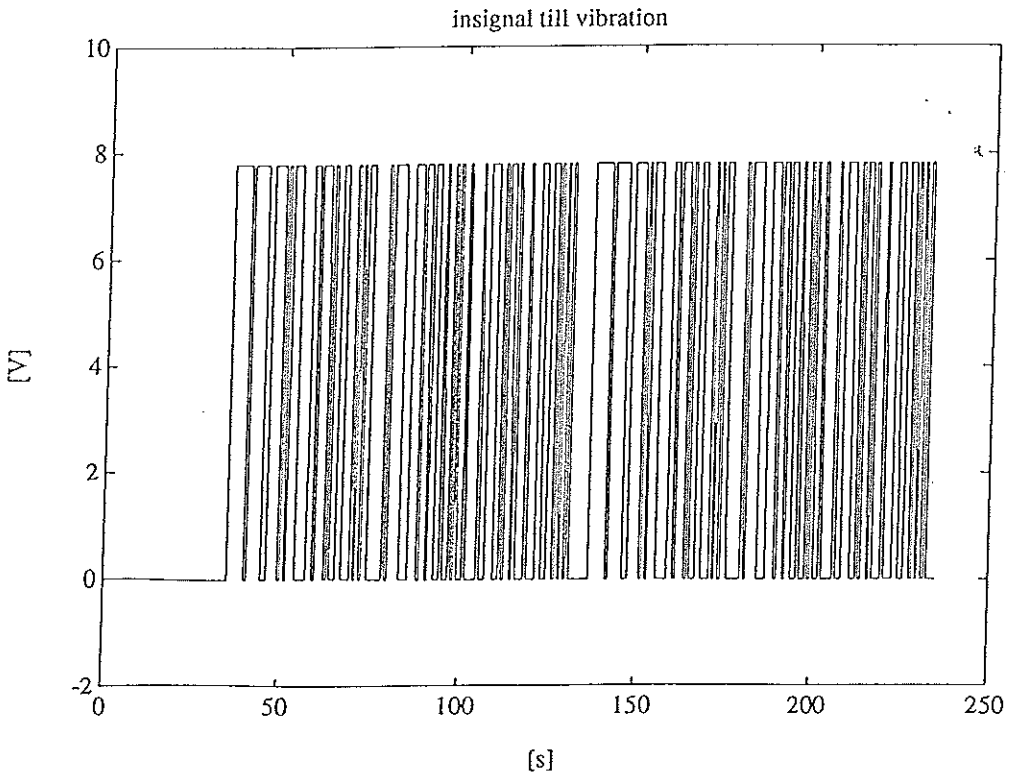

v11cea1
sagittalled [Nm]
lateralled [Nm]
10
5
0
-5
-10


v02cea3
sagittalled [Nm]
lateralled [Nm]
10
5
0
-5
-10


v13cea2
sagittalled [Nm]
lateralled [Nm]
10
5
0
-5
-10


v14cea3
sagittalled [Nm]
lateralled [Nm]
10
5
0
-5
-10


Insignal till galvanik
10
8
6
4
2
0
-2
-4
-6
-8
-10
[V]
0
50
100
150
200
250
[s]


insignal till vibration
10
8
6
4
2
0
-2
[V]
0
50
100
150
200
250
[s]