Trio program beskrivelse/forklaring

ZPOWERUP.bas

```
...
ETHERNET(2,0,7,1)
ETHERNET(2,0,9,1)
…
```

Disse to linjer er til opsætning af kommunikation over ethernet via Modbus/TCP.

1. Første linje definerer hvilken datatype der skal bruges i kommunikationen, 2 og 7 er faste argumenter ved denne definition, 0 er hvilket slot i TRIO'en ethernet-daughter board sidder på og 1 betyder at der er valgt 32bit signed floating point (havde der været 0 i stedet for 1, havde der været valgt 16bit signed integer, dette er også standard opsætning hvis der ikke er angivet andet).
2. Anden linje definere hvor igennem kommunikationen skal foregå, ligesom første linje er der faste argumenter (2 og 9) og TRIO slot nummer. Sidste tal (1) definerer at kommunikation skal foregå gennem TABLE (TABLE(0) til TABLE(16384)), havde det sidste været 0 ville kommunikationen have foregået gennem VR (VR(0) til VR(1023)), hvilket også er standard.

```
...
CONSTANT "axis_a_enabled",1
CONSTANT "axis_b_enabled",1
CONSTANT "axis_c_enabled",1
CONSTANT "axis_d_enabled",1
CONSTANT "axis_e_enabled",0
CONSTANT "axis_f_enabled",0
…
```

Disse 6 linjer er brugerdefinerede konstanter der angiver om en akse er aktiveret, i projekt/robot 328 (mihada) er der 4 akser aktiveret. På en standard robot vil der være 5 akser aktiveret, men da robot 328 ikke behøver at dreje sit tool er E-aksen ikke aktiveret. Jeg ved på nuværende tidspunkt ikke hvad F-aksen er/bliver brugt til.

```
...
CONSTANT "axis_t",0
CONSTANT "axis_a",1
CONSTANT "axis_b",2
CONSTANT "axis_c",3
CONSTANT "axis_d",4
CONSTANT "axis_e",5
CONSTANT "axis_f",6
CONSTANT "axis_vt",10
…
```

Disse 8 linjer er brugerdefinerede konstanter der angiver hvilket nummer en akse har.

```
…
CONSTANT "paxis_t",0
CONSTANT "paxis_a",paxis_t+29
CONSTANT "paxis_b",paxis_a+29
CONSTANT "paxis_c",paxis_b+29
CONSTANT "paxis_d",paxis_c+29
CONSTANT "paxis_e",paxis_d+29
CONSTANT "paxis_f",paxis_e+29
…
```

Disse 7 linjer er brugerdefinerede konstater der fungerer som pointerer til akserne (nok i enten TABLE eller VR), jeg ved ikke hvad disse bliver brugt til endnu.

```
…
CONSTANT "pscoperetur",paxis_f+29
CONSTANT "pscopeparam",pscoperetur+1
CONSTANT "pscopecommand",pscopeparam+20
CONSTANT "pscopetype",pscopecommand+1
CONSTANT "pscopelength",pscopetype+1
CONSTANT "pscopenumofsam",pscopelength+1
CONSTANT "pscopeascale",pscopenumofsam+1
CONSTANT "pscopebscale",pscopeascale+1
CONSTANT "pscopecscale",pscopebscale+1
CONSTANT "pscopedscale",pscopecscale+1
CONSTANT "pscopeescale",pscopedscale+1
CONSTANT "pscopefscale",pscopeescale+1
CONSTANT "pscopebuffer",pscopefscale+1
CONSTANT "pctlretur",pscopebuffer+20
CONSTANT "pctlcommand",pctlretur+11
CONSTANT "pmpnretur",pctlcommand+1
CONSTANT "pmpncommand",pmpnretur+11
CONSTANT "pmpnbasicerror",pmpncommand+1
CONSTANT "pmpnlinenumber",pmpnbasicerror+1
CONSTANT "pmpncancel",pmpnlinenumber+1
CONSTANT "pmpnrunning",pmpncancel+1
CONSTANT "pmpnstate",pmpnrunning+1
CONSTANT "pmpnspeedfactor",pmpnstate+1
CONSTANT "pmpntrio",pmpnspeedfactor+1
CONSTANT "pmpnlift",pmpntrio+50
CONSTANT "pmpnlast",pmpnlift+50
…
```

Disse linjer er brugerdefinerede konstanter der fungerer som pointerer til variabler. Pointerer der starter med *pscope* bliver brugt i MPNScope, jeg ved dog ikke hvordan disse bliver brugt og hvordan MPNScope virker, men tror det er til grafisk repræsentation af aksepositioner under bevægelse, eventuelt brugt til optimering eller opsætning af PID-regulering på akserne.
Jeg er ikke sikker på hvad de to konstanter der starter med *pctl* bliver brugt til.
Konstanter der starter med *pmpn* bliver så vidt jeg ved brugt til kommunikation mellem pallePC og TRIOen, til valg og start af path, ændring af hastighed og lignende.

```
…
CONSTANT "uploadgroup",5000
CONSTANT "psavetab",65600 'to 65799
CONSTANT "psavedata",66000 'to 99999
CONSTANT "pexpand",100000 'to 199999
CONSTANT "pscopedata",200000 'to 249999
…
```

På nuværende tidspunkt er jeg ikke sikker på hvad nogle af disse pointerer bliver brugt til.

```
…
'*************************************************
'* VR 800 - 899 globals                          *
'* use is depending on workcellid                *
'* HAC = slot 1, SAC = slot 2                    *
'*************************************************
'SLM asic asignment axis A
'VR(811)=2 'slot
'VR(821)=1 'asic
'SLM asic asignment axis B
'VR(812)=1 'slot
'VR(822)=0 'asic
'SLM asic asignment axis C
'VR(813)=2 'slot
'VR(823)=0 'asic
'SLM asic asignment axis D
'VR(814)=1 'slot
'VR(824)=1 'asic
'SLM asic asignment axis E
'VR(815)=1 'slot
'VR(825)=2 'asic
…
```

Disse linjer, som er udkommenteret, angiver hvor akserne er placeret på MultiAx'erne og hvilket slot på Motion Coordinator MultiAx'erne er placeret og skriver det til VR på de specificerede pladser til senere brug. Mit bedste gæt er at disse bliver kørt første gang programmet køres og derefter er det ikke længere nødvendigt at køre den hver gang, derfor er de udkommenteret.

```
…
GLOBAL "mpnhome_t",1013
GLOBAL "mpnhome_a",1014
GLOBAL "mpnhome_b",1015
GLOBAL "mpnhome_c",1016
GLOBAL "mpnhome_d",1017
GLOBAL "mpnhome_e",1018
GLOBAL "mpnhome_f",1019
GLOBAL "workcelltype",1020
GLOBAL "debug",1021
GLOBAL "workcellid",1022
GLOBAL "runningoffline",1023
```

```
…
```

De globale herover er definitioner af placering af *mpnhome_t* i VR til brug i alle programmer på TRIO'en, det vil sige at i stedet for at skrive VR(1014) så kan der skrives *mpnhome_a* i stedet for.

```
…
'*************************************************
'* LOCALS                                        *
'*************************************************
'GLOBAL "savenextpathidx",936:savenextpathidx=-1
oldspeedfactor=-1
maxspeed=0
mpnspeed=0
pathtype=-1
abspos=0
backup_val=778
debug = VR(1000)
inresetaxis=0 'show correct axis in error_routine (current_axis/dl_axis)
ON BASICERROR GOTO error_routine
…
```

Disse linjer er så vidt jeg ved initialisering af lokale variabler, så de ikke står med gamle eller random værdier når programmet starter. d*ebug* bliver initialiseret med værdi fra VR som er modtaget fra pallePC, sidste linje sender programmet til fejlhåndtering hvis der er fejl allerede under opstart.

```
…
'*************************************************
'* reset to default values                       *
'*************************************************
FOR i = 0 TO 15
  VR(600+i)=i
NEXT i

bvtest=0
kasseskaffet=0
boxunaligned=0

TABLE(pmpnstate,0) 'IDLE
'workcellid=401
TABLE(pmpnlift,0) 'disable=0 enable=1 will be set from pc

fe_limit_axis_a=10.0 'mm
fe_limit_axis_b=5.0 'deg
fe_limit_axis_c=5.0 'deg
fe_limit_axis_d=5.0 'deg
fe_limit_axis_e=5.0 'deg
fe_limit_axis_f=5.0 'mm
…
```

Disse linjer er nok en fortsættelse af initialisering af lokale variabler og sørge for at værdier i VR

på plads 600-615 er sat til standard værdier. Statussen for statemachines bliver sat til *IDLE* og lift bliver sat som ikke slået til, begge værdier vil blive ændret af pallePC. Til sidst bliver maksimalt tilladte følgefejl initialiseret.

```
…
GOSUB inittablevalues
GOSUB set_units_and_defaults
…
```

Her bliver der sprunget til subrutinerne *inittablevalues* og *set_units_and_defaults*, efter der er sprunget til første subrutine returnerer programmet og fortsætter med at springe til næste subrutine (var *GOTO* blevet brugt i stedet for ville programmet ikke have returneret).

```
…
IF debug>0 THEN
testloop:
'WAIT UNTIL IN(1)=ON
'WAIT UNTIL IN(1)=OFF
'TICKS=5000
'WAIT UNTIL TICKS<0
TRON
IF runningoffline=0 THEN GOSUB mpnstartup
'GOSUB set_units_and_defaults
GOSUB resetaxis
WDOG=ON
TRON
homeloop:
GOSUB set_mpn_home_all
IF slm_is_ok=0 THEN
  IF runningoffline=0 THEN GOSUB set_defpos
ENDIF
TRON
'GOSUB defpos_home_d
GOTO testloop
ENDIF
…
```

Dette stykke kode bliver kun kørt hvis debug er sat til 1. *TRON* er hvad der kaldes en break line, det vil sige hvis programmet startes fra TRIO'en i debug mode vil programmet stoppe ved *TRON* og skulle have input fra brugeren for at fortsætte programmet (dette gøre for at kunne tjekke diverse ting fx at værdier er sat rigtigt og er opdateret i VR og TABLE).

```
…
'RUN "ZMPNSCOPE",13
RUN "ZCTLCOMMAND",1
…
```

Her startes programmet op der ligger i filen *ZCTLCOMMAND.bas* med prioritet 1. 1 er en lav prioritet, de højeste prioriteter er 13 og 14, f.eks kører *ZPOWERUP.bas* med prioritet 14 og som

det kan ses i den udkommenterede linje ville *ZMPNSCOPE.bas* kører med prioritet 13 hvis den blev startet. *ZCTLCOMMAND* er det program der ligger på TRIO'en som kommunikerer med pallePC'en, TRIO'en får gennem *ZCTLCOMMAND* besked på hvad den skal gøre.

mainloop

```
…
mainloop:
   TABLE(pmpnstate,3) 'wating for mpncommand
   IF TABLE(pmpncommand)>1 THEN
      TABLE(pmpncancel,0)
      TABLE(pmpnretur,0)
…
   ENDIF
   GOTO mainloop
STOP
…
```

Det første stykke kode i *mainloop* sætter fortæller statemachines i pallePC at TRIO'en venter på kommando fra *Robostacker* softwaren. Den første if-sætning kontrollerer om *mpncommand* i TABLE er større end 1, hvis den ikke er startes *mainloop* forfra, hvis *mpncommand* er større end 1 gøres der forskellige ting alt efter hvad værdi *mpncommand* har:
2 – *mpnstartup*
3 – *set_units_and_defaults*
4 – *set_defpos*
5 – *resetaxis*
6 – *movehome and return new mpnhome*
7 – *PATHEXECUTER*
8 – *set_mpn_home_all* (+ test if power to slm has been lost)
9 – *set_mpn_auto_home/set_defpos*

```
…
'*********************************************************************************
'*     2 - mpnstartup
'*********************************************************************************
      IF TABLE(pmpncommand)=2 THEN
         IF runningoffline=0 THEN
            TABLE(pmpnstate,40) 'mpnstartup
            GOSUB mpnstartup
         ENDIF
         TABLE(pmpncommand,1)
         GOTO mainloop
      ENDIF
…
```

Herfra gås der til *mpnstartup* subrutinen hvis maskinen er online, efter den subrutine har kørt sættes *pmpncommand* til 1 og *mainloop* startes forfra og venter på en ny kommando.

```
…
'*********************************************************************************
'*     3 - set_units_and_defaults
'*********************************************************************************
      IF TABLE(pmpncommand)=3 THEN
```

```
        TABLE(pmpnstate,50)
        GOSUB set_units_and_defaults
        TABLE(pmpncommand,1)
        GOTO mainloop
    ENDIF
…
```

Her sættes *mpnstate* og der springes til subrutinen *set_units_and_defaults* hvor opsætning af akserne foregår og til sidst startes *mainloop* forfra. For mere detaljeret beskrivelse af subrutinen, se kodebeskrivelse længere nede i dokumentet.

```
…
'*************************************************************************
'*     4 - set_defpos
'*************************************************************************
    IF TABLE(pmpncommand)=4 THEN
        TABLE(pmpnstate,60) 'set_defpos
        IF runningoffline=0 THEN GOSUB set_defpos
        TABLE(pmpncommand,1)
        GOTO mainloop
    ENDIF
…
```

Her sættes *mpnstate* og der springes til subrutinen *set_defpos*, jeg ved endnu ikke hvad denne subrutine gør, det vil blive beskrevet senere i dokumentet.

```
…
'*************************************************************************
'*     5 - resetaxis
'*************************************************************************
    IF TABLE(pmpncommand)=5 THEN
        TABLE(pmpnstate,70) 'reset axis
        IF runningoffline=0 THEN
            GOSUB resetaxis
            WDOG=ON
        ENDIF
        TABLE(pmpncommand,1)

        GOTO mainloop
    ENDIF
…
```

Her sættes *mpnstate* og der springes til subrutinen *resetaxis*, denne subrutine resetter en akse hvis der er en fejl på. Detaljeret beskrivelse af subrutinen vil komme længere nede i dokumentet.

```
…
'*************************************************************************
'*     6 - movehome and return new mpnhome
'*************************************************************************
    IF TABLE(pmpncommand)=6 THEN
        GOSUB movehome
        TABLE(pmpnretur,VR(1013+TABLE(pmpnretur+1)))
        TABLE(pmpncommand,1)
        GOTO mainloop
    ENDIF
```

```
…
```

Her springes der til subrutinen *movehome*, jeg ved endnu ikke præcis hvad denne subrutien gør, men det vil blive beskrevet senere i dokumentet når jeg kommer til den i koden.

```
…
'*********************************************************************************
'*      7 - PATHEXECUTER
'*********************************************************************************
      IF TABLE(pmpncommand)=7 THEN
'pathexecuter:
        GOSUB mpnpath
        TABLE(pmpncommand,1)
        GOTO mainloop
      ENDIF
…
```

Her springes der til subrutinen *mpnpath* hvor bevægelse af robotarmen foregår, f.eks fra *conveyerA* til *palleA*.

```
…
'*********************************************************************************
'*      8 - set_mpn_home_all (+ test if power to slm has been lost)
'*********************************************************************************
      IF TABLE(pmpncommand)=8 THEN
         TABLE(pmpnstate,80) 'set_mpn_home_all
         IF runningoffline=0 THEN
            GOSUB set_mpn_home_all
         ENDIF

         TABLE(pmpncommand,1)
         GOTO mainloop
      ENDIF
…
```

Her sættes *mpnstate* og der springes til subrutinen *set_mpn_home_all* som sætter alle 0 positioner for akserne. Nærmere beskrivelse af subrutinen vil komme senere i dokumentet.

```
…
'*********************************************************************************
'*      9 - set_mpn_auto_home
'*********************************************************************************
      IF TABLE(pmpncommand)=9 THEN
         TABLE(pmpnstate,90) 'set_mpn_auto_home
         IF runningoffline=0 THEN
           GOSUB set_defpos
           'GOSUB set_mpn_auto_home
         ENDIF
         TABLE(pmpncommand,1)
         GOTO mainloop
      ENDIF
…
```

Her sættes *mpnstate* og der springes til subrutinen *set_defpos* som får robotten til at finde sinde home-/0-positioner på akse *B* og *C*, for derefter at gemme disse.

inittablevalues

```
…
inittablevalues:
TABLE(pscoperetur,0)
TABLE(pscopecommand,1)
TABLE(pctlretur,0)
TABLE(pctlcommand,1)
TABLE(pmpnretur,0)
TABLE(pmpncommand,1)
TABLE(pmpnspeedfactor,0.5)
TABLE(pmpnbasicerror,0)
TABLE(pmpnlinenumber,0)
TABLE(pmpncancel,0)
TABLE(pmpnrunning,5)
RETURN
…
```

I denne subrutine initialiseres værdier i TABLE, navnene i paranteserne er konstanter der bruges som pointerer og tallene i paranteserne er de værdier der bliver skrevet ind i TABLE.

mpnstartup

```
…
mpnstartup:
'Start DLink Section
'****************************************
'*** B B B B B B B B B B B B B B B B B         ***
'****************************************
IF axis_b_enabled = 1 THEN
 motor_default=FALSE
 dl_slot=VR(812):dl_asic=VR(822):dl_axis=2
 current_scaling=200.0000:filter=1000.0000
 IF debug = 1 THEN
  TRON
 ENDIF
 IF workcelltype=1 THEN
  drive_current=9.5000
  '4 deg - 29kgcm 180
  kp=0.027709:ki=2.932726:kd=0.003271
 ELSE
  IF workcelltype=2 THEN
   drive_current=15.0000
   '2 deg - 0.015kgm^2 200
   'kp=0.023386:ki=2.088949:kd=0.006495
    kp=0.027709:ki=2.932726:kd=0.003271
  ELSE
   drive_current=7.5000 'Denne plejer at være 15.0000
   '4 deg - 29kgcm 180
   kp=0.027709:ki=2.932726:kd=0.003271
   '2 deg - 200kgcm 180
   'kp=0.019603:ki=1.467906:kd=0.009243
```

```
   ENDIF
  ENDIF
  GOSUB init_dlink_axis
  'kp=0.019603:ki=1.467906:kd=0.009243
  'GOSUB set_motor_pid
ENDIF
…
```

Denne del af *mpnstartup* tager sig af opsætningen af akserne, i starten ses at der laves et check på om B-aksen er slået til, hvis den er startes opsætningen af denne. Jeg er endnu ikke sikker på hvorfor *motor_default* bliver sat til *false*, men efter det er gjort tjekkes VR for på hvilket slot motoren til B-aksen er forbundet og på hvilken kanal på den pågældende MultiAx motoren sidder (TRIO slot er i VR(812) og kanal på MultiAx er i VR(822)). *current_scaling* og *filter* bliver brugt under initialiseringen af motoren, dette sker når der med *GOSUB* bliver sprunget til *init_dlink_axis* til sidst i denne blok. Der er indsat et break point, *TRON*, der bliver stoppet ved hvis *debug* er sat til 1. Inden aksen bliver initialiseret bliver der lavet et tjek på hvilken *workcelltype* vi har at gøre med, jeg er ikke helt sikker på hvad det reelt betyder, men ud fra *workcelltype* bliver der valgt parametre som *drive_current* og PID konstanter, senere bliver den vidst også brugt til noget udregning på gearing. I dette udklip af koden kan det ses at *drive_current* bliver sat til *7.5000* hvis *workcelltype* ikke er 1 eller 2, dette er en ændring i koden jeg har haft lavet for at få en robot til at virke halvt.

```
…
'****************************************
'*** D D D D D D D D D D D D D D D D     ***
'****************************************
IF axis_d_enabled = 1 THEN
  motor_default=FALSE
  dl_slot=VR(814):dl_asic=VR(824):dl_axis=4
  current_scaling=200.0000:filter=1000.0000
  IF debug = 1 THEN
    TRON
  ENDIF
  IF workcelltype=1 THEN
    drive_current=2.5000
    kp=0.018725:ki=2.25:kd=0.002081
    'set motor parameters
  ELSE
    drive_current=2.5000
    kp=0.018725:ki=2.25:kd=0.002081
    'set motor parameters
  ENDIF
  GOSUB init_dlink_axis
ENDIF
…
```

Den eneste forskel mellem dette stykke kode og det først beskrevne for B-aksen er at der her ikke er speciel opsætning hvis *workcelltype* er 2, der er også brugt andre værdier for *drive_current* og selvfølgelig PID værdierne.

```
…
'****************************************
'*** E E E E E E E E E E E E E E E E     ***
'****************************************
```

```
IF axis_e_enabled = 1 THEN
  motor_default=FALSE
  dl_slot=VR(815):dl_asic=VR(825):dl_axis=5
  'kp=0.027709:ki=2.932726:kd=0.003271
  'current_scaling=100.0000:filter=1000.0000
  current_scaling=200.0000:filter=1000.0000
  IF debug = 1 THEN
    TRON
  ENDIF
  IF workcelltype=1 THEN
    drive_current=2.5000
    kp=0.022933:ki=2.25:kd=0.002548
  ELSE
    drive_current=2.5000
    kp=0.022933:ki=2.25:kd=0.002548
  ENDIF
  GOSUB init_dlink_axis
ENDIF
…
```

Denne ligner til forveksling opsætningen af D-aksen, dog med andre PID værdier.

```
…
'****************************************
'*** C C C C C C C C C C C C C C C C      ***
'****************************************
IF axis_c_enabled = 1 THEN
  motor_default=FALSE
  dl_slot=VR(813):dl_asic=VR(823):dl_axis=3
  current_scaling=200.0000:filter=1000.0000
  IF debug = 1 THEN
    TRON
  ENDIF
  IF workcelltype=1 THEN
    drive_current=2.5000
    kp=0.027709:ki=2.932726:kd=0.003
  ELSE
    drive_current=9.5000
    kp=0.027709:ki=2.932726:kd=0.003
  ENDIF
  GOSUB init_dlink_axis
ENDIF
…
```

Den eneste forskel mellem dette stykke kode og det først beskrevne for B-aksen er at der her ikke er speciel opsætning hvis *workcelltype* er 2 og at *drive_current* værdierne er lavere da der er en mindre motor på C-aksen.

```
…
'****************************************
'*** A A A A A A A A A A A A A A A A      ***
'****************************************
IF axis_a_enabled = 1 THEN
  motor_default=FALSE
```

```
  dl_slot=VR(811):dl_asic=VR(821):dl_axis=1
  current_scaling=200.0000:filter=1000.0000
  IF debug = 1 THEN
    TRON
  ENDIF
  IF workcelltype=1 THEN
    drive_current=9.5000
    kp=0.016556:ki=1.04703:kd=0.009163
  ELSE
    drive_current=9.5000
    kp=0.016556:ki=1.04703:kd=0.009163
  ENDIF
  GOSUB init_dlink_axis
ENDIF
…
```

Ud over andre værdier ligner denne til forveksling opsætningen af de andre akser.

```
…
'****************************************
'*** F F F F F F F F F F F F F F F F     ***
'****************************************
IF axis_f_enabled = 1 THEN
  motor_default=FALSE
  dl_slot=VR(816):dl_asic=VR(826):dl_axis=6
  current_scaling=200.0000:filter=1000.0000
  IF debug = 1 THEN
    TRON
  ENDIF
  IF workcelltype=1 THEN
    drive_current=2.5000
    kp=0.022933:ki=2.25:kd=0.002548
  ELSE
    drive_current=2.5000
    kp=0.022933:ki=2.25:kd=0.002548
  ENDIF
  GOSUB init_dlink_axis
ENDIF
RETURN
'slut mpnstartup
…
```

Ud over andre værdier ligner denne til forveksling opsætningen af de andre akser, jeg er dog stadig ikke sikker på hvad F-aksen bliver/er blevet brugt til.

init_dlink_axis

Den næste lange subrutine bliver kørt for hver akse, når en akse er kommet gennem sin opsætning bliver forbindelse og kommunikation til SLM board, MultiAx og motor oprettet, testet og sat op så robotten er klar til at bevæge sig. Så vidt jeg kan se er dele af denne subrutine genereret eller kopieret fra noget standard kode udarbejdet af TRIO/Control Techniques.

```
…
```

```
'************************************************************
'Initialisation Subroutine for SLM Technology Daughter boards
 dl_version = 2.0
'For use with System Software 1.52 and above
'Trio Motion Technology - 23-Jul-2003
'(based on dl_version 1.8 - Control Techniques - 28 Sept 2001)
'************************************************************
init_dlink_axis:
start_attempt = 0
dlink_status = 0
TABLE(pmpnstate,40+dl_axis) 'init axis X, 41 = A, 42 = B etc.
retry:
' PRINT "initialising Axis ";dl_axis [0]
' check if first initialisation attempt for this axis
 IF ATYPE AXIS(dl_axis)<>11 THEN
  ' check that the SLM module is detected
  IF DLINK(2,dl_slot,dl_asic)=0 THEN
   dlink_status=2
   GOTO error_handling
  ENDIF
  WA(5)

  ' check that the drive is detected
  IF DLINK(3,dl_slot,dl_asic)=0 THEN
   dlink_status=3
   GOTO error_handling
  ENDIF
  ' initialise the axis
  DLINK(4,dl_slot,dl_asic,dl_axis)
'  PRINT "axis:";dl_axis[0];" assigned"
 ENDIF
 WA(10)
…
```

Det første der sker i *init_dlink_axis* er at variabler for status og antal forsøg sættes til nul, herefter opdateres *TABLE*-værdien *pmpnstate* så den afspejler hvilken akse der er ved at blive initialiseret (dette bruges i Robostacker software til at give information til brugeren). Det første tjek der bliver lavet er et tjek af *ATYPE* for den aktuelle akse, hvis denne er ligmed 11 er aksen allerede forsøgt initialiseret en gang. Hvis *ATYPE* er forskellig fra 11 bliver der efterfølgende lavet tjek på om SLM modulet og motoren er forbundet, hvis de er kobles aksen sammen med motoren på den pågældende kanal på MultiAx, som er forbundet til det pågældende SLM board i *TRIO'en*. Hvis der er et tjek der fejler springes der til *error_handling* subrutinen.

```
…
' reset the amplifier in case of trip
 tries = 0
 reset_x:
  IF MOTION_ERROR THEN DATUM(0)
  sum=0
  FOR check=1 TO 10
   DLINK(7,dl_axis,252)
   sum=sum+DLINK(7,dl_axis,251)
  NEXT check
```

```
 IF sum<>-10 THEN
   tries = tries+1
   IF tries > 5 THEN
'     PRINT "unable to reset drive axis ", dl_axis
TRON
      RETURN
'     STOP
   ELSE
     WA(10)
     GOTO reset_x
   ENDIF
 ENDIF

 DLINK(10,dl_axis,22)' Dummy read encoder eprom
…
```

Her startes med med at se om *MOTION_ERROR* er forskellig fra 0, hvis *MOTION_ERROR* er forskellig fra 0 eksekveres *DATUM(0)* som sætter forlangte positioner til aktuelle positioner for alle akserne, når dette er gjort kan en akse med fejl på resettes. I den følgende *for*-løkke køres der nogle SLM kommandoer (jeg ved ikke hvilke, da jeg ikke kan finde dokumentation for hvad 251 og 252 betyder) i forbindelse med at resette en akse. Hvis der er brugt mere end 5 forsøg på at resette aksen, uden at dette er sket afbrydes forsøget på at resette. Er der forsøgt at restte mindre end 5 gange, ventes der 10 millisekunder ved *WA(10)* før der prøves igen. Den sidste linje i dette stykke kode er som der står i kommentaren et dummy read fra EPROM, Jeg er ikke helt sikker på hvorfor det gøres.

```
…
'*******************************************************
' ADDED BY THN 02.11.2003
 'skip initialization if 24V backup maintained
 IF debug = 1 THEN
   TRON
 ENDIF
 host_flag = DLINK(5,dl_axis,62)
 IF host_flag = backup_val THEN
   GOTO enable_axis
 ENDIF
…
```

Dette stykke kode tjekker om *backup_val* stadig er gemt i motoren, hvis den er kan der springes direkte til *enable_axis*.

```
…
'*******************************************************
 ' Perform Checksums to make sure that the data is valid
 ' Calculate the checksum for the motor object
 instances = DLINK(10,dl_axis,21)
 IF instances > 27 THEN
   dlink_status=5
   GOTO error_handling
 ENDIF
 calcheck = 0
 FOR object = 21 TO (20 + instances)
   calcheck = calcheck + DLINK(10,dl_axis,object)
```

```
  IF calcheck > 65535 THEN calcheck = calcheck - 65536
 NEXT object
 IF DLINK(10,dl_axis,20) <> calcheck THEN
  dlink_status=5
  GOTO error_handling
 ENDIF
…
```

I første linje læser programmet fra palds 21 i EPROM og gemmer det i *instances*, jeg er ikke sikker på hvad de instances er, da jeg mangler noget dokumentation. Ud fra koden kan jeg se at hvis der er mere end 27 *instances* så er der sket en fejl, hvis der er mindre end 27 bliver der udregnet en checksum over alle *instances,* herefter bliver den udregnede checksum sammenlignet med værdien der ligger på plads 20 i EPROM og hvis de ikke stemmer overens er der også en fejl og der bliver sprunget til *error_handling*.

```
…
' Check that the encoder data version is correct
 ' Motor Object
 IF DLINK(10,dl_axis,22) < 3 OR DLINK(10,dl_axis,22) > 99 THEN
  dlink_status = 4
  GOTO error_handling
 ENDIF
 'Calculate the checksum for the Encoder Object
 calcheck = 0
 FOR object = 0 TO 18
  calcheck = calcheck + DLINK(10,dl_axis,object)
  IF calcheck > 65535 THEN calcheck = calcheck - 65536
 NEXT object
 IF DLINK(10,dl_axis,19) <> calcheck THEN
  dlink_status=6
  GOTO error_handling
 ENDIF
…
```

Der bliver først lavet et tjek på plads 22 i EPROM, *?Motor objekt?,* for at være sikker på at værdien er mellem 3 og 99, de to tal inklusiv. Her efter bliver der lavet et checksums check på encoderen, værdierne der regnes checksum for ligger i EPROM på plads 0 til 18, og denne checksum bliver holdt op mod værdien på plads 19 i EPROM. Er der fejl i checksum springes til *error_handling*.

```
…
 'Calculate the checksum for the performance object
 IF motor_default = TRUE THEN
  instances = DLINK(10,dl_axis,51)
  IF instances > 40 THEN
   dlink_status=7
   GOTO error_handling
  ENDIF
  calcheck = 0
  FOR object = 51 TO (50 + instances)
   calcheck = calcheck + DLINK(10,dl_axis,object)
    IF calcheck > 65535 THEN calcheck = calcheck - 65536
   NEXT object
   IF DLINK(10,dl_axis,50) <> calcheck THEN
```

```
      dlink_status=7
      GOTO error_handling

    ENDIF
    objver = DLINK(10,dl_axis,52)
    IF (objver < 1) OR (objver > 100) THEN
      dlink_status=8
      GOTO error_handling
    ENDIF
 ENDIF
…
```

Først bliver der udregnet checksum for performance objektet, jeg er ikke sikker på hvad det er og hvad det gør, og så bliver den udregnede checksum sammenlignet med objektets værdi. Til sidst i koden bliver der lavet et tjek på om performance objektet har en accepteret version.

```
…
' Check for CT-Coder Failure
 DLINK(6,dl_axis,28,0)
 IF DLINK(5,dl_axis,28) <> 0 THEN
    dlink_status=10
    GOTO error_handling
 ENDIF

 ' Detect High Speed Motors
 IF DLINK(10,dl_axis,37) > 3700 THEN
    ss = 2
    DLINK(6,dl_axis,39,1)
 ELSE
    ss = 1
 ENDIF

 ' if using default values then read them
 IF motor_default THEN
  kp=DLINK(10,dl_axis,53)/10000
  ki=DLINK(10,dl_axis,54)/1000
  kd=DLINK(10,dl_axis,55)/10000
 ENDIF

' PRINT "calculate the parameters"
 ts=125/1000000
 digit_limit=524272
 a=(((kp/ki)+kd)*4*ss)/ts
 b=ki*32768*ts/kp
 c=ki*64*ss
 d=(kp*ss)/ts
 e=d*65535/(d+ki)
 f=digit_limit/((ki+(kp/ts))*ss)

 ' store the parameters
 DLINK(6,dl_axis,1,a)
 DLINK(6,dl_axis,2,b)
 DLINK(6,dl_axis,3,c)
```

```
DLINK(6,dl_axis,4,d)
DLINK(6,dl_axis,5,e)
DLINK(6,dl_axis,6,f)
…
```

Først bliver der tjekket om CT-Coder virker som den skal, dette gøres ved at skrive en værdi til CT-Coderen og bagefter efter tjekke om den har gemt den skrevne værdi som den skal. Herefter laves der er tjek på om motoren er high speed motor eller ej, dette gøres ved at læse motorens *rated RPM* og hvis denne er over 3700 sættes *ss* til 2 og der bliver skrevet et 1-tal til SLM modulet på adresse 39. Som det næste bliver der testet om motores standard PID-værdier skal bruges, og hvis de skal hentes de fra motoren/SLM-modulet. Som det sidste bliver der udregnet nogle akse specifikke værdier og disse bliver skrevet til det pågældende SLM-modul. Jeg er ikke helt sikker på hvad det er for nogle værdier der bliver udregnet og hvad de skal bruges til.

```
…
' p7: store the current scaling

 ' read motor current
 motor_current=DLINK(10,dl_axis,30)/10

 IF motor_current < 2.6 AND drive_current > 6.5 THEN
   dlink_status = 11
   GOTO error_handling
 ENDIF
 'THN 10.11.2004 motor has no power
 IF motor_current >= 2.6 AND drive_current <= 6.5 THEN
   dlink_status = 11
   GOTO error_handling
 ENDIF

 ' Read the recommended scaling
 IF motor_default = TRUE THEN current_scaling = DLINK(10,dl_axis,56)

 ' calculate current scale in SLM units proportional to drive current
 current_scale = (65536 * motor_current/drive_current)*(current_scaling/200)

 ' Check within range and download parameter
 IF current_scale > 65535 THEN current_scale = 65535
 DLINK(6,dl_axis,7,current_scale)
…
```

Her bliver motorstrømmen læst fra SLM-modulet (jeg ved ikke om det er aktuel strøm til motoren eller om det er datablads continuous current), herefter bliver der tjekket om der uoverensstemmelser mellem *motor_current* og *drive_current*, om de er for store eller små i forhold til hinanden. Herefter bliver der tjekket hvilken *current_scaling* der skal bruges, skal der bruges en tidligere defineret værdi eller motorens standard værdi, efter dette tjek udregnes skaleringen og efterfølgende skrives den udregnede værdi til SLM-modulet.

```
…
' p8: poles
 poles=DLINK(10,dl_axis,31)
 DLINK(6,dl_axis,8,poles/2)

 ' p9
```

```
p9=65535-(current_scale*0.25/16)
DLINK(6,dl_axis,9,p9)

' p10
p10=65535-(current_scale*0.25)
DLINK(6,dl_axis,10,p10)
…
```

Her startes med at læse fra SLM-modulet hvor mange poler motoren til den pågældende akse har. Herefter laves der to udregninger der indvolverer strømskaleringen, jeg er ikke helt sikker på hvad det er udregning af, men efter udregningen skrives de udregnede værdier til SLM-modulet.

```
…
' p11: absolute speed maximum trip threashold

DLINK(6,dl_axis,11,32700)

' p12: MultiAx Read Address
DLINK(6,dl_axis,12,32768)

' p13: speed loop filter cut off frequency
IF motor_default THEN
  filter=DLINK(10,dl_axis,57)
ENDIF
DLINK(6,dl_axis,13,2*PI*filter*ts*65535)

' p14: phase advance
DLINK(6,dl_axis,14,60000)

' p15: flux alignment position
flux=DLINK(10,dl_axis,29)
DLINK(6,dl_axis,15,flux)
…
```

I dette stykke kode skrives 5 værdier til SLM-modulet, ud fra kommentarene kan det ses hvilke værdier der bliver skrevet hvor, det eneste tjek der bliver lavet her er på om der skal bruges motorens standard værdi til beregning af cut off frequency eller tidligere defineret værdi. Ved flux allignment position, læses denne fra EPROM og skrives herefter til SLM-modulet.

```
…
' set I2t parameters

' p29: winding time constant
IF VERSION > 1.49 THEN
 p29=(DLINK(10,dl_axis,32)*100)/16
 p36=(((32*motor_current*129)/(drive_current*100))^2)/4
ELSE
 p29=(DLINK(10,dl_axis,32)*100)/8
 p36=(((32*motor_current*129)/(drive_current*100))^2)
ENDIF
DLINK(6,dl_axis,29,p29)
DLINK(6,dl_axis,36,p36)
' p37: hysteresis
p37 = p36 - ((p36*10)/100)
```

```
 DLINK(6,dl_axis,37,p37)

 ' p35: I2t backoff level (65535=peak current,32768=nom current
 DLINK(6,dl_axis,35,32768)
…
```

I dette stykke af kode bliver der udregnet noget i forbindelse med vindings tids konstanten, alt efter hvilken software version motion coordinatoren kører. Efterfølgende bliver der sat en hysterese værdi og til sidst bliver overstrøms beskyttelsen sat.

```
…
' set ctcoder parameters
 DLINK(6,dl_axis,20,DLINK(10,dl_axis,0))
 DLINK(6,dl_axis,22,DLINK(10,dl_axis,1))
 DLINK(6,dl_axis,21,DLINK(10,dl_axis,2))
 DLINK(6,dl_axis,23,DLINK(10,dl_axis,3))
 DLINK(6,dl_axis,16,DLINK(10,dl_axis,4))
 DLINK(6,dl_axis,18,DLINK(10,dl_axis,5))
 DLINK(6,dl_axis,17,DLINK(10,dl_axis,6))
 DLINK(6,dl_axis,19,DLINK(10,dl_axis,7))

 ' update the pid
 DLINK(7,dl_axis,247)

 ' update the number of poles and current scale
 DLINK(7,dl_axis,245)
 DLINK(7,dl_axis,246)
…
```

Her bliver der bare skrevet tidligere udregnede værdier til SML-modulet og til sidst bliver der kørt 3 SLM-kommandoer der opdaterer PID-reguleringen, antal motor poler og strømskalering.

```
…
enable_axis:
 ' enable the axis
 DLINK(7,dl_axis,253)

 RETURN
…
```

Her bliver SLM-kommandoen kørt der aktiverer aksen og dermed motoren. Til sidst bliver der returneret og kode afviklinge starter fra hvor *init_dlink_axis* blev kaldt.

```
…
' Error Handling Routein
 error_handling:
'    PRINT "Started Error Handling..."
   ' Each axis has a chance to retry the initialisation twice
   IF start_attempt < 2 THEN
'     PRINT "Retry to configure axis "; dl_axis
     start_attempt = start_attempt + 1
     GOTO retry
   ENDIF

'   PRINT "error initialising Axis number "; dl_axis[0]
```

```
'   PRINT "slot=";dl_slot[0];" asic=";dl_asic[0]

    IF dlink_status = 2 THEN
'     PRINT "DL.ER - No slm module detected"
    ENDIF

    IF dlink_status = 3 THEN
'     PRINT "DL.ER - No Drive Detected"
    ENDIF

    IF dlink_status = 4 THEN
'     PRINT "Motor Object ", DLINK(10,dl_axis,22)
'     PRINT "incompatible with initialisation routine version ", dl_version
    ENDIF

    IF dlink_status = 5 THEN
'     PRINT "SL.ER 2 - Motor Data Checksum Incorrect"
    ENDIF

    IF dlink_status = 6 THEN
'     PRINT "SL.ER 1 - Encoder Data Checksum Incorrect"
    ENDIF

    IF dlink_status = 7 THEN
'     PRINT "SL.ER 4 - Motor Default gains not available"
    ENDIF

    IF dlink_status = 8 THEN
'     PRINT "SL.ER 64 - Wrong version of performance object loaded"
    ENDIF

    IF dlink_status = 9 THEN
'     PRINT "SL.ER 8 - CT-Coder Failed"
    ENDIF

    IF dlink_status = 10 THEN
'     PRINT "Ctc - CT-Coder Failed"
    ENDIF

    IF dlink_status = 11 THEN
'     PRINT "Dr.Si - Motor / Drive Current Mismatch is too large"
    ENDIF

    TABLE(pmpnstate,7000+(dlink_status*10)+dl_axis)
    TICKS=5000
    WAIT UNTIL TICKS<=0
RETURN
…
```

Som det første i *error_handling* bliver der tjekket hvor mange forsøg der er gjort på at initialisere en akse, hvis det er mindre end 2 gøres der et forsøg mere. Hvis der er forsøgt at initialisere en akse 2 gange og der stadig er fejl, bliver den fundne TRIO fejl omregnet til et tal mellem 7000 og

7120, som det ses i andet argument i *TABLE* kommandoen i slutningen af subrutinen. Efter fejlen er blevet skrevet i *TABLE* ventes der 5 sekunder før der returneres til der hvor springet til *error_handling* skete. Som det kan ses i subrutinen har der været skrevet til terminalen i Motion Perfect hvad fejlen er i alle de udkommenterede linjer der starter med; *'PRINT "*. Mere præcis beskrivelse af fejlkoderne/fejlværdier er beskrevet i *TRIO fejlkodenumre i Robostacker software.odt*.

```
…
set_motor_pid:
 ' Detect High Speed Motors
 IF DLINK(10,dl_axis,37) > 3700 THEN
    ss = 2
    DLINK(6,dl_axis,39,1)
 ELSE
    ss = 1
 ENDIF
' PRINT "calculate the parameters"
 ts=125/1000000
 digit_limit=524272
 a=(((kp/ki)+kd)*4*ss)/ts
 b=ki*32768*ts/kp
 c=ki*64*ss
 d=(kp*ss)/ts
 e=d*65535/(d+ki)
 f=digit_limit/((ki+(kp/ts))*ss)
 ' store the parameters
 DLINK(6,dl_axis,1,a)
 DLINK(6,dl_axis,2,b)
 DLINK(6,dl_axis,3,c)
 DLINK(6,dl_axis,4,d)
 DLINK(6,dl_axis,5,e)
 DLINK(6,dl_axis,6,f)
 DLINK(6,dl_axis,13,2*PI*filter*ts*65535)
RETURN
…
```

Som navnet antyder så sætter *set_motor_pid* PID-værdierne for den aktuelle motor. Først laves der et tjek om det er en almindelig eller hurtig motor, alt efter hastigheden på motoren sættes *ss*, efterfølgende udregnes værdierne og til sidst skrives disse til motoren.

```
…
set_units_and_defaults:
DATUM(0)
'************************************************
'* AXIS  A                                      *
'************************************************
TABLE(pmpnstate,51)
BASE(axis_a)
UNITS=(65536*5)/(50*PI) 'tandhjul dele-diameter 50mm 5:1 gearing
P_GAIN=0.1
I_GAIN=0.0
D_GAIN=0.15
'D_GAIN=0.0
OV_GAIN=0.0
```

```
VFF_GAIN=8.0
AFF_GAIN=0.0
SRAMP=5.0
SPEED=40 ' Saet hastighed
ACCEL=40 ' Saet acceleration
DECEL=400 ' Saet deceleration
CREEP=0.5
JOGSPEED=1.0
FE_LIMIT=10
DAC=0.0
REP_DIST=999999.0
FWD_IN=-1
REV_IN=-1
DATUM_IN=4
FHOLD_IN=-1
FS_LIMIT=9999.0
RS_LIMIT=-9999.0
…
```

Denne subrutine, *set_units_and_defaults*, laver opsætningen for alle akserne. Som det første køres *DATUM(0)* som resetter fejl på alle akserne. Først sættes *pmpnstate* til 51, hvilket fortæller at *set_units_and_defaults* er i gang med A aksen. Herefter sættes *BASE* til A aksen og herefter kan standard værdier sættes med deres kommandoer.

```
…
'************************************************
'* AXIS  B                                      *
'************************************************
TABLE(pmpnstate,52)
BASE(axis_b)
IF workcelltype=1 THEN
  UNITS=65536*105/360
  P_GAIN=0.10
  I_GAIN=0.0
  D_GAIN=0.15
ELSE
  IF workcelltype=2 THEN
    UNITS=65536*191/360 '
    P_GAIN=0.20
    I_GAIN=0.0
    D_GAIN=0.15
  ELSE
    UNITS=65536*101/360 '
    P_GAIN=0.10
    I_GAIN=0.0
    D_GAIN=0.15
  ENDIF
ENDIF
VFF_GAIN=8.0
AFF_GAIN=0.0
SRAMP = 5.0
SPEED=4 ' Saet hastighed
ACCEL=8 ' Saet acceleration
```

```
DECEL=8 ' Saet deceleration
CREEP=0.1
JOGSPEED=1.0
FE_LIMIT=fe_limit_axis_b
DAC=0.0
REP_DIST=999999.0
FWD_IN=-1
REV_IN=-1
DATUM_IN=5
FHOLD_IN=-1
FS_LIMIT=128.0
RS_LIMIT=-128.0
…
```

Først sættes *pmpnstate* til 52, hvilket fortæller at *set_units_and_defaults* er i gang med B aksen. Der laves en test på hvilken type workcell det er, og *UNITS* sættes herefter på baggrund af forskellige gearinger (om det reelt bliver brugt ved jeg ikke, for det bliver defineret i ID.robot filen). Herefter sættes *BASE* til B aksen og herefter kan standard værdier sættes med deres kommandoer.

```
…
'*************************************************
'* AXIS  C                                       *
'*************************************************
TABLE(pmpnstate,53)
BASE(axis_c)
IF workcelltype=1 THEN
  UNITS=65536*105/360
  P_GAIN=0.12
  I_GAIN=0.0
  D_GAIN=0.15
ELSE
  IF workcelltype=2 THEN
    UNITS=65536*169/360 '
    P_GAIN=0.12
    I_GAIN=0.0
    D_GAIN=0.15
  ELSE
    UNITS=65536*105/360 '
    P_GAIN=0.12
    I_GAIN=0.0
    D_GAIN=0.15
  ENDIF
ENDIF
OV_GAIN=0.0
VFF_GAIN=8.0
AFF_GAIN=0.0
SRAMP = 5.0
SPEED=4 ' Saet hastighed
ACCEL=8 ' Saet acceleration
DECEL=8 ' Saet deceleration
CREEP=0.1
JOGSPEED=1.0
```

```
FE_LIMIT=5
DAC=0.0
REP_DIST=999999.0
FWD_IN=-1
REV_IN=-1
DATUM_IN=6
FHOLD_IN=-1
FS_LIMIT=160.0
RS_LIMIT=-160.0
…
```

Først sættes *pmpnstate* til 53, hvilket fortæller at *set_units_and_defaults* er i gang med C aksen. Der laves en test på hvilken type workcell det er, og *UNITS* sættes herefter på baggrund af forskellige gearinger (om det reelt bliver brugt ved jeg ikke, for det bliver defineret i ID.robot filen). Herefter sættes *BASE* til C aksen og herefter kan standard værdier sættes med deres kommandoer.

```
…
'*************************************************
'* AXIS  D                                       *
'*************************************************
TABLE(pmpnstate,54)
BASE(axis_d)
'UNITS=65536*4*5/360
'UNITS=65536*60/360
IF workcelltype=1 THEN
  UNITS=65536*100/360
  P_GAIN=0.15
  I_GAIN=0.0
  D_GAIN=0.20
ELSE
  IF workcelltype=2 THEN
    UNITS=65536*135/360 '
    P_GAIN=0.3
    I_GAIN=0.0
    D_GAIN=0.20
  ELSE
    UNITS=65536*100/360 '
    P_GAIN=0.15
    I_GAIN=0.0
    D_GAIN=0.20
  ENDIF
ENDIF
OV_GAIN=0.0
VFF_GAIN=8.0
AFF_GAIN=0.0
SRAMP = 5.0
SPEED=4 ' Saet hastighed
ACCEL=8 ' Saet acceleration
DECEL=8 ' Saet deceleration
CREEP=0.1
JOGSPEED=1.0
FE_LIMIT=5
```

```
DAC=0.0
REP_DIST=999999.0
FWD_IN=-1
REV_IN=-1
'*HStomat
DATUM_IN=7
FHOLD_IN=-1
FS_LIMIT=999999.0
RS_LIMIT=-999999.0
…
```

Først sættes *pmpnstate* til 54, hvilket fortæller at *set_units_and_defaults* er i gang med D aksen. Der laves en test på hvilken type workcell det er, og *UNITS* sættes herefter på baggrund af forskellige gearinger (om det reelt bliver brugt ved jeg ikke, for det bliver defineret i ID.robot filen). Herefter sættes *BASE* til D aksen og herefter kan standard værdier sættes med deres kommandoer.

```
…
'************************************************
'* AXIS  E                                      *
'************************************************
TABLE(pmpnstate,55)
BASE(axis_e)
'UNITS=65536*5/360
'UNITS=65536*60/360
IF workcelltype=1 THEN
  UNITS=65536*50/360 '
  P_GAIN=0.20
  I_GAIN=0.0
  D_GAIN=0.25
ELSE
  IF workcelltype=2 THEN
    UNITS=65536*135/360 '
    P_GAIN=0.40
    I_GAIN=0.0
    D_GAIN=0.25
  ELSE
    UNITS=65536*50/360 '
    P_GAIN=0.20
    I_GAIN=0.0
    D_GAIN=0.25
  ENDIF
ENDIF
OV_GAIN=0.0
VFF_GAIN=8.0
AFF_GAIN=0.0
SRAMP = 5.0
SPEED=4 ' Saet hastighed
ACCEL=8 ' Saet acceleration
DECEL=8 ' Saet deceleration
CREEP=0.1
JOGSPEED=1.0
FE_LIMIT=5
```

```
DAC=0.0
REP_DIST=999999.0
FWD_IN=-1
REV_IN=-1
'*HStomat
DATUM_IN=8
FHOLD_IN=-1
FS_LIMIT=999999.0
RS_LIMIT=-999999.0
…
```

Først sættes *pmpnstate* til 55, hvilket fortæller at *set_units_and_defaults* er i gang med E aksen. Der laves en test på hvilken type workcell det er, og *UNITS* sættes herefter på baggrund af forskellige gearinger (om det reelt bliver brugt ved jeg ikke, for det bliver defineret i ID.robot filen). Herefter sættes *BASE* til E aksen og herefter kan standard værdier sættes med deres kommandoer.

```
…
'************************************************
'* AXIS  F                                      *
'************************************************
TABLE(pmpnstate,56)
BASE(axis_f)
UNITS=(65536*10)/(50*PI) 'tandhjul dele-diameter 50mm 7:1 gearing
P_GAIN=0.20
I_GAIN=0.0
D_GAIN=0.25
OV_GAIN=0.0
VFF_GAIN=8.0
AFF_GAIN=0.0
SRAMP = 5.0
SPEED=4 ' Saet hastighed
ACCEL=8 ' Saet acceleration
DECEL=8 ' Saet deceleration
CREEP=0.1
JOGSPEED=1.0
FE_LIMIT=5
DAC=0.0
REP_DIST=999999.0
FWD_IN=-1
REV_IN=-1
'*HStomat
DATUM_IN=-1
FHOLD_IN=-1
FS_LIMIT=999999.0
RS_LIMIT=-999999.0
…
```

Først sættes *pmpnstate* til 56, hvilket fortæller at *set_units_and_defaults* er i gang med F aksen. Herefter sættes *BASE* til F aksen og herefter kan standard værdier sættes med deres kommandoer.

```
…
'************************************************
'* AXIS  T                                      *
```

```
'*************************************************
TABLE(pmpnstate,57)
BASE(axis_t)
UNITS=1
P_GAIN=0.0
I_GAIN=0.0
D_GAIN=0.0
OV_GAIN=0.0
VFF_GAIN=0.0
AFF_GAIN=0.0
SRAMP = 0.0
SPEED=30000 ' Saet hastighed
ACCEL=100000000 ' Saet acceleration
DECEL=100000000 ' Saet deceleration
CREEP=100
JOGSPEED=1000
FE_LIMIT=1
DAC=0.0
REP_OPTION=1
REP_DIST=99999999
FWD_IN=-1
REV_IN=-1
DATUM_IN=-1
FHOLD_IN=-1
FS_LIMIT=99999999
RS_LIMIT=-99999999
…
```

Først sættes *pmpnstate* til 57, hvilket fortæller at *set_units_and_defaults* er i gang med T aksen. Herefter sættes *BASE* til T aksen og herefter kan standard værdier sættes med deres kommandoer.

```
…
'*************************************************
'* AXIS  VT                                      *
'*************************************************
TABLE(pmpnstate,58)
BASE(axis_vt)
UNITS=1
P_GAIN=0.0
I_GAIN=0.0
D_GAIN=0.0
OV_GAIN=0.0
VFF_GAIN=0.0
AFF_GAIN=0.0
SRAMP = 0.0
SPEED=30000 ' Saet hastighed
ACCEL=100000000 ' Saet acceleration
DECEL=100000000 ' Saet deceleration
CREEP=100
JOGSPEED=1000
FE_LIMIT=1
DAC=0.0
REP_OPTION=1
```

```
REP_DIST=99999999
FWD_IN=-1
REV_IN=-1
DATUM_IN=-1
FHOLD_IN=-1
FS_LIMIT=99999999
RS_LIMIT=-99999999
RETURN
…
```

Først sættes *pmpnstate* til 58, hvilket fortæller at *set_units_and_defaults* er i gang med VT aksen. Herefter sættes *BASE* til VT aksen og herefter kan standard værdier sættes med deres kommandoer.

```
…
resetaxis:
inresetaxis=1
WDOG=OFF
WA(100)
IF axis_a_enabled = 1 THEN
  current_axis=axis_a
  GOSUB reset_drive
ENDIF
IF axis_b_enabled = 1 THEN
  current_axis=axis_b
  GOSUB reset_drive
ENDIF
IF axis_c_enabled = 1 THEN
  current_axis=axis_c
  GOSUB reset_drive
ENDIF
IF axis_d_enabled = 1 THEN
  current_axis=axis_d
  GOSUB reset_drive
ENDIF
IF axis_e_enabled = 1 THEN
  current_axis=axis_e
  GOSUB reset_drive
ENDIF
IF axis_f_enabled = 1 THEN
  current_axis=axis_f
  GOSUB reset_drive
ENDIF
DATUM(0)

WA(100)
IF axis_a_enabled = 1 THEN SERVO AXIS(axis_a)=ON
IF axis_b_enabled = 1 THEN SERVO AXIS(axis_b)=ON
IF axis_c_enabled = 1 THEN SERVO AXIS(axis_c)=ON
IF axis_d_enabled = 1 THEN SERVO AXIS(axis_d)=ON
IF axis_e_enabled = 1 THEN SERVO AXIS(axis_e)=ON
IF axis_f_enabled = 1 THEN SERVO AXIS(axis_f)=ON
'WDOG=ON
```

```
inresetaxis=0
RETURN
…
```

Størstedelen af denne subrutine går på at tjekke hvilke af akserne der er *enabled* og hvis de er springe til en anden subrutine der resetter den pågældende akse. Efter alle de akser der skal resettes er blevet resat, cleares eventuelle fejl med *DATUM(0)* og til sidst tændes/gives der strøm til de akser der er *enabled* og som tidligere blev resat.

```
…
'*************************************************
'********** T E S T   F U N C T I O N S **********
'*************************************************
' reset the amplifier in case of trip
reset_drive:
TABLE(pmpnstate,70+current_axis) 'reset axis num
tries = 0
reset_xx:
BASE(current_axis)
'PRINT "*****************************************"
'PRINT "*** R E S E T   A X I S ";current_axis[0];"  ***************"
'PRINT "*****************************************"
  IF MOTION_ERROR THEN DATUM(0)
  sum=0
  FOR check=1 TO 10
   sum=sum+DLINK(7,current_axis,251)
  NEXT check
  IF sum<>-10 THEN
    tries = tries+1
    IF tries > 5 THEN
'     PRINT "unable to reset drive axis ", current_axis
      STOP
    ELSE
      WA(10)
      GOTO reset_xx
    ENDIF
  ENDIF
 ' enable the axis
  DLINK(7,current_axis,253)
RETURN
…
```

I denne subrutine er det første der bliver gjort at *pmpnstate* bliver sat til 7*x* (71 for akse A og 72 for akse B) hvilket fortæller at *x* aksen er ved at blive resat og så bliver *BASE* sat til *current_axis*. Hvis der er en fejl på aksen så bliver der udregnet en checksum for den aktuelle akse. Hvis aksen ikke kan resettes efter 5 forsøg stoppes subrutinen ellers bliver der forsøgt igen, og hvis aksen er blevet resat, aktiveres aksen igen.

```
…
set_mpn_home_all:
TABLE(pmpnretur,0)
slm_is_ok = 1
GOSUB test_if_slm_is_ok
IF axis_a_enabled = 1 THEN GOSUB quick_datum_a
```

```
IF axis_b_enabled = 1 THEN GOSUB quick_datum_b
IF axis_c_enabled = 1 THEN GOSUB quick_datum_c
IF axis_d_enabled = 1 THEN GOSUB quick_datum_d
IF axis_e_enabled = 1 THEN GOSUB quick_datum_e
IF axis_f_enabled = 1 THEN GOSUB quick_datum_f
IF slm_is_ok=0 THEN
  TABLE(pmpnretur,1) 'tell master to show calib screen
  BASE(axis_b)
  FS_LIMIT=999.0
  RS_LIMIT=-999.0
  BASE(axis_c)
  FS_LIMIT=999.0
  RS_LIMIT=-999.0
ELSE
  IF axis_a_enabled = 1 THEN GOSUB set_mpnhome_a
  IF axis_b_enabled = 1 THEN GOSUB set_mpnhome_b
  IF axis_c_enabled = 1 THEN GOSUB set_mpnhome_c
  IF axis_d_enabled = 1 THEN GOSUB set_mpnhome_d
  IF axis_e_enabled = 1 THEN GOSUB set_mpnhome_e
  IF axis_f_enabled = 1 THEN GOSUB set_mpnhome_f
ENDIF
RETURN
…
```

Først bliver *slm_is_ok* sat til 1, herefter springes der til subrutinen *test_if_slm_is_ok*. Efter *test_if_slm_is_ok* har kørt springes der til subrutiner der resetter fejl på de enkelte akser. Herefter hvis *sml_is_ok* er 0 ændres *FS_LIMIT* (Forward Software Limit) og *RS_LIMIT* (Reverse Software Limit) til 999.0 og -999.0, ellers springes der til subrutiner der sætter home for den aktiverede akser.

```
…
get_abspos:
'  PRINT "*******************************"
  pos = DLINK(5,dl_axis,74)
  flux = DLINK(5,dl_axis,75)
  zero = DLINK(5,dl_axis,76)
  rev = DLINK(5,dl_axis,90)
'  PRINT "AXIS ";dl_axis;" pos=";pos;" zero=";zero;" rev=";rev;" flux=";flux
  IF rev > 32767 THEN
'    PRINT "AXIS ";dl_axis;" rev>32767 rev=";rev
     rev = rev - 65536
'    PRINT "AXIS ";dl_axis;" rev=rev-65536 rev=";rev
  ENDIF
  abspos=rev*65536+pos-zero
'  PRINT "AXIS ";dl_axis;" abspos=";abspos
RETURN
…
```

Subrutinen *get_abspos* læser motorparametre fra SLM-modulet (position, flux, nulpunkt og omdrejninger) herefter udregnes den absolutte position og herefter returneres til stedet der blev sprunget til subrutinen fra.

```
…
'mpnhome positions are absolute SLM positions
```

```
set_mpnhome_a:
  BASE(axis_a)
  WAIT IDLE
  WA(50)
  OFFPOS = -mpnhome_a/UNITS
  WAIT UNTIL OFFPOS=0
RETURN
set_mpnhome_b:
  BASE(axis_b)
  WAIT IDLE
  WA(50)
  OFFPOS = -mpnhome_b/UNITS
  WAIT UNTIL OFFPOS=0
RETURN
set_mpnhome_c:
  BASE(axis_c)
  WAIT IDLE
  WA(50)
  OFFPOS = -mpnhome_c/UNITS
  WAIT UNTIL OFFPOS=0
RETURN
set_mpnhome_d:
  BASE(axis_d)
  WAIT IDLE
  WA(50)
  OFFPOS = -mpnhome_d/UNITS
  WAIT UNTIL OFFPOS=0
RETURN
set_mpnhome_e:
  BASE(axis_e)
  WAIT IDLE
  WA(50)
  OFFPOS = -mpnhome_e/UNITS
  WAIT UNTIL OFFPOS=0
RETURN
set_mpnhome_f:
  BASE(axis_f)
  WAIT IDLE
  WA(50)
  OFFPOS = -mpnhome_f/UNITS
  WAIT UNTIL OFFPOS=0
RETURN
…
```

Her er alle subrutinerne der kører akserne tilbage til *home* position, det gøres ved at sætte *OFFPOS* til encoder værdien for *home* delt med den enhed som aksen er delt op i (afhægig af gearing og lignende).

```
…
quick_datum_a:
  TABLE(pmpnstate,81)
  BASE(axis_a)
  DATUM(2)
```

```
  WAIT IDLE
  WA(50)
  dl_axis = axis_a
  GOSUB get_abspos
  DEFPOS(abspos/UNITS)
  WAIT UNTIL OFFPOS=0
  WA(50)
RETURN
quick_datum_b:
  TABLE(pmpnstate,82)
  BASE(axis_b)
  IF workcelltype = 2 THEN
    DATUM(1)
  ELSE
    DATUM(2)
  ENDIF
  WAIT IDLE
  WA(50)
  dl_axis = axis_b
  GOSUB get_abspos
  DEFPOS(abspos/UNITS)
  WAIT UNTIL OFFPOS=0
  WA(50)
RETURN
quick_datum_c:
  TABLE(pmpnstate,83)
  BASE(axis_c)
  DATUM(1)
  WAIT IDLE
  WA(50)
  dl_axis = axis_c
  GOSUB get_abspos
  DEFPOS(abspos/UNITS)
  WAIT UNTIL OFFPOS=0
  WA(50)
RETURN
quick_datum_d:
  TABLE(pmpnstate,84)
  BASE(axis_d)
  DATUM(1)
  WAIT IDLE
  WA(50)
  dl_axis = axis_d
  GOSUB get_abspos
  DEFPOS(abspos/UNITS)
  WAIT UNTIL OFFPOS=0
  WA(50)
RETURN
quick_datum_e:
  TABLE(pmpnstate,85)
  BASE(axis_e)
  DATUM(2)
```

```
  WAIT IDLE
  WA(50)
  dl_axis = axis_e
  GOSUB get_abspos
  DEFPOS(abspos/UNITS)
  WAIT UNTIL OFFPOS=0
  WA(50)
RETURN
quick_datum_f:
  TABLE(pmpnstate,86)
  BASE(axis_f)
  DATUM(1)
  WAIT IDLE
  WA(50)
  dl_axis = axis_f
  GOSUB get_abspos
  DEFPOS(abspos/UNITS)
  WAIT UNTIL OFFPOS=0
  WA(50)
RETURN
…
```

Her er alle *DATUM*/homing for alle akserne. Først bliver *pmpnstate* sat til at aksen er i gang med at finde home position, så sættes *DATUM(1)* (flytter aksen fremad i *creep* hastighed) eller *DATUM(2)*(flytter aksen baglæns i *creep* hastighed) indtil aksen er i nulposition.

```
…
set_defpos:
IF axis_a_enabled = 1 THEN GOSUB defpos_home_a
IF axis_b_enabled = 1 THEN GOSUB defpos_home_b
IF axis_c_enabled = 1 THEN GOSUB defpos_home_c
IF axis_d_enabled = 1 THEN GOSUB defpos_home_d
IF axis_e_enabled = 1 THEN GOSUB defpos_home_e
IF axis_f_enabled = 1 THEN GOSUB defpos_home_f
BASE(axis_t)
DEFPOS(0)
WAIT UNTIL OFFPOS=0' Ensures DEFPOS is complete before next line
RETURN
…
```

Denne subrutine laver bare spring til andre subrutiner for de enkelte aktiverede akser og til sidst bliver *DEFPOS* til 0 på T aksen.

```
…
movehome:
  IF TABLE(pmpnretur+1) = 0 THEN
    BASE(axis_t)
    DEFPOS(0)
    WAIT UNTIL OFFPOS=0' Ensures DEFPOS is complete before next line
    mpnhome_t=0
  ENDIF
  IF TABLE(pmpnretur+1) = 1 THEN GOSUB defpos_home_a
  IF TABLE(pmpnretur+1) = 2 THEN GOSUB defpos_home_b
  IF TABLE(pmpnretur+1) = 3 THEN GOSUB defpos_home_c
```

```
  IF TABLE(pmpnretur+1) = 4 THEN GOSUB defpos_home_d
  IF TABLE(pmpnretur+1) = 5 THEN GOSUB defpos_home_e
  IF TABLE(pmpnretur+1) = 6 THEN GOSUB defpos_home_f
RETURN
…
```

Denne subrutine, ligesom *set_defpos*, laver bare spring til andre subrutiner for at få robotten i home position, alt efter hvilken akse der står til at skulle flyttes til home i registret *pmpnretur+1*.

```
…
defpos_home_a:
  BASE(axis_a)
'  DATUM(4)
  DATUM(2)
  WAIT IDLE
  WA(50)
  dl_axis = axis_a
  GOSUB get_abspos
  mpnhome_a = abspos
  DEFPOS(0)
  WAIT UNTIL OFFPOS=0
  WA(50)
  DLINK(6,axis_a,62,backup_val) 'set host flag for backup
RETURN
…
```

Denne subrutine får A aksen til at køre i home position. Først sættes A aksen som *BASE*, herefter vil alle funktioner bliver udført på denne akse. *DATUM(2)* kører A aksens motor baglæns i krybehastighed indtil home positionen er fundet, herefter sættes home positionen til den aktuelle position aksen har og *DEFPOS* sætter punktet som nulpunkt for aksen.

```
…
defpos_home_b:
  BASE(axis_b)
  IF workcelltype = 2 THEN
    DATUM(3)
  ELSE
    DATUM(4)
  ENDIF
  WAIT IDLE
  WA(50)
  dl_axis = axis_b
  GOSUB get_abspos
  mpnhome_b = abspos
  DEFPOS(0)
  WAIT UNTIL OFFPOS=0
  WA(50)
  DLINK(6,axis_b,62,backup_val) 'set host flag for backup
RETURN
…
```

Subrutinen for at køre B aksen i home position ligner den for A aksen, den eneste forskel er at der bruges *DATUM(3)* eller *DATUM(4)*, alt efter *workcelltype*, fordi B aksen har en fysisk homesensor som bruges til at finde home positionen.

... ...
... ...
... ...
... ...
... ...
... ...