ISSN 0280-5316
ISRN LUTFD2/TFRT--5475--SE

Implementering av experimentellt robotstyrsystem

Mats Karlsson

Institutionen for Reglerteknik
Lunds Tekniska Högskola
November 1993

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

Document name
MASTER THESIS

Date of issue
November 1993

Document Number
ISRN LUTFD2/TFRT--5475--SE

Author(s)
Mats Karlsson

Supervisor
Klas Nilsson, Rolf Johansson

Sponsoring organisation

Title and subtitle
Implementering av experimentellt robotstyrsystem
(Implementation of an experimental robot control system)

Abstract

An experimental platform containing robots, a Sun-workstation, a joy-stick and external sensors was used forresearch in robot programming and sensor-based motion control. The purpose of this work was to implement a real-time system that integrate these parts, into a system for advanced robotics and control experiments. For external sensors forming an external closed loop feedback, with the system. There are hard real-time demands on the parts inside this loop, otherwise you get time-delay. The general programming language to be used in the system is Modula-2.

Key words
Industrial robots, Real-time programming, Control systems

Classification system and/or index terms (if any)

Supplementary bibliographical information

ISSN and key title
0280–5316

ISBN

Language
Swedish

Number of pages
26

Recipient's notes

Security classification

Innehållsförteckning

Förord

Denna rapport avser ett examensarbete gjort på Institutionen för Reglerteknik vid Lunds Tekniska Högskola. Arbetet har inneburit utveckling av realtidsprogramvara för experimentell robotstyrning även ett gränssnitt för en yttre givare har implementerats. Test och utprovning av programvaran har varit tidskrävande. Att beskriva alla problem som stötts på under denna fas är inte meningsfullt. Därför har jag valt att försöka underlätta för ev. efterföljares arbete genom en handboksliknande utformning av denna rapport.

Jag vill här passa på att framföra ett tack till Klas Nilsson för hans uthållighet och aktiva vägledning av arbetet, och till Sofia Öberg som hjälpt mig med att läsa korrekturet.

1. Inledning

På institutionen för reglerteknik finns ett robotsystem som används inom såväl forskning som undervisning. Systemet används för både generell och tillämpad robotstyrning. Ett exempel på tillämpning är sensorbaserad rörelsestyrning där återkoppling görs från en yttre tillämpningsspecifik givare. För att inte tidsfördröjning och därmed också fasförskjutning skall uppkomma i ovanstående exempel p.g.a. att vissa delar, som tidigare låg utanför reglerloopen, hamnar inuti den yttre reglerloopen, ställs det höga realtidskrav även på dessa. Dessutom krävs en tät koppling mellan de olika delarna av realtidsprogramvaran, för att kunna åstadkomma nya yttre reglerloopar, samt för att erhålla effektivt utförande av robotrörelser. Syftet med detta examensarbete, har varit att implementera ett sådant sammanbundet system . I detta system skall det vara enkelt att utföra olika avancerade robotik-, servo- och reglertekniska experiment. Ett problem är att det är ett heterogent realtidssystem med såväl tidskritisk sampling av regulatorn som sporadiska beräkningsoperationer. Man vill ha en koppling mellan beräkningsrutiner och regulatorn där man utnyttjar den tillgängliga datorkraften så effektivt som möjligt. För att kunna användas i undervisningen och eftersom det finns en väl utprovad realtidskärna, har Modula-2 valts som språk.

Den yttre givare som används är en avståndsmätare av lasertyp. För att kommunikationen med denna skall kunna ske på ett enkelt och smidigt sätt behövs en koppling mellan givare och styrprogram. För detta har ett gränssnitt implementerats.

Denna rapport har följande uppläggning: I kapitel 2 ges en översikt av systemet. Kapitel 3 beskriver användargränssnitten, dvs. styrning av roboten från värddator resp. styrspak. I kapitel 4 förklaras programstrukturen. Detta innebär att modul och processtruktur presenteras och att kontrollen över roboten definieras. Kapitel 5 innehåller kommentarer kring implementeringen av de olika modulerna. Hur uppstart och igångkörning av systemet sker beskrivs i kapitel 6. Slutligen finns en sammanfattning i kapitel 7.

Rapporten har delvis givits en handboksliknande utformning. Min förhoppning är att den därigenom skall kunna tjäna som handledning för nya användare av systemet.

2. Systemöversikt

I detta kapitel ges en översikt av vad som fanns av systemet och vad som utvecklats och lagts till. Översikten är uppdelad i två delar, hårdvarudel och mjukvarudel. Målet med detta kapitel är att läsaren generellt skall förstå hur systemet hänger ihop.

2.1 Hårdvara

Figur 2.1 ger en översikt av hårdvaran i systemet. Av det ursprungliga robotsystemet, en ASEA IRB-6, utnyttjas endast kraftelektroniken och nödstoppsanordningen. Användarens gränssnitt gentemot systemet består av tre delar: fönster på värddatorn,nödstoppen och styrspaken med dess knappsats.

För dataöverföring mellan värddatorn och målsystemet används två olika metoder. En seriell kommunikation, RS-232, används för igångkörning av

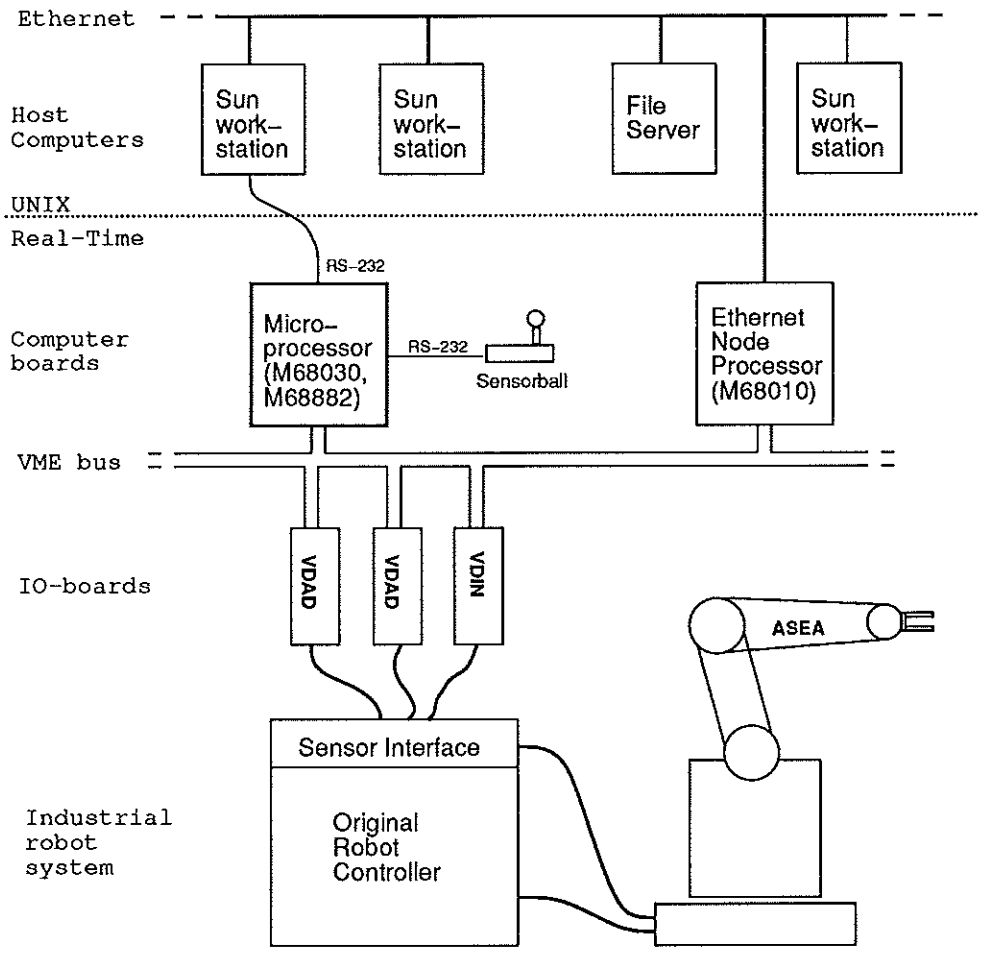

Figure 2.1 Systemöversikt.

systemet och smärre operatörskommunikationer. Denna kommunikation används också via en annan port för överföring av data från styrspaken. Den andra metoden är till för stora och tidskrävande dataöverföringar. Detta sker över Ethernet. Ethernet-processorn1 är kopplad till VME-bussen och används för att skicka ner rörelsespecifikationer (robotprogram, trajektorier) till målsystemet och för att hämta upp olika signaler (loggade data) till värddatorn. Där används de bl.a. till plottning med hjälp av Matlab.

Kommunikation med VDAD och VDIN-korten ingår också som en del i trafiken över VME-bussen. Ärvärdena kan fås på två sätt, antingen analogt via VDAD-korten eller digitalt via VDIN-korten. VDAD-kortens digitala kanaler används får att adressera aktuell robotled vid digital läsning, återställa resolvermätsystemet, styra gripdonet och styra lasern. All hårdvara, beskriven ovan, fanns tillgänglig när arbetet började. Det som gjorts gällande hårdvaran är att en yttre givare, en laseravståndsmätare tillförts.

En av fördelarna med denna öppna experimentstruktur är den stora frihet som finns att t.ex. byta regleralgoritm eller andra moduler på ett smidigt sätt. Detta är inte möjligt i ett kommersiellt system, som optimerats för att på enklaste sätt klara normal industriell användning. Sådana system blir mer eller mindre slutna.

2.2 Mjukvara

De delar av mjukvaran som fanns innan arbetet startade var kommunikation med värddatorn via terminallinje och Ethernet. Dessutom fanns också all mjukvara relaterad till ServoIO-modulen och till den underliggande lager så som ResolverIO, DigitalIO och AnalogIO. Det enda hårdvarunära programmering som gjorts är implementering av ett interface för laseravståndsmätaren. Denna styrs via VDAD-korten. Därför har modulerna ResolverIO och VDAD modifierats. Dessutom har en modul LaserIO tillkommit.

Detta arbetes tyngdpunkt har legat i att implementera ett program som sammanbinder de olika delarna av systemet. I figur 2.2 visas de olika gränssnitten som programmet måste arbeta mot.

---

1 Vid uppgradering av hårdvaran efter detta arbetes utförande, slopades Ethernet-processorn. Denna funktionalitet finns istället på det modernare CPU-kortet.

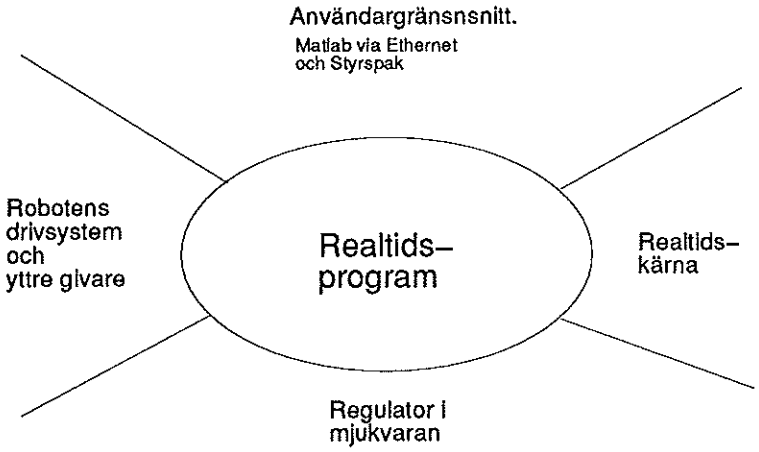

Figure 2.2 Omgivning för det utvecklade realtidsprogrammet.

Det gäller alltså att binda ihop drivsystemet till roboten med en mjukvaruregulator och ett användargränssnitt. Mjukvaruregulatorn ligger förvisso inuti programmet, men den kan anses som en egen enhet. Den har därför lagts i en egen modul och kan betraktas som drivrutiner för robotens servon. Rent reglertekniskt kan man se regulatorn som den väsentliga delen av systemet, men programvarutekniskt utgör regulatorn en liten del av styrsystemets programvara. Huvuddelen utgörs istället av det realtidsprogram som tar emot och tolkar rörelsebeskrivningar och kommandon, och omsätter detta till parametrar och börvärden för den underliggande regleringen. Det är detta realtidsprogram, som arbetet är inriktat på. Förutom detta har också en användarvänlig man-maskin kommunikation implementerats. Denna består i dels ett grafiskt användargränssnitt på värddatorn och dels en förenkling av körningen via styrspaken.

För programmeringen har Modula-2, samt en realtidskärna som utvecklats på institutionen, använts.

3. Användargränssnitt

Detta kapitel innehåller en handledning i hur gränssnittet på värddatorn respektive sensorbollen används. Implementeringen av dessa finns i kapitel 5.

3.1 Styrning av roboten via värddatorn

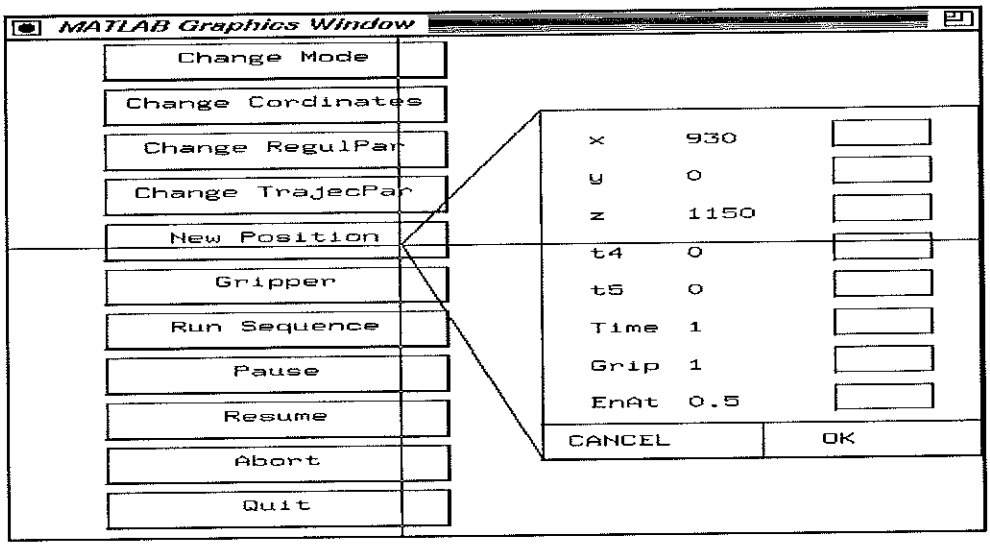

Figure 3.1 Användargränssnitt på Sun-datorn impl. i Matlab3.

Efter uppstart av systemet kommer ett antal knappar upp i ett av matlabfönstren på värddatorns skärm, se figur 3.1. Dessa knappars funktion beskrivs nedan. Defaultvärdena på de parametrar och värden som kan ändras anges i ändringsfönstren. Notera att vid inmatning från tangentbordet till grafikfönstret för Matlab3, måste vagnretur slås efter att ett nytt värde matats in(dvs. nytt musklick duger ej). Knapparna har följande innebörd :

- **Quit:** Terminerar hela programmet.
- **Abort:** Om roboten är i rörelse, stannas den. Därefter kastas alla begärda förflyttningar.
- **Resume:** Om roboten är , körs återstoden av begärda förflyttningar. Annars ignoreras kommandot.
- **Pause:** Roboten stannas och man väntar på att antingen Resume eller Abort skall aktiveras.
- **Run Sequence:** Kör hela sekvenser som definierats i Matlab. Formatet på dessa macron kan ses i kapitel 5 under rubriken Matlabmacron. Om man anger SYNC som sekvens så utförs synkronisering av robotens mätsystem.

- **Gripper:** Stänger gripdonet med kommandot On och öppnar detsamma med Off.
- **New Position:** Om rörelsen till den nya positionen ligger inom robotens arbetsområde utförs en förflyttning i det koordinatsystem man för tillfället befinner sig i. Förflyttningen utföres från föregående slutposition till positionen x, y, z, t4 och t5 eller till t1, t2, t3, t4 och t5. Kartesiska förflyttningar anges i millimeter och ledvisa förflyttningar i grader. Tiden man vill att rörelsen skall ske på, Time, anges i millisekunder. Gripdonet, Grip, kan även styras under rörelsen genom att det sätts till Open, Close eller NoMove. Om man vill att det skall ske någon förändring av gripdonets läge så sker det efter EnAt av den totala tiden för rörelsen.
- **Change TrajecPar:** Ändrar trajektorieparametrarna maximalhastighet, V,och maximal acceleration, A, för varje led eller för kartesisk förflyttning då kallad Vcrt resp. Acrt. Man kan även ändra Trajektorieprocessens samplings intervall, Ts. Tiden skall då anges i millisekunder.
- **Change RegulPar:** Ändrar regulator parametrarna Kff och Kp för varje led.Man kan även ändra regulatorns samplings intervall, Ts. Även denna tid skall anges i millisekunder.
- **Change Cordinates:** Byter till kartesiska koordinater eller ledvinklar genom att klicka på Cart. respektive Joint.
- **Change Mode:** Om styrningen av roboten sker via skärmen, Auto, kan man lämna över kontrollen till styrspaken för manuell körning(fortsättningsvis kallad Sensorbollen) genom att klicka på Man. Att byta från Man. till Auto måste ske via sensorbollens knappar.

3.2 Styrning av roboten via sensorbollen

Sensorbollen är en sexdimensionell styrspak för roboten. För att kunna styra roboten med sensorbollen krävs att man klickar på knappen Change Mode och därefter på knappen Man på värddatorns skärm. Då övergår robotens rörelsekontroll till sensorbollen. Vissa kommandon såsom att ändra regulator och trajektorie parametar kan fortfarande utföras från skärmen. Kontrollen behålls av sensorbollen tills någon trycker ned knapp nummer 8 på sensorbollens sockel. Då återgår kontrollen till värddatorn.

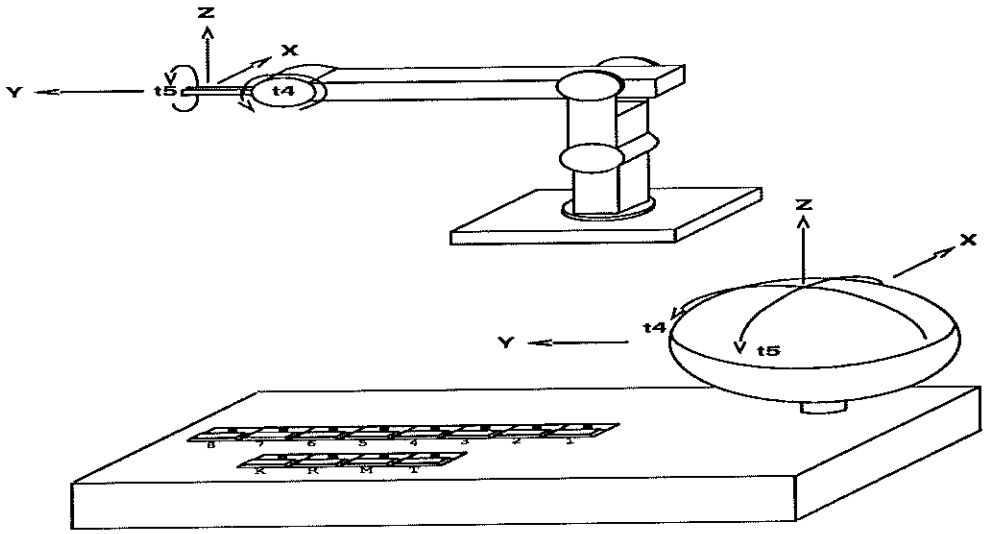

Figure 3.2 Koppling mellan sensorboll och robot då kartesiska koordinater används

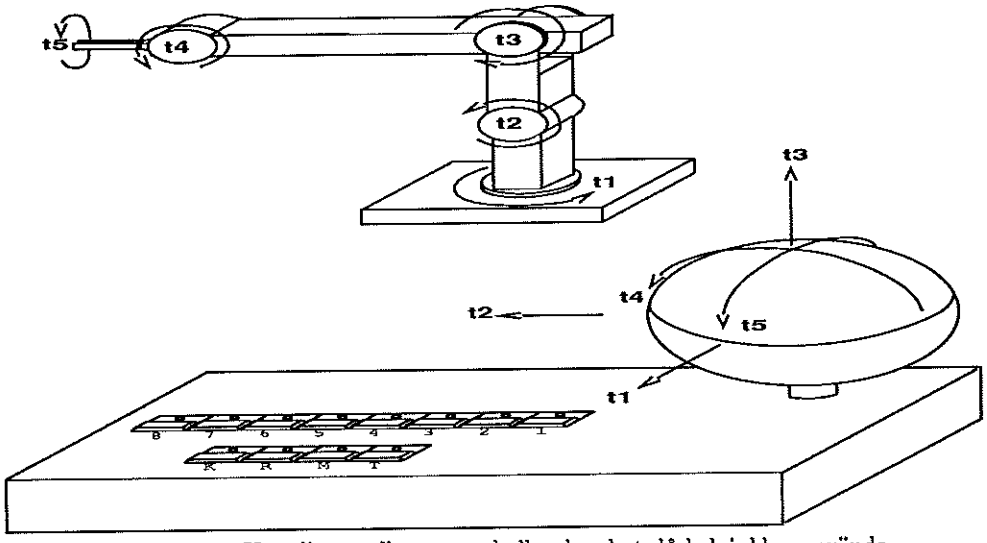

Figure 3.3 Koppling mellan sensorboll och robot då ledvinklar används.

Sensorbollens koppling till robotens rörelse kan ses i fig. 3.2 och fig. 3.3. Denna koppling och knapparna 1, 2 och 3 på sensorbollens sockel är beroende av vilket koordinatsystem som används för tillfället. Knapparnas funktion är enligt följande:

Om ledvinklar används :

Knapp	**Aktiverade**	
1	t1	t4
2	t2	t5
3	t3	

Om kartesiska koordinater används :

Knapp	Aktiverade	
1	X trans	t4
2	Y trans	t5
3	Z trans	

Om R eller T också är nedtryckt utesluts andra resp. tredje kolumnen i tabellerna ovan. Ett exempel : Om knapparna 1 och R är nedtryckta och man använder sig av ledvinklar är alltså endast led fyra, t4, aktiverad.

Oberoende av vilka koordinater som används :

Knapp

4 **Pos. def.** : Medför att aktuell position, beskriven i det koordinatsystem som gäller för tillfället, skrivs ut i console-fönstret. Här finns en dödtid på 3 sekunder mellan två tryckningar för att förhindra ofrivilliga dubbeltryckningar.

5 **Gripper** : Växlar mellan stängt och öppet gripdon och vice versa. Minimum tid mellan växlingarna är 2 sekunder för att förhindra ofrivilliga dubbeltryckningar.

6 **ToggleCoord**: Byter koordinatsystem i vilket förflyttninngen skall ske. Här är dödtiden också 2 sekunder. Från början är jointkoordinater inställda.

7 **Spare bit** : Används ej.

8 **Återlämnar** kontrollen av robotens rörelse till Auto, dvs till värddatorn.

4. Programstruktur

Detta kapitel behandlar hur modul och processuppdelningen är gjord. Frågan om vem som har den övergripande kontrollen av roboten besvaras också. Slutligen beskrivs vad som händer när kommandona pause, abort eller resume anropas.

4.1 Modul och processtruktur

Moduluppdelningen enligt fig 4.1 är gjord för att ge en konstruktion av återanvändbara och flexibla programkomponenter. Tanken är att man skall kunna byta/ändra varje lager som man vill, bara man håller de olika gränssnitten intakta.

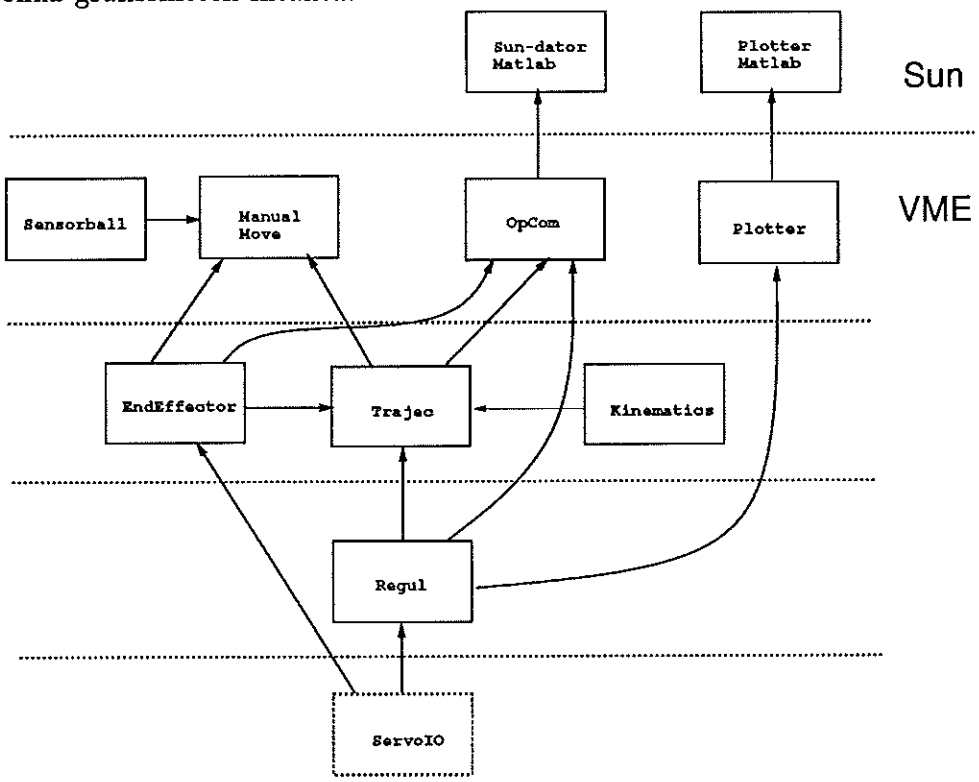

Figure 4.1 Modulgraf med export och import indikationer.

Underst ligger modulen Regul. Den innehåller som namnet anger regleringen. Ovanför denna ligger Trajecmodulen. Här sker genereringen av trajektorierna. Beräkningsprocedurer för trajektorierna exporteras från denna modul till nästa nivå. Detta görs för att man skall förbättra prestandan hos systemet, vad det gäller körning av flera trajektorier i följd. Vid beräkningarna används matematikmodulen Kinematics som innehåller omvandling mellan de olika koordinatsystemen och kontroll om en punkt eller en rörelse ligger inom robotens arbetsområde. Opcom har

hand om all kommunikation med operatören utom manuell körning via sensorbollen. Det senare är modulen ManualMove huvudsakliga uppgift. Här finns även en modul Plotter som ombesörjer plottningen av olika signaler från regulatorn på Sun-datorns skärm. Överst ligger ett antal Matlab-macron för användargränssnittet i värddatorn.

EndEffector tar hand om robotens gripdon. Eftersom man skall kunna styra gripdonet på tre olika sätt: från sensorbollen, i en trajektoria och direkt från skärmen behövs kommunikation med denna modul på flera olika nivåer.

Processerna följer i stort modulernas uppdelning. Enda undantagen, vilket kan ses i fig 4.2, är att Opcom modulen innehåller två processer, Commpro och Matlabhandler, och att Kinematics inte motsvaras av någon process. Processen Matlabhandler har hand om att omvandla kommandon, utföra skiftet mellan körning av roboten från värddatorn respektive via sensorbollen och kontrollera pause, abort och resume medan Commpro utför alla trajektorieberäkningar. Om dessa uppgifter legat i samma process hade inte kommandona pause, abort och resume fungerat tillfredställande.

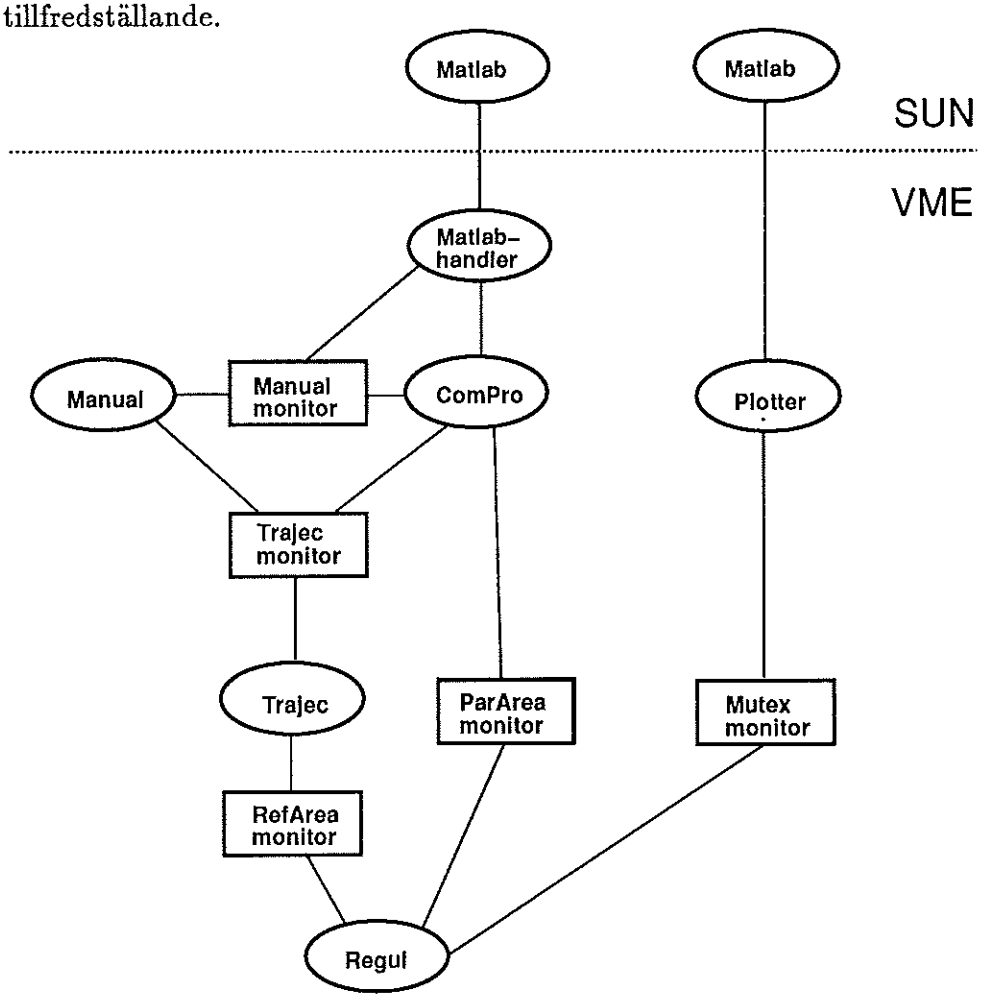

Figure 4.2 Processgraf.

4.2 Kontrollstruktur för robotens rörelse

Målet med denna kontrollstruktur är att endast en av processerna ManualMove och MatlabHandler skall ha kontrollen över robotens rörelse vid varje tillfälle. Detta vore enkelt om den ena processen kunde vara överordnad den andra. Ett annat sätt kunde vara att den ena processen hängde på en semafor, medan den andra hade kommandot och vice versa. Men man vill kunna utföra vissa kommmandon, som inte direkt har med rörelsen av roboten att göra, under tiden som den andra processen har kontrollen över rörelsen. Exempel på sådana kommandon är att ändra regulator eller trajektorie parametrar. Detta medför en något mer komplicerad struktur, vars tillståndsgrafer visas i figur 4.3.

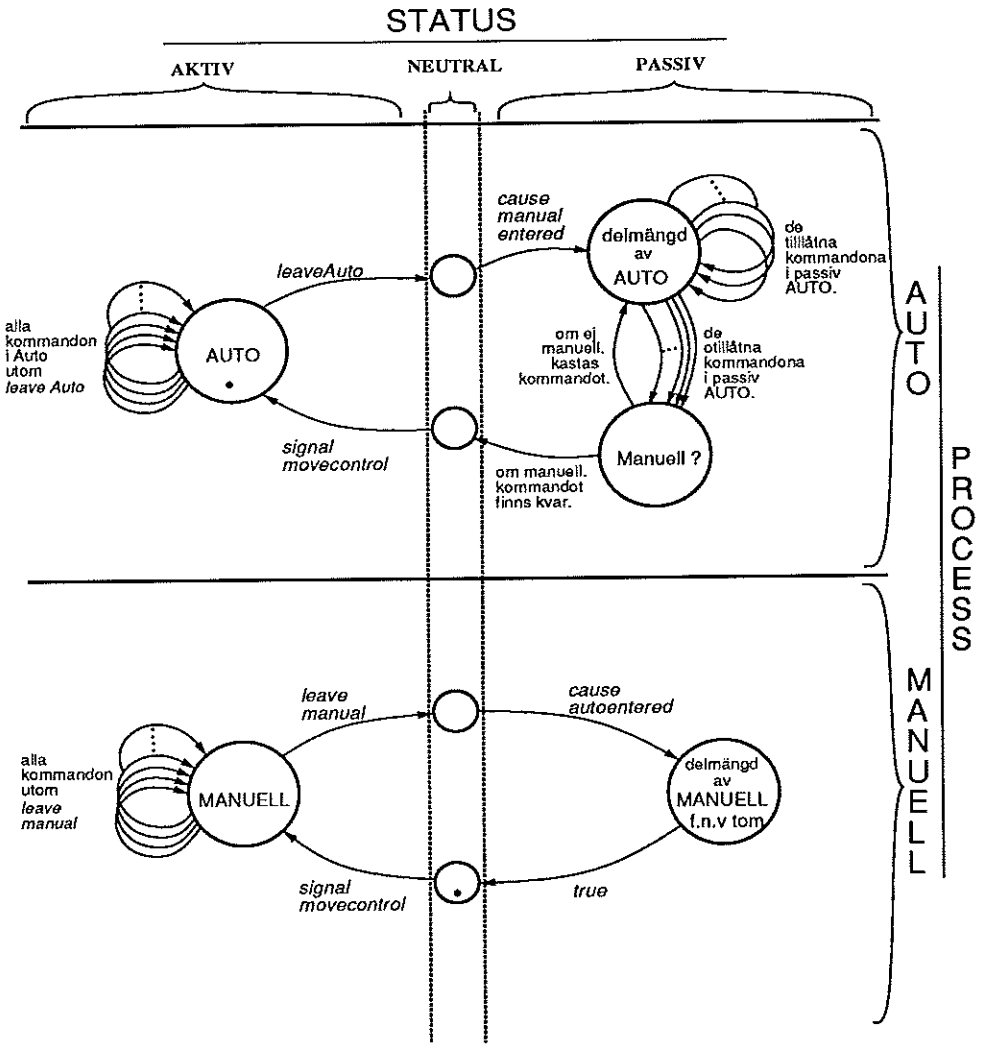

Figure 4.3 Tillståndsgrafer för MatlabHandler, här kallad Auto, och ManualMove, kallad Manuell.

Processerna har tilldelats tre möjliga övergripande tillstånd: Aktiv, Neutral och Passiv.

- Att vara i **aktivt** tillstånd innebär att man har kontrollen över

robotrörelserna. Därför får bara en process befinna sig i detta tillstånd vid en och samma tidpunkt, för att ömsesidig uteslutning skall uppnås.

- I det **passiva** tillståndet kan man utföra kommandon som inte har direkt med rörelsen av roboten att skaffa.
- Det **neutrala** tillståndet är till för att klara skiftet mellan aktivt och passivt tillstånd utan kapplöpningseffekter mellan processerna.

Skiftet mellan tillstånden sker på följande sätt. Den process som har kontrollen, dvs. är i aktivt tillstånd, går till neutralt tillstånd. Där väntar den tills den andra processen också kommit dit. Därefter kan de båda processerna fortsätta till sina respektive nya tillstånd utan risk för kapplöpning.

4.3 Kontrollstruktur för Abort och Pause

Som operatör från skärmen finns det möjlighet att med hjälp av kommandot pause stanna roboten under en rörelse. Därefter kan man kasta alla kvarvarande trajektorier med kommandot abort, eller fortsätta där man slutade med kommandot resume. Det är processen Matlabhandler som sköter detta. Tillstånden denna process kan befinna sig visas i fig. 4.4. Denna tillståndsgraf är endast aktuell när man kör från skärmen eftersom pause och abort annars är otilllåtna kommandon.

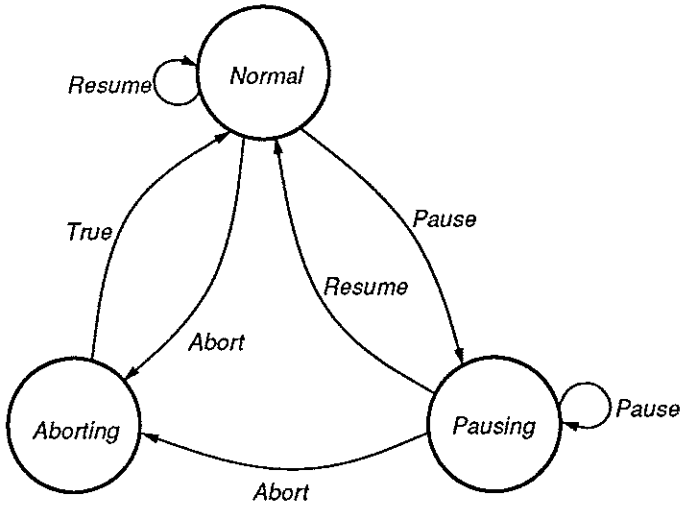

Figure 4.4 Tillståndsgraf för processen Matlabhandler.

5. Implementering

Implementeringen av reglering, referensgenerering, arbetsområdeskontroll, operatörskommunikation och manuell förflyttning beskrivs i detta kapitel. Lasermätarens I/O-modul beskrivs också.

5.1 Reglering

Regulatormodulen **Regul** innehåller den del av regleringen som implementerats i programvara enligt figur 5.1. Förutom själva regleralgoritmen ingår metoder för ändring av parametrar o.d. vilka beskrivs nedan. I figur 5.1 visas även den analoga delen av regleringen som ingår i robotstyrningens hårdvara.

Det finns fem stycken robotservon. För varje servo finns en uppsättning av de block som visas i figur 5.1. Den återkopplade servovinkeln, theta, och börvärdet för läget, lägesref., bildar reglerfelet, e, genom att den förstnämnda subtraheras från den sistnämnda. För att kunna köra längs en bana utan eftersläpning, som innebär banavvikelse, används framkoppling från börvärdet. Detta sker genom att börvärdet för hastigheten, hast.ref., multipliceras med framkopplingsfaktor, Kff, och adderas till reglerfelet som har multiplicerats med en förstärkningsfaktor, Kp. Denna summa skickas sen till regulatorns hårdvarudel. Allt detta sker i processen Regul. Observera att deriveringen av lägesreferensen sker i tra-

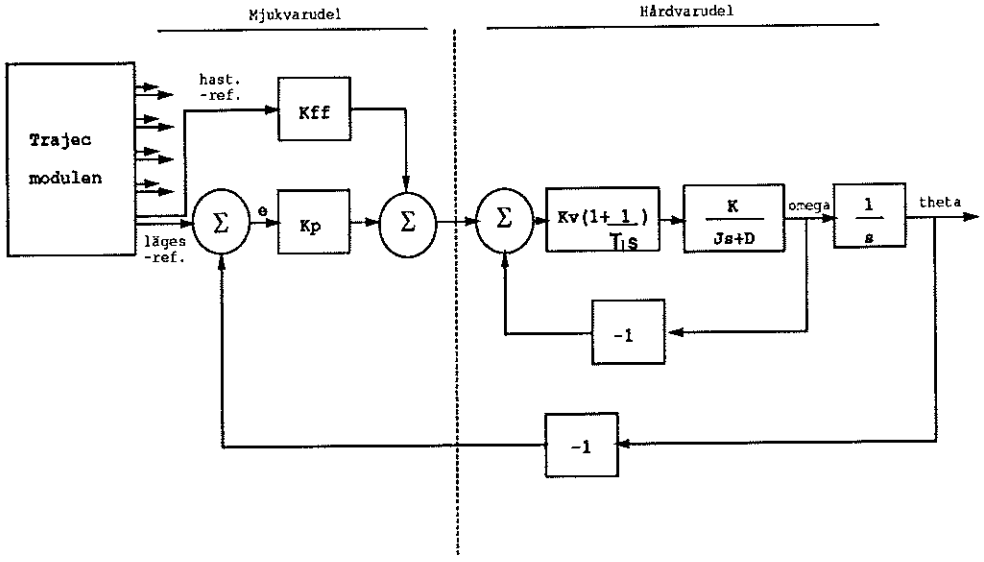

Figure 5.1 Blockschema för trajektoriegenerering och reglering. Endast en förenklad beskrivning av regleringen för en led visas.

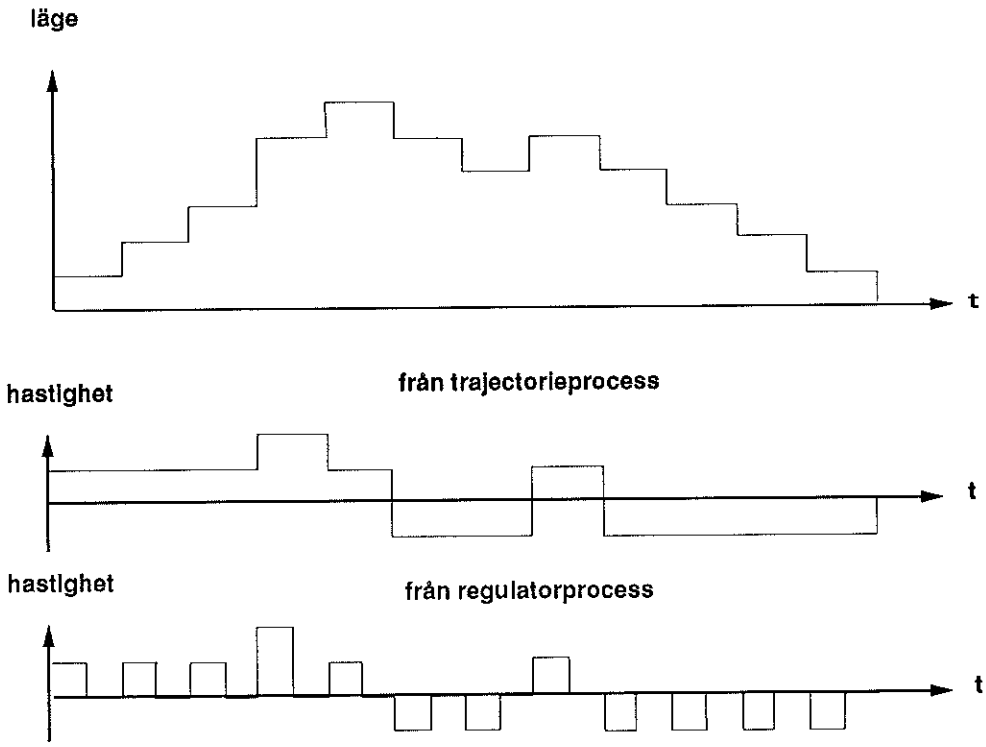

Figure 5.2 Exempel på skillnad i beräknad hastighet från trajektorieprocessen och från regulatorprocessen då regulatorn samplats dubbelt så snabbt som trajektorieprocessen.

jektorieprocessen för att tillåta olika samplingsfrekvenser i regulatorprocessen och trajektorieprocessen. I annat fall kommer framkopplingstermen endast att bli korrekt under de regulatorsampel då lägesreferensen förändrats. Ett exempel på detta visas i figur 5.2. För att kunna kommunicera med modulen finns ett antal funktioner och procedurer enligt följande:

- **SetReference:** Skickar nya läges och hastighetsbörvärden till regulatorerna.
- **SetRegulTime:** Sätter samplingstiden för Regulprocessen.
- **GetRegulTime:** Ger samplingstiden för Regulprocessen.
- **SetRegulPar:** Sätter regulaturparametrarna Kp och Kff för de fem regulatorerna.

5.2 Referensgenering

Modulen **Trajec** är systemets referensgenerator. Processen hämtar ett meddelande från en brevlåda. Meddelandet kan antingen vara ett trajektoriekommando eller ett metakommando. Metakommandona används för att kunna stanna roboten tillfälligt, pause, eller mer slutgiltigt, abort. Om ett trajektoriekommando erhålles så exekveras denna trajektoria tills den är avslutad om inte pause, abort eller terminate anropats under tiden. Vad som sker när dessa tre specialkommandona anropas, beskrivs närmare nedan.

- **Pause:** Den approximativt snabbast möjliga inbromsningstrajektorian genereras och exekveras. Därefter väntar processen på att antingen **Resume** anropas, då återstoden av den ursprungliga trajektorian exekveras, eller att **Abort** anropas. Notera att om pause anropas utan att det exekveras någon trajektoria måste det åtföljas av ett resume eller abort för att kunna fortsätta körningen av programmet.
- **Abort:** Om inte roboten står stillla genereras och exekveras snabbast möjliga inbromsningstrajektoria. Sedan kastas alla begärda förflyttningar. För att kunna skilja de förflyttningar som beräknats före abort anropats från de som beräknats efter, används Metakommandot som skickas ner genom systemet.
- **Terminate:** Exekvering avbryts direkt. Ingen inbromsning sker utan stopp slås direkt. Alla processer termineras också.

Förutom ovanstående kommandon tillhandahåller denna modul ett antal procedurer för att bl.a beordra robotrörelser. De viktigaste beskrivs nedan. Några av dessa anropas från andra processer. Detta medför en balansering av CPU-belastningen eftersom bl.a. trajektorieberäkningen kan utlokaliseras till andra processer.

- **CartGoto:** Försöker generera en trajektoria som utför en rätlinjig förflyttning från den senast beräknade slutpositionen till den kartesiska positionen, TargetCartPosition, på tiden InTime. Om InTime är för kort sättes kortast möjliga tid. Handen, gripper, kan även styras under rörelsen. Detta sker genom att gripPosition sättes till Open, Close eller NoMove. Om det skall ske någon förändring av handens tillstånd så sker det efter GripEventInPartOfInTime av den totala tiden för förflyttningen. GripEventInPartOfInTime sättes alltså till mellan noll och ett. Om trajektorian är tillåten blir OK sann och ett trajektoriemeddelande sänds till trajektorieprocessen, där sedan trajektorian genereras.
- **JointGoto:** Identisk med proceduren ovan förutom att förflyttningen sker i ledvinklar.
- **IncCartLagGoto:** Förändrar robotpositionen med incpos på tiden, Reftime, om slutpunkten ligger inom tillåtet område. Detta sker genom en rätlinjig kartesisk förflyttning. Här sätts framkopplingsfaktorn till noll. Denna och nästa procedur används vid manuell eller sensorbaserad styrning då rörelsen inte är känd på förhand, dvs. då framkoppling kan ske utan att införa extra fördröjning.
- **IncJointLagGoto:** Identisk med proceduren ovan förutom att förflyttningen sker i ledvinklar.
- **SetTrajecTime:** Sätter samplingstiden för trajecprocessen.
- **GetTrajecTime:** Ger samplingstiden för trajecprocessen.
- **SetMaxSpeedAcc:** Möjliggör inställning av maximal acceleration och maximal hastighet för de olika servona inom vissa gränser.
- **GetMaxSpeedAcc:** Ger maximala accelerationer och maximala

hastigheter för de olika servona.

- **GetCurrentPos:** Ger robotens aktuella position i motorvinklar.

Trajektorie generering

Det genereras två olika trajektorietyper i denna modul. Den ena är väldigt simpel och ser ut som i fig. 5.3. Denna typ användes i IncCartLagGoto och IncJointLagGoto. I detta fall låter man regulatorn korrigera det fel som uppstår.

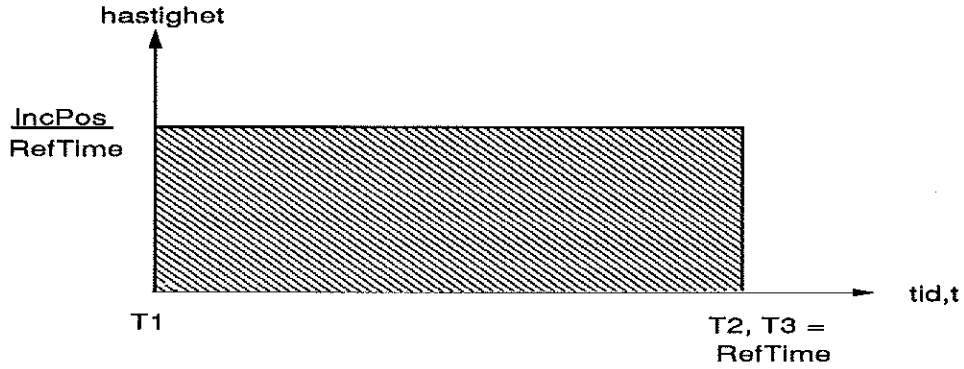

Figure 5.3 Trajektoria beräknad med IncCart eller IncJointLagGoto. Den sträckade arean motsvarar den sträcka eller vinkel som skall avverkas.

Den andra typen är lite mer sofistikerad, men fortfarande mycket enklare än de optimala metoder som finns i robotlitteratur. Man börjar med att bestämma ett antal hastighetsprofiler, se fig. 5.4,

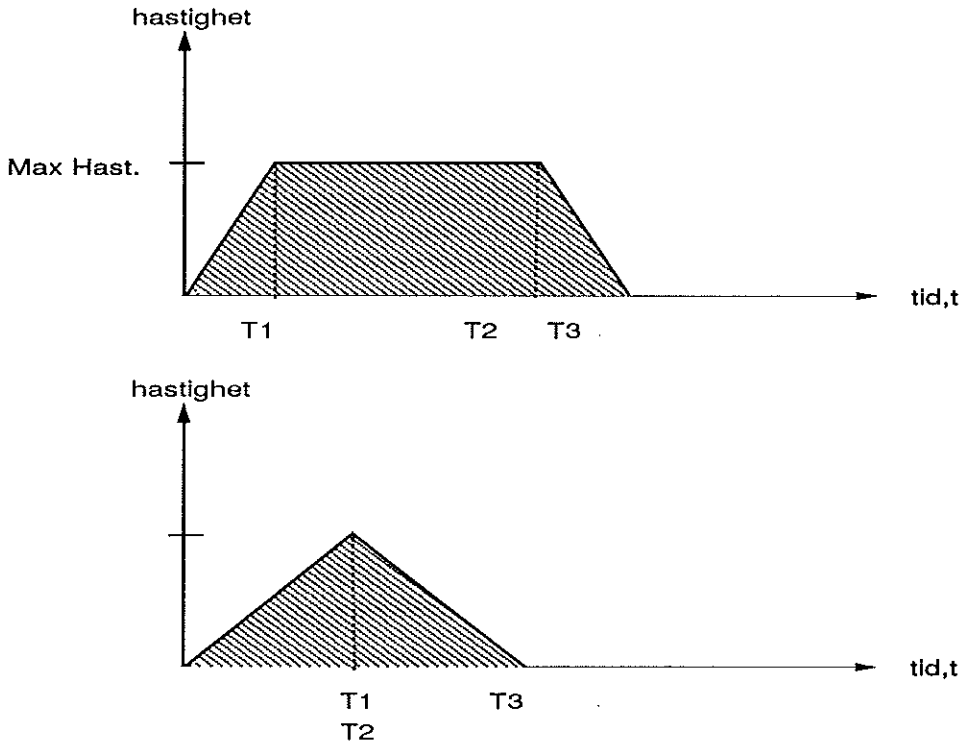

Figure 5.4 Två olika typer av Trajektorier beräknade med Cart eller JointGoto. Den streckade arean motsvarar den vinkel eller sträcka som skall avverkas.

där antalet är beroende av om man skall göra förflyttningen i karte-

siska koordinater eller ledvinklar. I ledvinklar används det fem profiler, en för varje led. I kartesiska koordinater behövs det tre stycken, en för ToolCenterPoints(TCPs) förflyttning och två för handens vridning, t4 och t5. Profilerna visar den kortaste tid som varje led behöver för att utföra sin del av rörelsen. Profilerna bestäms med utgångspunkt av den sträcka eller vinkel som skall avverkas, max acceleration och max hastighet för varje led. De två sistnämnda kan bestämmas av användaren upp till en viss nivå. Det som bestämmer denna nivå är motorernas prestanda. Ur dessa profiler väljs den med störst T3 till att vara den bestämmande eftersom det är den led som behöver längst tid på sig för att utföra sin del av rörelsen. Därför måste den sätta takten. När detta är klart kontrolleras om den tid man önskat utföra förflyttningen på är längre än ovannämnda T3. Är så fallet utförs en linjär skalning av tiderna T1, T2 och T3. Till slut justeras hastigheten, för de olika lederna, så att de beräknade hastighetsprofilernas överensstämmer med sträckorna som skall avverkas. Egentligen borde man även se till att ingen led överskred sin maximala acceleration. Detta har dock inte gjorts.

5.3 Kinematik och kontroll av arbetsområdet

Vid beräkningen av trajektorierna behöver man dels kunna byta koordinatsystem snabbt och dels avgöra om rörelsen håller sig inom robotens arbetsområde. Därför har procedurer för detta samlats i en speciell matematikmodul kallad **Kinematics**. En stor fördel med att denna modul är att den underlättar om man vill ändra tillåtet arbetsområde eller robotens fysiska data som t.ex. längden på robotens armar. Procedurerna som exporteras från Kinematics beskrivs nedan.

- **JointToCart:** Konverterar ledvinklar (t1 tom t5), som skall anges i radianer, till kartesiska koordinater (x, y, z, t4, t5) i millimeter och radianer.
- **CartesianToJoint:** Konverterar kartesiska koordinater till ledvinklar.
- **JointToActuator:** Konverterar ledvinklar till motorvinklar.
- **ActuatorToJoint:** Konverterar motorvinklar till ledvinklar.
- **InverseKinematic:** Konverterar kartesiska koordinater till motorvinklar.
- **Kinematic:** Konverterar motorvinklar till kartesiska koordinater.
- **CartPointCheck:** Kontrollerar om den kartesiska punkten är tillåten, dvs ligger inom robotens arbetsområde.
- **CartRouteCheck:** Kontrollerar om den kartesiska förflyttningen är tillåten.
- **JointPointCheck:** Kontrollerar om den i ledvinklar specificerade punkten är tillåten.
- **JointRouteCheck:** Kontrollerar om den i ledvinklar specificerade rörelsen är tillåten.

- **GetResetCoord:** Returnerar ledvinklar för reset positionen. I denna procedur är reset positionen definierad. Den går att ändra, om man så önskar.

5.4 Operatörs kommunikation

I modulen **Opcom** finns två processer, Matlabhandler och Compro, som tar hand om kommandon som kommer från operatören.

Matlabhandler

Denna process konverterar Matlab meddelanden, som kommer över Ethernet från värddatorn, till avkodade meddelanden för realtidsprogrammet. En del av dessa sänds vidare till Compro för vidare bearbetning där. Dessutom har matlabhandlern hand om bytet från Auto till Manual. En tredje uppgift denna process har är att kontrollera Pause, Abort och Resume. För närmare information om dessa se kap. 4.3 och 5.2. Vid anrop av Abort och Pause hänger denna process ända tills besked kommit att roboten verkligen står still.

Compro

Processen tar emot meddelanden i en brevlåda, commandmailbox, som tolkas och beroende på vilken typ de är av, utförs olika operationer. Dessa operationer är exporterade från berörd modul. På detta sätt behöver inte implementering av underliggande moduler vara känd. Här beräknas t.ex. trajektorierna eftersom detta, som nämnts tidigare, avlastar den högre prioriterade processen Trajec som har hand om den tidskritiska genereringen av trajektorierna.

5.5 Manuell förflyttning

Modulen **ManualMove** utnyttjar modulen **Sensorball** för att kunna köra roboten manuellt. Processen Manual läser sensorbollens värde och vilka knappar som är nertryckta.(För närmare beskrivning av dessa se kap. 3.2). Med ledning av detta utförs de operationer som är begärda. Dessa är exporterade från berörd modul. Här finns även monitorn som används för att uppnå ömsesidig uteslutning av de båda sätten att kontrollera robotens rörelse, alltså manuellt via sensorbollen eller automatiskt via skärmen. Om man utnyttjat enbart en semafor, eller andra standardmetoder för ömsesidig uteslutning hade man inte kunnat utföra vissa önskvärda operationer från skärmen under tiden som manuell har kommandot över robotrörelsen. Se kap. 4.2 för vidare information angående denna kontroll.

5.6 Plottning

Proceduren SendToPlotter i modulen **Plotter** används för att skicka referens och läge till ett av Sun-datorns Matlabfönster. Där tar matlab-macrot datregplot hand om utskriften av dessa värden.

5.7 Matlab-macron

Ett macro som råkat få namnet **loop3** startar det grafiska användargränssnittet i Matlab. Plottningen i Matlab sköts med **datregplot** för referens och läge och med **distanceplot** för lasern.

Rörelsesekvenser i Matlab

För att kunna köra en följd av trajektorier måste Matlab-macrot, som man vill köra, ha ett visst format. Macrot STest följer nu som exempel.

```
GripOrPosCol = 1;
modecol = 2;
xcol = 3;
ycol = 4;
zcol = 5;
t4col = 6;
t5col = 7;
timecol = 8;
Gripcol = 9;
enatcol = 10;
t1col = 3;
t2col = 4;
t3col = 5;

Sequence =
[0 0  45   0   0    0 0 70  -1 0.5;   % Ledvinklar
 0 0  90   0   0    0 0 50  1  0.5;   % Ledvinklar
 1 -1 0    0   0    0 0 0   0  0;     % grip
 0 1  1000 0   690  0 0 300 1  0.2;   % Kartesisk
 0 1  550  650 1200 0 0 100 0  0;     % Kartesisk
 0 1  930  0   1150 0 0 100 0  0;     % Kartesisk
 0 0  -135 0   0    0 0 50  -1 0.5;]; % Ledvinklar
```

Om första kolonnen sätts till en nolla blir kolonn två den som bestämmer vilket koordinatsystem som skall användas. En nolla ger kartesiska koordinater allt annat ger ledvinklar. Men om första kolonnen sätts till något annat än en nolla blir hela kommandot endast en order om att styra grippern, alltså sker ingen förflyttning. I så fall bestäms gripperns öppning och stängning med en etta resp. minus ett i kolonn två. Ovanstående förfarande kunde förenklas avsevärt om man utnyttjade något robotspråk

för detta. Detta har också gjorts efter det att jag avslutat mitt utvecklingsarbete.

5.8 LaserIO modulen

Laseravståndsmätaren som ingår i systemet är av märket OMRON Z4W-A29 med skyddsklass 3b. Detta innebär att lasern är skadlig för ögonen, vid direkt belysning. Därför har en lägesbrytare installerats. Denna bryter strömmen till lasern om den lutar mer än +/- 10 grader från vertikalt läge. Lasern är av halvledartyp och har ett mätområde mellan 45 och 55 mm. Nollnivån ligger på 50 mm. Kommunikation med avståndsmätaren sker med modulen **LaserIO** som jag tillfört systemet. Procedurerna i modulen är:

- **InitLaserIO:** Initierar ResolverIO. Anropa denna procedur endast om inte InitServoIO eller InitResolver, anropas.
- **Operate:** Sätter på och stänger av avståndsmätaren.
- **IsOperating:** Kontrollerar om avståndsmätaren är på eller avslagen. Om avståndsmätaren är avslagen ger nedanstående procedurer ingen relevant information.
- **Distance:** Returnerar avståndet i mm.Om man är inom mätområdet, +/- 5 mm, är avståndet relevant och exakt. Då returneras "OK". Utanför detta område, men innanför alarmgränserna, är avståndet relevant men kan vara inexakt, och "Sat" returneras. Utanför alarmgränserna returneras "Alarm" och avståndet sätts till noll.
- **Level:** Returnerar "far", "close" eller "alarm" beroende på avståndet lasern registrerar."Far" ges om avståndet är mellan 0 och 7 mm. Close ges om det är mellan -7 och 0 mm. Utanför dessa områden ges alarm.

6. Igångkörning av systemet

I detta kapitel beskrivs hur systemet skall vara kopplat och inställt för att man skall kunna köra igång systemet. Därefter vägleds man genom uppstarten av systemet.

6.1 Hårdvaruinställningar och kopplingar

För att kunna starta systemet krävs att det är kopplat och inställt på ett speciellt sätt. Därför måste man se till att följande punkter är uppfyllda:

- Sun-datorn är ansluten via port A till serieporten på VME-datorns CPU-kort.
- Ethernet-kortet är installerat och ansluten till nätet.
- Samtliga brytare till regulatorerna är i läge disable.
- Regulatorerna är inställda på PI-reglering.
- Det finns tryckluft till roboten.
- Nödstoppsknappen finns lätt tillgänglig.
- Strömmen är påslagen till alla enheter.

6.2 Uppstart av mjukvaran

När ovanstående punkter är genomförda skall följande operationer göras för att starta systemet :

1. Starta fyra fönster på värddatorn.
2. Gör cd till aktuell katalog i alla dessa fönster.
3. Starta MATLAB3 i två av fönstren.
4. Skriv SIMCOM i ett av de andra fönstrena.
5. Tryck på ABORT knappen på vme-kortet. Detta skall ge utskriften BUG 143 i simcom fönstret.
6. När den röda lampan på vme-kortet släcks skall m 900 000 skrivas i simcom fönstret.
7. Skriv sedan g fff20000 på samma fönster.
8. Välj nu det kvarvarande fönstret och skriv VMEGO vme/Main.sr.
9. Starta Operatörs kommunikationen genom att skriva loop3 i ett av matlab fönstrena.
10. Starta plottningen genom att skriva datregplot i det andra matlab fönstret.
11. Starta roboten genom att trycka på ROBOT OPERATE-knappen.
12. Slå om regulatorbrytarna till enable för de leder du skall använda.

13. Nu kan roboten köras antingen manuellt eller via skärmen. Man avslutar genom att klicka på Quit.

Punkterna 1 tom 10 kan med fördel läggas i ett script. Min handledare på institutionen har genomfört detta, och också uppdaterat detta för de kraftfullare datorkort som tagits i bruk efter detta arbetes utförande.

7. Sammanfattning

Programvara för en labmiljö bestående av bl a en robot och en värddator har utvecklats och implementerats. För att få en flexibel programstruktur har programmet uppdelats i olika moduler. Tanken har varit att varje modul skall kunna bytas ut, utan stora ingrepp i de andra modulerna. Programmeringen har utförts i Modula-2 tillsammans med en realtidskärna som utvecklats på institutionen. Ett gränssnitt till en yttre givare, en laseravståndsmätare, har även implementerats.

Roboten styrs från två enheter, antingen från värddatorns användargränsnitt eller från en styrspak kallad sensorboll. Detta har ställt speciella krav på kontrollstrukturen. Det största problemet har varit att ta hänsyn till alla realtidskrav så som olika samplings intervall av processerna och skydd av gemensamma resurser.

Förutom att prestanda och funktionalitet hos den utvecklade programvaran för realtidsstyrning har verifierats, är slutsatsen att labmiljön bestående av en värddator och programvaror som utvecklingsmiljö, väl integrerad med en fristående realtidsdator, utgör en utmärkt bas för utveckling av avancerade styr och reglersystem.

Fortsatt arbete

Banföljning med hjälp av laseravståndsmätaren, som jag har implementerat ett gränssnitt till, och lastadaptering är två exempel på tänkbara utvecklingar av mitt examensarbete. Detta har också genomförts som projekt inom kurserna realtidssystem och adaptiv reglering, under tiden som denna rapport färdigställts. Det har kommit en ny version av Matlab, Matlab-4.0, sen jag avslutade mitt utvecklingsarbete. Att uppdatera min programvara för denna version, skulle också vara önskvärt.

8. Referenser

BLIXT, B., EDBLAD, L-O. M.FL (1991): *Styrning av ASEA IRB-6*, Department of Automatic Control, Lund Institute of Technology, Sweden.

BRAUN, R., NIELSEN L. and K. NILSSON (1990): *Reconfiguring an ASEA IRB-6 Robot System for Control Experiments*, Department of Automatic Control, Lund Institute of Technology, Sweden.

CRAIG, JOHN J. (1989): *Introduction to robotics: mechanics and control 2nd ed.*, Addison-Wesley, USA.

KING, K.N. (1988): *Modula-2 A Complete Guide.*

MOTOROLA (1987): *MVEME133A-20 VMEmodule, 32-Bit Monoboard Microcomputer, Users Manual.*

NILSSON, P. (1991): *Programmoduler för styrning och identifiering av en industrirobot TFRT-5432*, Department of Automatic Control, Lund Institute of Technology, Sweden.

PEP MODULAR COMPUTERS (1988): *VDAD, Universal Analog to Digital / Digital to Analog I/O Module for the Vmebus, Users Manual.*

THE MATHWORKS INC (1987): *Pro-Matlab.*

ÅSTRÖM, K. J. and B. WITTENMARK, (1990): *Computer-Controlled Systems, Theory and Design 2nd ed.*, Prentice-Hall, USA.