



UPPSALA UNIVERSITET

Rapport Tillämpad Mekanik II

Hampus Björklin
Axel Djurberg
Fredrik Forsberg
Oscar Jacobson

29 december 2022

Innehåll

1	Sammanfattning	1
2	Introduktion	2
3	Labb 1	3
3.1	Uppgift	3
3.2	Systemets uppbyggnad	3
3.3	Metod och utförande	7
3.4	Resultat	9
4	Labb 2	10
4.1	Uppgift	10
4.2	Systemets uppbyggnad	10
4.3	Metod och utförande	11
4.4	Resultat	13
5	Allmän diskussion om båda labbarna	14
6	Utvärdering	19
7	Appendix	20

1 Sammanfattning

Följande rapport är baserad på två laborationer inom ämnet robotik och de styrssystem som kan användas för att kontrollera robotar. De två laborationerna utgick från den relativt enkla uppgiften att plocka upp tre olika cylindrar från ett löpande band och placera dem i respektive tilldelat cylinderställ. Detta gjordes både online med direkt kontroll över en industrirobot och offline simulerat i programmet ABB Robotstudio med hjälp av en dator.

Rapporten visar på tillvägagångssätt, svårigheter och förbättringsider för att lösa denna uppgift samt skillnader och likheter för de olika situationerna och programmeringsspråken. I rapporten ingår även diskussioner kring viktiga delar av robotiken som gör denna prestation möjlig. Detta inkluderar industrirobotens olika maskinelement samt samspelet mellan dessa, samspelet med andra robotar och hur robotten kan beordras till att utföra väldigt precisa rörelser med hjälp av relativa koordinatsystem, referenspunkter och tillämpade matematiska satser. Laborationen upplevdes som lärorik och rolig trots några oväntade förhinder.

2 Introduktion

Programmerbara robotar har blivit väldigt vanliga inom industrin, de kan effektivisera repetitivt arbete och utför tunga lyft så att människor slipper. En robot kan också enkelt byta mellan olika uppgifter då de endast behöver köra annan kod eller programmeras om. Dessutom kan vi simulera robotar i datorprogram kan vi enkelt skriva nya program och testköra utan att behöva stoppa pågående produktion.

I denna rapport presenteras resultatet av två laborationer där vi undersökt närmare hur robotarna fungerar och vad de kan användas till samt hur processen från simulering och programmering till fungerade robot ser ut.

3 Labb 1

3.1 Uppgift

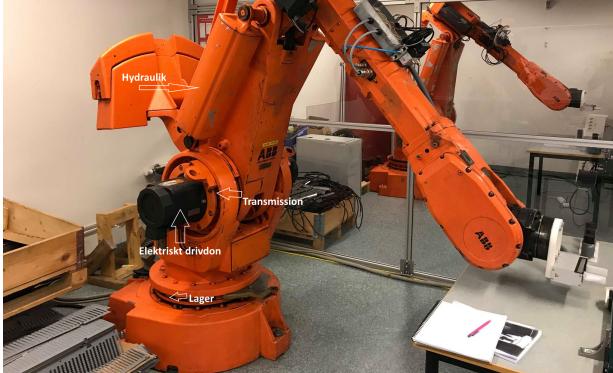
I denna laboration kommer vi prova använda en verlig industrirobot för att sortera cylindrar av olika storlekar. Tanken är att roboten ska plocka upp en cylinder från ett rullband, känna av vilken storlek den har och placera den på rätt ställe. För att roboten ska fungera som tänkt behöver vi undersöka hur programmeringen av systemet går till samt hur man hanterar in och utgångssignaler.

3.2 Systemets uppbyggnad

I denna laboration använder vi oss av en sex-axlig robot från ABB (modellen ABB IRB6000 M91A), roboten liknar de flesta robotar som används i industrier oavsett tillverkare. Den här typen av robotar påminner om en människas arm, där manipulatorn motsvarar handled, över och underarm. På manipulatorn kan vi fästa någon typ av verktyg, i detta fall ett gripdon, ungefär som vår hand. Tre av robotens sex axlar används för att förflytta "handleden" till önskad position och resterande tre används för att vrinda handleden i olika riktningar så verktyget hamnar rätt. För att robotens rörelser ska ha hög precision används återkoppling till robotens stryrdon. Med återkoppling blir systemet dynamiskt och beror inte endast på styrsignalen, utan även på den nuvarande uppmätta positionen.

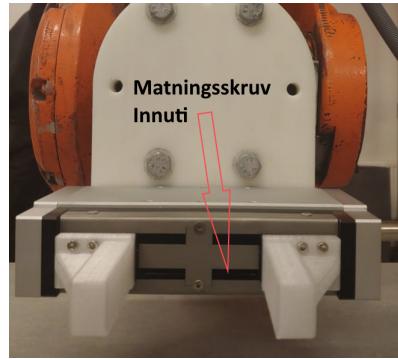
Manipulatorn består, som de flesta andra mekaniska konstruktioner, av många olika maskinelement. Lager för att minska friktion vid rörelser, transmissioner för att kontrollera krafter och hastigheter, hydrauler för dämpning och elektriska drivdon för rörelse, för att nämna några. Manipulatorns rörelser drivs främst av elektriska motorer. Dessa driver sedan en axel som i sin tur driver transmissionerna. Här är det planetväxlar och dess solhjul som drivs. Planetväxlar består av tre lager kugghjul, dessa är det redan nämnda solhjulet i centrum med utvändiga kuggar, ett ringhjul med invändiga kuggar och planethjul som sitter mellan sol- och ringhjul. Planethjulen sitter alla fast i en så kallad medbringare som roterar kring samma punkt som sol- och ringhjul. Som sagt drivs robotens rörelse av elektriska motorer vilka oftast har för hög rotationshastighet och för litet vridmoment för robotens önskade rörelse. Således i transmissionerna till manipulatorns leder krävs utväxling för att anpassa hastigheten och belastningen. Utväxlingen (i) är en avvägning mellan rotationshastighet (ω) och vridmoment (T) som beskrivs av formeln $i = \frac{T_{ut}}{T_{in}} = \frac{\omega_{in}}{\omega_{ut}}$. Som beskrivet ovan används främst planetväxlar vilka utövar en nedväxling mellan den snabbt roterande motorn till en rotationshastighet som är önskad för styrningen av manipulatorleden. I och med nedväxlingen av rotationshastigheten ökar vridmomentet med samma faktor som rotationshastigheten minskar. En önskad rotationshastighet skulle även kunna uppnås med att rotera motorn mycket långsamt. Detta skulle dock ge ett mycket lägre vridmoment jämfört med en snabb rotationshastighet

med nedväxling till önskade rotationshastigheten. Nedväxlingen är således nödvändig för att manipulator exempelvis ska kunna lyfta tung last. Det omvända förhållandet gäller även, uppväxling där rotationshastigheten ökar således vridmomentet minskar. I manipulatorn är det dock nedväxling som främst används.



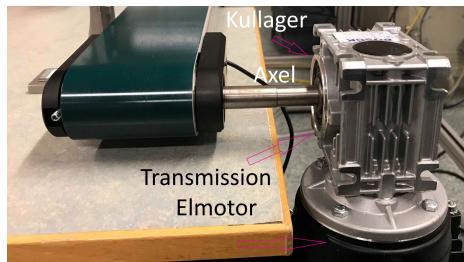
Figur 1: Exempel på några maskinelement i roboten

Gripdonet som användes bestod också av vanliga maskinelement. En matningsskruv som drivs av en elmotor används för att klämma. Matningsskruven består av en fast mutter, eller liknande, som kommer att glida på gångorna i skruven när den roteras. På vårt gripdon har man satt plastbitar på dessa muttrar vilket får dem att röra sig mot varandra när skruven roteras. Här används också återkoppling för att veta när matningsskruven ska sluta mata. Detta är viktigt då vi i förväg inte vet hur långt vi behöver mata då vi inte vet cylindrarnas storlek. Med hjälp av återkopplingen kan vi bestämma storleken också, vilket möjliggör sorterings. Återkopplingen fungerar genom att strömförbrukningen i motorerna övervakas. När cylindern grips bromsas motorn vilket får förbrukningen att öka. Detta är programmerat att tolkas som en signal att stoppa motorn. Med hjälp av en rotationssensor kan vi dessutom hålla koll på hur många varv som skuren har vridits, detta gör att vi kan avgöra avståndet mellan klorna och sedan cylinderns diameter.



Figur 2: Matningsskruven får de vita plastbitarna att åka ihop. Återkopplingen styr när drivdonent stannar

Fler exempel på maskinelement hittar vi i drivningen av bandet vi plockar upp cylindrarna ifrån.



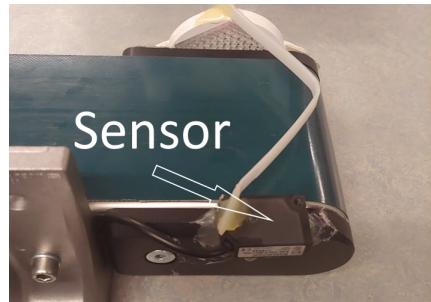
Figur 3: Några av de maskinelement vi hittar på bandet.

Detta system består utav en elektrisk motor samt en snäckväxel som vrider rotationen 90° . Motorn är monterad vinkelrät mot bandet, vilket sparar på plats. Snäckväxeln består av ett kugghjul som ligger i skåror på en matningsskruv som sitter på motorns axel. Detta gör att snäckväxeln även ger en nedväxling av motorns rotation. Den utgående vinkelhastigheten för bandet är lägre än den ingående vinkelhastigheten från motorn samtidigt som det utgående vridmomentet är högre.

Mellan bandet och transmissionen har vi en fast axel. Hade detta varit ett system med större krafter eller rotationshastigheter hade det varit väldigt viktigt att axel varit rak och monterad rätt för att undvika vibrationer. Om det av olika anledningar är svårt att uppnå en rak drivaxel kan en bälchkoppling varit ett bra alternativ, då den kan böjas och därav tål visst monteringsfel.

Återkoppling används även på bandet. En sensor märker när en cylinder har nått läget där den ska bli upplockad av roboten och stänger av bandet så att

cylindern stannar.



Figur 4: Sensor som känner av när bandet ska stanna.

Bandet kontrolleras av samma styrenhet som roboten, dvs. samma program som styr roboten ska även se till att bandet slås på och av vid rätt tillfällen.

3.3 Metod och utförande

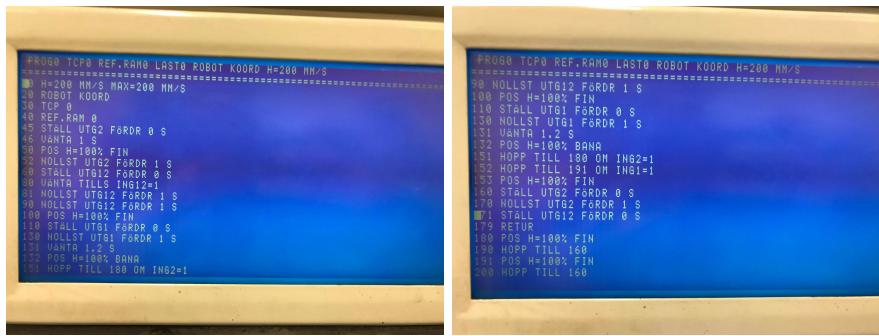
Det första som görs är att definiera robotens hastighet och koordinatsystem i koden. Vi valde att använda det förinställda koordinatsystemet och valde den maximala hastigheten 200 mm/s. För att köra roboten manuellt och programmera den använder vi kontroll-dosan. Roboten körs till olika positioner som sedan sparas och används i koden. Exempel på detta finns på rad 50 i figur 5 nedan. Kommandot POS flyttar roboten till den sparade positionen. När koden kompileras skapar styrenheten en bana som roboten ska följa för att nå positionen, om man vill undvika att kollidera med något som ligger mellan nuvarande position och dit roboten är på väg kan man spara en extra punkt som roboten ska åka till först. Om man ej valt hög noggrannhet "fin" kommer roboten dessutom börja med nästa instruktion en stund innan den pågående är klar, vilket leder till att roboten kommer att ta små mjuka genvägar förbi punkter.

Vi började med att spara en position i närheten av där cylindern kommer stanna på bandet. Vi såg även till att skicka en signal att öppna gripdonet, ifall det var stängt. Roboten väntar där tills vi har en "etta" på ingången från sensorn på bandet som känner av när en cylinder är redo att plockas upp.

När en cylinder är på plats skickas en signal att bandet ska stängas av samt att roboten ska föras fram så den är redo att gripa cylindern.

När roboten är i rätt position skickas en signal att först klämma med gripdonet och sedan lyfta.

Gripdonet har tre olika utgångssignaler, en för varje storlek på cylindern. När roboten lyft cylindern körs olika bitar av koden beroende på vilken utgångssignal som är "etta". Detta görs med hjälp av HOPP kommandot. När roboten är framme släpps cylindern och programmet upprepas.



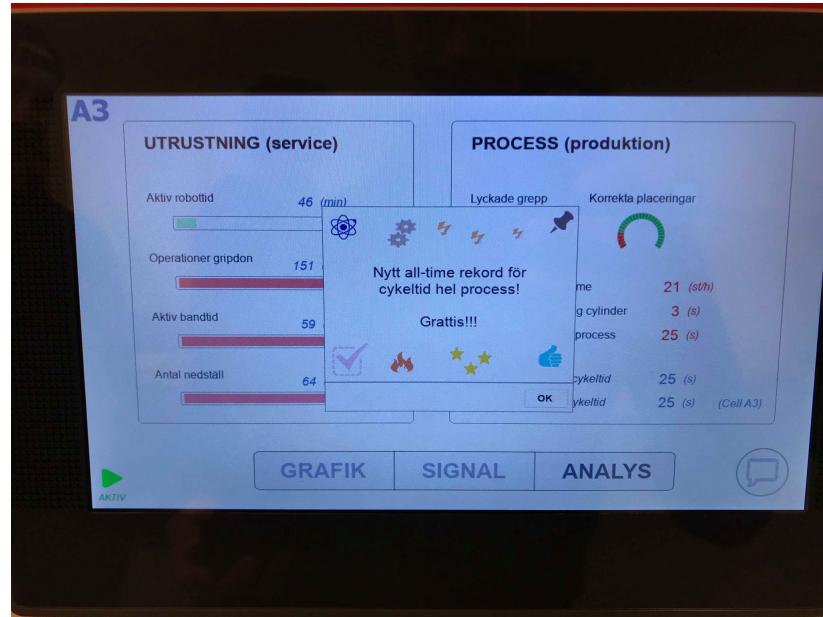
Figur 5: Slutgiltig kod som användes

Nedan följer koden i figur 5. för förtydligande:

```
PROGO TCPO REF.RAMO LASTO ROBOT KOORD H=200 MM/S
=====
10 H=200 MM/S MAX=200 MM/S
20 ROBOT KOORD
30 TCP 0
40 REF.RAM 0
45 STÄLL UTG2 FÖRDR 0 S
46 VÄNTA 1 S
50 POS H=100% FIN
52 NOLLST UTG2 FÖRDR 1 S
60 STÄLL UTG12 FÖRDR 0 S
80 VÄNTA TILLS ING12=1
81 NOLLST UTG12 FÖRDR 1 S
90 NOLLST UTG12 FÖRDR 1 S
100 POS H=100% FIN
110 STÄLL UTG1 FÖRDR 0 S
130 NOLLST UTG1 FÖRDR 1 S
131 VÄNTA 1.2 S
132 POS H=100% BANA
151 HOPP TILL 180 OM ING2=1
152 HOPP TILL 191 OM ING1=1
153 POS H=100% FIN
160 STÄLL UTG2 FÖRDR 0 S
170 NOLLST UTG2 FÖRDR 1 S
171 STÄLL UTG12 FÖRDR 0 S
179 RETUR
180 POS H=100% FIN
190 HOPP TILL 160
191 POS H=100% FIN
200 HOPP TILL 160
```

3.4 Resultat

Roboten fungerade som den skulle och den kunde sortera cylindrarna till rätt koppar, oavsett ordning de kom på bandet. vilket var målet. Den bästa cykeltiden vi uppnådde var 21 sekunder, ett rekord!



Figur 6: Resultat efter optimering

4 Labb 2

4.1 Uppgift

I denna laboration 2 var uppgiften likt laboration 1 att sortera tre stycken cylindrar med olika storlekar från ett löpande band, känna av storleken och placera dem på rätt ställe. I denna laboration utfördes dock uppgiften med moderna industrirobotar i simuleringssverktyget ABB Robotstudio med programmeringsspråket ABB RAPID. Cylindrarna är förprogrammerade med storlekarna S, M och L vilka gripdonet känner av.

4.2 Systemets uppbyggnad

I ABB Robotstudio erhölls en förberedd virtuell robotcell lik den verkliga i labb 1. Cylindrarna var förmärkta med S, M och L (relaterat dess storlek i labb 1). Genom att manuellt ge utsignal kommandot DO11, DO12 eller DO13 startas bandet och respektive cylinder för kommandot åker fram. När cylindern passerat en sensor i slutet av bandet känner den av cylinderns märkning genom att ansätta insignalen DI1, DI2 eller DI3 för respektive storlek och bandet stängs automatiskt av.

Koordinatsystemet relaterat till robotcellen är den referensram som alla andra koordinatsystem relaterar till. Sedan definieras ett "WorkObject" koordinatsystem (wobj0) vilket sätts till robotens bas. Det är detta referenssystem som alla definierade punkter relaterar till. Ett koordinatsystem definieras även vid gripdonets mittpunkt, ett TCP koordinatsystem, vilket är relativt wobj0.

Genom att markera en av manipulatorns olika länkarmar kan man styra denna beroende på vad för led den är kopplad till. Genom att markera TCP koordinatsystemet kan denna förflyttas godtyckligt i förhållande till wobj0 förutsatt att manipulatoren tillåter denna rörelse.

TCPs position kan sparas som ett Target där TCPs position tillsammans med manipulatorns ledkonfiguration sparas. Varje Target innehåller informationen position, orientering, robotkonfiguration och extern axel. Roboten kan sedan röra sig mellan dessa sparade Targets i vald ordning.

I Rapidkoden kan MoveL användas för att förflytta mellan valda Targets med en linjär rörelse. I MoveL kan förflyttningshastigheten och noggrannheten definieras. Med standard noggrannhet (z100) kan robotten påbörja nästa kommando lite innan punkten är uppnådd, vilket kan resultera i en rörelse där TCP tar små genvägar mellan punkterna. Genom noggrannheten "fine" säkerställs att TCP uppnår de definierade punkten.

I Rapidkoden manipuleras gripklon genom kommandon DO1 (stänga klon) och DO2 (öppna klon). Genom kommandot "SET" kan dessa utföras (en "1" ges) följt

av kommandot "RESET" (för att ge "0"). Kommandot "GOTO" används för att hoppa till en del i koden som är definierad av en variabel efter GOTO-satsen.

4.3 Metod och utförande

Denna uppgift inleddes likt labb 1 med att först definiera de nödvändiga punkterna som Targets. I denna uppgift var vi till största del intresserade av gripdonets mittpunkt (TCP) dvs. robotmålet var mest en fråga om TCPs position och robotens axelkonfiguration var inte lika relevant eftersom rummet var relativt fritt. Dock fick roboten inte kollidera med bordet eller liknande, således krävdes en viss hänsyn till axelkonfigurationen.

Först sparades position (p1) där gripdonet befann sig framför bandet. Då med en fungerande axelkonfiguration. Sedan för att uppnå resterande punkter förändrades endast TCPs koordinatsystem och genom detta behölls en så lik axelkonfiguration som möjligt, vilket var fungerande för alla punkter. Sedan sparades ytterligare Targets: med TCP där klossen stannade på bandet (p2), en punkt lite ovanför p2 (p3), tre punkter ovanför respektive ställe där cylindrarna skulle placeras (ös, öm, öL) och punkterna i respektive ställe (is, im, iL).

I koden användes MoveL för att få en linjär rörelse mellan de sparade Targets. Hastigheten "v1000" användes för alla rörelser. För punkterna p2, ös, öm och öL användes noggrannheten "fine". Dessa lägen var kritiska, p2 där cylindern skulle plockas upp. Här användes även kommandot "Waittime" med 1 s både innan och efter klon stängdes med kommandot "Set DO1". Följt av "RESET DO1". Sedan används kommandot "IF" och "ELSEIF" beroende på vilken insignal som får en "1" av sensorn (beskrivet i sektion 4.2).

Exempelvis i IF-satsen om insignalen är för DI1 innebär det att det är cylinder s som klon har plockat upp. Den förflyttar sig till position p3 och sedan ös. position ös (och öm, öL) har noggrannheten "fine" eftersom det var viktigt att denna position uppnåddes innan positionerna i ställena uppnåddes. Annars kunde cylindern kollidera med cylinderställets kanter. Kommandot "WaitTime" (1s) användes innan klon öppnades med kommandot "Set DO2" följt av "RESET DO2". Detta för att säkerställa att punkterna i stället uppnåtts. Liknande kod användes för resterande cylindrar (insignalerna DI2 och DI3) efter en ELSEIF-sats. Koden avslutades med en "GOTO a" och ett a skrevs först i koden för att koden skulle börjas om på nytt efter att en cylinder placerats.

Nedan följer den färdiga koden. De tre punkterna illustrerar en kodbit som ej skrivits ned, vilken återfinns i Appendix. Det är de sparade punkterna (p2, p3, ös, is, öm, im, öL, iL) vilka alla är på samma form som den illusterade punkten p1.

```

MODULE Module1
  CONST robotarget p1:=[[1616.955986384,-296.507382971,
    875.807830622],[0.68724615,0.000000837,0.726424621,
    -0.00000291],[-1,-3,2,0],
    [9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    •
    •
    •

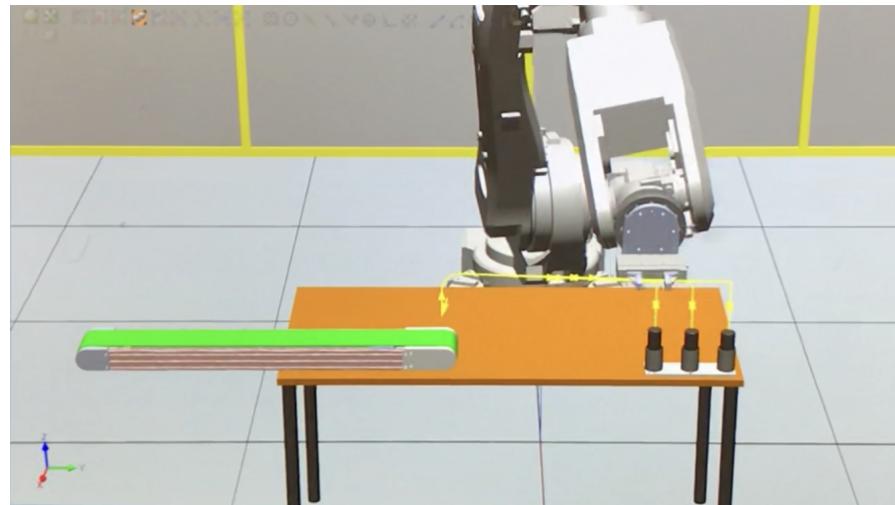
PROC main()
  a:
    MoveL p1,v1000,z100,Gripper_Tool_1\WObj:=wobj0;
    WaitDI DI12, 1;
    MoveL p2,v1000,fine,Gripper_Tool_1\WObj:=wobj0;
    Waittime 1;
    Set D01;
    Waittime 1;
    Reset D01;
  IF DI1 = 1 THEN
    MoveL p3,v1000,z100,Gripper_Tool_1\WObj:=wobj0;
    MoveL ös,v1000,fine,Gripper_Tool_1\WObj:=wobj0;
    MoveL is,v1000,z100,Gripper_Tool_1\WObj:=wobj0;
    WaitTime 1;
    Set D02;
    Reset D02;
    MoveL ös,v1000,fine,Gripper_Tool_1\WObj:=wobj0;
  ELSEIF DI2 = 1 THEN
    MoveL p3,v1000,z100,Gripper_Tool_1\WObj:=wobj0;
    MoveL öm,v1000,fine,Gripper_Tool_1\WObj:=wobj0;
    MoveL im,v1000,z100,Gripper_Tool_1\WObj:=wobj0;
    WaitTime 1;
    Set D02;
    Reset D02;
    MoveL öm,v1000,fine,Gripper_Tool_1\WObj:=wobj0;
  ELSEIF DI3 = 1 THEN
    MoveL p3,v1000,z100,Gripper_Tool_1\WObj:=wobj0;
    MoveL öL,v1000,fine,Gripper_Tool_1\WObj:=wobj0;
    MoveL iL,v1000,z100,Gripper_Tool_1\WObj:=wobj0;
    WaitTime 1;
    Set D02;
    Reset D02;
    MoveL öL,v1000,fine,Gripper_Tool_1\WObj:=wobj0;
  ENDIF
  GOTO a;

ENDPROC
PROC Path_10()
ENDPROC
ENDMODULE

```

4.4 Resultat

Den simulerade roboten fungerade som den skulle och kunde sortera cylindrarna till rätt koppar, oavsett ordning de kom på bandet, vilket var målet. Cykeltiden som uppnåddes var cirka 16 sekunder.



Figur 7: En färdig cykel, roboten har sorterat alla cylindrar

5 Allmän diskussion om båda labbarna

Programmeringsspråket, ARLA, var inte så likt de andra språk som gruppen har erfarenheter med. Det kändes omodernt och det var knöligt att lägga till eller ändra rader. Det hade varit intressant om det nyare RAPID-språket fanns tillgängligt vid en fysisk maskin så att man kunnat jämföra skillnaden. Cykeltiden som uppnåddes var snabb men tanke på att roboten var i manuellt läge, det vill säga att toppfarten var 200 mm/s. Genom att säkra salen och låsa in robotcellen så kan automatiskt läge slås på. Detta gör det möjligt att öka topophastigheten markant, och således öka produktionshastigheten. Roboten som användes var väldigt stor med avseende på den uppgift som tillgavs. Istället skulle en mindre robot kunnat användas för att utföra samma arbete samtidigt som den drar mindre ström. En mindre robot är dock inte lika flexibel då det kommer till att utföra olika typer av arbeten. En större maskin kan utföra en bredare grad av uppgifter än en liten, exempelvis så krävs det ibland att stora laster flyttas, eller att objekt ska förflyttas längre sträckor. Ett problem som uppstod under laborationen var att bordet rörde sig vid en kollision. Om bordet satt fast i marken hade detta inte varit ett problem om roboten hade haft ett automatiskt nödstopp. Det går helt klart att implementera, men det är också en stor kostnad, vilket kan kännas överflödigt då experiment lätt kan utföras i RobotStudio utan risk för skador på egendom.

Det finns många likheter mellan det nyare RAPID-språket och ARLA-språket. Det nyare programmeringsspråket RAPID är ett modernare språk där logiska uttryck finns tillämpliga, t.ex. if-satser. I det äldre programmeringsprogrammet som användes vid laboration ett användes istället kommandot GOTO som används för att hoppa mellan rader. Det nyare språket liknar ett avancerat programmeringsspråk medan det äldre är ett mer klassiskt maskinspråk. Under laborationen märktes att det nyare språket har betydligt fler funktioner. I RAPID fanns fler Move-kommandon, t.ex. MoveL och MoveC, medan i ARLA fanns det enbart ett. Under denna grundliga laboration så var det inga märkbara skillnader och båda språk kunde tillämpas för att uppnå målen.

De sparade positionerna i ARLA-språket gick inte att namnge, till skillnad från Rapid-språket. Detta gjorde att vi var tvungna att memorisera de olika punkterna. Efter en längre passerad tid krävdes ibland att manuellt gå till punkterna med roboten för att säkerställa att rätt punkt användes, exempelvis när HOPP-satser infördes i slutskedet av programmeringen.

Offlinesimulering är särskilt fördelaktigt då man vill experimentera eller hitta nya lösningar till problem med tanke på att det är ett sandlåde-läge. Det är enklare att få en översikt i hur man vill att systemet ska bete sig och man kan testa sig fram enklare utan att riskera att förstöra något. För att få en bekräftelse för att offlinesimuleringen fungerar online så krävs det att man testar det online. Det är många faktorer som påverkar onlinesimuleringen och sannolikt kommer den att skilja sig från offlinesimuleringen. Här krävs det att man simu-

lerar och programmerar online, så att allt funkar i praktiken. I onlinesimulering så får man en mer praktiskt överblick och man ser allt i realtid med en riktig maskin. Problem som uppstår är enkla att förstå då det inte är en simulering i ett program, allt händer i ”verkligheten”. Om man ska ändra något i en maskin som redan används till exempelvis produktion, så blir det olönsamt att inte ha maskinen igång. Därför är det bra om man redan har en idé hur man ska implementera den nya koden och då är offlineläget en bra start.

För denna laboration användes linjär rörelse genom MoveL. Noggrannheten av denna rörelse kan ändras, t.ex. användes vid laborationen ”fine” för att få en noggrann rörelse vid känsliga lägen. Nackdelen till att alltid använda den bästa noggrannheten är att maskinen slits i större grad än när man använder lägre noggrannhet. Detta eftersom att robotens rörelser blir mer hackiga vilket sliter mer på lederna. Dessutom en lägre noggrannhet med genvägar i rörelsen leder till en snabbare cykeltid. Således är lägre noggrannhet fördelaktigt där högre inte är nödvändigt.

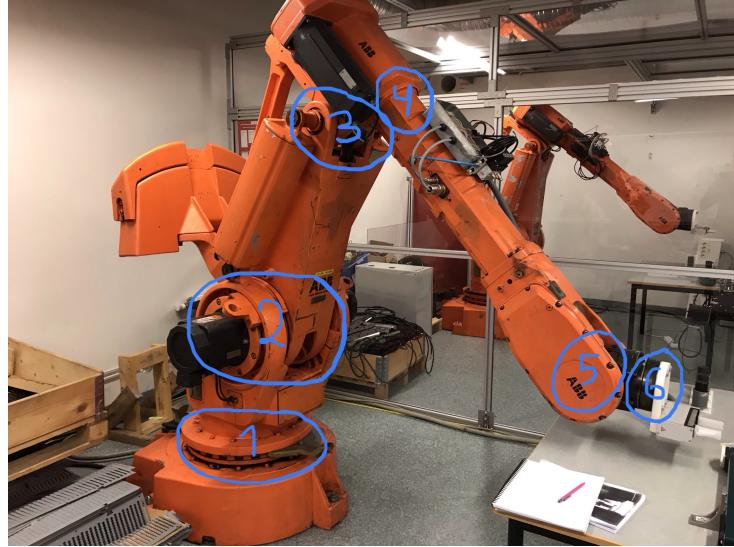
Istället för linjär rörelse kan man t.ex. använda MoveC, vilket är en cirkulär rörelse. Detta underlättar just då man vill ha en cirkulär rörelse, vilket kan vara komplicerat att uppnå genom MoveL. Vill man t.ex. undvika en kollision så kan MoveC användas tillsammans med ett ToPoint kommando samt ett CircPoint kommando, där CircPoint är en punkt som kommer vara en del av den cirkulära rörelsen, och ToPoint slutpositionen. Detta ger en jämnare rörelsegång i jämförelse med att använda flera MoveL-kommandon. Ytterligare är det kortare kod vilket oftast är fördelaktigt.

Som beskrivet för labb 2 ställdes roboten med en acceptabel axelkonfiguration på punkten p1, sedan förflyttades endast TCP för att uppnå de andra positionerna. Rörelsetypen handlade således om PTP-styrning (Point-To-Point). Robotmålet var fokuserat på Target-punkterna men eftersom dessa punkter var relativt nära och okomplicerade translationer för TCP så gjorde manipulatorn inga större förändringar i axelkonfigurationen eftersom den tog den kortaste och jämnast möjliga justeringen. Detta kan ses i informationen som beskriver robotens konfiguration för olika Targets (se appendix). Konfigurationen beskrivs i tredje vektorn vilket kan observeras vara densamma för p1, p2 och p3 samt vara samma för de resterande punkterna. Konfigurationen för de olika Targets kan dock enkelt ändras genom att att högerklicka på robotmålet, välja ”configuration” och ändra dess konfiguration. Då kan ses att de lagrade värdena ändras.

Roboten som arbetades med var en klassisk industrirobot med sex stycken axlar och dess sammansättning av axlarna ger den sex stycken frihetsgrader. Således kan roboten utföra både translation (räta linjig rörelse) längs XYZ i ett kartesiskt koordinatsystem och rotation kring XYZ.

Om man tittar på ett robotmål består det både av TCPs läge, dvs. dess position och orientering, men även manipulatorns axelkonfiguration. Ett och samma

läge för TCP kan oftast uppnås med olika axelkonfigurationer. Roboten som arbetades med illustreras i bilden nedan (figur 8) och dess sex axlar är markerade.



Figur 8: Roboten med dess ledar markerade

Den positionen som roboten befinner sig i (i figur 8) kan jämföras med position p1. Samma läge för TCP skulle kunna uppnås genom att axel 2 vrider nedåt med samma vinkel, axel 3 bildar samma vinkel åt andra hållet och axel 5 justeras så att samma läge för TPC uppnås. DVS. roboten skulle ha liknande form men omvänt så att axel 3 pekar nedåt. Problemet med detta skulle vara att roboten förmögligen skulle slå i bordet och positionen skulle inte kunna uppnås för att marken är i vägen (robotens del vid axel 3 skulle slå i marken). Således skulle detta inte vara en fungerande axelkonfiguration i denna robotcell.

Således för att skapa dessa olika robotmål i robotstudio kan man direkt grafiskt förflytta TCP och även dess olika ledar för att uppnå olika axelkonfigurationer. Denna kan även modifieras som beskrivit ovan genom ”configurations”. Dessa robotmål sparas sedan. Som beskrivits i sektion 4.2 består de sparade robotmålen i RAPID-koden av position, orientering, robotkonfiguration och extern axel. Således kan olika robotmål skapas/modiferas genom att direkt skriva in dem i koden. Detta är dock mycket omständligt. Exempelvis som beskrivs senare i denna sektion lagras orienteringen som kvarternioner vilket inte är så intuitiva och således svåra att skriva in direkt. Rotationen kan dock modifieras likt konfigurationen genom att högerklicka targets och ändra dess Tait-Bryan vinklar (Mer beskrivet om detta nedan).

I robotmålen är således axelkonfigurationen en viktig parameter för att anpassa

manipulatorn till omgivningen. Som nämnt var axelkonfigurationen mycket liknande för alla våra sparade punkter och mellan dessa punkter rörde sig roboten på att sådant sätt för att få en jämn konfigurationsövergång. Det kan dock vara mycket nödvändigt att manipulatorn rör sig på ett visst sätt mellan punkterna. Som beskrivet tidigare tog vi till största hänsyn till slutlägena och det handlade om PTP-styrning. Styrning för hur manipulator rör sig mellan lägena kallas för CP-styrning. Detta kan vara avgörande för att robotar ska kunna utföra sin uppgift utan att exempelvis kollidera med något. Exempelvis i en stor industri där många robotar måste samverka på en liten yta är CP-styrning en nödvändighet, att mer noggrant styra alla rörelser. Att tillägga är dock att vi använder oss av "MoveL" vilket innebär som sagt att roboten följer en linjär bana och är definierad som en CP-rörelse. Således använde vi oss av CP-styrning även om det i princip hade räckt att ta hänsyn till PTP-styrningen.

Ett problem inom robotik som också kan uppstå för vår robot är så kallad singularitet. Detta innebär att ledar som verkar på samma vis är parallella. Detta resulterar i att frihetsgrader kan tappas. Exempelvis om roboten studeras i figur 8. Både led 4 och led 6 är roll-leder som utför liknande rotation. Om led 5 är vinkelrakt ut resulterar detta i att led 4 och 6 är parallella. Eftersom de ger upphov till samma rörelse och är parallella så tappas en frihetsgrad. Detta kan leda till att vissa rörelser blir omöjliga för den definierade banföljningen. Vid programmering krävs således en viss kännedom om robotens mekanik.

Som sagt hade roboten definierats med ett koordinatsystem i dess TCP (mittpunkten av gripdon). Denna relaterar till koordinatsystemet wobj0, vilket var definierat vid robotens bas. punkten tool0 är den punkt där verktyget monteras. Således är verktygskoordinatsystemet förskjutet i förhållande till TCPs koordinatsystem. Som beskrivits i sektion 4.2 innehåller punkterna bl.a. information om position och orientering. Detta innebär exempelvis om tool0 koordinatsystem utsätts för en rotation ändras endast dess orientering medan TCP koordinatsystemet kommer en förändring både i orientering och position. Informationen extern axel kommer även få värden om ett verktyg monteras. Det är denna förskjutning som innebär den externa axeln. Utan ett monterat verktyg så att ett verktygskoordinatsystem blir definierat innehåller informationen av externa axel i RAPID-koden inga värden. Det kan ses att informationen för extern axel är samma för alla sparade Targets (i appendix), vilket är förväntat eftersom ett och samma verktyg användes.

Som beskrivet i sektion 4.2 lagras informationen om ett Target som: position, orientering, konfiguration och extern axel. Positionen är ett kartesiskt koordinatsystem (XYZ). Orienteringen där rotationen ingår beskrivs med kvarternioner. Detta kan ses i de sparade Targets. Det som beskriver orienteringen består av fyra variabler. Kvarternioner är ett hyperkomplext tal där talområdet kan ses som ett fyrdimensionellt rum med en reell och tre imaginära axlar och skrivs på formen $\mathbf{q} = a + bi + cj + dk$.

Om man studerar alla targets i appendix så kan man observera att kvarternionerna (andra vektorn i Targets) är i princip identiska för alla våra sparade positioner. Detta innebär att i princip ingen rotation skedde vid förflyttning av cylindrarna utan endast translation skedde. Studeras translationen så kan man se att positionsbeskrivningen (första vektorn i Targets) för exempelvis p2 och p3 är samma för X och Y koordinater, men skiftar i Z koordinaten. Samma sak för positionerna över cylinderstället där de olika positionerna är förskjutna i X och Y led. Även exempelvis positionerna ös, öm och öL är samma fast förskjutna cirka 100 i y-led. Som sagt är kvarterionerna nästan samma för alla punkter eftersom TCPs position i princip endast ändrades, men de skiljer en aningen eftersom en vis rotationsändring gjordes vid kritiska lägen, exempelvis där gripdonet greppade cylindern och där de placerade cylindern för att precisionen var kritisk.

En rotationsjustering kan således utföras genom att ändra kvarternionerna i Rapid-koden. Dessa är dock inte så intuitiva och det är därför enklare att i simuleringen ändra dessa genom att markera TCP och fritt rotera verktyget. Alternativt kan rotationen ändras genom att skriva in hur mycket varje axel för det definierade koordinatsystemet ska rotera (i grader) i förhållande till referenskoordinatsystemet. Detta beskrivs då med Tait-Bryan vinklar, vilket är en variant av Euler-vinklar där rotationen sker kring tre olika axlar. Om så önskas kan rotationen fås utskrivet som Tait-Bryan vinklar utifrån kvarternionerna sparade i punkterna.

6 Utvärdering

Laboration 1 var enkel att komma igång med och efter att gruppen fått förståelse för programmeringsspråket, framförallt Go-to-kommandot, så gick det väldigt smidigt att få ett fungerande program som kunde utföra en hel cykel. Det var kul att man fick operera en riktig robot och kunde se hur det fungerar i praktiken. Det var enkelt att felsöka de problem som uppstod och det var inget problem med varken robot eller styrdon. Det enda problemet som uppstod var att under optimering av cykeltiden så flyttades bordet något då roboten kolliderade med det. Detta påverkade hela cykeln då bordets position i koordinatsystemet relativt roboten ändrades. Detta löstes genom positionsförändringar.

Laboration 2 gick inte lika smidigt. RobotStudio tog lång tid att starta vilket gjorde att det tog tid att komma igång. Ytterligare problem uppstod även då simuleringen startades men inte kunde avbrytas vilket medförde att programmet behövde startas om. Det tog över en timme innan gruppen kunde starta laborationen på riktigt, och även då hade några av oss problem med att sensorn saknades i den förbyggda robotcellen som användes. Den fungerande koden som tagits med i laborationen hade en fungerande sensor, men de som inte hade den behövde lösa uppgiften på ett annat sätt, där vikterna försvann efter dom blivit tilldelade sin position, detta då det löpande bandet och cylindrarna hade ”samma” kommando. Själva programmeringen kändes mer lik den som samtliga i gruppen använt tidigare, t.ex. IF-satser istället för GOTO-kommandon. Ibland uppstod problem som var svåra att förstå, då detta inte hade hänt i verkligheten, t.ex. så kolliderade en vikt upprepade gånger med gripdonet under en rörelse. Detta var pågrund av att noggrannheten inte hade ställts in till ”fine”.

7 Appendix

I sektion 4.3 är det följande positioner som ersatts med **•••**:

```
robotarget p2:=[[1744.80100396,-309.570133822,875.807808958],  
[0.687246135,0.000000958,0.726424635,-0.000002784],[-1,-3,2,0],  
[9E+09,9E+09,9E+09,9E+09,9E+09]];  
  
robotarget p3:=[[1744.801003243,-309.570115033,1007.338029677],  
[0.687246139,0.000000912,0.726424631,-0.00000278],[ -1,-3,2,0],  
[9E+09,9E+09,9E+09,9E+09,9E+09]];  
  
robotarget ös:=[[1804.025966039,285.150407923,1007.33803833],  
[0.687246135,0.000000748,0.726424635,-0.000002825],[0,-2,1,0],  
[9E+09,9E+09,9E+09,9E+09,9E+09]];  
  
robotarget öm:=[[1804.025969227,383.042528634,1007.338007572],  
[0.68724613,0.000000778,0.726424639,-0.000002787],[0,-2,1,0],  
[9E+09,9E+09,9E+09,9E+09,9E+09]];  
  
robotarget öL:=[[1804.025977246,486.810848295,1007.338047741],  
[0.68724614,0.000000844,0.72642463,-0.000002776],[0,-2,1,0],  
[9E+09,9E+09,9E+09,9E+09,9E+09]];  
  
robotarget is:=[[1800.045930673,285.150395067,832.006837357],  
[0.687246134,0.000000765,0.726424636,-0.000002894],[0,-2,1,0],  
[9E+09,9E+09,9E+09,9E+09,9E+09]];  
  
robotarget im:=[[1804.025978639,386.900797399,829.491347344],  
[0.687246153,0.000000839,0.726424618,-0.000002798],[0,-2,1,0],  
[9E+09,9E+09,9E+09,9E+09,9E+09]];  
  
robotarget iL:=[[1804.025975994,486.79279622,823.395983386],  
[0.687246136,0.00000083,0.726424634,-0.000002815],[0,-2,1,0],  
[9E+09,9E+09,9E+09,9E+09,9E+09]];
```
