Jämförelse och implementering av visionsystem

LUNDS UNIVERSITET
Campus Helsingborg

LTH Ingenjörshögskolan vid Campus Helsingborg
Industriell elektroteknik och automation

Examensarbete:
Viktor Hedberg
Eric Karlsson

LTH Ingenjörshögskolan vid Campus Helsingborg
Lunds universitet
Box 882
251 08 Helsingborg

LTH School of Engineering
Lund University
Box 882
SE-251 08 Helsingborg
Sweden

Tryckt i Sverige
Lunds universitet
Lund 2016

Sammanfattning

Detta examensarbete har utförts på uppgift av Automationsteknik i Hässleholm. Arbetet gick ut på att undersöka två olika visionkameror från olika tillverkare och skapa en applikation som kan förbättra en del i ett befintligt automationssystem.

Processen som Automationsteknik ville förbättra paketerar skåpsluckor med hjälp av en robot och ett packbord. Hur packbordet ser ut är inte intressant i detta arbete bortsett från att skivan som lyfts med robot ska kunna placeras på bordet med 2 mms noggrannhet. Problemet grundar sig i att skivorna som kommer in för paketering är placerade 100 st. på en pall med otillräcklig noggrannhet. I det befintliga systemet korrigeras skivans position med hjälp av ett vinkelbord. Med hjälp av visionsystemet har det tagits fram en metod för att ersätta vinkelbordet som ökar hastigheten i processen och har skapat ett mer anpassningsbart system.

En PLC från Siemens styr hela systemet som även består av en robot från Yaskawa Motoman, ett lyftverktyg på roboten och extern belysning. PLCn skickar ut positioner till roboten som instruerar vart och hur den ska förflytta sig beroende på var i processen den befinner sig. Först lyfter roboten skivan på en position, därefter flyttas skivan så att den placeras med ett av hörnen framför kameran. Väl framme vid kameran tar den ett foto av skivans ena hörn. Kamerorna är konstruerade så att de innehåller hårdvara och mjukvara för att kunna detektera skivans kant och göra vissa beräkningar. Skivans detekterade position skickas till PLCn som beräknar hur skivan ska justeras för att kunna lägga ner den på packbordet med hög noggrannhet.

Nyckelord: PLC, Visionsystem, Automation, Yaskawa Motoman robot, förflyttningsberäkningar.

Abstract

This thesis project has been conducted in collaboration with Automationsteknik in Hässleholm. The issue of this thesis was to investigate two vision cameras from different manufacturers and create an application that can improve part of an existing automation system.

The process that was to be improved packs cupboard doors with an industrial robot arm. Mechanical aspects of the packing process will not be covered unless the required accuracy of 2mm is affected. The base of the problem lies in the fact that the doors are stacked 100 to a pallet and not oriented properly when they arrive to this process. In the existing system this problem is solved with an angled table. A method for replacing the angled table with vision cameras has been developed to speed up the process and make it more versatile.

A PLC from Siemens operates the system which also consists of a robot from Yaskawa Motoman, a grip tool on the robot and camera lights. The PLC sends out positons that instructs the robot how to operate depending on the size of the door and where in the process the robot is. First, the robot lifts the door at one position, moving the door so that it's placed with one of its corners in front of the camera. Once there, the camera takes a photo of the corner. The vision system is constructed so that it has hardware and software built in the camera, making it able to detect the position of the corner and to do necessary calculations for the operation. This information is sent to the PLC which calculates how to adjust the doors position before putting it down on its final position with the right accuracy.

Keywords: PLC, Vision systems, Yaskawa Motoman robot, adjustment calculations.

Förord

Denna rapport är resultatet av ett examensarbete på 22,5 HP. Examensarbetet har utförts under vårterminen 2016 på Automationsteknik AB i Hässleholm. Examensarbetet ingår som avslutande moment på högskoleingenjörsutbildningen vid Lunds Tekniska Högskola Campus Helsingborg.

Ett stort tack riktas till personal på Automationsteknik AB i Hässleholm. Främst till vår handledare Jonas Olsson och till Håkan Marke för all support under arbetes gång.
Vi vill även tacka vår handledare på LTH Johan Björnstedt för hans engagemang under dessa veckor.

Innehållsförteckning

1 Inledning

1.1 Bakgrund

Industrin i dagens samhälle blir alltmer automatiserad för att hantering och tillverkning av produkter ska utföras på kortast möjliga tid. Produkter inom logistik och tillverkning ska hanteras på olika sätt i fler steg med väldigt hög hastighet. Det kan t.ex. vara ett paket som ska lyftas från A till B eller en etikett som ska monteras på en produkt. Det här examenarbetet har utförts i samarbete med Automationsteknik i Hässleholm som är ett företag som skapar lösningar för denna typ av applikationer. I den här automationsprocessen är det skivor som ska tryckas ner i en paketeringsmaskin med 1-2mm:s noggrannhet i horisontalplanet. För att produkten ska kunna hanteras automatiskt med t.ex. en robot behöver roboten styras med någon form av dator eller logisk enhet. I denna process är det en PLC som programmeras för styrning av roboten. Programmet i PLCn behöver veta produktens position i förhållande till roboten och maskinen som ska hantera produkten i nästa steg. Ett sätt är att produkten alltid placeras på samma position där den först plockas upp av roboten, på så sätt vet programmet var produkten lyfts och kan då lägga ner den med hög noggrannhet. Ett annat sätt är att använda sig av någon form av fast monterat riktningsbord som roboten antingen kan släppa produkten på eller trycka produkten mot. När produkten har kommit på plats i denna anordning lyfter roboten den på dess uträknade mittpunkt. Skivorna i den här processen är t.ex. garderobsdörrar, hyllplan eller köksluckor. När de kommer in till paketering är de staplade 100 st. på en pall, placeringen av skivorna är gjord med väldigt låg noggrannhet och kan därför variera någon cm och vara vridna någon grad. Även dimensionerna på skivorna kan variera mellan de olika pallarna. Om skivor på en ny pall har andra dimensioner än de föregående matar en operatör in de nya dimensionerna i PLCn för beräkningar senare i processen.

En del av arbetet är att förbättra en befintlig process genom att byta ut mellanstationen som utför positionsbestämningen mot ett alternativ som gör det med högre hastighet. I processen idag används ett vinkelbord. Ett vinkelbord är ett bord med en lutande toppyta av hjul. När roboten släpper ner skivan på bordet glider den ner till två stoppar som utformar ett hörn där den stannar. Roboten lyfter skivan igen på en position uträknad efter hur stor skivan är och i förhållande till vinkelbordets position. På detta sätt

vet programmet exakt var på skivan roboten lyfter och kan placera den i paketeringsmaskinen med den noggrannhet som krävs. Packning av en skiva med den befintliga lösningen tar 12 sekunder. När skivan släpps för att rulla ner i det fasta hörnet förloras det 2.5 sekunder. Vinkelbordet ska bytas ut mot ett visionsystem som kan utföra positionsbestämning på bråkdelar av en sekund. Om en visionlösning sparar 2 sekunder på hela packningsprocessen har den effektiviserats med nästan 20 %.

Ett mindre visionsystem består av 1-2 kameror med inbyggd hårdvara och mjukvara som kan utföra beräkningar. Om mer än två kameror behövs för att utföra arbetet används kameror som inte kan utföra några beräkningar utan skickar istället bilderna till en enhet som utför beräkningarna med bilder från alla kamerorna. Ett visionsystem är dyrt, en kamera från Cognex kostar ca 50000 kr respektive ca 25000kr från en tillverkare som heter Sensopart men med ett visionsystem behövs inget vinkelbord. Ett vinkelbord ska konstrueras, byggas och justeras. Vinkelbordet blir väldigt specifikt för varje lösning och kan inte återanvändas medan koden i ett visionsystem kan återanvändas. Ett vinkelbord kräver fler arbetstimmar för att sätta upp än ett visionsystem. Därför blir det förhoppningsvis till och med billigare med ett visionsystem.

Programmering av kamerorna görs via dator, där är det möjligt att ställa in vilka mått i fotot som ska beräknas, vilka konturer som kameran ska följa och hur vinklar ska mätas ut. I examensarbetet behövdes endast en kamera eftersom det räcker att ta en bild på ett hörn för att kunna räkna fram skivans position och robotens lyftpunkt.
En annan del av examensarbetet är att jämföra visionkameror från två olika tillverkare. Kamerorna som ska testas är från Cognex och Sensopart, båda modellerna ser väldigt likvärdiga ut på pappret men vilken som passar bäst för projektet återstår att se. Priser är så klart en del som har vägts in i analysen men det viktigaste är att komma fram till vilken kamera som är bäst lämpad för projektet.

1.2 Syfte

Syftet har delats upp i två delar. Den första delen går ut på att undersöka och analysera två olika visionsystem. Dra slutsatser om någon av dem är bättre än den andra och vad fördelarna med de olika systemen är. Eftersom inköpskostnaden skiljer sig väldigt mycket kan detta vara bra att använda som hjälp för företagets beslut vid framtida inköp.
Den andra delen går ut på att realisera en lösning med ett av visionsystemen för att minska produktionstiden och därmed öka intäkterna.

1.3 Avgränsningar

För att kunna lägga ner skivan på packbordet måste vridning och förflyttning ske. Arbetet har inte tagit hänsyn till om produkten har blivit defekt någon gång tidigare i processen utan tittar bara på hur en perfekt produkt ska förflyttas för att paketeras på korrekt sätt. Denna lösningsmetod kontrollerar bara ett hörn och då måste skivorna som kommer in på pallarna vara korrekt i sina dimensioner och vinkelräta. Motoman robot, Siemens PLC, Cognex visionsystem och Sensopart visionsystem är utrustning Automationsteknik tillhandahöll och arbetet har endast utförts på dessa.

2 Teori

Då roboten lyfter skivorna är dessas position i ett koordinatsystem känd vilket innebär att koordinaterna för mittpunkten av en perfekt placerad skiva också är känd. Eftersom skivorna inte är perfekt placerade på pallen blir den positionen inte mitten på skivorna. Eftersom skivorna ligger ovan på varandra på pallen är det svårt för en visionkamera att lokalisera den översta skivans position och vinkel på pallen då underliggande skivor kan sticka ut och störa lokaliseringen i visionsystemet.
Om skivan lyfts upp och ett hörn istället placeras framför en kamera utan

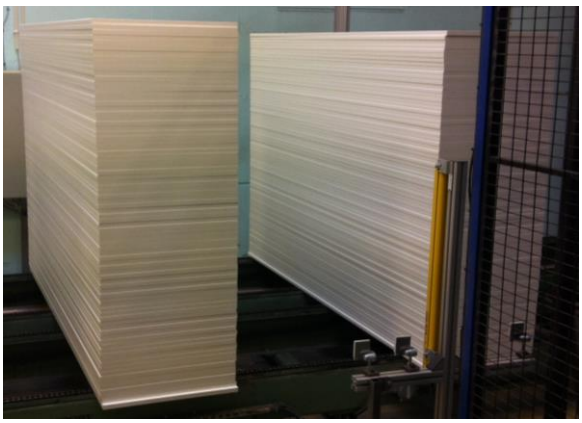

Bild 1: Så här kan skivorna ligga stapplade på varandra innan roboten ska packa ner dem

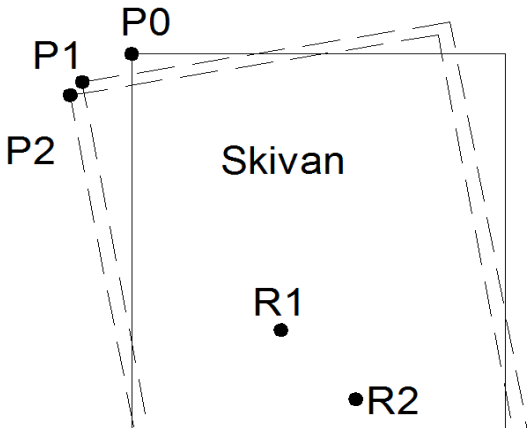

Bild 2: R1 och R2 är olika rotationspunkter för skivan med hörnet i P_0. P_1 och P_2 är punkterna skivans hörn kommer hamna i efter vridning med 10° i de olika rotationspunkterna

andra saker som stör bakom skivan kan programmet med hög noggrannhet lokalisera hörnet i bilden. Med hjälp av olika funktioner i visionprogrammet kan positionen och vridningen i förhållande till en perfekt lyft skiva räknas ut. När vridning sker förflyttas hörnets x- och y-koordinater. Hur mycket de förflyttas beror på var roboten har lyft upp skivan eftersom detta blir rotationspunkten. Förflyttningen som blir ett resultat av vridningen måste därefter ställas i förhållande till den ursprungliga felplaceringen på pallen, det vill säga avståndet från en perfekt lyft skiva innan skivan har vridits. Bild 2 beskriver hur förflyttningen påverkas av rotationspunkten.

2.1 Belysning

För att kameran ska kunna identifiera skivorna och deras hörn kan extra belysning behöva användas. Både Sensopart och Cognex arbetar med gråskala d.v.s. de mäter hur mycket svart eller vitt det är i pixlarna. Där skivans kanter är måste det ske en förändring i pixlarna antingen från ljusare till mörkare eller mörkare till ljusare. Detta kan fungera utan belysning om t.ex. skivan som ska paketeras är vit och ytan under skivan är svart. Då kommer det ske en förändring i gråskalan där skivans hörn är och kameran kan detektera detta. Det skulle också fungera om skivan var svart och ytan under var vit. Med hjälp av olika belysningsmetoder kan skugga eller ljussken skapas. Med andra ord behöver det skapas kontraster mellan skivans kanter och ytan under skivan.

Belysning kan ske uppifrån som belyser skivans kanter så de blir ljusare och ytan under får då inte samma belysning utan blir därför mörkare. För att detta ska fungera behöver ytan under vara mörk och måste även skyddas så andra ljuskällor inte kan lysa upp denna yta och störa. Det kan också ske underifrån då kommer ytan under skivan bli väldigt ljus och skivan i sig själv bli mörkare och som i exemplet ovan så får inte externa ljuskällor påverka skivan uppifrån. Blir skivan också belyst så kan skivan bli lika ljus som ytan under och då kan inte kameran detektera detta. Med hjälp av belysningen så kan skivor med olika färger paketeras och lösningen blir mer dynamisk.

2.2 Kamerainställning

För att få en bra bild av objektet som ska mätas krävs det korrekt belysning och vinkel på kameran. Det är viktigt att belysningen inte riktas på ett sådant sätt att skuggor och blänk kan störa mätningarna. Vinklas kameran eller belysningen fel kan bilden bli förvrängd. Desto längre ut från bildens centrum, desto mer förvrängs vinkelräta linjer så de kan se böjda ut. Var belysningen placeras i förhållande till objektet och kameran avgör hur mycket programmet klarar av att objektet varierar från bild till bild. Eftersom projektet bara ska fokuseras på var i bilden skivan är behöver kameran bara leta efter skivans yttre konturer och inte efter mönster på skivan. Då skivan kan variera i färg och storlek är det viktigt att dessa egenskaper vägs in vid positionering av belysning och kamera.

2.3 Visionsystem

Ett visionsystem kan t.ex. bestå av en kamera och programvara på en dator. Med hjälp av datorn går det att ställa in kamerans programvara så den hittar information i en bild som är tagen av kameran. Kameran kan programmeras för att hitta ett avstånd mellan två föremål på en bild. Denna funktion kommer användas för att avgöra hur roboten ska anpassa skivans position. Två olika visionsystem kommer användas och utvärderas under arbetet. Cognex är visionsystemet som Automationsteknik använder i de flesta fallen. Sensopart är en nyare och mindre kostsam tillverkare. Därför kommer systemen jämföras, Sensopart blir kanske en bättre och billigare lösning för framtiden.

2.3.1 Cognex

Cognex har fyra stycken visionserier: In-Sight Micro, 5600/5705, 5000 och 7000. Det finns i stora drag 4 skillnader mellan kamerorna. Det första är vilken maxupplösning kameran har. Om kameran kan arbeta med färger eller gråskala är också bra att känna till. Kameran kan komma med ett fast objektiv eller med så kallat C-mount utan objektiv. Det finns många olika modeller av objektiv från flera tillverkar att beställa separat. Till sist är det vilken hastighet kameran arbetar med d.v.s. hur många bilder den klarar av att ta i sekunden[1]. Cognex system är konstruerat så programmeraren skapar ett eller flera jobb. Det är i dessa jobb som verktygen definieras. Verktygen kan vara av olika slag som att identifiera en kant, placera ut en fast linje i bilden eller mäta avstånd/vinkel mellan en kant och en linje. I kameran sparas jobbet som blivit uppbyggt. Det kan sparas flera jobb då kameran kanske ska göra olika arbetsuppgifter beroende på vilken produkt som är aktiv i processen. Det är alltid samma jobb som startas vid omstart av kameran och det går att välja vilket jobb det ska vara. När kameran är igång kan jobbet bytas till ett annat via PLCn.

2.3.1.1 Installation

In-Sight 7000 använder sig av Ethernet kabel med M12pins kontakt i kamerahuset och RJ45 till PC för kommunikation. Strömkabeln som behövs är med M12 12pins kontakt i kamerahuset och 12-pin till nätverksdelen där bara två av ledarna används.

2.3.1.2 In-Sight Explorer

Alla modeller i In-Sight serien använder sig av mjukvaran In-Sight Explorer. Allt som ska göras i mjukvaran kan göras under Easy Builders fyra steg. Start, Set Up Tools, Configure Results och Finish.

2.3.2 Sensopart

Kameramodellerna från Sensopart heter Visor V10 och V20. V20 är gjord för att täcka en bredare yta på kortare avstånd. Med ett 12mms objektiv täcker den 16x13mm, motsvarande för V10 modellen är 8x6. V20 har även en högre upplösning och mindre pixlar. 1280x1024, 5.3µm för V20 och 736x480, 6.0µm för V10. Båda är IP65/67 klassad vilket innebär att den är helt dammskyddad och att den klarar en vattenstråle under 3 minuter med ett tryck på 30kPa. Båda modellerna går att få med ett fast 12mms objektiv eller C-mount så att externt objektiv kan skiftas. V10 finns även med 6mm och 25mms fasta objektiv[2].

2.3.2.1 Installation

Visorkameran använder sig av ethernetkabel med M12-4pins kontakt i kamerahuset och RJ45 till PC för kommunikation och strömkabel med M12 12pins kontakt i kamerahuset och 12-pin till PSU.

2.3.2.2 SensoFind

Mjukvaran till Visor består av tre delar, SensoFind, SensoConfig och SensoView. SensoFind är som namnet implicerar till för att hitta de kameror som finns i nätverket och göra nödvändiga inställningar för dem, här kan t.ex. namn, IP-adress och olika lösenord för åtkomst till kameran för “admin” och “worker” ställas in. För att kunna ansluta till kameran som kopplats in måste IP-adressen för datorn vara inställd som statisk.

2.3.2.3 SensoView

För att ställa in hur bilder ska loggas från Visorkameror används SensoView. Loggning av antingen alla tagna bilder, bara de som är rätt och bara de som är fel går att ställa in. I samband med att en bild loggas kan även resultatet av bildanalysen sparas, det ger utdata som t.ex. position och vinkel.

2.3.2.4 SensoConfig

SensoConfig är programmet som används för att ställa in bildanalyseringen till Visorkameror. Programmet är uppdelat i sex huvudfunktioner Job, Allignment, Detector, Output, Result och Start sensor.

2.4 PLC

PLCn som används till detta projekt är en Siemens ETS 1215-SP. Det är en mindre serie från Siemens. Den vanligaste Siemens PLCn är S7 och de båda serierna programmeras med samma mjukvara. PLCn drivs med ett 24V DC nätaggregat från Siemens av modell SITOP PSU200M. Samma nätaggregat driver också kamerorna, ventilerna till verktyget på roboten och två LED-lampor. PLCn är modulär vilket gör att den enkelt kan byggas ut med extramoduler för olika syften. Kommunikation med PLCn görs via protokollet Profinet som använder Ethernet kabel, därför kan en RJ45-kontakt användas vid kommunikation till dator.

2.4.1 Mjukvara

PLCn programmeras via ett program som heter TIA Portal. TIA är en förkortning för Totally Integrated Automation. TIA Portal är en relativt ny programvara som har ersatt det gamla programmet Step 7. För att använda TIA Portal krävs en licens, eftersom Automationsteknik vill skydda sina licenser så har två virtuella maskiner erhållits från dem där TIA Portalen är förinstallerad och ett program som sköter licenshanteringen till TIA Portal. PLCn är den del som ser till så att allting samspelar. Kamerorna, roboten och verktyget på roboten kommunicerar med PLCn och den bestämmer när och vad som ska ske.

2.4.2 Programmering

För att det ska bli så lätt som möjligt att använda programmen som har skapats för detta projekt till andra projekt så kommer funktionsblock att skapas. Ett funktionsblock (FB) kan vara allt från en avancerad beräkningfunktion skriven i SCL till enklare logiska funktioner uppbyggda med Ladder eller FBC. SCL står för Structured Controll Language och är ett programmeringsspråk som på många sätt kan liknas med JAVA. FBC är färdiga funktioner som läggs till som grafiska block med fördefinierade in- och utgångar. Vid skapandet av ett FB definieras vilka in- och utgångar som behövs till funktionen. Detta kan t.ex. vara variabler för att göra uträkningar eller datablock med ingångar från en extern enhet via Profinet. Ett funktionsblock illustrerar grafiskt för programmeraren på ett enkelt sätt vad som behövs för att använda en skriven funktion. Datablock är listor som kopplas till en eller alla funktioner, i listorna går det att definiera olika typer av variabler. Datablocket håller koll på hur mycket minnesarea som behövs för variablerna. Funktioner som används i programmen skriver antingen till ett eget datablock eller till ett globalt som används för att förmedla variabler mellan de olika programmen. Om flera kopior av

samma funktion nyttjas skapas automatiskt datablock till de olika funktionerna. När ett nytt FB läggs till är det enda som behöver göras att mata in vilka inputs och outputs det ska vara till just det blocket. En input till ett FB kan t.ex. vara hur stor en skiva är eller var i bilden kameran hittat en punkt. En output kan t.ex. komma ifrån beräkningen och skicka ut någon variabel som definierar hur skivan ska förflyttas. Användare behöver också definiera hur kommunikation ska ske mellan enheten och funktionsblocket. För att kunna göra detta måste GSD-filer för just den enheten installeras i TIA portal. GSD-filerna följer med programvarorna till enheterna och beskriver för programmeraren och PLCn på vilka adresser som olika inputs och outputs ligger[3]. Dessa adresser ligger under olika moduler som är döpta till vad de hanterar för olika typer av data.

2.5 Robot Yaskawa Motoman

Roboten som har använts i detta projekt är en 6-axlig Yaskawa Motoman av modell MH200II. Den har en lyftkapacitet på 200 kg och en räckvidd på ca 2.5 meter[4]. Skivorna som ska lyftas i detta projekt väger mellan 4.6 - 11kg. Robotens servon styrs av en styrenhet från Yaskawa Motoman. Styrenheten är av modell DX100 från Motoman och kan hantera upp till 8 robotar samtidigt med koordinerade rörelser mellan de olika robotarna, något som inte används i detta projekt. Roboten programmeras via en handdator som är kopplad till styrenheten. Programmeringsspråket som används på handdatorn heter INFORM III och är en typ av instruktionslista. Styrenhetens kommunikation med PLCn sker via Profinet. Alternativa kommunikationsprotokoll som är värt att nämna är t.ex. Ethernet/IP och Profibus[5].

2.6 Lyftverktyg

Verktyget som har använts för att roboten ska kunna lyfta skivorna är ett befintligt lyftverktyg på Automationsteknik. Verktyget är uppbyggt av fyra konsoler med vardera två rader av tio sugproppar. Totalt 80 sugproppar. Varje konsols sugproppar är kopplade till en vakuumpump från Piab som drivs med tryckluft.

Bild 3: Undersidan av lyftverktyget. Fyra konsoler med 20 sugproppar var.

Varje vakuumpump är kopplad till en ventil som styrs med en I/O Kontroller från Murr Elektronik. Kontrollern styrs på bitnivå, d.v.s. genom att ettställa olika utgångar väljs om vakuumpumpen ska suga eller blåsa, vilket gör att verktyget greppar eller släpper en skiva.

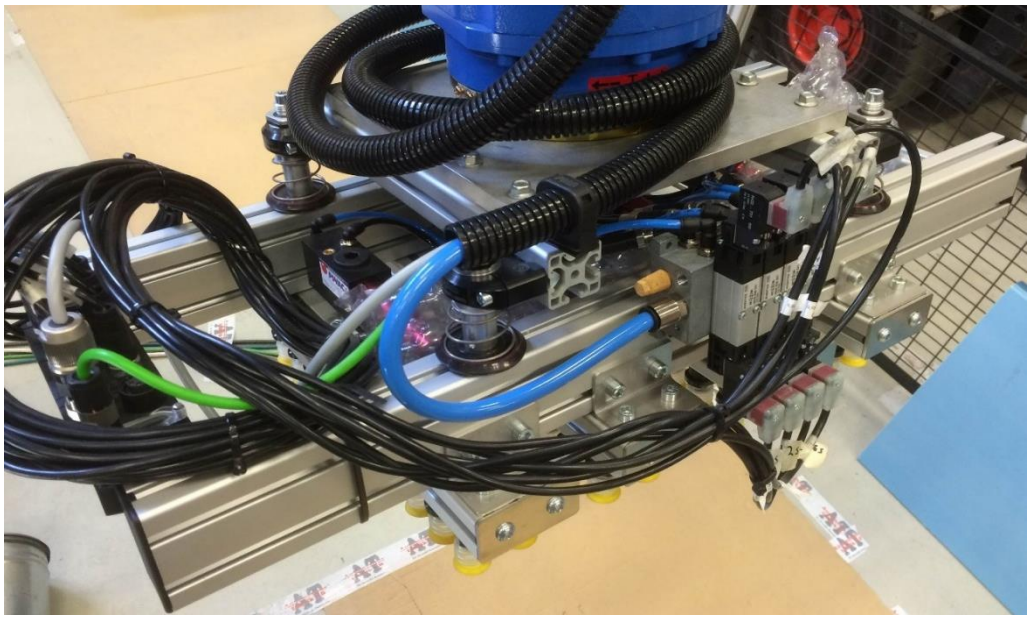

Bild 4: Lyftverktyget från ovansidan. Ventiler till höger i bild och I/O kontroller till vänster.

3 Genomförande

Innan undersökningen av kamerasystemen började behövdes det bildas en uppfattning om vilka värden som kameran behöver ta fram för de beräkningar som ska göras i PLCn. Skisser ritades över de olika bilderna som kameran kan ta och hur vinkel och förflyttning kommer räknas ut. Det som behövs för att räkna ut vridning och förflyttning är de olika vinklarna och punkternas koordinater i bild 5 och 6.
För att fastställa hur mycket skivans hörn rör sig i x- samt y-led när skivan vrids rätt togs en beräkning fram. För denna beräkning behöver punkterna P_0 och P_2 fastställas via kameran och även vinkeln Alpha. P_0 är punkten dit skivans hörn ska flyttas till för att ligga rätt. P_2 är skivans hörn och Alpha vinkeln är så mycket skivan behöver vridas för att ligga rakt. Punkten P_1 behöver räknas ut eftersom det är lyftverktygets mittpunkt även kallad rotationspunkten för skivan. Varje punkt P_n består av X_n och Y_n.

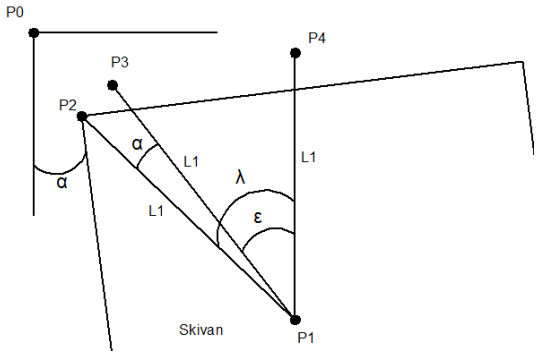

Bild 5: Medurs vridning.

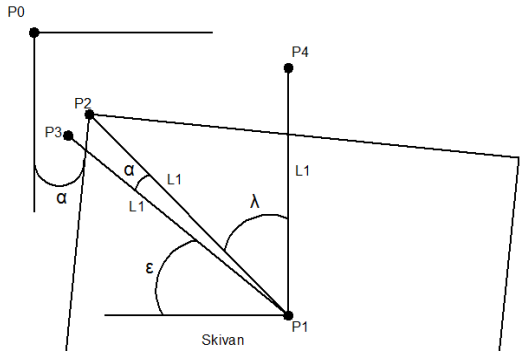

Bild 6: Moturs vridning.

$$P_1 = (x_1, y_1) = (x_0 + {}^{x_{Skiva}}/_2), (y_1 = y_0 + {}^{y_{Skiva}}/_2)$$

Eftersom skivan kan vridas både medurs och moturs behövdes två olika beräkningar tas fram beroende på vridningens riktning. Bild 5 illustrerar när skivan vrids medurs och bild 6 när den vrids moturs.

För att kunna beräkna fram var skivans hörn kommer att hamna i koordinatsystemet efter vridning med Alpha behöver först vinkeln Lambda räknas ut. Punkten där skivans hörn kommer att vara efter vridning är P_3. Med hjälp av punkten P_4 kan Lambda räknas ut.

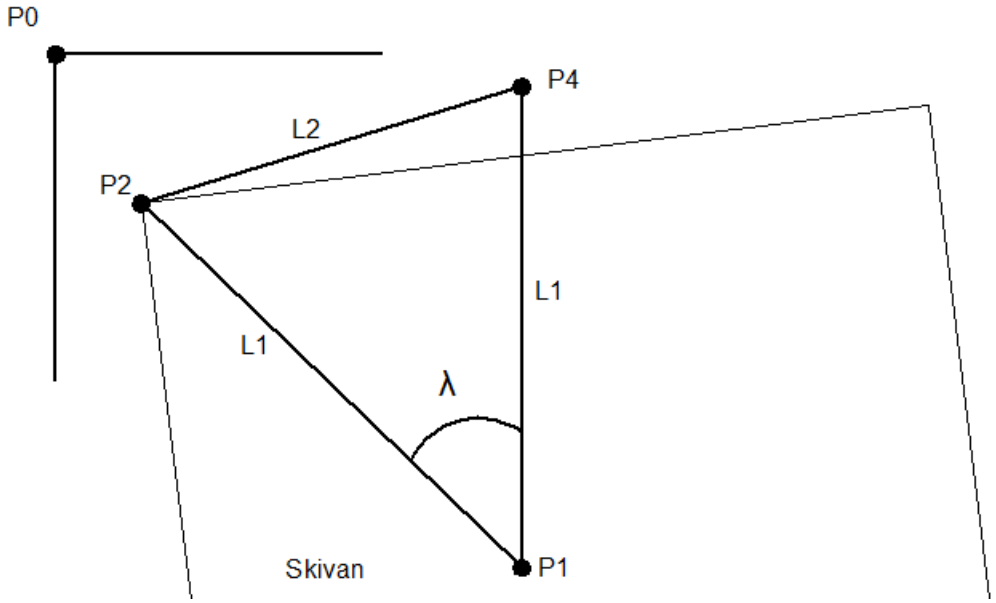

Bild 7: Beräkning av Lambda.

P_4 sätts ut i koordinatsystemet med samma x-koordinat som P_1 men med en y-koordinat som är förskjuten med längden L_1 som ligger mellan punkterna P_1 och P_2. Därefter kan L_2 beräknas som ligger mellan P_2 och P_4 och med hjälp av den kan vinkeln Lambda beräknas fram med cosinus-satsen.

$$P_4 = ((x_4 = x_1), (y_4 = y_1 + L_1))$$

$$L_1 = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

$$L_2 = \sqrt{(x_2 - x_4)^2 + (y_2 - y_4)^2}$$

$$\lambda = \cos^{-1}(1 - (L_2^2/2 * L_1^2)$$

När lambda har beräknats fram kan vinkeln Epsilon räknas ut. Epsilon är olika för medurs och moturs vridning och har därför en unik formel för varje vridning.

Medurs: $\varepsilon = \lambda - \alpha$
Moturs: $\varepsilon = 90 - \lambda - \alpha$

Bild 8 visar hur P_3 slutligen kan räknas ut. Illustrationen till vänster är för medurs vridning och den högra för moturs vridning. Med sträckan P_1-P_3 som har samma längd som L_1 och vinkeln Epsilon kan differensen i x och y från P_1 till P_3 räknas ut med Pythagoras sats. Differensen i x-led och y-led ska adderas eller eventuellt subtraheras med P_1s x och y värde för att få fram P_3.

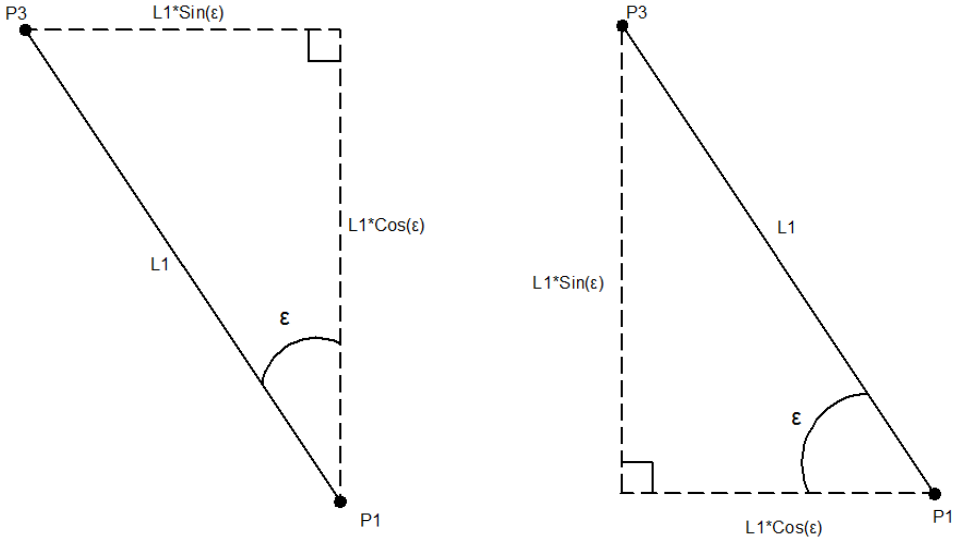

Bild 8: Punkt 3 beräknas fram med Pythagoras sats för de olika vridningarna.

Medurs: $x_3 = x_1 + L_1 \sin \varepsilon \qquad y_3 = y_1 + L_1 \cos \varepsilon$
Moturs: $x_3 = x_1 + L_1 \cos \varepsilon \qquad y_3 = y_1 + L_1 \sin \varepsilon$

Förflyttning i x-led: $x_{Förflyttning} = x_o - x_3$
Förflyttning i y-led: $y_{Förflyttning} = y_o - y_3$

Istället för att beräkna vinkeln Lambda kan Cognexkameran programmeras så en lyftpunkt kan skickas in via PLC. Med hjälp utav den punkten kan vinkeln Lambda mätas ut med samma verktyg som mäter fram vinkeln Alpha. Nackdelen med att mäta fram Lambda på detta sätt är att endast Cognex klarar av det. Därför valdes denna metod bort så även Sensopart kan användas.

3.1 Cognex

Cognexs system undersöktes först. Till Cognex används programmet In-Sight Explorer v.5.2.1. Modellen från Cognex är In-Sight 7402 vilket är en kamera för gråskala med 1280 x 1024 i upplösning. De första testerna som genomfördes skedde på kontoret och illustrerar endast problemet ute i verkstaden. Testerna har gått ut på att hitta ett hörn på ett vitt A4 papper och en grå skåplucka. På arbetsplatsen tillhandahölls ett bord med laborationsstativ som kan justeras i höjdled samt två rundstavar som belysning kan fästas på. Rundstavarna kan justeras i höjdled samt vridas mot stativet. Fästet för Cognexkameran är ett fäste gjort för visionkameror som går att justera i två dimensioner. Till Cognexkameran följde inga kablar med så det första som gjordes var att hitta kablar som passade kameran. Strömkabeln är med M12-kontakt i ena änden och lösa kablar i andra. I bruksanvisningen står det vilken pinne som ska kopplas till 24 +VDC och vilken som ska kopplas till GND. Kommunikationen sker via en Ethernetkabel som kopplas till en switch.

Det första som gjordes i In-Sight Explorer var att lägga till kameran. In-Sight Explorer har en inbyggd funktion som söker efter kompatibla sensorer i nätverket och där ska kameran dyka upp om allt är rätt kopplat. När kameran hittats konfigureras nätverksinställningarna för kameran som från start är fabriksinställd. För att smidigare kunna etablerar en stabil uppkoppling behövs statiska IP-adresser. IP-adresser tilldelades med nummer 192.168.17.220-229. Cognexen tilldelades IP-adressen 192.168.17.225, default gateway sattes till 192.168.17.0 och mask sattes till 255.255.255.0. När dessa har matas in på rätt sätt väljs “add sensor” och anslutning till kameran är nu etablerad.

Innan kameran kunde användas korrekt behövdes ett objektiv monteras då denna kamera modell inte har något fast objektiv. Olika objektiv erhölls för att undersöka vilket som var bäst för uppgiften. De olika objektiven kommer från Omron, Edmund, Tamron och Kowa. Brännvidden på de olika objektiven är 8mm, 12mm och 25mm. Bländaren är 1.4, 1.6 och 1.8. Arbetet började med att testa om det var något av objektiven som direkt kunde uteslutas med brännvidd och bländare i åtanke. Målet var att få med så stor yta som möjligt utan att behöva ha kameran för högt upp. Anledningen till detta är för att kunna avbilda så mycket som möjligt av skivorna och då kunna få så stor noggrannhet som möjligt. Hur mycket skivorna kan placeras fel påverkas också av hur stor yta som fotograferas.

Vid programmering i In-Sight Explorer finns det två olika metoder, Spreadsheets och Easybuilder. Spreadsheets är som Excel, all information som kameran uppfattar i bilden läggs in i rader och kolumner. Detta är en äldre metod och det är mest experter på Cognex som använder sig av Spreadsheets då det är en väldigt invecklad metod som gör det svårt att förstå var informationen som är relevant för uppgiften hamnar. Easybuilder är som namnet antyder en enklare och mer grafiskt illustrerande metod för att programmera kameran. De flesta funktioner som finns i Spreadsheets finns även i Easybuilder men är lättare att använda sig av. För att t.ex. överföra data via Profinet behöver detta göras i flera steg. Data måste läsas, buffras, omvandlas och skrivas. Varje steg behöver en egen funktion. Även om programmet fortfarande gör detta i bakgrunden är det inget som behöver läggas tid på i Easybuilder eftersom programmet gör det automatiskt.

De första försöken resulterade i att det inte blev någon läsbar bild på grund av undermålig belysning och för kort slutartid. Slutartiden ökades och kameran tog in mer ljus, då kunde pappret avbildas av kameran. Nackdelen med detta är att det tar tid och minsta lilla rörelse på kameran eller pappret gör att bilden blir otydlig.

För att kunna detektera hörnet av skivan behövs detekteringsverktyget patternMax, vilket är en funktion där sökområdet som kameran ska arbeta inom först ställs in. Därefter definieras vad kameran ska söka efter, i detta fall var det skivans hörn mot en slät bakgrund. PatternMax följer den definierade formen när skivan flyttas runt i bilden. Resultatet blir positionen i x, y och den relativa vinkeln till hur patternMax har vridits från den ursprungliga inställningen. Positionen som returneras är i förhållande till verktygets mittpunkt och inte skivans hörn. För att få hörnets exakta position användes även detekteringsverktyget edgeDetection som lades tills så det följer patternMax. Det kombinerades två edgeDetection för att skapa en edgeIntersection. När de två verktygen som detekterar kanterna kombineras räknar kameran ut var de kommer korsa varandra och returnerar den punktens position och vinkel.

Vid de tidiga testerna märktes det att denna punkt inte hamnade där den behöver vara för att beräkningarna ska bli korrekt, efter fler tester löstes det problemet genom att inte detektera dessa kanter hela vägen ut till hörnet. Anledningen till att det blev problem är att när kanterna närmar sig hörnet rundas de av och möter varandra i en båge, detta gör att kameran har väldigt svårt att avgöra var det rätta hörnet befinner sig.

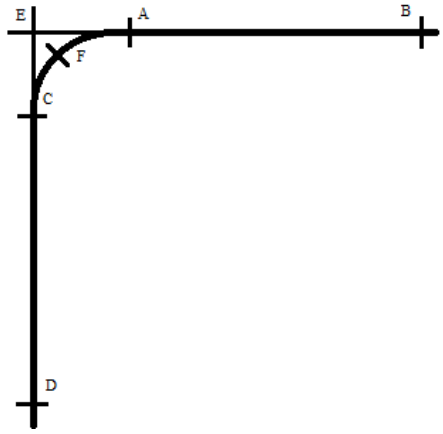

Bild 9: Om hörnet räknas ut med hjälp av linjerna mellan punkt F till B och F till D returneras punkt F som korsningspunkt. Däremot om linjerna mellan punkt A till B och C till D används returneras punkt E. Det är denna punkt som behövs för att beräkningarna ska bli korrekta.

Det finns många inspekteringsverktyg som kan kopplas till ett detekteringsverktyg för att göra mätningar, sätta ut punkter och räkna former med mera. Inspekteringsverktygen kopplas till detekteringsverktygen så att de följer med kanten och att mätningarna alltid sker på rätt ställe. För att kameran ska ta en bild behöver den "triggas". Detta kan ställas in på olika sätt, om kameran är offline och arbetar mot datorn går det att trigga genom att trycka på f5. Kameran kan även trigga kontinuerligt, då tar den bilder i den hastighets som det tar för kameran att utföra ett jobb.

Det finns många fler metoder för att trigga kameran. Den metod som används vid kommunikation med PLCn heter "external". Triggning sker då från PLCn i fyra steg. Det första steget är att aktivera triggningsfunktionen genom att ettställa biten "TriggerEnable" på AquisitionControl-modulen och hålla den ettställd tills triggningen är färdig. Därefter kontrolleras biten "TriggerReady" på AquisitionStatus-modulen. Tredje steget är att ettställa "Trigger" som också ligger på samma modul som "TriggerEnable". Sista steget kontrollerar så att kameran har tagit emot triggern, detta görs genom att kolla så att biten "TriggerAck" på AquisitionStatus är ettställd.

Cognexkameran ställs in enligt bild 10. Tre fasta punkter sätts ut med hjälp av detekteringsverktyget ComputeFixture. Mellan dessa tre punkter dras två linjer med inspekteringsvektyget Geometry/Line. Tillsammans utgör de hörnet där skivan ska hamna om den lyfts perfekt. Det som matas ut till PLCn efter en bild blivit tagen är den fasta punktens hörn, hörnet på skivan som fotograferas och alphavinkeln. Alpha mäts mellan en av de fasta linjerna och motsvarande kant på skivan.

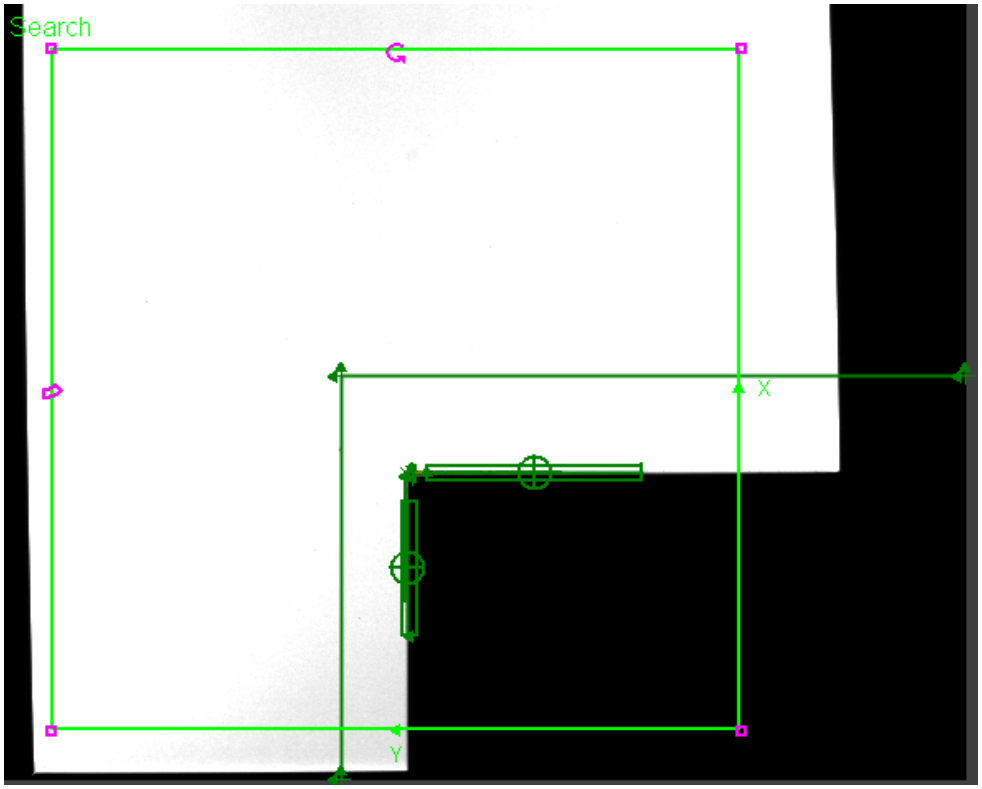

Bild 10: Inställning av Cognex.

In-Sight Explorer har en inbyggd funktion för att omvandla pixlar till mm. Användaren märker ut två referenspunkter i bilden på den höjd som objektet ska avbildas, mäter upp avståndet mellan punkterna i verkligheten och matar in det i programmet. Omvandling hade enkelt kunnat göras i PLCn. Efter kontroll av kalibreringen valdes det att använda den inbyggda funktionen då den är tillräckligt noggrann.

För att få ut värdena som kameran har analyserat behöver inställningar göras under fliken Communications. Här går det välja olika typer av överföringsprotokoll som t.ex. Profinet eller anslutning till en OPC-server. Profinet valdes eftersom all annan kommunikation görs via den metoden.

Via Profinet kan data skickas in till kameran och även skickas ut. Funktionen att skicka in är överflödig för detta projekt. Under fliken output kan data som ska ut till PLCn läggas till. Utdata som kan läggas till är värden från verktygen som blivit programmerade. Det går också att välja vilken datatyp som ska matas ut. Här gäller det att hålla koll på ordningen som värdena matas ut eftersom det är i den ordningen som data kommer in till PLCn. Bild 11 visar hur kommunikationsinställningarna ser ut och vad som skickas ut.

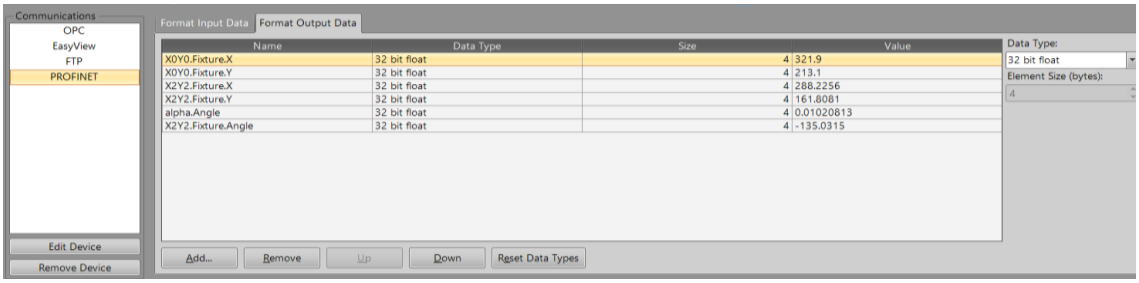

Bild 11: Kommunikationsinställningar i In-Sight Explorer.

3.2 Sensopart

Till Visorkameran installerades den senaste versionen av deras programvara, en nackdel med deras programvara var att den inte hade någon kalibreringsfunktion. Till en början programmerades en funktion i PLCn för att omvandla pixlar till mm, efter några veckors test kom det ut en ny betaversion med kalibrering inbyggd. Kalibreringsfunktionen testades och jämfördes med uträkningarna, eftersom resultatet blev lika noggrant valdes det att arbeta med den inbyggda funktionen därefter. Samma grundläggande tester som gjordes för Cognex med A4 papper görs även för Sensopart för att få en förståelse för hur Sensopart fungerar. All kringutrustning som användes till Cognex används här också, dock har Sensopart sina egna kablar men med samma standard på kontakter.

För att få kontakt med Sensopart behövde en fast IP-adress väljas och eftersom ett antal IP-adresser hade tilldelats användes 192.168.17.220, default gateway valdes till 192.168.17.0 och masken fick 255.255.255.0. Efter namnet valts till Visor123 är allt konfigurerat så programmering av kameran kan göras via datorn. Även Visor V20 är en modell med C-mount. Samma objektiv som användes till Cognex monterades på Visorkameran. Visor vision sensor är mycket annorlunda jämfört med In-Sight Explorer, programmet kan inte söka efter en kant utan här används ett detekteringsverktyg som heter Contour. Verktyget Contour är en detektor

som letar efter ett specifikt mönster som användare väljer i bilden. Det är enkelt att programmera Sensopart med denna detektor. Först behövs en bra bild som visar hur skivan ska ligga när den plockats upp rätt. Med hjälp av den bilden sätter användaren upp detektorn för att definiera vad kameran ska leta efter. Detektorn måste flyttas för hand i bilden för att träffa hörnet precis rätt. Korrekt vinkel på skivan måste också ställas in. Det är här det börjar bli mindre noggrant med Sensopart program eftersom det är nästintill omöjligt att träffa hörnet för hand. Det går att granska den valda konturen och även sudda ut delar som kameran inte ska söka efter. Om det är några pixlar inom ett visst område som ändras väldigt mycket från bild till bild går det att ställa in kameran så den ignorera dessa för att öka prestandan.

När detektorn är inställd som bild 12 kommer kameran skicka ut detektorns mittpunkt i x och y koordinater för beräkningarna med X_2 och Y_2. Den kommer också skicka ut hur långt ifrån ursprungspositionen X_0, Y_0 den är med ΔX och ΔY. X_0, Y_0 matas inte ut från kameran men kan enkelt räknas ut med deltapositionerna. Contour kommer också skicka ut vilken vinkel den har jämfört med originalet som sattes upp och det är den som döpts till Alpha. Användare kan också bestämma begränsningar som säger hur mycket detektorn får vridas. Det finns även en inställning som bestämmer hur snabb kontra noggrann kameran ska vara när den letar upp konturen. Ställs denna inställning in på mest noggrann kan sökningen ta över 5 sekunder vilket är för mycket för applikationen. Denna inställning behöver användaren helt enkelt testa sig fram till för att se vad som ger bäst resultat.

Det går också att ställa in slutartiden för kameran, d.v.s. hur länge den ska släppa in ljus till sensorn vid varje bild. Desto längre slutartiden är desto ljusare blir bilden då mer ljus har släppts in. Men den blir även långsammare samt att om något rör sig det minsta så kommer bilden att bli suddig. Med en kortare slutartid blir programmet snabbare, dock behövs starkare belysning för att bilden inte ska bli för mörk. Detta behöver användare justera med hjälp av ljusinsläppet via objektivet och slutartiden för att uppnå resultatet som eftersträvas. När resultatet är bra ställs utbenen in i programmet och där väljs i vilken ordning och vad som ska skickas ut till PLCn.

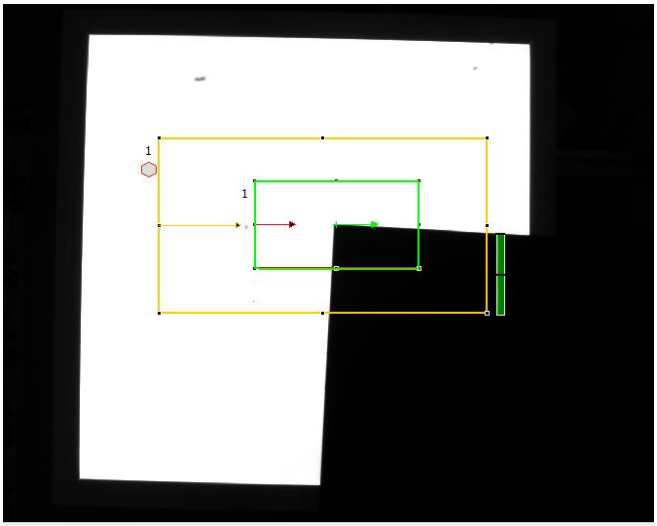

Bild 12: Detekteringsverktyget som ställs in i SensoConfig. Den röda pilen behöver vridas så den är exakt i linje med skivans övre kant.

För att kameran ska kunna triggas ifrån en extern enhet som PLCn måste kameran sättas i "trigger mode" d.v.s. att den ska ta emot triggning via en ingång. Triggningssekvensen till Sensorpart är lite annorlunda men idén är densamma. Först skickas en signal för att trigga, denna signal registreras med en P_Trig som håller signalen hög igenom cykeln. Sen finns det en bit som heter Ready i kameran, då kollar PLCn om kameran är redo för att bli triggad. Om Ready är ettställd samt att en trigger har blivit begärd kommer ett SR minne att ettsättas och tilldela trigger biten inuti i kameran. SR minnet återställs när biten triggerAck har ettställts ifrån kameran och sedan kan PLCn trigga om på nytt igen.

För att skicka ut värdena som kameran har analyserat behövs rätt inställningar under fliken Output. Här går det precis som för Cognex att välja olika typer av överföringsprotokoll. Även här väldes Profinet. Via Profinet kan data skickas ut. Sensopart klarar inte av att ta emot data av annan typ än på bitnivå. Under fliken Telegram läggs data till som ska skickas till PLCn. Här kan endast data ifrån Contour väljas då det är det enda verktyget som används i Sensopart. Även här är det viktigt att hålla koll på ordningen som värdena matas ut eftersom det är i den ordningen som data kommer in till PLCn.

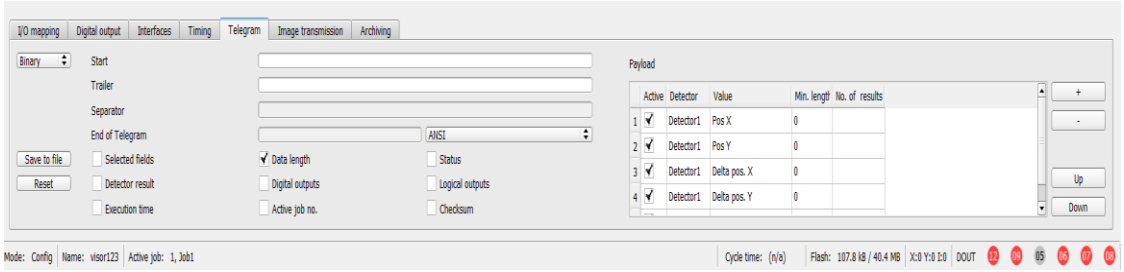

Bild 13: Kommunikationsinställningar i SensoConfig.

3.3 Belysning

Vid de första testerna användes ingen extra belysning förutom kontorets vanliga taklampor. När de första inställningarna gjorts i kamerornas program så att de var inställda på att söka efter kanter, för att detektera kanterna bäst behövdes belysningen laboreras med. Den första extra belysningen som testades var en vanlig skrivbordslampa med 11W lysrör. Belysning gjordes uppifrån och bilderna blev relativt bra. Problem som uppstod var bland annat att flera bilder av samma skiva inte ser likadana ut, detta påverkar hur kameran ska tyda vinkeln och beräknar därför fel. En av anledningarna till detta är att skrivbordslampan inte har någon form av högfrekvensmodul, därför blinkar lampan med dubbla nätets frekvens. Beroende på vilken slutartid kameran är inställd på kan det hända att bilden tas mellan lampans cykler. Även om detta inte var något större problem vid testerna på kontoret skulle det komma att bli ett problem när systemet testades med roboten.

Det diskuterades med handledare på företaget och kom fram till ett par lampor som beställdes för test. Den första varianten var två LED-skenor. Under väntan på att dessa skulle levereras fortsatte laborering med den belysning som fanns tillgänglig. Ganska snabbt blev det uppenbart att beroende på vilken färg och textur som skivorna hade behövdes det olika mycket belysning. Beslut togs att testa belysning från undersidan av skivorna. Två halogenlampor avsedda för kamerabelysning på 60 watt fanns tillgängliga, dessa monterades på stativ och riktades i ca 10° vinkel längs med golvet. Dessa belyste en ljus kartongskiva så att den på bilden i kamerorna ser vit ut. När en skiva placeras ovanför kartongskivan stoppar den en del av ljuset väg till kameran, detta bildar en stark kontrast som är lätt att fånga upp med kameran. Det spelar ingen roll vilken färg eller textur skivan har eftersom det är de yttersta kanternas form som bestämmer vilket ljus som ska tas upp i kameran.

Bild 14: Laborering med två halogenlampor.

När denna metod var framtagen diskuterades det med handledaren igen om någon alternativ belysning kunde beställas eftersom problemet med lampornas frekvens fortfarande kvarstod och ledskenorna inte hade kommit. En Osram Luxiled 600x600 beställdes. Luxiled från Osram är en LED-vägg på 600X600 och 3250 lumen. LED-väggen lades under kameran och skivan placerade mellan kameran och LED-väggen. Detta visade sig vara en väldigt bra metod, alla problem som tidigare uppvisats försvann och kontrasten mellan LED-väggen och skivan blev väldigt stark. Slutartiden i kamerorna kunde sänkas till 2ms vilken betyder att bilden tas väldigt snabbt. LED-skenorna testades ändå men visade sig inte alls vara lika bra som LED-väggen. Eventuellt skulle dessa lampor även kunna fungera men för att få samma effekt som med LED-väggen hade en låda med vitt glas behövt byggas som lamporna hade monterats i. Det hade även behövts fyra 4 lampor för samma effekt. Det lades ingen mer tid på att undersöka LED-skenorna eftersom belysningen som behövs redan hade uppnåtts med LED-väggen.

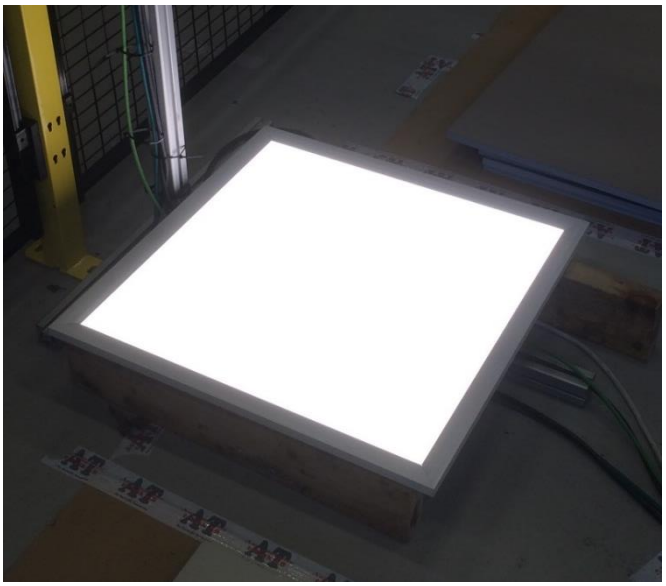

Bild 15: LED-väggen placerad på golvet under kameran.

3.4 PLC

Till PLCn som används i projektet kopplas två extra moduler som gör det möjligt att koppla in 24V:s digitala in- och utsignaler. Med hjälp av dessa in- och utgångar ansluts en knappsats med tre knappar och tre lampor. Knapparna gör det möjligt att på ett enkelt och snabbt sätt påverka PLCn med en signal för att t.ex. återsätta en sekvens. Lamporna är väldigt användbara då de kan tändas under en viss sekvens för att enkelt upptäcka när något specifikt har inträffat under programmets gång.

3.4.1 PLC-programmering - Visionsystemen

Eftersom projektet började med laboration på kamerorna var det naturligt att börja med programmera funktionsblock för kamerorna. För att upprätta kommunikation mellan PLCn och kamerorna behöver GSD-filerna för kameramodellen installeras. Dessa filer följer med i programfilerna till kamerorna. Efter installation läggs kameran till i TIA under fliken “other field device”. En Profinetanslutning definieras genom att grafiskt dra en linje mellan PLCn och kameran under fliken “Devices and Network”. Nästa steg är att välja en IP-adress och ett device-namn för kameran. Device-namnet behöver stämma överens med namnet som har tilldelats i In-Sight Explorer respektive Visor. Den valda IP-adressen behövde tilldelas till kameran genom att först söka upp kompatibla enheter, välja den som ska få IP-adressen och klicka på “assigns IP-adress. Detta behövde göras för båda kamerorna. Alla moduler som hanterar de olika in- och utgångarna till kamerorna installeras automatiskt när kamerorna läggs till med undantag för vilken DATA-modul som ska användas till Visorkameran. Storleken på datamodulen till Visorkameran väljs av användaren, modulerna finns i storlek på 16-256 bytes. Det valdes en variant som hanterar 32 byte data och 2 byte kontrolldata. Därefter gick det att kompilera PLCn och se så den fått kontakt med kamerorna och inga varningar dykt upp. När detta var gjort kunde implementering av funktionsblock påbörjas.

När ett funktionsblock ska programmeras väljs först vilket programspråk som ska användas inom funktionsblocket. Kommunikationsblocket programmerades i Ladder. Först i funktionen så läses utgångarna från kameran av via en Get_IO funktion. Till Get_IO kopplas en vektor där PLCn sparar ner datan. Det är viktigt att vektorn är lika stor som informationen som skickas från kameran. Sedan behöver användaren själv ta reda på vilka bitar eller bytes som innehåller data från kameran. Varje

ingång respektive utgång är av storleken en byte. Om data som skickas är fyra byte sträcker den sig över fyra ingångar osv. Därför måste storleken definieras på varje variabel som ska tas ut från datablocket. Definieras ingången till bitstorlek går det att titta på varje bit på ingången och sedan spara den i en variabel. Definieras ingången som en DINT kommer den sträcka sig över fyra ingångar med start på t.ex. ingång 4. Granskas då ingång 5 kommer datan som avläses inte gå att tyda till något användbart. D.v.s. varje variabel som skickas till PLCn måste definieras till den storlek som den har när den skickas, är den större än 1 byte kommer den sträcka sig över flera ingångar med start på den första byten efter 2- 4 kontrollbyte. 2 kontrollbyte för Visorkameran och 4 för Cognexkameran. Nästa variabel kommer starta på ingången efter där den förra slutade.

Efter att PLCn har läst av från kamerans utgång bearbetats informationen. I detta fall skickas variabler X, Y- koordinater och vinkeln till beräkningen. En funktion för att trigga kameran implementeras också i funktionsblocket. Att trigga kameran betyder att kameran tar ett nytt kort, analyserar bilden och sätter de nya värdena på kamerans utgångar. Exempelsekvenser på hur kamerorna ska triggas finns i deras manualer och beskrivning på hur implementering gjorts finns under respektive underrubrik. Om funktionsblocket för kameran endast ska sköta kommunikationen mellan PLC och kameran så implementeras en Set_IO funktion efter triggerfunktionen som skickar ut värdena från PLCn till kameran. Ett beslut togs att implementera beräkningen inuti dessa funktionsblock eftersom beräkningarna skiljer sig en del mellan de två olika kamerorna. Beräkningarna implementerades i ett eget funktionsblock som sedan används inom funktionsblocket för kameran. I detta funktionsblock valdes programspråket SCL.

Som tidigare förklarats behövs två olika beräkningar beroende på vridningens riktning. För Visorkameran är det enkelt att avgöra vilken beräkning som ska användas eftersom den alltid skickar ut Alpha som negativ eller positiv beroende på vilket håll den ska vridas. Cognexkameran byter inte tecken beroende på vridningen, dess vinkel beror på hur kameran är inställd. För att bestämma vilken beräkning som ska användas kontrolleras vinkeln på hörnet av skivan. Om vinkeln på själva korsningen av hörnen är <-135° ska den vridas moturs och annars gör den beräkningar för medurs vridning. Efter kontroll av vinkeln görs respektive beräkning. Eftersom alphavinkeln inte ska omberäknas så skickades den direkt ut till

programloopen, den skickas även till beräkningen där den behövs för att räkna ut förflyttning. Innan alpha skickas ut behöver den dimensioneras rätt så roboten får in den i rätt storleks ordning. Eftersom roboten tar emot variabler av typen INT sker även en omvandling innan de sätts till PLCn utgångar.

När alla beräkningar och kontroller har gjort så skriver PLCn ut till kameran via en SET_IO funktion. Här är det viktigt precis som i GET_IO funktionen att alla platser skrivs till i minnet på kameran. Eventuellt behöver bara 2 bytes skickas till kameran medan kameran kan ta in 16 bytes. Skickas endast 2 bytes så kan kameran misstolka detta antingen direkt eller vid nästa insättning. Därför är det viktigt att skicka två bytes med information som ska komma fram och sedan fylla ut med en tom vektor för de andra 14 bytes.

3.4.2 PLC-programmering - Motoman

Precis som för kamerorna behövs GSD filer för roboten. Dessa filer fick vi utav Automationsteknik. För roboten fungerar det på samma sätt som med ovanstående enheter. Roboten läggs till under fliken “other field devices” och kopplas ihop via Profinet med PLCn. Funktionsblocket till roboten är endast ren kommunikation mellan PLCn och roboten. Först läses utgångarna från roboten via en GET_IO och sparar ner dessa. Efter att den informationen har erhållits tilldelas robotens ingångar med en SET_IO. Mellan dessa två funktioner används fyra MOVE funktioner. MOVE-funktioner tilldelar endast lokala platserna inom detta funktionsblock. Det är de lokala platserna som skickas ut via SET_IO.

3.4.3 PLC-programmering - Lyftverktyg

Lyftverktyget fungerar precis som alla andra enheter när den ska kopplas samman med PLCn. GSD-filerna som är länken mellan PLCn och verktyget som med de andra enheterna. GSD-filerna fanns också tillgängliga på automationsteknik eftersom verktyget redan satt monterat på roboten och har används i andra applikationer. I/O kontrollern som styr lyftverktyget har 8 st. ingångar, 4 för att styra sugning och fyra för blås. Funktionsblocket för I/O kontrollern är uppdelat i fyra steg. Första är en Get_IO funktion som hämtar variablernas värden, variablerna är boolska d.v.s. etta eller nolla. Nästa steg aktiverar blåsning, detta görs genom att ettställa en variabel på robotens utgångar. När blåsningen aktiveras blåser alla sugpropparna samtidigt i 100ms. Steg tre är till för att aktivera sugning. Sugning är programmerat med självhållning så att när den aktiveras från roboten med en puls kommer den att hållas hög d.v.s. suga tills blåsning

har aktiverats. Användaren kan även stänga av sugningen via knapp tre på den externa knappsatsen. Detta återställer hela programloopen. Den har programmerats så att alla sugproppar gör samma sak samtidigt. Behövs det inte att alla ska suga eller blåsa går det att stänga av varje konsol individuellt i programmet. Det finns även förreglingar som gör så att verktyget inte kan suga och blåsa samtidigt.

3.4.4 PLC-programmering - Mainfunktion

I mainfunktionen knyts allting ihop och säger till alla enheter i systemet vad de ska göra. Mainfunktionen är uppbyggd av en case-sats som är det samma som en switch-sats inom java. Den här case-satsen består av 3 grenar. Gren 1 arbetar i användardefinierade koordinatsystemet 4 och det är där roboten hämtar skivor från pallen. Gren 2 arbetar vid kameran, opererar inom användardefinierade koordinatsystem 5 och ser till så skivans ena hörn placeras framför kameran och triggar kameran så koordinaterna där den ska lämna skivan på beräknas. Gren 3 arbetar också inom koordinatsystem 5 och det är där skivan lämnas. Koordinaterna varierar från skiva till skiva och det beror på att skivorna ligger olika på pallen där de hämtas. Positionerna inom de olika grenarna är förbestämda i roboten, det PLCn gör är att förskjuta positionerna med de inmatade dimensionerna och de uträknade variablerna från kameran. Detta gör att förflyttning mellan positionerna går fortare eftersom alla positioner inom varje gren laddas samtidigt. Alla positionerna beräknas baserat på koordinatsystemens offset och skivornas storlek. Följande punktlista förklarar positionerna som roboten får från PLCn.

1. Ovanför skivorna där de ska hämtas. Fast höjd på 300mm. Hastighet på 400mm/s.
2. Ner för att plocka upp skiva, höjden räknas ut beroende på hur många skivor som är inmatat och hur många den redan har sorterat. Hastighet på 80mm/s.
3. Upp till samma position som punkt 1 med en hastighet på 400mm/s
4. Positionering 1mm ifrån position 5. Hit går roboten i 1200mm/s. Positionen blir inte så noggrann när den går med denna hastighet.
5. Positionering framför kameran, fast höjd på 300mm och alltid samma X,Y för samma storlek på skiva. Hastighet på 80mm/s för att positionen ska bli så exakt som möjligt.
6. Mellan position som är på väg till position 7. Denna position finns här för att roboten ska hinna ladda de uträknade variablerna som definierar position 7. Hastighet på 1200mm/s.
7. Uträknad position med variablerna från kameran. Höjd på 300mm. Hastighet på 1200mm/s
8. Ner för att lämna skivan. Höjd beroende på hur många skivor som redan lämnats. Hastighet på 80mm/s
9. Upp till position 8 för att sedan kunna gå rakt till position 1 utan att slå i marken eller belysningen. Hastighet på 400mm/s.

3.5 Yaskawa Motoman

Roboten programmerades via en handdator som är kopplad med kabel till robotens styrenhet. Det är denna enhet som göra alla beräkningar som behövs för att styra robotens sex servon. Programspråket som roboten använder är sekventiellt. Programmen exekveras uppifrån och gör en rad i taget och stegar sedan neråt till nästa rad. Roboten kan programmeras så den förflyttar sig mellan två positioner på olika sätt. Ett sätt är att använda sig av MOVL. L står för linjär och då rör roboten sig rakt mellan punkterna inom det valda koordinatsystemet. Det finns även MOVJ som är en metod då roboten räknar ut den snabbaste vägen att ta sig från ett ställe till ett annat. Detta är en svepande rörelse som utnyttjar alla axlar till max. Med denna rörelsetyp behöver användaren försäkra sig om att det finns tillräckligt med svängrum för roboten så att den kan röra sig fritt och inte slå emot saker. Det går även att jogga den manuellt efter alla olika koordinatsystem eller varje servo för sig. Med hjälp av detta kan användaren köra roboten till en position och spara den. När den är sparad kommer den upp som en koordinat i handdatorn där kan det väljas vilken rörelsemetod som roboten ska använda för att ta sig dit. Det går också att programmera in en positionsvariabel med förbestämda parametrar.

Vid överföring av en position från PLCn skickas varje del av positionen som en INT, därefter läggs alla delarna ihop med ett befintligt program. Det som kan ställas in är positioner för robotens alla servon, vilket verktyg som används och inom vilket koordinatsystem positionen befinner sig.

Roboten har ett bas koordinatsystem som är baserat från robotens fot. Där x pekar rakt fram, y till vänster samt z uppåt i förhållande till kontakpanelen på robotens baksida. Det kan bli komplicerat att utifrån robotens baskoordinatsystem definiera de exakta variablerna som ska användas när roboten ska hämta en skiva från en pall. För att förenkla en sådan operation kan det programmeras ett eget koordinatsystem för roboten som t.ex. arbetar inom pallarna. I detta fall är pallens ena hörn origo och sidorna x, y. Det är viktigt att positionerna y och x hamnar på rätt håll så att z blir riktat åt det håll som är tänkt. Läggs en av x eller y axlarna inverterat kommer z bli inverterad.

För att lyfta skivan så nära dess mittpunkt som möjligt utgår roboten från det nya koordinatsystemet, går sedan in halva skivans bredd och längd för att lyfta där. Det definierades ett koordinatsystem för där skivorna hämtas

och ett för där de lämnas. Vid kamerapositionen används samma koordinatsystem som för lämna. Vid definiering av ett användarkoordinatsystem monteras en spets på det befintliga verktyget för att med högre noggrannhet kunna gå ner till de punkter som används för att definiera det.

4 Slutsatser

Bild 16: Uppställning av hela projektet i Automationstekniks verkstad.

4.1 Kameramodeller

I slutprodukten är det Cognex visionsystem som används då Sensopart har visat sig inte varit tillräckligt bra för denna uppgift. Problemet med Sensopart är noggrannheten i vinkeln som kameran mäter att skivan ligger fel. Om två identiska bilder på en skiva tas när roboten håller den stilla framför kameran kan vinkeln skilja sig upp till 1 grad emellan bilderna. Detta problem har försökts lösas med olika slutartider på kameran samt justering av fokus på objektivet. Området där kameran fotograferar har även mörklagts med hjälp av wellpappskartonger för att stänga ute eventuellt störande ljus. Ingenting av detta har hjälpt och samma problem har kvarstått. Anledningen är detektorn som letar rätt på skivan i programmet, den är inte bra för detta ändamål. I och med att den letar efter hörnet och som det har förklarats i avsnitt 3.1 så är det svårt att hitta det exakta hörnet när fotografering sker på detta vis. Cognex klarar av detta då den kan beräkna fram med hjälp av två kanter var de kommer skära varandra och märker ut hörnet. Sensopart klarar inte av detta då den inte har Edge detectors eller funktionen som beräknar fram hörnet med hjälp av två kanter. Därför kan vinkeln variera mellan två bilder och det gör att kameran är obrukbar för denna uppgift.

4.2 Objektiv

Objektivet som fungerade bäst för denna applikation är av tillverkaren Edmund med en brännvidd på 12mm och en bländare på f=1.8. Efter tester drogs slutsatsen att en brännvidd på 12mm är den som fungerar bäst för detta projekt. Avståndet från kameran till skivan när den fotograferas är 800mm, med en brännvidd på 12mm ger det en bra storlek på fotona. 8mm ger en för stor bild vilket gör att detaljer som ligger utanför arbetsområdet kan komma med på fotot och påverka hur kameran uppfattar skivan. Med en brännvidd på 25mm blir bilden för liten och ger kameran inte tillräckligt stor bild av skivan för att kunna detektera hörnet i alla fall. Området där objektivens fokus behöver ställas in slutar kring 550mm vilket gör att alla objektiven ger en klar fokus av skivan vid 800mm:s avstånd om de är inställda på yttersta fokus.

4.3 Belysning

Efter åtskilliga tester med olika typer av belysning har slutsatsen dragits att LED-väggen är den variant som fungerar bäst. Om arbetet ska appliceras på en process ute i industrin är det denna modell som är att föredra. En

storlek större kan eventuellt bli aktuellt men det beror på hur applikationen kommer att se ut.

Efter flera veckors tester märktes det att damm ganska snabbt samlas på LED-väggen, detta var inget problem under testerna men om belysningen ska användas under en längre tid och under smutsigare förhållanden kommer det eventuellt behövas en fläkt som kan stå och blåsa bort damm under intervaller.

4.4 Lyftverktyg

Ytan som sugpropparna täcker är ca 300x550 mm, detta funkar bra för skivor upp till storlekar på ca 600x600 mm, även i höga hastigheter. Under projektet arbetades det mycket med skivor på 1941x495mm och 2290x495 mm. Verktyget har inga problem med att lyfta dessa skivor, dock börjar de gunga en hel del i z-led när roboten flyttar dem. Detta gör att roboten måste stanna framför kameran och vänta tills skivan slutat gunga för att resultatet ska bli bra. Om projektet ska användas i industrin kommer ett verktyg som greppar en större yta att användas, därför har det inte implementerats någon funktion för roboten att stanna framför kameran om skivan gungar. När verktyget släpper skivan gör den en kort blåsning för att skivan ska lossna fort. Eftersom skivorna placeras ovan på varandra skapas det en lite luftbubbla mellan skivorna som får skivan att glida i sidled. När skivorna glider ser det ut som om de inte har placerats perfekt ovan på varandra. Detta kan motverkas genom att släppa skivan med en vinkel genom att ena sidan först nuddar skivan under och sedan vinklas nedåt. Inte heller detta implementerades eftersom skivorna inte ska släppas på varandra ute i industrin. Varje skiva kommer att tryckas ner på en kartongskiva i en ram vilket gör att det inte blir några problem med att skivan glider.

4.5 Robot

En åtgärd som kan förbättra noggrannheten och förenkla programmeringen vore att definiera ytterligare ett koordinatsystem som används när roboten ska stanna under kameran. Detta hade gjort det lättare att hitta den exakta positionen som roboten ska gå till.
Det har inte analyserats hur lång tid det tar att flytta en skiva ifrån hämtningspositionen till lämningspositionen då det är irrelevant för arbetet. Tiden som examensarbetet och Automationsteknik är intresserad av är hur länge roboten behöver stå still framför kameran för att få en bra bild. Det vi har kommit fram till är att den inte behöver vänta över huvud taget utan

bara går över till nästa position direkt. Tricket är att köra roboten väldigt snabbt till en närliggande position där fotot ska tas, för att sedan köra den lite långsammare till den korrekta positionen. När roboten körs långsammare kommer den vara mer exakt i sina rörelser och hackar inte. Tiden det tar för denna metod att utföra jobbet som vinkelbordet tidigare har gjort är 0.4 sekunder. Denna tid går med all säkerhet att korta ner ännu mer om mer tid hade kunnat läggas på utvecklingen av robotens styrning. Mer erfarenhet och kunskap behövs för att programmera roboten på ett bättre sätt. Med en tid på 0.4 sekunder är det en förbättring på 2.1 sekunder jämfört med vinkelbordet. Detta har minskat hanteringstiden med 17.5 %.

5 Framtida utvecklingsmöjligheter

För att öka processeringshastigheten behöver roboten programmeras mer effektivt. I projektet har endast linjära rörelser utnyttjats. Byts rörelsemetoden till svepande rörelser får roboten mer frihet att själv räkna ut den snabbaste väg till slutmålet, vilket gör att den totala tiden för att hantera en skiva minskar.

Kameran och belysningen behöver ett stativ som sitter fast i marken för att utrustningen inte ska störas på något sätt av andra processer runt om kring.

Programmering av PLCn har inte skett med någon standard. Detta gör att om projektet överlämnas till någon annan kan det ta onödig tid att sätta sig in i koden. En standard lämnades ut av Automationsteknik, tyvärr kom den när en stor del av programmet redan var skrivet och tid inte fanns att göra om.

Ett HMI behöver programmeras och installeras så att en operatör kan mata in dimensioner och styra processen utan behov av en dator med direkt uppkoppling till PLCn.

Referenser

[1]http://www.cognex.com/?langtype=2057 (23-05-2016)
[2] http://www.sensopart.com/en/ (23-05-2016)
[3] http://w3.siemens.com/mcms/automation/en/Pages/automation-technology.aspx (23-05-2016)
[4] http://www.motoman.com/datasheets/MH280II.pdf (23-05-2016)
[5] http://www.motoman.com/datasheets/DX100%20Controller.pdf (23-05-2016)

Bilagor

PLCkod - Mainmetod

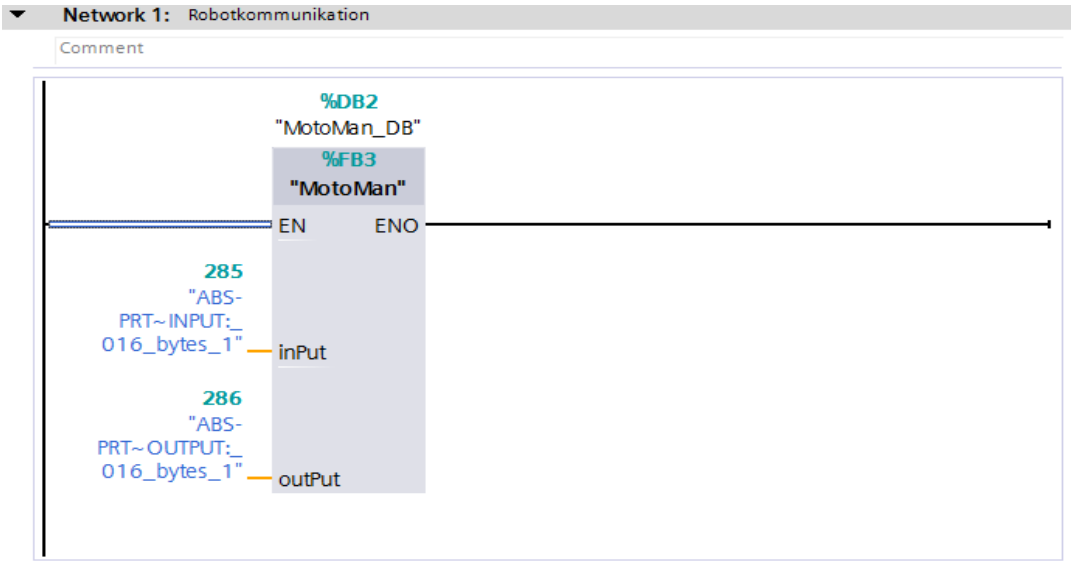

Bild 17: Kommunikationsblocket för Yaskawa Motoman

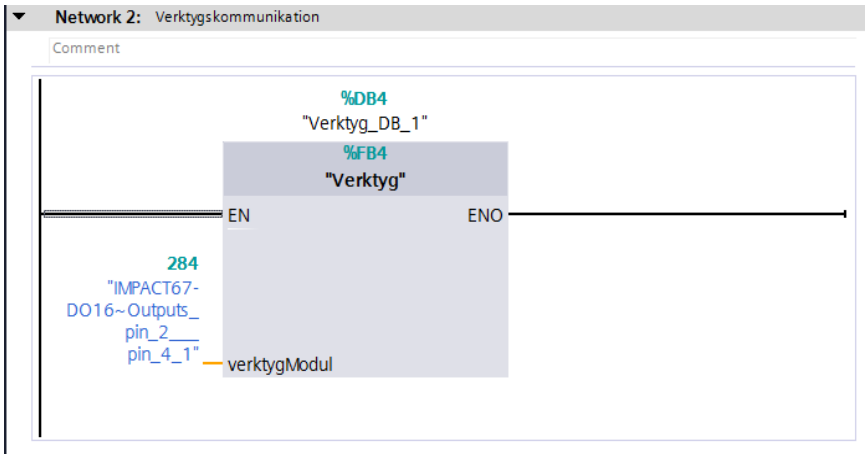

Bild 18: Kommunikationsblocket för lyftverktyget.

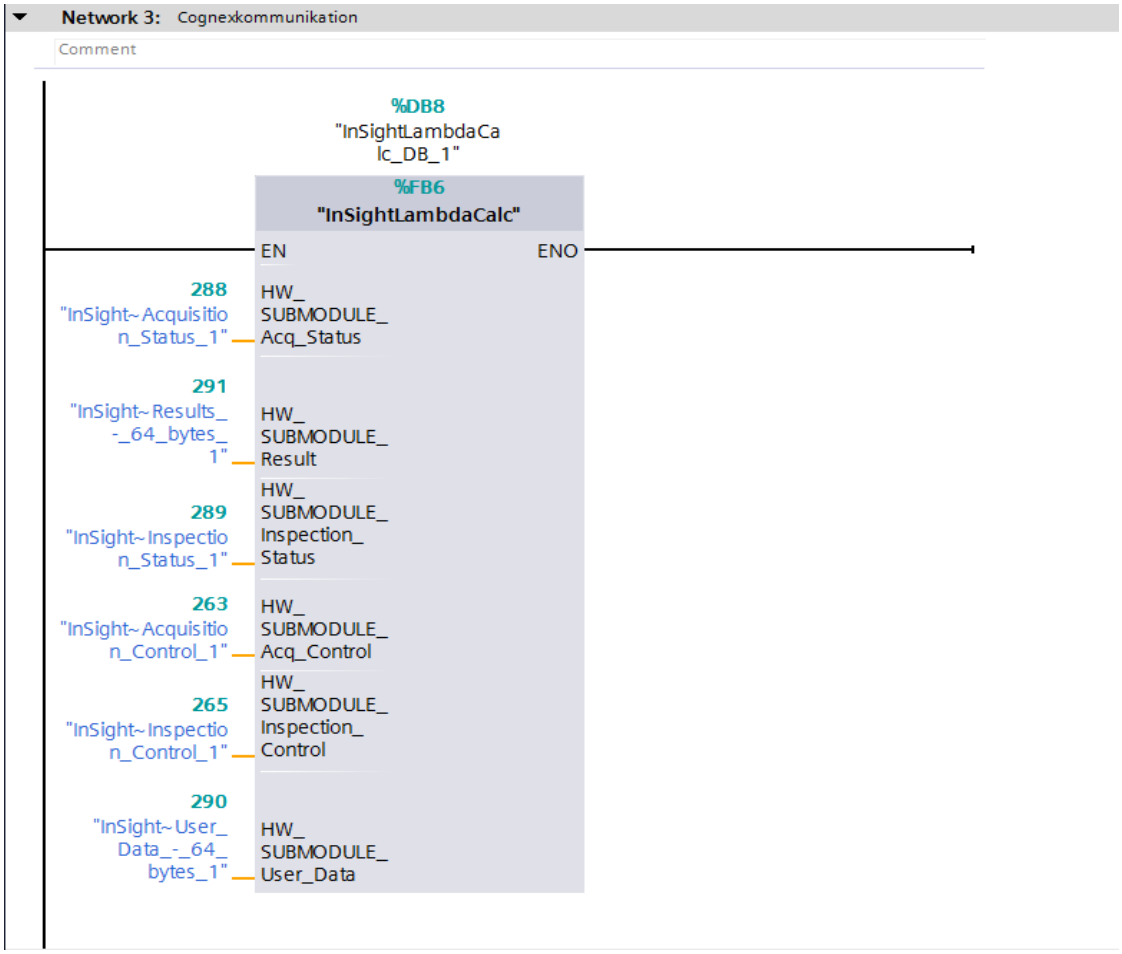

Bild 19: Kommunikationsblocket för Cognex, här finns även uträkningar.

Verktyg

		Name	Data type	Default value	Retain	Accessible f...	Visible in ...	Setpoint
1		Input				☐	☐	☐
2		verktygModul	HW_SUBMODULE	16#0	Non-retain	☑	☑	☐
3		Output				☐	☐	☐
4		<Add new>				☐	☐	☐
5		InOut				☐	☐	☐
6		<Add new>				☐	☐	☐
7		Static				☐	☐	☐
8		SETIO_Instance	SETIO			☑	☑	☐
9		outVerktyg	Struct		Non-retain	☑	☑	☐
10		blås.0	Bool	false	Non-retain	☑	☑	☐
11		blås.1	Bool	false	Non-retain	☑	☑	☐
12		blås.2	Bool	false	Non-retain	☑	☑	☐
13		blås.3	Bool	false	Non-retain	☑	☑	☐
14		sug.0	Bool	false	Non-retain	☑	☑	☐
15		sug.1	Bool	false	Non-retain	☑	☑	☐
16		sug.2	Bool	false	Non-retain	☑	☑	☐
17		sug.3	Bool	false	Non-retain	☑	☑	☐
18		reserv	Byte	16#0	Non-retain	☑	☑	☐
19		blåsaktiv	Bool	false	Non-retain	☑	☑	☐
20		sugaktiv	Bool	false	Non-retain	☑	☑	☐
21		blås	Bool	false	Non-retain	☑	☑	☐
22		sug	Bool	false	Non-retain	☑	☑	☐

Bild 20: Datablock för verktyget.

PLCkod - Verktygskommunikation

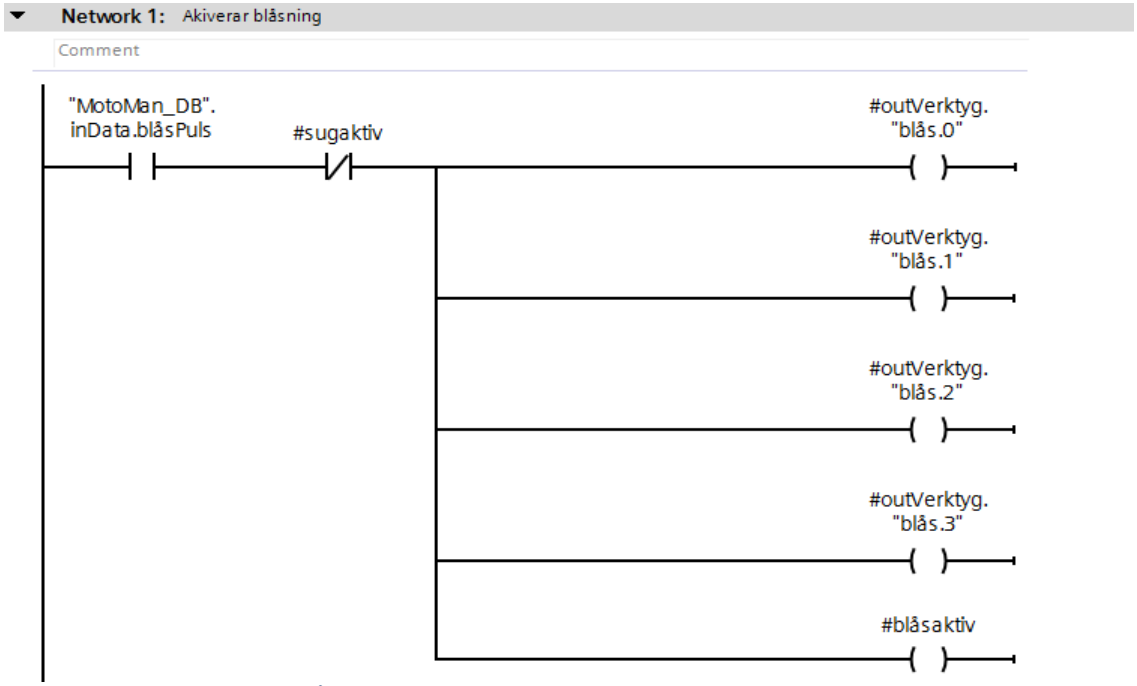

Bild 21: Gren för att aktivera blås.

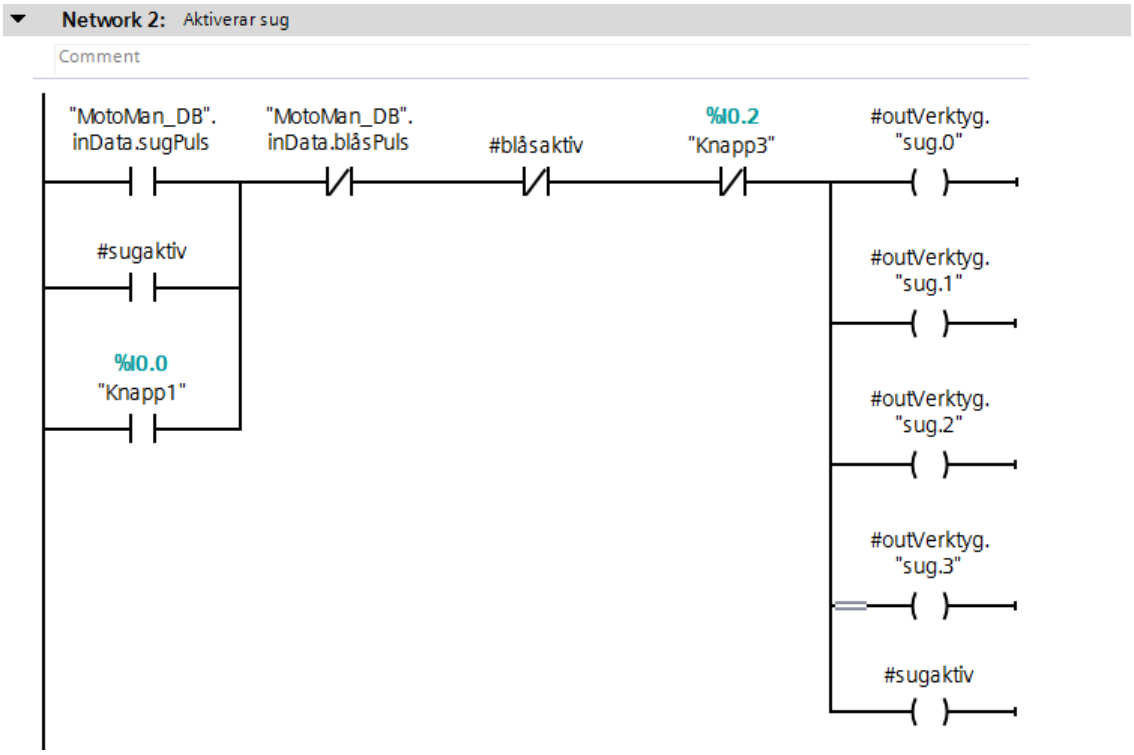

Bild 22: Gren för att aktivera sug.

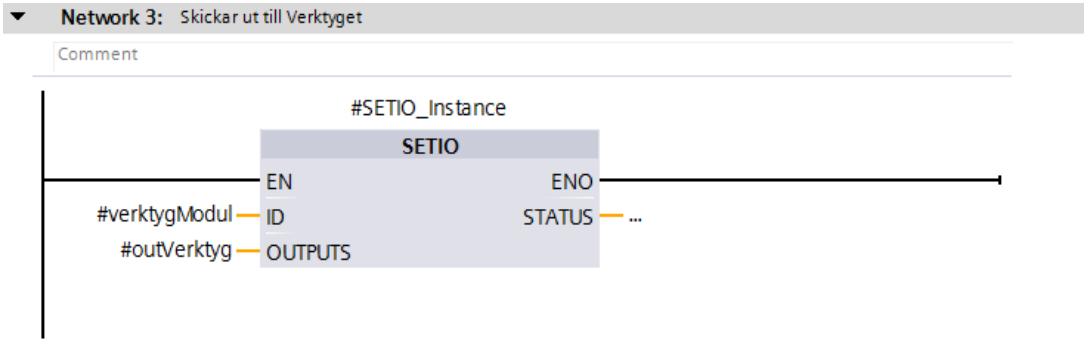

Bild 23: Gren för att sätta utgångarna.

PLCkod - Motomankommunikation

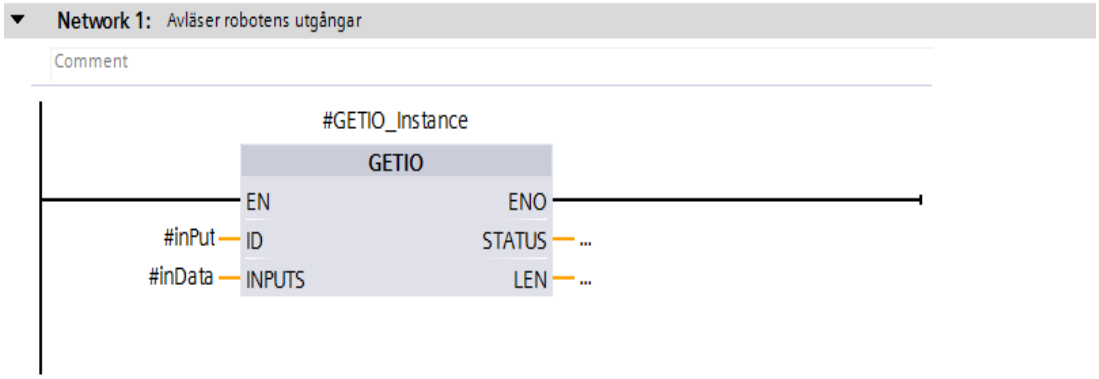

Bild 24: Läser av ingångarna till PLCn från roboten.

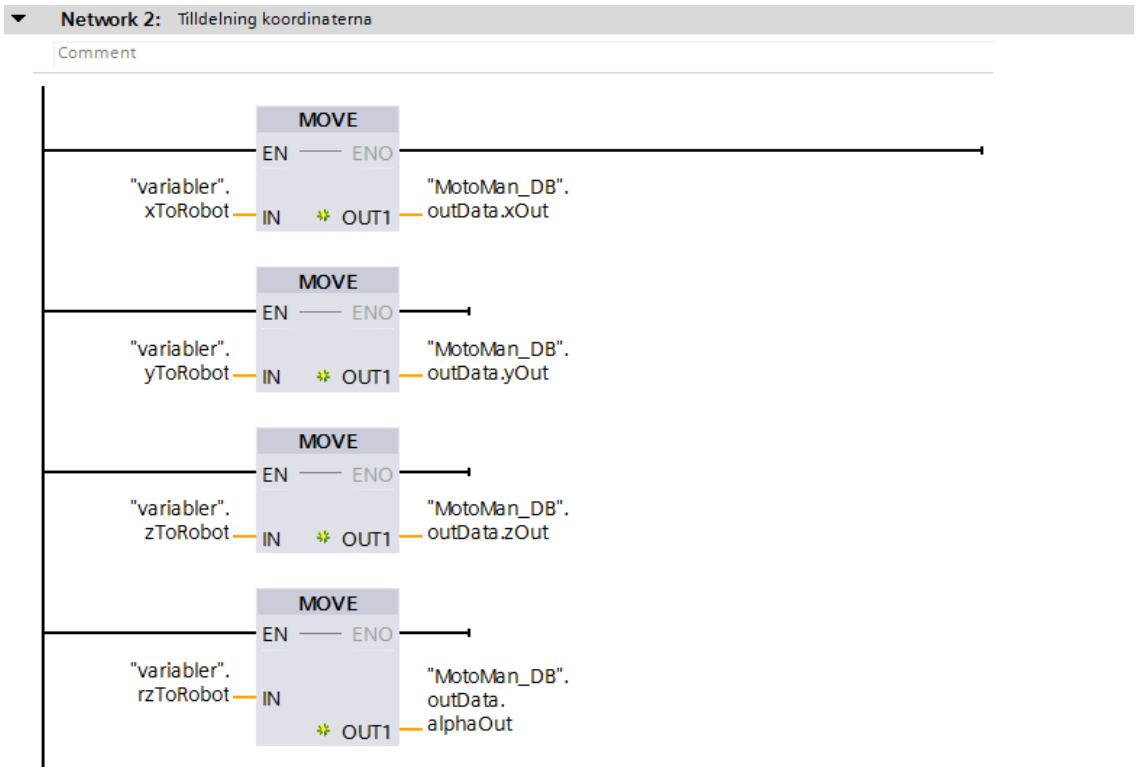

Bild 25: Flyttar de uträknade variablerna till PLCn:s utgångar

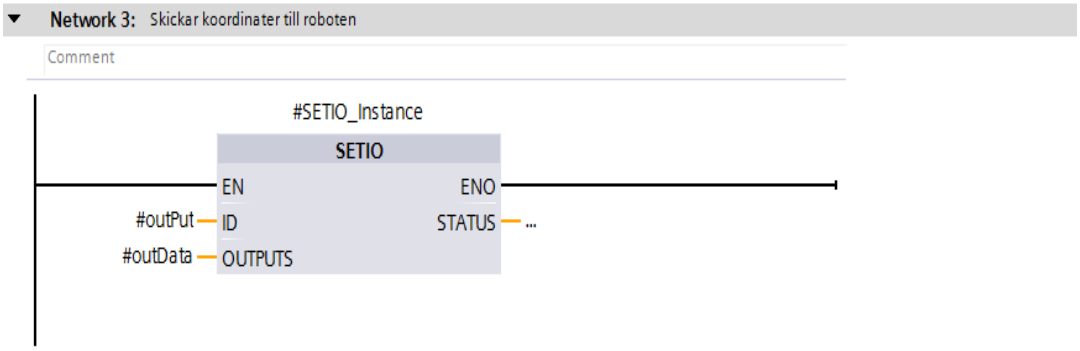

Bild 26: Sätter PLCn:s utgångar till roboten.

MotoMan

		Name	Data type	Default value	Retain	Accessible f...	Visible in ...	Setpoint	Comment
1		Input							
2		inPut	HW_SUBMODULE	16#0	Non-retain	☑	☑		
3		outPut	HW_SUBMODULE	16#0	Non-retain	☑	☑		
4		Output							
5		<Add new>							
6		InOut							
7		<Add new>							
8		Static							
9		GETIO_Instance	GETIO			☑	☑		
10		outData	Struct		Non-retain	☑	☑		
11		Reserv	Int	0	Non-retain	☑	☑		
12		xOut	Int	0	Non-retain	☑	☑		
13		yOut	Int	0	Non-retain	☑	☑		
14		zOut	Int	0	Non-retain	☑	☑		
15		alphaOut	Int	0	Non-retain	☑	☑		
16		reservArray	Array[9..15] of Byte		Non-retain	☑	☑		
17		inData	Struct		Non-retain	☑	☑		
18		sugPuls	Bool	false	Non-retain	☑	☑		
19		blåsPuls	Bool	false	Non-retain	☑	☑		
20		Trigger	Bool	false	Non-retain	☑	☑		
21		Step	Bool	false	Non-retain	☑	☑		
22		0.4	Bool	false	Non-retain	☑	☑		
23		0.5	Bool	false	Non-retain	☑	☑		
24		0.6	Bool	false	Non-retain	☑	☑		
25		0.7	Bool	false	Non-retain	☑	☑		
26		1	Array[1..15] of Byte		Non-retain	☑	☑		
27		SETIO_Instance	SETIO			☑	☑		

Bild 27: Datablock för kommunikation med roboten.

PLCkod - Cognexkommunikation

InSightLambdaCalc

	Name	Data type	Default value	Retain	Accessible f...	Visible in ...	Setpoint
1	Input				☐	☐	☐
2	HW_SUBMODULE_Ac...	HW_SUBMODULE	16#0	Non-ret...	☑	☑	☐
3	HW_SUBMODULE_Res.	HW_SUBMODULE	16#0	Non-retain	☑	☑	☐
4	HW_SUBMODULE_Ins..	HW_SUBMODULE	16#0	Non-retain	☑	☑	☐
5	HW_SUBMODULE_Ac...	HW_SUBMODULE	16#0	Non-retain	☑	☑	☐
6	HW_SUBMODULE_Ins..	HW_SUBMODULE	16#0	Non-retain	☑	☑	☐
7	HW_SUBMODULE_Use.	HW_SUBMODULE	16#0	Non-retain	☑	☑	☐
8	Output				☐	☐	☐
9	<Add new>				☐	☐	☐
10	InOut				☐	☐	☐
11	<Add new>				☐	☐	☐
12	Static				☐	☐	☐
13	SetOffline	Bool	false	Non-retain	☑	☑	☐
14	P_TRIG	Array[1..1] of Bool		Non-retain	☑	☑	☐
15	SR	Array[1..2] of Bool		Non-retain	☑	☑	☐
16	GETIO Acq Status	GETIO			☑	☑	☐
17	GETIO_Result	GETIO			☑	☑	☐
18	GETIO Inspection Statu	GETIO			☑	☑	☐
19	SETIO_Acq_Control	SETIO			☑	☑	☐
20	SETIO_Inspection_Co...	SETIO			☑	☑	☐
21	SETIO_User_Data	SETIO			☑	☑	☐
22	In_Acq_Status	Struct		Non-retain	☐	☐	☐
23	In_Result	Struct		Non-retain	☐	☐	☐
24	In_Inspection_Status	Struct		Non-retain	☐	☐	☐
25	Out_Acq_Control	Struct		Non-retain	☐	☐	☐
26	xMove	Real	0.0	Non-retain	☑	☑	☐
27	yMove	Real	0.0	Non-retain	☑	☑	☐
28	Out_Inspection_Contro	Struct		Non-retain	☑	☑	☐
29	Out_User_Data	Struct		Non-retain	☑	☑	☐
30	Calc_Instance	"CalcMedurs"			☑	☑	☐
31	CalcMoturs_Instance	"CalcMoturs"			☑	☑	☐
32	AlphaCalc_Instance	"AlphaCalc"			☑	☑	☐
33	triggerBegärd	Bool	false	Non-retain	☑	☑	☐
34	Temp				☐	☐	☐
35	aktiveraMoturs	Bool			☐	☐	☐
36	AlphaToCalc	Real			☐	☐	☐

Bild 28: Datablock för kommunikation med Cognex.

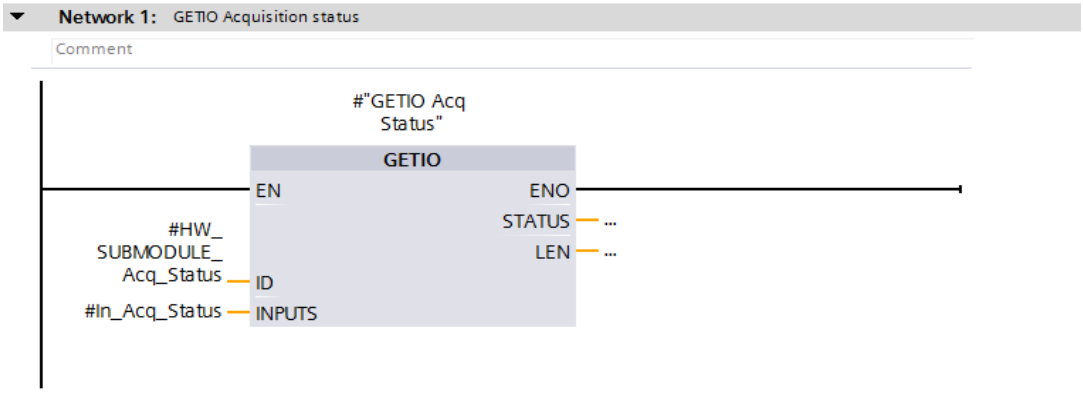

Bild 29: Läser av ingångarna från Cognex som ligger under modulen för bildstatus.

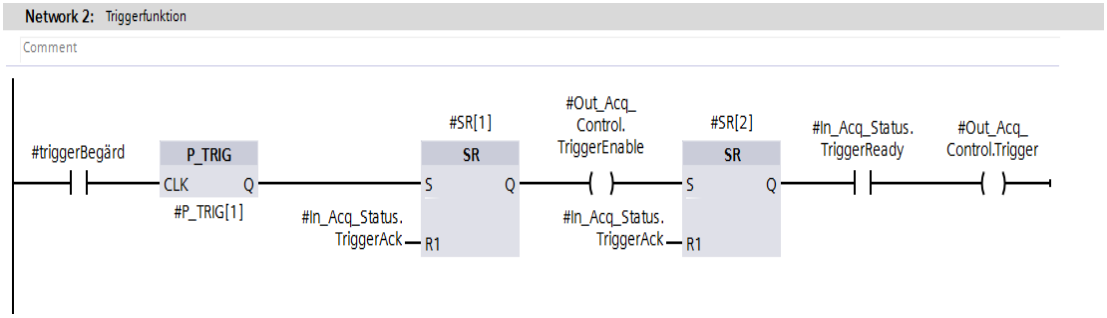

Bild 30: Sekvensen för att trigga Cognexkameran.

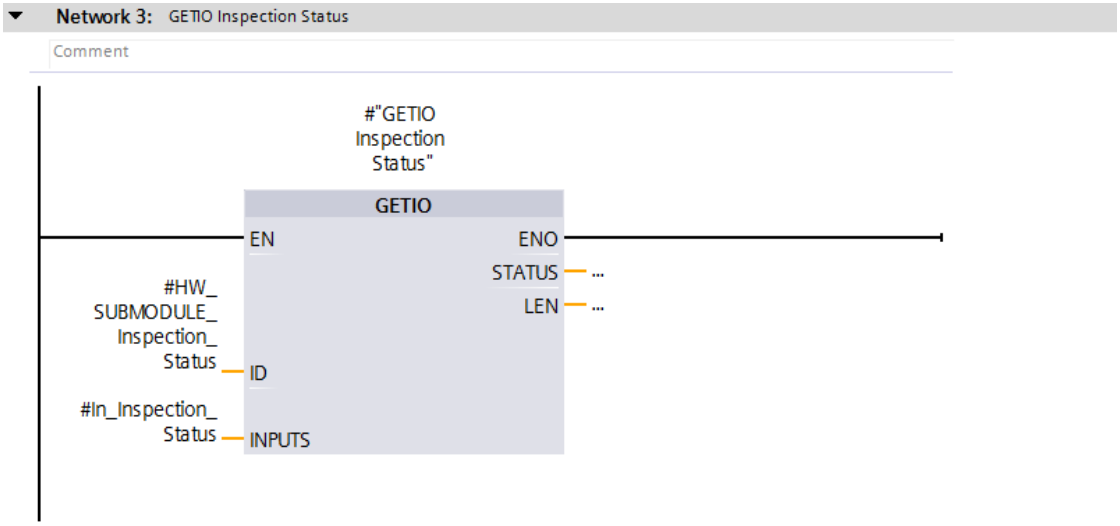

Bild 31: Läser av ingångarna från Cognex som ligger under modulen för inspektionsstatus.

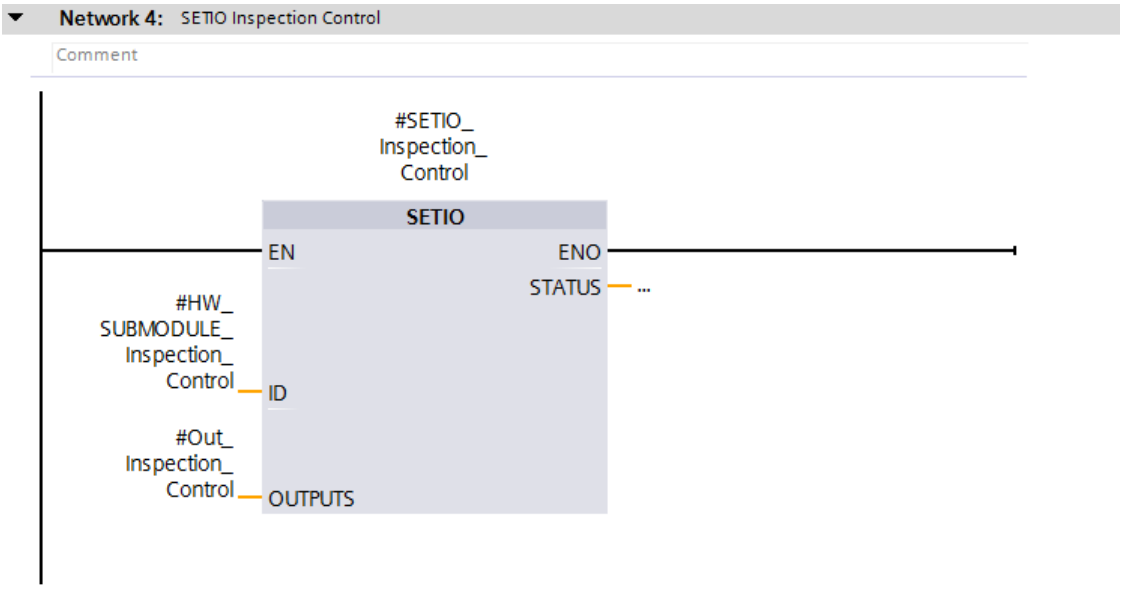

Bild 32: Sätter utgångarna till Cognex som ligger under modulen för inspektionskontroll.

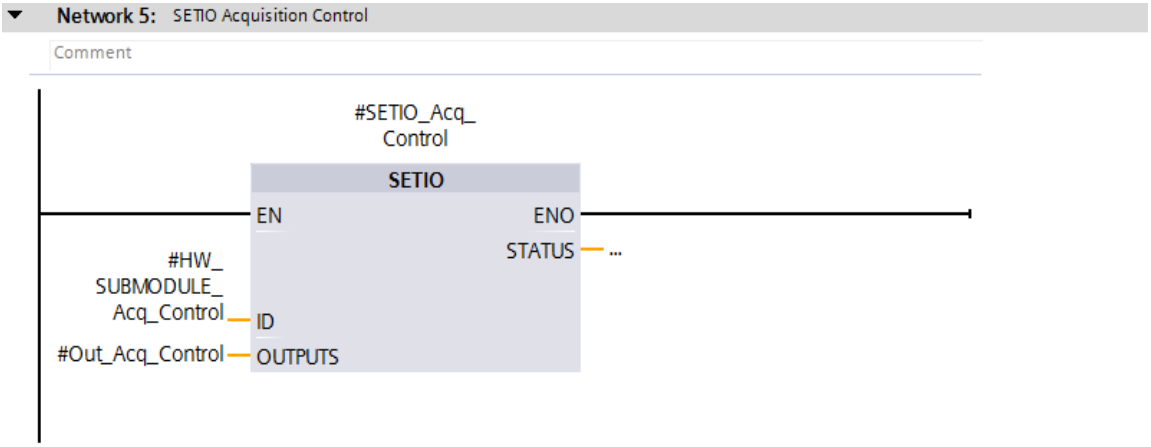

Bild 33: Sätter utgångarna till Cognex som ligger under modulen för kontroll av bildtagning.

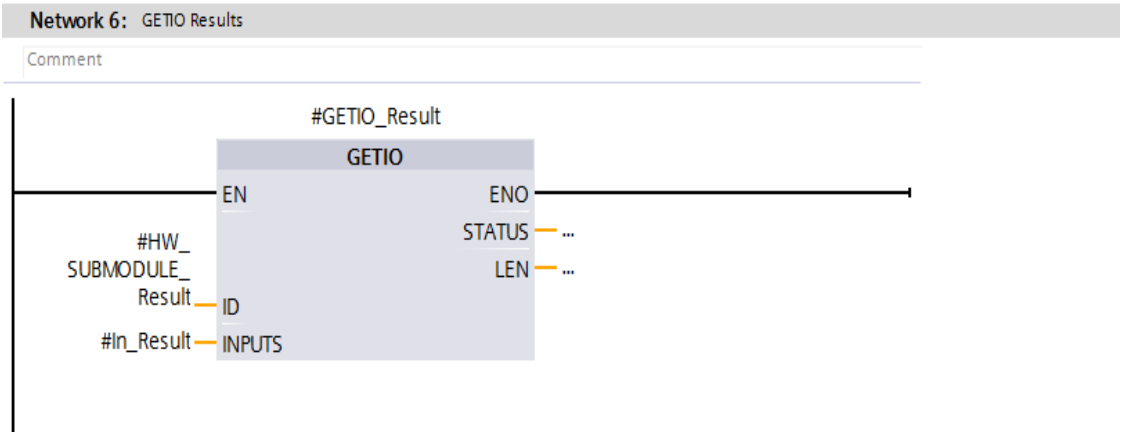

Bild 34: Läser av ingångarna från Cognex som ligger under modulen för resultat av kamerans bildanalys.

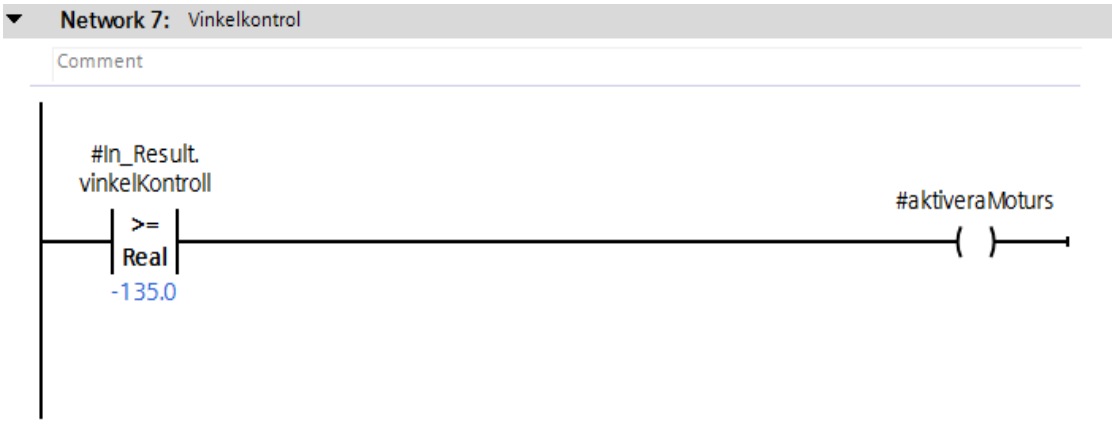

Bild 35: Kontrollerar alphavinklen och aktiverar den berörda beräkningsfunktionen.

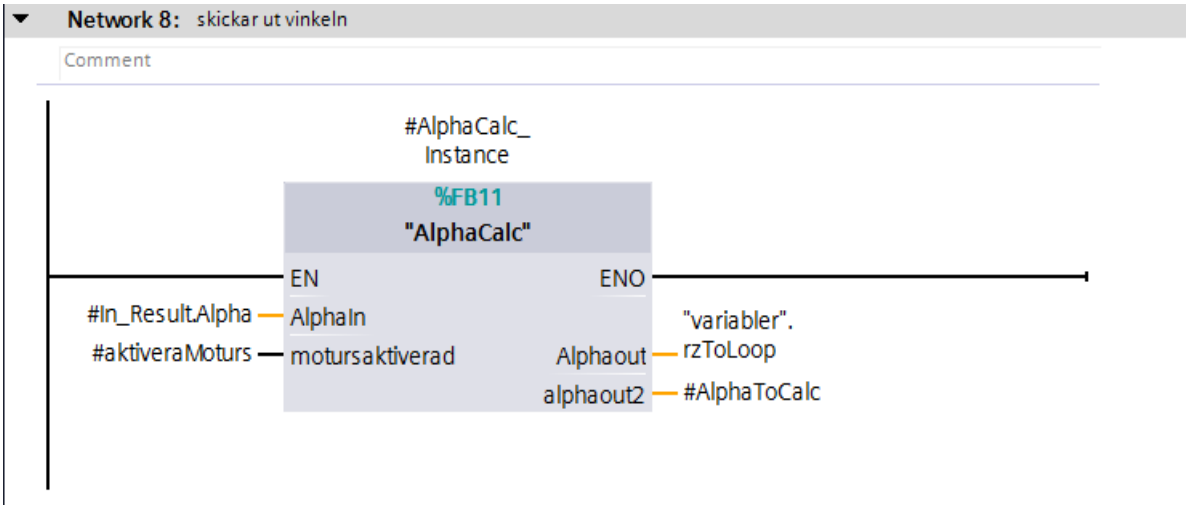

Bild 36: Aktiverar funktionen AlphaCalc.

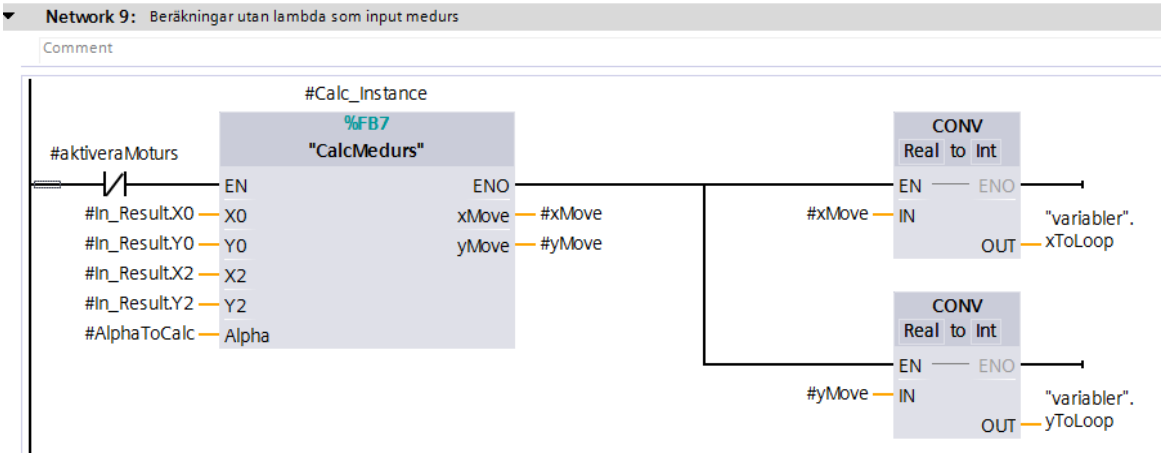

Bild 37: Funktionen CalcMedurs och konvertering av dess resultat till INT.

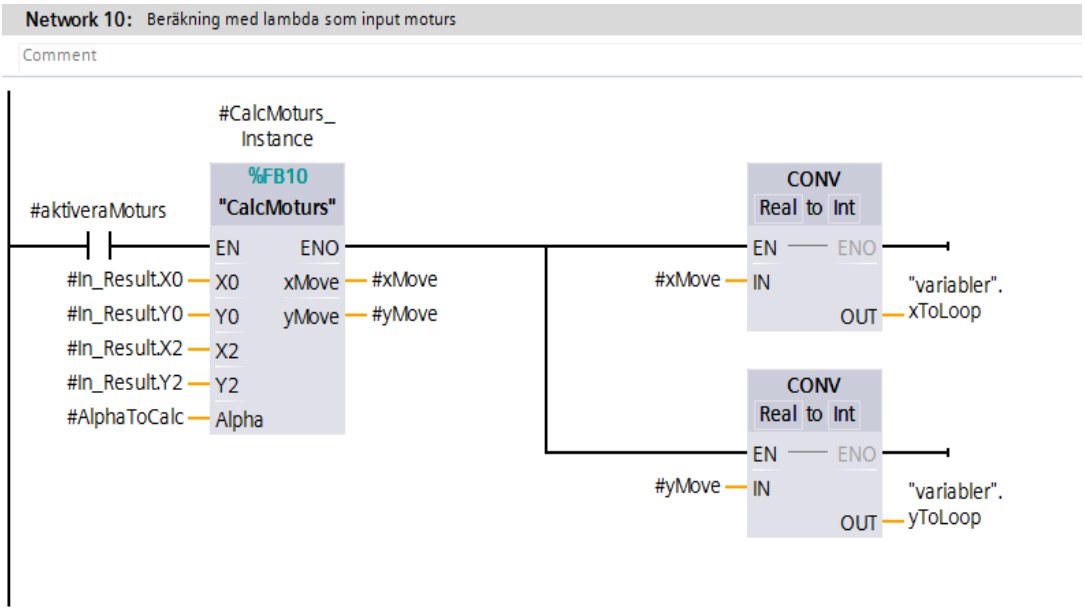

Bild 38: Funktionen CalcMoturs och konvertering av dess resultat till INT.

PLCkod - Beräkning

```
#X1 := #X0 - INT_TO_REAL("variabler".xSkiva)/10 / 2; //beräknar lyftpunktens x värde.
#Y1 := #Y0 - INT_TO_REAL("variabler".ySkiva)/10 / 2; //beräknar lyftpunktens y värde.
#L1 := SQRT(SQR(#X1 - #X2) + SQR(#Y1 - #Y2)); //beräknar fram sträckan L1
#X4 := #X1; // x4 tilldelas
#Y4 := #Y1 + #L1; // y4 beräknas
#L2 := SQRT(SQR(#X2 - #X4) + SQR(#Y4 - #Y2)); // Sträckan L2 beräknas
#LambdaR := ACOS(1 - (#L2 * #L2/ (2 * #L1 * #L1))); // vinklen Lambda beräknas fram i rad
#Lambda := #LambdaR * 180 / #Pi; // omvandlar Lambda till grader
#Epsilon := #Lambda - #Alpha; // beräknar fram Epsillon
#EpsilonR := (#Epsilon * #Pi) / 180; // omvandlar Epsillon till rad
#X3 := #X1 + #L1 * SIN(#EpsilonR); // beräknar hörnets x koordinat efter vridning
#Y3 := #Y1 + #L1 * COS(#EpsilonR); //beräknar hörnets y koordinat efter vridning
#xMove := (#X0 - #X3)*10; // beräknar fram förflyttning i x led
#yMove := (#Y0 - #Y3)*10; // beräknar fram förflyttning i y led
```

Bild 39: Beräkning för meduras vridning.

```
#X1 := #X0 - INT_TO_REAL("variabler".xSkiva/10) / 2; //beräknar lyftpunktens x värde.
#Y1 := #Y0 - INT_TO_REAL("variabler".ySkiva/10) / 2;//beräknar lyftpunktens y värde.
#L1 := SQRT(SQR(#X1 - #X2) + SQR(#Y1 - #Y2));//beräknar fram sträckan L1
#X4 := #X1; // x4 tilldelas
#Y4 := #Y1 + #L1;// y4 beräknas
#L2 := SQRT(SQR(#X4 - #X2) + SQR(#Y2 - #Y4));// Sträckan L2 beräknas
#LambdaR := ACOS(1 - (#L2 * #L2/ (2 * #L1 * #L1)));// vinklen Lambda beräknas fram i rad
#Lambda := #LambdaR * 180 / #Pi;// omvandlar Lambda till grader
#Epsilon :=90-#Lambda - #Alpha;// beräknar fram Epsillon
#EpsilonR := (#Epsilon * #Pi) / 180;// omvandlar Epsillon till rad
#X3 := #X1 + #L1 * COS(#EpsilonR);// beräknar hörnets x koordinat efter vridning
#Y3 := #Y1 + #L1 * SIN(#EpsilonR); //beräknar hörnets y koordinat efter vridning
#xMove := (#X0 - #X3)*10;// beräknar fram förflyttning i x led
#yMove := (#Y0 - #Y3)*10; // beräknar fram förflyttning i y led
```

Bild 40: Beräkning för moturs vridning.

PLCkod - AlphaCalc

```
#alphaout2 := #AlphaIn; //tilldelar alphaout2 som används i beräkningarna
#Alphaout := REAL_TO_INT(#AlphaIn * 100); // tilldelar alphout och skalar upp den. Ska till roboten.
IF #AlphaIn >90 THEN // Vänder vinklarna om de är felvända
    #Alphaout := 18000 - #Alphaout;
    #alphaout2 := 180 - #AlphaIn;
    ;
END_IF;

IF #motursaktiverad THEN //inverterar vinklen så roboten vrider åt rätthåll
    #Alphaout := #Alphaout * -1;
    ;
END_IF;
```

Bild 41: Räknar om Alpha om den kommer ut som fel värde. Skalar även om vinkeln så roboten kan ta emot den.

PLCkod - Programloop

```
CASE #Step OF

    0:  // hämtposition x,y,z beräknas fram för var skiva. Vinklen är standard
        "variabler".xToRobot := ("variabler".xSkiva / 2); // x-koordinat
        "variabler".yToRobot := ("variabler".ySkiva / 2); //y-koordinat
        "variabler".zToRobot := "variabler".zOffsetHämta + ("variabler".zSkiva * "variabler".antalSkivor) -
        (#antalPlock * "variabler".zSkiva); //z-koordinat
        "variabler".rzToRobot :=0; // vinkel på verktyget
        ;

    1: // kameraposition x,y,z beräknas fram för var skiva. Vinklen är standard

        "variabler".xToRobot := - "variabler".xSkiva / 2+ "variabler".xOffsetLämna;// x-koordinat
        "variabler".yToRobot := -(-"variabler".ySkiva /2 + "variabler".yOffsetLämna);//y-koordinat
        "variabler".zToRobot :=0; //z-koordinat
        "variabler".rzToRobot :=9000;// vinkel på verktyget
        #R_TRIG_Instance_5(CLK:="MotoMan_DB".inData.Step, //aktiv om step kommer ifrån roboten
                           Q=>"InSightLambdaCalc_DB_1".triggerBegärd); //triggar kameran
        ;

    2: //lämnaposition x,y,z och vinklen rz beräknas fram för var skiva.
        "variabler".xToRobot := ("variabler".xSkiva / 2 + "variabler".xOffsetLämna +"variabler".xToLoop);// x-koordinat
        "variabler".yToRobot := -("variabler".ySkiva / 2 + "variabler".yOffsetLämna +"variabler".yToLoop);// y-koordinat
        "variabler".zToRobot := "variabler".zOffsetLämna + #antalPlock * "variabler".zSkiva;// z-koordinat
        "variabler".rzToRobot :=9000-"variabler".rzToLoop;// korigerad vinkel
END_CASE;
```

Bild 42: Första delen av programloopen. Case-satsen som består av tre steg. Hämt-, Kamera- och lämnaposition.

```
#R_TRIG_Instance(CLK:="MotoMan_DB".inData.sugPuls,
                 Q=>#plock);
IF #plock THEN
    #antalPlock := #antalPlock + 1;
    ;
END_IF;

#IEC_Counter_0_Instance(CU:="MotoMan_DB".inData.Step,
                        R:=#resetCounter,
                        PV:=0,
                        CV=>#Step);
    #R_TRIG_Instance_6(CLK:=#Step=3,
                       Q=>#resetCounter);

    IF "Knapp3" THEN
        #antalPlock := 0;
        ;
    END_IF;
    IF "Knapp3" THEN
        #Step := 0;
        ;END_IF;
```

Bild 43: Andra delen av programloopen. Räknare för att hålla koll på antalet skivor och R_Trig för att registrera pulser från roboten.

Robotkod

```
*START
CALL JOB:POS_XYZ
SFTON P010 UF#(4)
MOVL C00000 V=350.0 PL=0
MOVL C00001 V=100.0 PL=0
PULSE OT#(33)
PULSE OT#(36)
MOVL C00002 V=350.0 PL=0
SFTOF
CALL JOB:POS_XYZ_2
SFTON P011 UF#(5)
MOVL C00003 V=1200.0 PL=0
MOVL C00004 V=150.0 PL=0
PULSE OT#(36)
SFTOF
MOVL C00005 V=1200.0 PL=0
CALL JOB:POS_XYZ_2
SFTON P011 UF#(5)
MOVL C00006 V=700.0 PL=0
MOVL C00007 V=100.0 PL=0
PULSE OT#(34) T=0.50
PULSE OT#(36)
MOVL C00008 V=350.0 PL=0
SFTOF
JUMP *START
END
```

Bild 44: Programkod för roboten.