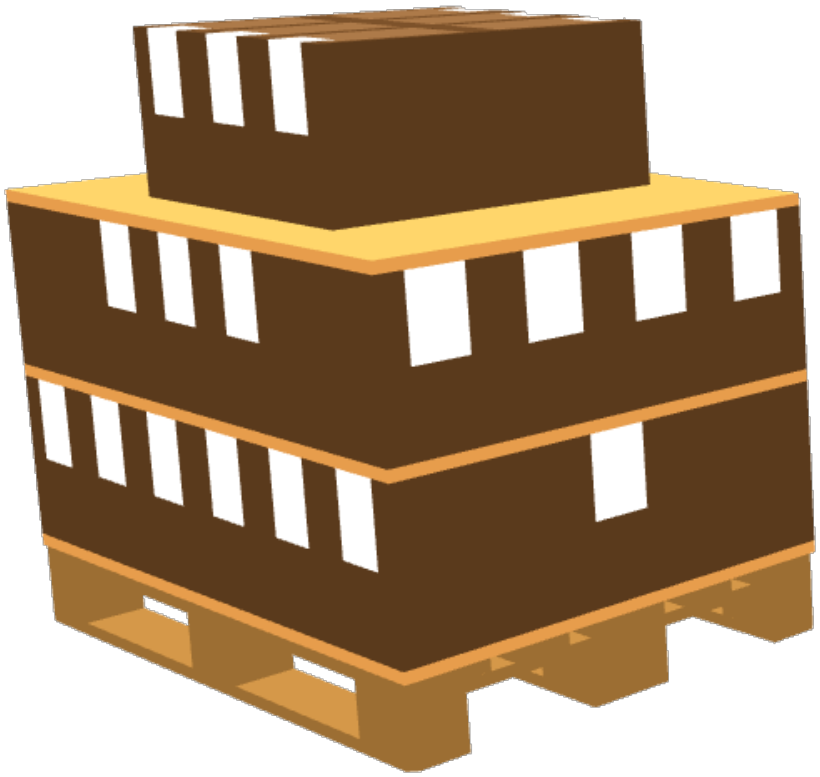
CHALMERS



Utveckling av skalbar PLC-styrning till palleteringscell

Examensarbete inom högskoleingenjörsprogrammet Elektroteknik

JOEL PERSSON

INSTITUTIONEN FÖR ELEKTROTEKNIK

CHALMERS TEKNISKA HÖGSKOLA

Göteborg, Sverige 2023

www.chalmers.se

Examensarbete 2023

Utveckling av skalbar PLC styrning till palleteringscell

Joel Persson

CHALMERS

Institutionen för Elektroteknik
CHALMERS TEKNISKA HÖGSKOLA
Göteborg, Sverige 2023

Utveckling av skalbar PLC styrning till palleteringscell
Joel Persson

Handledare:
Veronica Olesen, Institutionen för Elektroteknik
Jonas Persson, Evomatic AB

Examinator: Veronica Olesen, Institutionen för Elektroteknik

Examensarbete 2023
Institutionen för Elektroteknik
Chalmers Tekniska Högskola
SE-412 96 Göteborg
Telefon +46 31 772 1000

Omslag: Tredimensionell modell av en pall med produkter från användargränssnittet som utvecklats i projektet.

Institutionen för Elektroteknik
Göteborg, Sverige 2023

Abstract

The objective of this project, carried out at Evomaitc AB, was to research and develop the possibilities of scalable PLC program code that controls the picking pattern of a palletizing robot in projects. Integration of HMI is made to allow building of custom pallet pattern layouts to suit a broad application base. This means flexible regulations of the pallet pattern that is palletized by operators and reduced repetative programming for programmers.

Sammandrag

Målet för detta projekt, utfört på Evomatic AB, har varit att undersöka och utveckla möjligheterna av skalbar programkod för PLC som styr plockmönster för en palleteringsrobot i projekt. Integration med HMI har gjorts för att möjliggöra uppbyggnaden av unika pallmönster för att kunna passa produktspecifikationer i flera projektåtaganden. Den skalbara programkoden för PLC tillsammans med användargränssnittet medför en flexibel reglering av pallmönstret som palleteras av operatör och reducerad repetativ programmering för programmerare.

Förord

Detta examensarbete är utfört av student på Chalmers tekniska högskola under institutionen för elektroteknik. Projektet är den avslutande delen i utbildningen. Arbetet och skrivandet av rapporten har utförts under våren 2023.

Stort tack till företaget för given möjlighet av projektarbete.
Stort tack till handledare och examinator Veronica Olesen.

Joel Persson, Göteborg, juni 2023

Innehållsförteckning

1
Inledning

Följande kapitel ger en kort presentation av företaget där projektet har utförts, en beskrivning av systemet som utvecklats samt syfte, precisering av frågeställningen och avgränsningar för projektet.

1.1 Bakgrund

Projektet har utförts på Evomatic AB. Företaget är en komplett automationsleverantör som specialiserar sig inom mekanik, elektronik, robotteknik, programmering och installation. Projektåtaganden för företaget involverar hela robotlinjer inom flera olika industriområden.

Automatisering av produktions- och hanteringslinjer av produkter inom industrier innebär att hela eller delar av de arbetsstationer som finns längsmed linjen utförs av robotar istället för människor. Arbetsuppgifterna vid arbetsstationerna är väldigt olika och speglar varierade behov inom diverse industri. Det kan exempelvis innebära att svetsa komponenter inom bilindustrin, ihopsättning av möbler inom generell industri eller packning av mat inom livsmedelsindustrin.

En väsentlig arbetsstaion av linjen som frekvent automatiseras och som medför snabbt genomflöde samt strukturerad vidarehantering av produkter är palleteringscellen. Med cellen menas att det är ett komplett system inkluderat robot och all annan kringutrustning för att arbetsprocessen ska fungera. Palleteringsroboten används ofta i slutet av linjen för att palletera, det vill säga stapla, färdighanterade produkter som är ordnade tillsammans i något hanteringsmedium. I väldigt många fall används rätblocksformade wellpapp kartonger som hanteringsmedium.

Produkterna som palleteras av roboten placeras nästan alltid på någon sorts lastpall av standardiserad storlek. Europapall är ett exempel på en mycket vanlig lastpall i trä som är standardiserad i Europa. Pallens dimensioner är anpassade för transporter med järnväg och landsvägstransporter med lastbil är också anpassade efter dimensionerna. Pallen är utformad med en hålighet som möjliggör att den kan lyftas med gaffeltruck och handtruck. Detta gör vidarehantering av lastpall som blivit färdigpalleterad av robot till exempelvis ett lager med färdiga produkter som är redo för leverans av operatör mycket smidig.

I projektet ska det utvecklas ett system som är tänkt att underlätta integrationen av

palleteringsceller i framtida projektåtaganden genom att på ett mer dynamiskt och skalbart vis både bygga upp samt skicka plockmönster koordinater för palletering från mänskligt gränssnitt till robot.

Information om hur en pall ska staplas upp med produkter av palleteringscell byggs i form av lager. Ett lager består antingen av produkter, vanligtvis placeras produkterna i rätblocksformade wellpapp kartonger för hanteringsmedium, ett lager kan också bestå av ett mellanlägg, det vill säga ett tunt wellpapp ark som läggs för att antingen avgränsa produkter lager emellan och/eller för att skapa stabilitet.

Datan för lager som består av produkter blir då de tredimensionella kartesiska koordinaterna (X,Y,Z). Dessa koordinater beskriver var produkterna befinner sig i lagret relativt origo som sätts i något hörn på lagret. Utöver detta består datan av produkternas dimensioner, dess rotation och möjligtvis vald etikettriktning som är av intresse att hålla koll på, då den ibland önskvärt placeras utåt för ökad avläsbarhet. Datan för lager som består av mellanlägg blir en koordinat för placering i lagrets centrum.

I det utvecklade systemet konfiguerar användare först grundläggande parametrar för palluppbyggnad, val av palltyp, anger storlek på produkter och om eventuell riktning av etikett är av intresse. Nästa steg är att bygga lager med olika mönster genom att placera produkter enligt behov. Sista steget är välja ut och kombinera de nyss skapade lagren till komplett pallstruktur. Det är här mellanlägg kan läggas till efter behov. En tredimensionell rendering av pallens aktuella stadie, det vill säga de produktlager och mellanlägg, om sådana är tillagda, visas som visuellt stöd under pallens uppbyggnad för att få lättare och bättre uppfattning av det som konstrueras.

När pall är färdigbyggd skickas koordinater från alla ingående lager från redigerarens gränssnitt till utformat område i minnet hos det programmerbara styrsystemet Programmable Logic Controller (PLC). Lagringsområdet som används är Data Block (DB) för samling av vald data i det programmerbara styrsystemet. Vidare kommuniceras de sparade koordinaterna från PLC till kontrollenhet Industrial Robot Controller (IRC) för robot. Överföring sker med en kommunikationsstandard för fältbussar inom industriell automation, Process Field Buss (PROFIBUS) för att sedan avläsas och användas av roboten som positioner att gå till.

1.2 Syfte

Systemet som beskrivs i 1.1 är tänkt att undersöka och utveckla koncept för mer interaktivt användargränssnitt mellan människa och maskin, Human-Machine Interface (HMI) för interaktion mellan användare och PLC. Detta medför också skalbar del i PLC och IRC programkod som styr robots plockmönster eftersom att mönstret som ska plockas kan ändras genom gränssnitt av exempelvis operatör eller programmerare.

Det blir också smidig reglering och startkonfiguration av robotdrift för operatör

samt reducerad upprepning av grundläggande PLC programmering innehållande hårdkodning av olika varianter på produktstorlekar för programmerare.

1.3 Precisering av frågeställningen

Huvuduppdraget i projektet är att undersöka och utveckla vad det finns för framtida möjlighet till systemintegration med ökad skalbarhet inom programmering mellan modernare HMI på en PLC och en IRC till robot.
För att uppnå detta har två delmål utformats:

Delmål 1. Grafiskt användargränssnitt för att skapa palleterings pallmönster
Delmål 2. Programkod kommunikation mellan HMI, PLC och IRC

Vidare följer lite mer konkreta krav på de båda delmålen:

Krav 1. Gränssnittet ska kännas intuitivt, enkelt och responsivt så att det blir smidigt att använda.

Krav 2. Kommunikationen är optimerad, det vill säga så snabb och tydlig som möjligt. Den görs på ett resonabelt smidigt vis som tillåter varierande mängd data att skickas.

En strukturerad validering av konceptet för det utvecklade system och därigenom också de båda delmålen kommer också att genomgöras i projektet.

1.4 Avgränsningar

Funktionaliteten hos användargränssnittet för mönster uppbyggnad arbetar endast med rätblocksformade produkter som antas antingen ligga eller stå upp, det vill säga med en rotation på 0°, 90°, 180° eller 270°. Detta görs dels för att rotationen på en majoritet av produkter som palleteras i verkligheten placeras med räta vinklar relativt varandra för platseffektivisering. Men dels också för att kollisionskontroll vid utplacering av rätblocksformade produkter blir enklare för att produkter inte ska kunna överlappa varandra.

Omfattning av systemet som utvecklas blir endast utveckling av användargränssnitt och utveckling av kommunikation för palldata från användargränssnitt till styrenhet för robot. Utveckling och praktiskt testning av programkod för styrning av robot till de överförda koordinaterna kommer inte omfattas.

2
Teori

Följande kapitel beskriver system, programvaror, programspråk och bibliotek som använts under projektarbetets gång.

För att programmera och testa kommunikation i praktiken mellan webbaserat användargränssnitt som ligger på webbserver som PLC är värd för till PLC och sedan till styrenhet för robot har både nödvändig mjukvara och hårdvara tillhandahållits av företaget.

2.1 RobotStudio 2022

RobotStudio är ett program för offline programmering och simulering av robotapplikationer från ABB [1]. I projektet användes det till att på ett smidigare vis genom programmet koppla sig till styrenhet hos robot och Input/Output signaler konfigurerades samt programmering med RAPID kod.

RAPID är ett högnivåspråk som används för att styra industrirobotar från ABB [2]. I projektet användes det för att kommunicera med PLC för överföring av bland annat koordinater som sedan kan användas i programkoden för punkter som robot går till.

2.2 Siemens TIA Portal V17

Siemens TIA Portal är ett program från Siemens. Det är deras egna ingenjörsplatform som är en helhetslösning för att effektivt driftsätta, programmera och diagnosera Siemens automations hårdvara och mjukvara [3]. I projektet har det använts till att programmera, överföra kod till och diagnosera en PLC från Siemens.

Ladder logik (LAD) är ett programspråk som används för att programmera PLC. Det är grafiskt, vilket innebär att det representerar logiska operationer med symbolsika notationer [4]. Det är ett av programspråken som är definerade i den internationella standarden IEC 61131-3 för programmering av PLCs [5]. I projektet har det använts eftersom att det primära organsiationsblocket som utför programkod cykliskt i PLC är programmerad i detta språk. Det programmerades med hjälp av TIA Portalen.

Structured Control Language (SCL) är ett textbaserat högnivåspråk baserat på programspråket PASCAL som används för att programmera PLC. Det är Siemens egna

expansion på programspråket Structured Text (ST) som också är ett av språken i den internationella standarden IEC 61131-3 [6]. I projektet har det använts för att programmera programdel i PLC som kommunicerar med styrenhet till robot.

2.3 Visual Studio Code

Visual Studio Code är en gratis programkodsredigerare från Microsoft [7]. I projektet har det använts för att strukturera, programmera och simulera det webbaserade användargränssnittet till PLC.

HyperText Markup Language (HTML) är ett märkspråk för hypertext och är del i den grundläggande standarden för webbsidor [8]. I projektet har det använts för strukturen i det webbaserade användargränssnittet.

JavaScript (JS) är ett programspråk som används som skriptspråk för webbsidor [9]. Det kan hjälpa till att göra webbsidor interaktiva och responsiva. I projektarbetet har det använts för att göra användargränssnittet responsivt på inmatning från användaren.

Cascading Style Sheets (CSS) är ett programspråk som används för att beskriva hur stilen på HTML dokument ska se ut [10]. I projektet har det används för programmera design på användargränssnittet.

Three.js är ett bibliotek till JavaScript som gör det möjligt att skapa och visa animerad tredimensionell datorgrafik i en webbläsare med hjälp av WebGL [11]. I användargränssnittet används det för att tredimensionellt visualisera status på pall som håller på att byggas av användare.

jQuery är ett kompakt, snabbt och innehållsrikt bibliotek till JavaScript [12]. I användargränssnittet används det till så kallad drag and drop funktionalitet samt hjälper till att göra manipulering och händelsehantering enklare.

jQuery UI är en förlängning av funktionaliteten från jQuery med inriktning mot användargränssnitt [13]. Det används till att göra element flyttbara.

2.4 Blender

Blender är ett gratis modelleringsprogram för tredimensionell grafik med öppen källkod [14]. Det användes för att tillverka tredimensionella modeller av kartonger, mellanlägg och pallar för visuellt stöd till rendering i gränssnittet.

2.5 Inkscape

Inkscape är en gratis bildredigerare för vektorgrafik med öppen källkod [15]. Det användes för att tillverka skalbara bilder till användargränssnittet.

2.6 PLC

PLC är ett programmerbart styrsystem som används vid styrning inom automation av industriella processer [16]. Processorenheten programmeras efter behov för att med uppbyggd logik styra industriprocesser med hjälp av signaler på ingångar och utgångar. I projektet har en Siemens processorenhet CPU 1510SP F-1 PN PLC använts. Figur 2.1 nedan visar PLC som användes i projektet. Moduler kan anslutas till PLCn för att möjliggöra kommunikation med andra externa enheter. Till PLCn i projektet har en modul för Profibus fältbusskommunikation med Input/Output, I/O, kapacitet 128/128 bytes av data använts för att utbyta information med en IRC5 styrenhet för robot. PLCn programmerades och I/O portar för fältbusskommunikation konfigurerades med hjälp av Siemens egna mjukvara TIA Portal.

Figur 2.1. PLC som användes i projektet.

2.7 Profibus

Profibus är en standard inom fältbusskommunikation inom industriell automation. Fältbuss är en digital kommunikationsbuss som används inom industrin för att koppla samman automationsustrustning i ett nätverk. Profibus är tillverkaroberoende. Kommunikation mellan produkter från olika tillverkare ska därför kunna utföras utan någon särskild anpassning eller speciellt program [17]. Varianten av profibus som har applicerats i projektet har varit att PLC har varit styrande enhet och att IRC har varit arbetande enhet. I projektet har fältbusskommunikation använts mellan PLC och IRC5 styrenhet för att kommunicera palleteringsdata från användargränssittet.

2.8 IRC5 Controller

IRC5 är en styrenhet för industriella robotar från ABB [18]. I projektarbetet programmerades styrsystemet i RAPID genom RobotStudio. Figur 2.2 nedan visar styrenheten.

Figur 2.2. styrenheten som användes i projektet.

2.8.1 FlexPendant

FlexPendant är en handhållen programmeringsdosa som är kopplad till IRC5 styrenhet för robot [19]. Dosan har användargränssnitt och joystick som bland annat möjliggör manuell styrning av robot. Den är exemeplvis smidig för att snabbt läsa av signalvärden på ingång eller utgång. RAPID programmering kan utföras men projektet har programmerat styrenheten genom RobotStudio då datorprogrammet gör det visuellt mer tydligt. Figur 2.3 nedan visar pendanten.

Figur 2.3. Pendanten som användes i projektet.

2.9 IRB 360 FlexPicker

IRB 360 FlexPicker är en höghastighets plocka och placera robot från ABB. Robotcellen i projektet har använts som ett hjälpmedel för att praktisk förstå hur roboten agerar på styrning genom handhållen styrenhet FlexPendant och genom programkod. IRC5 styrenheten som beskrevs ovan var kopplad styrenhet till robot och hade hand om kommunikation och programkod. Figur 2.4 nedan visar roboten som styrenheten var kopplad till och som användes i projektet.

Figur 2.4. Roboten som användes i projektet.

3
Metod

Följande kapitel beskriver de metoder som använts under arbetet. Projektet kunde delas upp i tre delar: konstruktion av användargränssnitt för inmatning av data och uppbyggning av pallmönster, konstruera och etablera kommunikation mellan användargränssnitt, PLC och IRC5 kontrollenhet och slutligen implementering av resultat för konceptvalidering.

3.1 Användargränssnitt

Siemens har tillgängliga verktyg i TIA Portalen för att bygga upp grafiska användargränssnitt till skärmar och paneler kopplade till PLC. Dessa inbyggda verktyg fungerar bra till mindre interaktiva gränssnitt att exempelvis ta emot indata, visa värden på taggar, det vill säga variabler, i PLC och hantera enklare dragrörelser av användare.

Det första delmålet var att ett grafiskt användargränssnitt för att skapa palleterings pallmönster med tillhörande krav att användargränssnittet ska kännas intuitivt, enkelt och responsivt så att det blir smidigt att använda skulle utvecklas. Ett användargränssnitt av denna typen anses mer krävande funktionsmässigt än vad som på uppskattat rimligt vis kan utvecklas med ovannämnda standard verktyg.

Siemens har med några av deras senare HMI modeller introducerat konceptet av att integrera tekniken som redan används vid uppbyggnaden av webbplatser över hela internet till att programmera framtidens HMI med [20]. Det innebär att användargränssnitten är tänkta att kunna programmeras i HTML5, JavaScript och CSS, samt att grafiken som används är i SVG format.

Funktionaliteten som söks till användargränssnittet var nåbar med denna typ av teknik och gör att den läggs i framkant av vad som just nu är möjligt på detta planet. Detta gjorde att det därför valdes att utvecklas med hjälp av dessa verktyg.

Användargränssnittet utvecklades i programredigeraren Visual Studio Code som nämns 2.3. Det utnyttjades sedan att PLCn kan driva och vara värd för egen lokal webbserver som gränssnittet kan laddas till och agera användargränssnitt därifrån.

3.2 Kommunikation

Kommunikationen har implementerats för data som ska mellan användargränssnitt som finns på webbserver i PLC och PLC. Genom att en användare byggt upp en pall har användargränssnittet data på om ett lager innehåller produkter eller om det är ett mellanlägg, samt eventuella koordinater och rotationer på produkter. Både läsning och skrivning från och till PLC görs från programkod i gränssnittet med AWP kommandon. Det är en speciell sorts kommando syntax som används för utbyte av data mellan processorenhet på PLC och användargränssnitt, det vill säga HTML filer. Syntaxen och dess implementation förklaras mer utförligt i kommande kapitel.

Kommunikation har också implementerats för data som ska mellan PLC till kontrollenhet för robot. Här används Profibus fältbuss för att utbyta data då det är två fysiskt skiljda enheter som ska kommunicera. På PLC sidan är det ett programmerat funktionsblock som har i uppgift att cykliskt skicka över så mycket data den kan varje iteration sålänge som kontrollenheten speglar samma data som skickats, tillbaka, för valideringskontroll. Implementationsmässigt innebar det RAPID programming för kontrollenheten för styrning av programflöde vid kommunikation. Det behövdes också I/O konfiguration på kontrollenhetens sida då den behövde veta hur den ska tolka datan och på vilka portar den ska läsa av samt skicka tillbaks.

3.3 Konceptvalidering

Konceptvalidering förevisade systemets funktionalitet. Det gjordes genom att användargränssnittet laddades till TIA Portal projekt för PLCn för att agera genom webbserver. TIA Portal projekt laddades sedan till PLC med korrekt I/O konfiguration samt program block. RAPID programkod och I/O konfigueration laddades till kontrollenhet genom RobotStudio. Proceduren för hur systemet är tänkt att kunna användas testades genom att gå till webbplats som PLC är värd för. Uppbyggnad av pallmönster och ihopsättning av lager utfördes och skickades vidare. Manuell kontroll får sedan göras för att validera att den data som var tänkt att föras över till kontrollenheten verkligen har förts över.

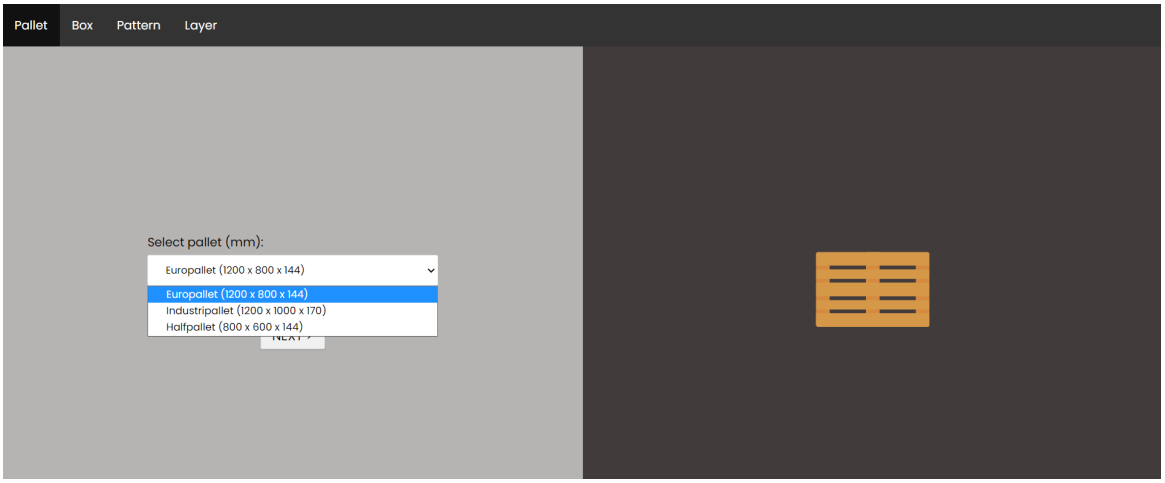
4

Resultat

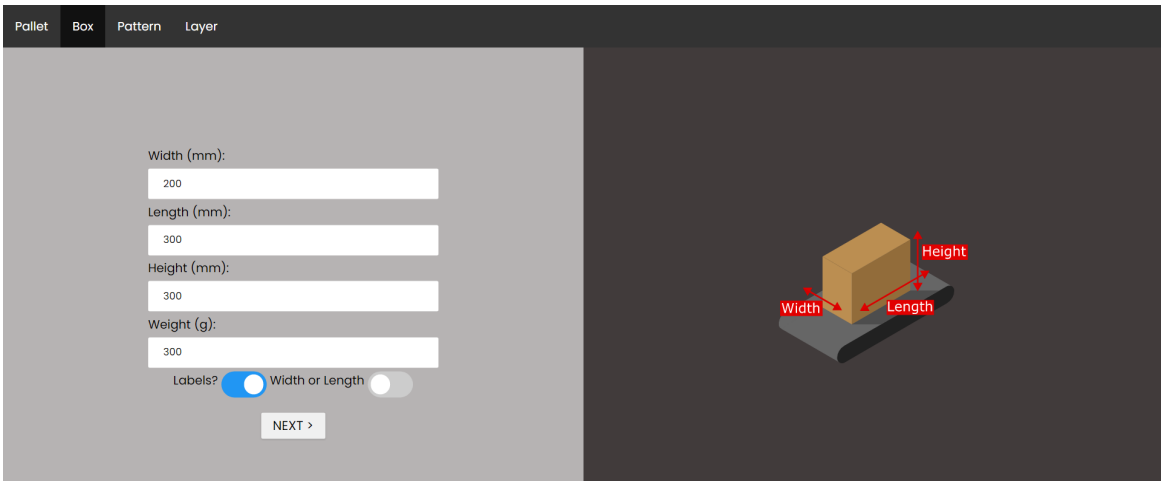
Följande kapitel beskriver arbetsprocessen som har utförts för att uppnå målen med projektet.

4.1 Användargränssnitt

Användargränssnittet utvecklades i programkodsredigeraren Visual Studio Code. Ett gratis tillägg användes i redigeraren, Live Server, som agerade drivande server under utvecklingen. Det innebar att programkod för webbsidorna som agerar användargränssnitt kunde simuleras och resultat av programkodsförändring uppdateras dynamiskt. I figur 4.1 och figur 4.2 nedan visas sidor i användargränssnittet där indata om egenskaper hos pall respektive låda som anges av användare.



Figur 4.1. Sida för val av palltyp ur lista som utvecklats för det grafiska användargränssnittet.



Figur 4.2. Sida för inmatning av produktdata som har utvecklats för det grafiska användargränssnitt.

Märkspråket HTML är grunden i webbsidorna som utgör användargränssnittet, syntaxen som består av text och symboler beskriver bland annat strukturen och formateringen för sidorna. Standardformuleringen av grundstrukturen för syntaxen visas nedan.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8" />
  <title>Hello, world!</title>
</head>
<body>
  <h1>Hello, world!</h1>
</body>
</html>
```

Programspråket CSS beskriver presentationsstilen på ett strukturerat dokument som HTML är. Exempel på egenskaper som går att ändra är typsnitt, textstorlek och färg. I projektet har ett gemensamt dokument används för att bestämma liknande egenskaper för allt som syns visuellt i användargränssnittet. Stilen som är designad är utformad för att kännas enkel och intuitiv att använda om man använder tekniska applikationer med grafiska användargränssnitt. Men också självlärande för att integration med nya operatörer ska gå så smidigt som möjligt. Exempel på strukturen för syntaxen visas nedan. I exemplet programmeras olika stilegenskaper såsom typsnitt och bakgrundsfärg för en oordnad lista (unordered list) som förekom i projektet.

```
ul {
  font-family: 'poppins';
  list-style-type: none;
  margin: 0;
  padding: 0;
  overflow: hidden;
  background-color: #333333;
}
```

Programspråket JavaScript användes som skriptspråk i webbsidorna som utgör användargränssnittet. Det används för att tillföra responsivitet, i de flesta fall för projektarbetet innebar det att behandla indata från tänkt operatör av användargränssnittet. Olika parametrar sätts beroende på pallstorlek som väljs i lista, indata fält för lådstorlek behöver valideras så att inmatad data är lämplig och för att uppnå intuitivitet förväntas något hända när man trycker på knappar och i menyer. Exempel på indata som behöver valideras visades i föregående bild 4.2. Indata från operatör där exempelvis lådans längd eller bredd är större än tidigare vald palls längd respektive bredd ska inte godtas. Nedanför visas syntaxen för språket genom en funktion som används för att detektera kollisioner mellan kanterna på rektangulära objekt.

```
function checkRectColl(rectOne, rectTwo){
    var r1 = $(rectOne);
    var r2 = $(rectTwo);
    var r1x = r1.offset().left -0.1;
    var r1w = r1.width()-0.1;
    var r1y = r1.offset().top-0.1;
    var r1h = r1.height()-0.1;
    var r2x = r2.offset().left -0.1;
    var r2w = r2.width()-0.1;
    var r2y = r2.offset().top-0.1;
    var r2h = r2.height()-0.1;
    if( r1y+r1h < r2y || r1y > r2y+r2h ||
        r1x > r2x+r2w || r1x+r1w < r2x ){
        return false;
    }else{
        return true;
    }
}
```

Funktionen ovan användes för att detektera kollisioner mellan tvådimensionella rektangulära representationer av lådorna i del där användare drar ut och släpper lådorna till önskad placering på pall. Om kollision uppstår illustreras detta visuellt, i enligthet mot Krav 1 1.3, genom att rött kryss hamnar över den låda som behandlas.

Det illustreras också på samma vis över samtliga redan placerade lådor som är under och därför i vägen för låda som hanteras. Beskriven händelse visas i figur 4.3 nedan.

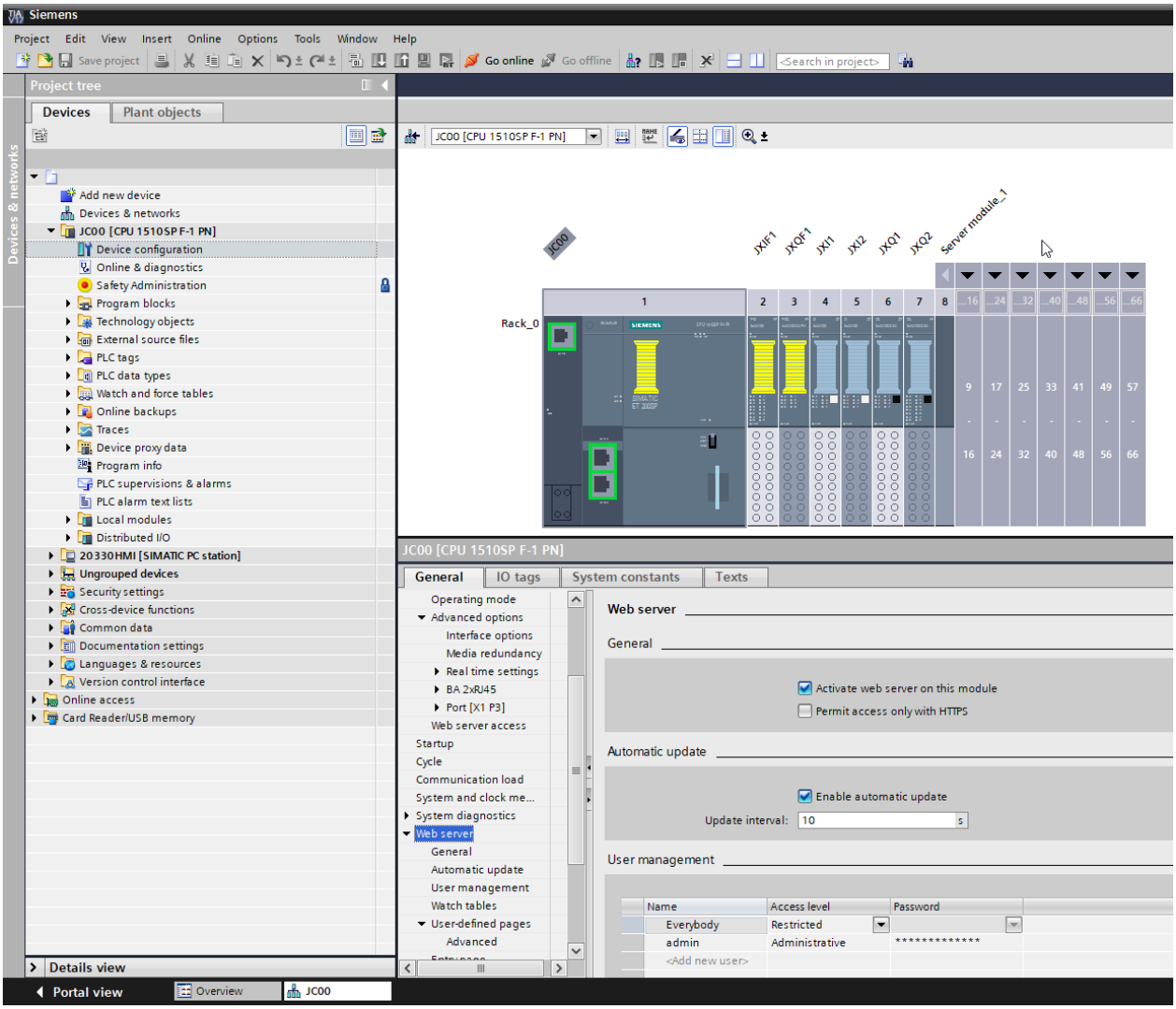
Figur 4.3. Exempel på visuell indikation för kollision mellan placerade produkter och hanterad produkt i sidan för mönsteruppbyggnad i användargränssnittet.

Produktmönster som byggs upp i 4.3 används sedan på sista sidan i användargränssnittet för att sätta samman uppbyggda lager och mellanlägg till hel pall efter behov. För att forsatt nå Krav 1 1.3 visas hela tiden en tredimensionell rendering av stadiet på pallen som hålls på att byggas. Detta ger ett visuellt stöd under sammansättningen. Lager markeras och läggs till genom knapptryckning. Mellanlägg läggs till genom knapptryckning. Tillagda lager och mellanlägg kan flyttas internt inom lagerhögen genom att trycka och dra till önskad position. Modellen uppdateras vid förändring av pallens stadie, alltså om lager eller mellanlägg tars bort eller läggs till samt om någon ordning skiftar. Denna sida visas nedan i figur 4.4.

Figur 4.4. Sida för pallbyggnad av skapade lager i användargränssnittet.

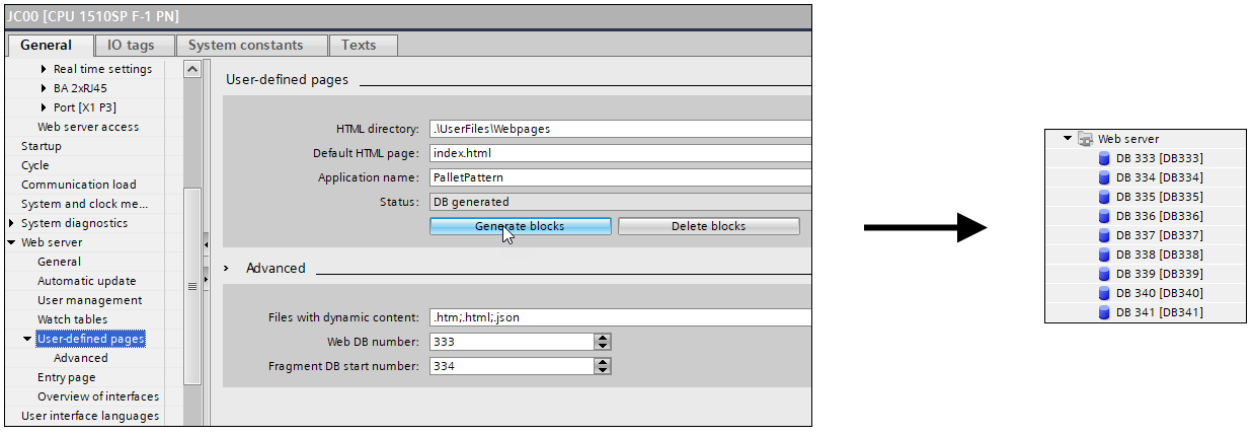
Implementationen av användargränssnittet i projektet använde sig av möjligheten att integrera användardefinerade webbsidor i Siemens PLC version S7-1500 [21]. Det innebar att egen utvecklad webbsida kunde laddas till integrerad webbserver på PLC och användas som användargränssnitt. Den integrerade webbservern på PLC

startades genom TIA Portal som visas i figur 4.5 nedan. För att data skulle kunna skickat mellan användargränssnit och PLC senare krävdes även att en användare lades till med rättigheter att skriva data. Under utveckling av projektarbetet användes exempelvis en användare benämnd admin med administrativa rättigheter. Detta visas också i figur 4.5 nedan.



Figur 4.5. Inställningfönster TIA Portal för start av integrerad webbserver samt skapa användare på PLC.

För att kunna använda det utvecklade användargränssnittet som användardefinerade webbsidor behövde alla filer placeras i bestämd mapp i projektmappen för TIA Portal tillhörande PLC. Det utvecklade användargränssnittet behövde därefter genereras till så kallade fragment som innebar att dess programkod delades upp i bytes som placerades i data block. Denna process illustreras i figur 4.6 nedan.

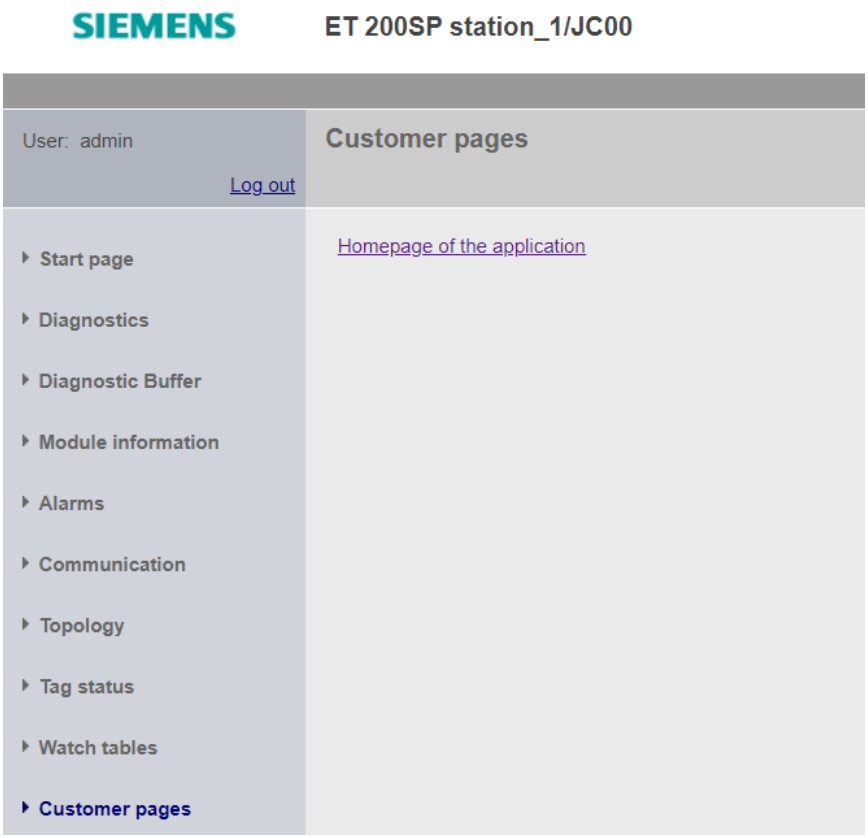


Figur 4.6. Inställningfönster TIA Portal fragmentuppdelning av det utvecklade användargränssnittet och några av de genererade data blocken.

De uppdelade webbsidorna behövde nu drivas på webbservern. Detta gjordes genom att implementera en tillgänglig instruktion kallad WWW. CPU på PLC visste inte att det var webbsidor som låg lagrade i fragment i data blocken som precis blivit genererade. Denna instruktion användes för att initialisera, det vill säga tala om detta för PLC vid uppstart. Den synkroniserade också de användardefinerade webbsidor för att kunna hantera interaktion i webbsidorna. För att hantera detta behövde instruktionen anropas cykliskt. Den anropades därför i det primära organisationsblocket, *Main OB1*, som kontinuerligt skannar och exekverar programkod i PLC. Grafisk representation av instruktionen WWW som användes visas i figur 4.7 nedan.

Figur 4.7. WWW instruktionen som initialiserar och cykliskt synkroniserar webbsidorna som utgör användargränssnittet.

Dessa konfigurationer kunde sedan laddas ned till PLC från TIA Portal via nätverkskabel. Om man sedan navigerade i en webbläsare till IP-adressen som blivit tilldelad PLC så kunde man nå de användardefinerade webbsidorna, det vill säga användargränssnittet. Först hamnade man på PLC webbsida i standard Siemens utformning som visas nedan i figur 4.8. Det gick sedan att logga in med hjälp av det tidigare konfiguerade kontot och navigera till användardefinerade webbsidorna där man möttes av samma användargränssnitt som i utvecklingsprocessen. Programkoden för det utvecklade användargränssnittet har valts att inte inkluderas i rapporten.



Figur 4.8. Standardwebbsidan med länk till användargränssnittet.

4.2 Kommunikation

Kommmunikation av data behövde göras på två platser i projektet. Första platsen är mellan det konstruerade användargränssnittet på webbserver och PLC. Den andra platsen är mellan PLC till kontrollenhet för robot.

4.2.1 Skrivning till PLC från HMI på webbserver

Som beskrivs i 3.2 är metoden att använda AWP-kommandon om man vill utföra skrivning och läsning av data mellan PLC och PLC-webbserver från Siemens [22]. Genom att i förväg i HTML programkoden för användargränssnittet definera vilka taggar i PLC som ska kommuniceras med möjliggörs skrivning och läsning av data. Exempel på strukturen för syntaxen presenteras nedan.

```
// (1)
<!-- AWP_In_Variable Name='"myDataBlock".myTag' -->

// (2)
:="myDataBlock".myTag:
```

Första raden programkod talar om för PLC att taggen *myTag* som är placerad i data block *myDataBlock* är skrivbar. Andra raden talar om att samma tagg kan komma att frågas efter dess värde, det vill säga läsbar. Denna kommunikation byggdes alltså upp genom att konfigurera denna syntax korrekt och skriva data till taggar som PLC är förberedd på att kan uppdateras. Integrationen gjordes när användargränssnittet redan var färdigbyggt. Detta eftersom att det då var tydligt vilken data som skulle överföras och var detta lämpligast gjordes i gränssnittets programkod.

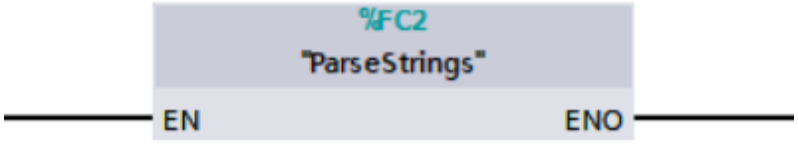
Av informationen som kunde hittas är det väldigt begränsat vad som idag är möjligt att göra med detta kommunikationsbibliotek. Det lämpar sig främst mot att läsa och skriva till enstaka taggar. Detta passade inte projektets behov då relativt komplicerade datastrukturer, fält med positionsstrukturer innehållande x-koordinat, y-koordinat, z-koordinat, rotation och mellanläggsflagga behövde skickas. Vilket hade inneburit att orimligt många taggar hade behövt definerats i förväg enligt ovan exempel. Om antalet taggar skulle behövas defineras i förväg skulle även skalbarheten och den dynamiska aspekten av programkoden minskat avsevärt, eftersom att de då måste regleras och ställas om manuellt beroende på antal positioner som är tänkt att skickas.

Projektet valde att lösa detta genom att sätta samman datan som skulle skickas till textsträngar. Datan i strängarna är avgränsade med kommatecken så att den enklare kan plockas isär och tolkas med hjälp av ett eget funktionsblock i PLC. Detta passade projektets behov bättre. En sträng i Siemens system har plats till 256 karaktärer varav 2 karaktärer är reserverade för stränghuvud. Vid försök att skicka strängar som användes till sin fulla kapacitet så erhölls problem i form av tappad data vid överföring. En faktor som ligger bakom begränsningen tror projektet är överföringskapaciteten på den interna webbservern. Genom testning fanns att placera 100 karaktärer per sträng fungerade bra. Detta gjorde att ett flera positioner alltså kunde skickat tillsammans på ett smidigare vis. Denna lösning är inte heller optimal, strängar, det vill säga taggar i PLC, behöver fortfarande defineras manuellt i användargränssnittets programkod. Det är dock enkelt att definera fler strängar än vad som behövs så att antalet positioner som i praktiken resulterar från en staplad pall täcks med god marginal.

4.2.2 Läsning i PLC från HMI på webbserver

När strängarna med koordinater skrivits till PLC skrevs även en flagga till hög. Den indikerade start för en egen funktion *ParseStrings* som anropades i det primära organisationsblocket, *Main OB1*. Funktionen skrevs i SCL och konstruerades för att iterativt analysera och tolka datan som placerats i strängarna. Den plockar ur datan skild med kommatecken och placerar in den i korrekt plats i ett datablock i PLC bestående av ett fält med koordinater. När all data i strängarna blivit hanterade, sattes ny flagga till hög vilket indikerade start av överföring till kontrollenhet. Koordinaterna var återigen lagrade på ett strukturerat sätt och därför redo att skickas vidare från PLC till kontrollenhet för robot. Grafisk representation av funktionen

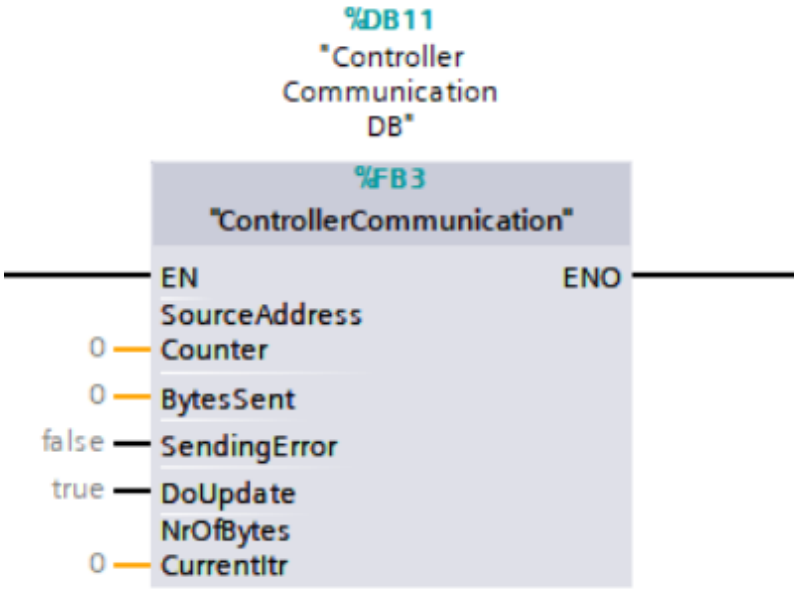
som konstruerats visas i figur 4.9 nedan. Programkod för funktionen återfinns i Appendix A.



Figur 4.9. Funktion ParseStrings som i PLC tolkar och plockar isär strängar.

4.2.3 Skrivning till styrenhet från PLC

För kommunikationen konstruerades ett eget funkionsblock *ControllerCommunication* som skrevs i SCL och som anropades från det primära organisationsblocket, *Main OB1*. Funktionsblocket utförde utbyte av data cykliskt med RAPID programkod i kontrollenhet. Modulen som användes i projektet för profibus fältbusskommunikation hade en maximal kapacitet på 128 bytes data in och 128 bytes data ut. Denna konfiguration användes. Varje koordinat använde 4 bytes per egenskap, x-koordinat, y-koordinat, z-koordinat och rotation, det vill säga 16 bytes per koordinat. Funktionsblocket skickar 7 koordinater, 112 bytes, per kommunikationsiteration. Resterande data användes för övrig kommunikation, exempelvis flagga för start av överföring samt antal koordinater som skulle kommuniceras totalt. Funktionsblocket skriver och läser till och från definerade ut- och ingångar med hjälp av instruktion för flytt av data som finns tillgänglig i programmet. Denna instruktionen skriver önskad mängd data från ett område i PLC till ett annat. Både läsaddress och skrivaddress godtar dynamiskt addressering som indata vilket möjliggör att variabel konfiguration kunde appliceras i slinga. För varje kommunikationsiteration av uppsättning koordinater jämfördes skickade koordinater med av kontrollenhet mottagna koordinater genom kontrollenhet speglade tillbaks mottagna koordinater till ingångar på PLC. För att nästa kommunikationsiteration skulle ske var egenskaperna hos koordinaterna tvungna att stämma överrens med varandra. Om någon olikhet uppstod, som det inte ska göra vid korrekt överföring, avbröts kommunicationsprocessen och flagga för indikation av fel sattes till hög. Grafisk representation av funktionsblocket som har konstruerats visas i figur 4.10 nedan. Programkod för funktionsblocket återfinns i Appendix B.



Figur 4.10. Funktionsblocket ControllerCommunication som i PLC kommunicerar med kontrollenhet för robot.

4.2.4 Läsning i styrenhet från PLC

Logik för läsning av data i kontrollenheten i kommunikationsiterationerna skrevs i RAPID genom RobotStudio via nätverkskabel kopplad till serviceport på kontrollenhet.

In och utgångar konfigurerades med hjälp av I/O konfigurator verktyg som fanns i RobotStudio. Detta innebar att portar i profibus fältbusskommunikationen kunde struktureras ihop korrekt och bindas till ett namn så att anrop i programkoden var möjlig. En viktig notering gällande hantering och tolkningen av bytes i konfigurationen var endian. Endian beskriver ordningen på bytes i ett digital heltal där flera bytes ingår. Siemens PLC använder sig av Big-Endian medans ABB kontrollenhet använder sig av Little-Endian. Detta innebar helt enkelt att obehandlad direkt överföring av datan skulle tolkats helt olika på de båda sidorna. Projektet löste detta genom att individuellt per byte manipulera ordningen bakvänt så att ordningen tolkades korrekt vid avläsning i kontrollenheten.

RAPID programkoden i kontrollenheten kördes cykliskt vid kommunikationen. Eftersom att kommunikationen började med att PLC skrev data till bestämda utgångar så började kontrollenheten med läsning av ingångar. Datan som lästes på ingångarna skrevs direkt till utgångarna på kontrollenhet för validering hos PLC. Om i nästa avläsningscykel flagga hade satts som godkände kontroll av skickade och mottagna koordinater placerade RAPID programmet koordinaterna i en lista och nästa kommunikationsiteration påbörjades. Listan som koordinaterna placerades i består av en speciellt struktur som beskriver gå punkter för robot, *robtargets*. Dessa gå

punkter kan sedan teoretiskt användas av senare programkod att styra robot att gå till. I omfånget av projektarbetet användes dessa punkter inte, men blir som en representation för hur man kan göra. Programkod för kommunikationen i RAPID återfinns i Appendix C.

4.3 Konceptvalidering

För att demonstrera koncept och funktionalitet hos det utvecklade systemet så utfördes en konceptvalidering i slutet av projektetarbetet. Det utfördes genom att simulera processen av att en operatör av användargränssnittet konfigurerar pall som beskrivits. Sedan utfördes manuell jämförelse av att koordinater för den uppbyggda pallen, produkter och mellanlägg korrekt kommunicerats och placerats i lista på kontrollenhet.

Palluppbyggnaden som användes vid konceptvalideringen är samma som visas i figur 4.4. När man sedan tryckte på överföra data så överfördes den först från användargränssnitt till PLC i form av strängar som beskrivits ovan. Överblick i PLC genom TIA Portal för detta stadiet visas figur 4.11 nedan. Notera strängarna längst upp i listan som innehöll egenskaper för koordinaterna.

tmpCoordDB

	Name	Data type	Offset	Start value	Monitor value	Retain	Accessible f...
1	Static						
2	tmpString0	String	0.0	''	'400,600,11,0,100...		
3	tmpString1	String	256.0	''	'350,500,172,90,3...		
4	tmpString2	String	512.0	''	'172,90,650,700,1...		
5	tmpString3	String	768.0	''	''		
6	tmpString4	String	1024.0	''	''		
7	stringsTransferDone	Bool	1280.0	false	FALSE		
8	nrTotStrings	Int	1282.0	5	5		
9	tmpStringArr	Array[0..4] o...	1284.0				
10	tmpStringArr[0]	String	1284.0	''	'400,600,11,0,100...		
11	tmpStringArr[1]	String	1540.0	''	'350,500,172,90,3...		
12	tmpStringArr[2]	String	1796.0	''	'172,90,650,700,1...		
13	tmpStringArr[3]	String	2052.0	''	''		
14	tmpStringArr[4]	String	2308.0	''	''		

Figur 4.11. Koordinaters egenskaper i strängar på PLC som blivit överförda från webbserver. Notera strängarna *tmpString0...4*.

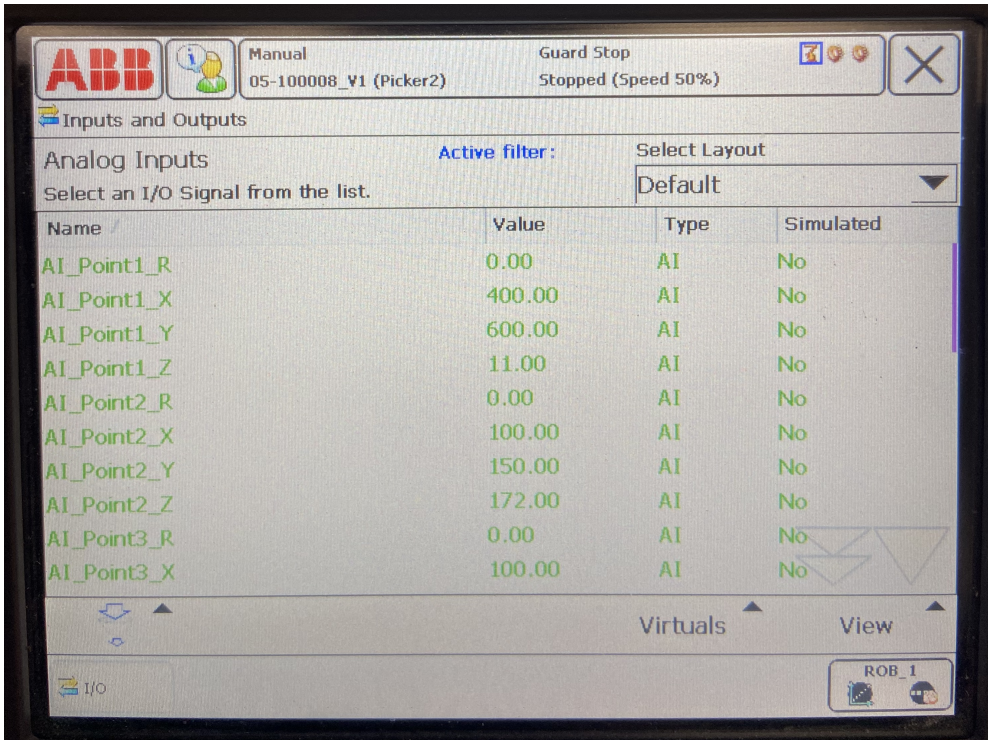
Nästa steg var tolkning koordinaternas egenskaper som befann sig i strängar till en lista av strukturellt korrekta koordinater för enklare vidarehantering. Detta visas i figur 4.12 nedan. Notera första värderna i *tmpString0* 400 och 600 som nu är placerade i X respektive Y värde för första koordinaten i *points*.

transferPointsDB

	Name	Data type	Offset	Start value	Monitor value	Retain	Accessible f...
1	Static						
2	points	Array[0..100] of "tr...	0.0				
3	points[0]	"transferPoint"	0.0				
4	X	DInt	0.0	0	400		
5	Y	DInt	4.0	0	600		
6	Z	DInt	8.0	0	11		
7	R	DInt	12.0	0	0		
8	points[1]	"transferPoint"	16.0				
9	X	DInt	16.0	0	100		
10	Y	DInt	20.0	0	150		
11	Z	DInt	24.0	0	172		
12	R	DInt	28.0	0	0		
13	points[2]	"transferPoint"	32.0				
14	X	DInt	32.0	0	100		
15	Y	DInt	36.0	0	450		
16	Z	DInt	40.0	0	172		
17	R	DInt	44.0	0	0		
18	points[3]	"transferPoint"	48.0				
19	X	DInt	48.0	0	100		
20	Y	DInt	52.0	0	750		
21	Z	DInt	56.0	0	172		
22	R	DInt	60.0	0	0		

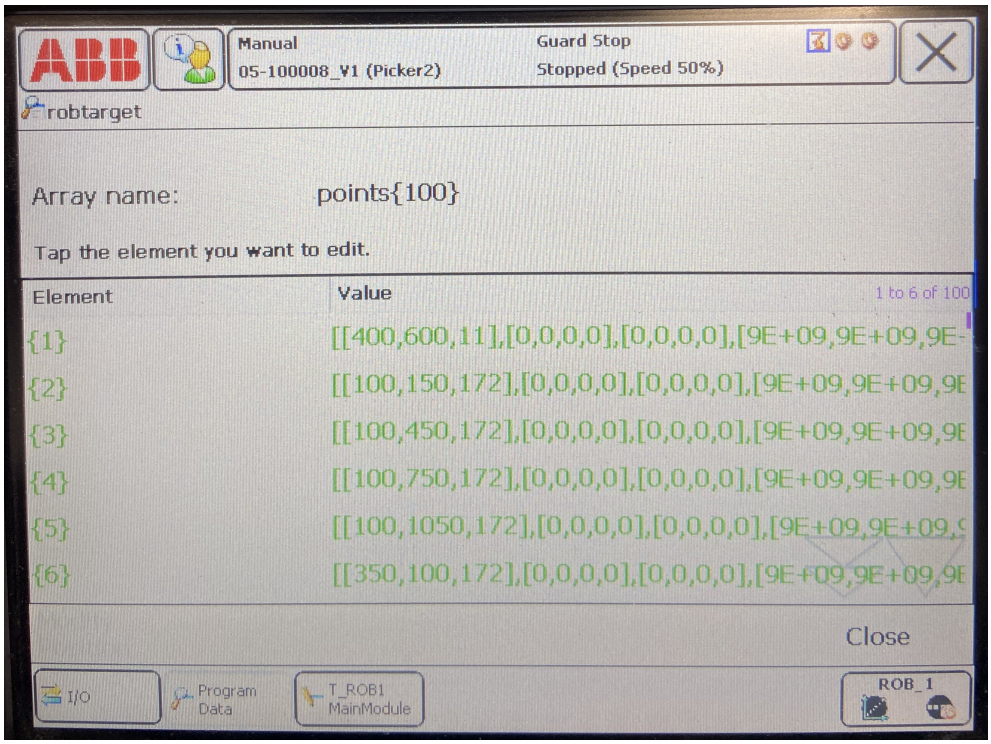
Figur 4.12. Koordinaters egenskaper tolkade och placerade i lista av koordinat strukturer.

Dessa koordinater kunde sedan överföras till kontrollenhet. PLC programkod placerar direkt första uppsättningen koordinater som ska överföras på konfigurerade utgångar. Hur det såg ut på de konfigurerade ingångarna på kontrollenheten visas nedan i figur 4.13. Notera att exempelvis *AI_Point1_X* och *AI_Point1_Y* stämmer överens med första punktens X och Y värde i PLC listan *points*.



Figur 4.13. Koordinaters egenskaper som indata på kontrollenheten sett på Flex-Pendanten.

Till slut exekveras RAPID programkod för iterativ igenomstegning av alla koordinater som skulle överföras. Mellan varje parti koordinater utfördes även validering. Koordinaterna placerades i lista med gå punkter till robot som kan användas vidare och blir slutprodukt för projektsystemet. Överblick över bit av listan visas i figur 4.14.



Figur 4.14. Listan innehållande robtarget objekt där koordinaterna är insatta som egenskaper.

5
Slutsats

Systemet som har utvecklats under projektarbetet har undersökt genom enklare konceptuell implementation vad det i framtiden finns för möjligheter inom systemintegrationer mellan PLC, HMI och IRC som tillför modernare användargränssnitt och en mer skalbar programkod till PLC. Sammanfattningsvis uppnåddes båda delmålen som var satta för projektet Krav 1 är uppnått. Krav 2 är delvis uppnått med god möjlighet till framtida utveckling.

5.1 Diskussion

Projektet har lyckats med att skapat ett verkligt system utifrån de delmål och krav som ställdes av företaget. Ett grafiskt användargränssnitt som agerar både representation och undersöker vad som idag är möjligt och fortfarande begränsat att integrera med PLC gällande en modern framkant. Användargränssnittet kan användas för att skapa unika pallmönster efter behov genom att kombinera egenbyggda lager och mellanlägg. Storleken på produkterna väljs också efter behov. Rotation på produkter hanteras. Kollisiondetektion, dra och släppa teknik samt etikett representation används för att få användargränssnittet att kännas intuitivt och responsivt.

Programkoden undersöker skalbarhet ur vinkeln att styrning ska kunna ändras dynamiskt utan förändringar i skriven programkod. Detta genom att palluppbyggnader skapar en abstraktion som gör det tillgängligt att ändra plockmönster utan programmeringserfarenhet. En breddad tillgänglighet bland operatörer och att programmeringskunnig inte behöver befinna sig på plats för eventuella omställningar av programkod skapar stor frihet i processen.

Framtida tillämplingar av resultatet går att se inom liknande systembehov. Delar av projektets implementationer kommer då troligen att användas som någon sorts referens för hur mer sofistikerade system inom samma inriktning kan utvecklas.

5.2 Hållbarhetsaspekter

Positiva arbetsmiljöaspekter kan tänkas finnas genom en mer givande arbetsmiljö för operatör av process där det utvecklade systemet appliceras. Detta för att proccessen kan regleras bättre och lättare på egen hand. En annan positiv arbetsmiljöaspekt är den minskande andelen upprepning av grundläggande programkod för programmerare. Med detta menas exempelvis när programkod för nytt plockmönster, till

skillnad från det utvecklade systemt, manuellt måste programmeras. En negativ arbetsmiljöaspekt kan vara att eventuell initial utveckling av ett mer avancerad system, där det utvecklade systemet ligger som referensram, kan bli väldigt omfattande. Det kan därför bli ett påfrestande projekt för företag.

En positiv miljöaspekt kan vara att det utvecklade systemet tillför mer anpassad palletering än nuvarande system. Detta gör att det är mindre slösat potentiellt lastningsutrymme per pall. Det bidrar till att vidare transport via exempelvis lastbil kan få med sig fler produkter på samma körd sträcka som tidigare. En annan positiv miljöaspekt kan vara att det utvecklade systemet kräver mindre resande för programmerare för att exempelvis programmera nytt plockmönster eller annat underhåll.

En etikaspekt som finns är att implementering av system enligt det utvecklade kan innebära att potentiella arbetstillfällen för programmerare i form av att manuellt programmera nya plockmönster blir färre. En annan etikaspekt är också på platser där palletering idag utförs för hand. Där implementering av system enligt det utvecklade skulle innebära att dessa arbetstillfällen försvinner helt eller minskar rejält.

5.3 Framtida utveckling

Som nämns i Krav 2 1.3 skulle kommunikationen i projektet vara optimerad. Den iterativa process som idag används vid kommunikation mellan PLC till kontrollenhet kan relativt enkelt optimeras bättre genom att egenskaperna hos koordinater skickas som en datatyp av mindre minne och därför använder färre bytes vid kommunikationen så att fler koordinater kan skickat samtidigt. I projektet använder varje egenskap till koordinaterna datatypen DInt som kräver 4 bytes det vill säga en summa av 16 bytes per koordinat. Anledningen är att implementationen av funktionsblocket som hanterar kommunikationen programmerades för att hantera egenskaper av samma storlek 4 bytes tidigare i projektet och fick helt enkelt leva kvar.

Återigen för att i högre grad uppnå Krav 2 1.3 och en tydligare kommunikation så ser projektet utvecklingsmöjlighet gällande överföring av koordinater mellan webbserver och PLC. Överföring av data placerat i strängar användes främst för att någon bättre metod inte upptäcktes inom projektet. Strängarna funkade i projektets begränsade omfattning för principiellt system. Hantering av strängar erhåller stor risk för att exempelvis felaktiga tecken eller oväntad ordning av förväntad teckenföljd kan introduceras ovetande. Detta skapar sedan problem eftersom att datan som placeras i koordinaterna exempelvis kan bli förskjuten åt något håll vilket ger fel resultat.

Referenser

[1] "RobotStudio Suite | ABB Robotics", [Online]. Available: https://new.abb.com/products/robotics/robotstudio (visited on 15/05/2023).

[2] "RAPID Instructions, Functions and Data types", [Online]. Available: https://library.e.abb.com/public/688894b98123f87bc1257cc50044e809/Technical%20reference%20manual_RAPID_3HAC16581-1_revJ_en.pdf(visited on 15/05/2023).

[3] "Totally Integrated Automation Portal", [Online]. Available: https://www.siemens.com/global/en/products/automation/industry-software/automation-software/tia-portal.html (visited on 16/05/2023).

[4] "Ladder Logic Basics", [Online]. Available: https://ladderlogicworld.com/ladder-logic-basics/ (visited on 16/05/2023).

[5] "IEC 61131-3: a standard programming resource - PLCopen", [Online]. Available: https://plcopen.org/sites/default/files/downloads/intro_iec_oct2016.pdf (visited on 16/05/2023).

[6] "Programming Guideline for S7-1200/1500", [Online]. Available: https://cache.industry.siemens.com/dl/files/040/90885040/att_970576/v1/81318674_Programming_guideline_DOC_v16_en.pdf (visited on 16/05/2023).

[7] "Visual Studio Code - Code Editing. Redefined", [Online]. Available: https://code.visualstudio.com/ (visited on 16/05/2023).

[8] "HTML: HyperText Markup Language - MDN Web Docs", [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/HTML (visited on 16/05/2023).

[9] "JavaScript - MDN Web Docs - Mozilla", [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/JavaScript (visited on 16/05/2023).

[10] "CSS - Cascading Style Sheets home page - W3C", [Online]. Available: https://www.w3.org/Style/CSS/Overview.en.html (visited on 16/05/2023).

[11] "Three.js - JavaScript 3D Library", [Online]. Available: http://www.threejs.org (visited on 15/05/2023).

[12] "jQuery", [Online]. Available: https://jquery.com (visited on 15/05/2023).

[13] "jQuery UI", [Online]. Available: https://jqueryui.com (visited on 15/05/2023).

[14] "blender.org - Home of the Blender project - Free and Open 3D Creation Software", [Online]. Available: https://www.blender.org (visited on 15/05/2023).

[15] "Inkscape: Draw Freely", [Online]. Available: https://inkscape.org (visited on 15/05/2023).

[16] "PLC - styrsystem i det mindre formatet", [Online]. Available: https://new.abb.com/se/om-abb/teknik/sa-funkar-det/plc (visited on 16/05/2023).
[17] "profibus.com - www.profibus.com", [Online]. Available: https://www.profibus.com/ (visited on 16/05/2023).
[18] "IRC5 - Industrial Robot Controller - ABB", [Online]. Available: https://new.abb.com/products/robotics/controllers/irc5-overview/irc5 (visited on 16/05/2023).
[19] "Operating manual - IRC5 with FlexPendant", [Online]. Available: https://icdn.tradew.com/file/201606/1569362/pdf/7083424.pdf (visited on 16/05/2023).
[20] "SIMATIC WinCC Unified System - Siemens", [Online]. Available: https://www.siemens.com/se/sv/produkter/industriautomation/simatic-hmi/wincc-unified.html#FramtidensHMIfamilj (visited on 16/05/2023).
[21] "Creating User- defined Web Pages on S7-1200 / S7-1500", [Online]. Available: https://cache.industry.siemens.com/dl/files/496/68011496/att_917318/v3/68011496_S7-1200_1500_Webserver_DOC_v22_en.pdf (visited on 24/05/2023).
[22] "S7-1500 Web server", [Online]. Available: https://www.hmkdirect.com/downloads/faq_docs/plc/s7-1500/s7-1500_webserver_function_manual.pdf (visited on 22/05/2023).

A

Appendix 1

PLC SCL programkod för funktion för tolkning av strängar. Notera att syntaxen på programkoden inte är helt korrekt på alla platser då långa referenser har behövts delas upp på flera rader för att få plats.

```
// Parse strings with coordinates in a comma separated
// format "X,Y,Z,R,X,Y,Z,R" into a data-block
// with an array of user defined PLC data types
// representing coordinates
IF "tmpCoordDB".stringsTransferDone THEN

    // Combine separate strings into an array
    // of strings for easier looping
    FOR #index := 0 TO 4 DO
        #byteOffsetIndex := #index * 256;
        #targetOffsetIndex := 1284 + (#byteOffsetIndex);
        POKE_BLK(area_src := 16#84,
                 dbNumber_src := 12,
                 byteOffset_src := #byteOffsetIndex,
                 area_dest := 16#84,
                 dbNumber_dest := 12,
                 byteOffset_dest := #targetOffsetIndex,
                 count := 256);
        ;
    END_FOR;

    #propCounter := 1;
    #pCounter := 0;
    FOR #i := 0 TO 4 DO
        #strLen := LEN("tmpCoordDB".tmpStringArr[#i]);
        #strTemp := "tmpCoordDB".tmpStringArr[#i];
        WHILE #strLen > 0 DO
            #delimPos := FIND(IN1 := #strTemp, IN2 := ',');
            IF #delimPos > 0 THEN
                #elementLen := #delimPos - 1;
                CASE #propCounter OF
                    1:
                        "transferPointsDB".points[#pCounter]
```

```
                    .X := STRING_TO_DINT(IN := LEFT(IN
                    := #strTemp, L := #elementLen));
                    ;
                2:
                    "transferPointsDB".points[#pCounter]
                    .Y := STRING_TO_DINT(IN := LEFT(IN
                    := #strTemp, L := #elementLen));
                    ;
                3:
                    "transferPointsDB".points[#pCounter]
                    .Z := STRING_TO_DINT(IN := LEFT(IN
                    := #strTemp, L := #elementLen));
                    ;
                4:
                    "transferPointsDB".points[#pCounter]
                    .R := STRING_TO_DINT(IN := LEFT(IN
                    := #strTemp, L := #elementLen));
                    ;
            END_CASE;
            #restLen := #strLen - #delimPos;
            #strTemp := RIGHT(IN := #strTemp,
                              L := #restLen);
        ;
    ELSE
        CASE #propCounter OF
            1:
                "transferPointsDB".points[#pCounter]
                .X := STRING_TO_DINT(IN := #strTemp);
                ;
            2:
                "transferPointsDB".points[#pCounter]
                .Y := STRING_TO_DINT(IN := #strTemp);
                ;
            3:
                "transferPointsDB".points[#pCounter]
                .Z := STRING_TO_DINT(IN := #strTemp);
                ;
            4:
                "transferPointsDB".points[#pCounter]
                .R := STRING_TO_DINT(IN := #strTemp);
                ;
        END_CASE;
        #strTemp := '';
    END_IF;
    #strLen := LEN(#strTemp);
```

```
            // Update property counter
            #propCounter := #propCounter + 1;
            IF #propCounter > 4 THEN
                #pCounter := #pCounter + 1;
                #propCounter := 1;
                ;
            END_IF;
            ;
        END_WHILE;
        ;
    END_FOR;

    // Clear strings with coords
    "tmpCoordDB".tmpString0 := '';
    "tmpCoordDB".tmpString1 := '';
    "tmpCoordDB".tmpString2 := '';
    "tmpCoordDB".tmpString3 := '';
    "tmpCoordDB".tmpString4 := '';

    // Reset flag that triggers this function
    "tmpCoordDB".stringsTransferDone := false;

    // Set total number of points to send robot controller
    "PalletPatternData".totNrPoints := #pCounter;

    // Set flag start sending data to robot controller
    "PalletPatternData".startSendingToController := true;
    ;
END_IF;
```

B

Appendix 2

PLC SCL programkod för funktionsblock för kommunikation med kontrollenhet. Notera att syntaxen på programkoden inte är helt korrekt på alla platser då långa referenser har behövts delas upp på flera rader för att få plats.

```
// Writing 4 byte sized (DInt) parts of points to profibus
// output addresses, 128 bytes out, max 112 (7 points,
// 16 bytes per point) bytes per cycle, first 16 bytes
// reserved for other communication, validation of sent
// bytes before sending next cycle

#TotalNrOfBytes := ("PalletPatternData".totNrPoints * 16);

IF "PalletPatternData".startSendingToController AND
   NOT #SendingError THEN

    // Keep track of how many bytes that is sent
    IF #DoUpdate THEN
        IF (#BytesSent + 112) <= #TotalNrOfBytes THEN
            #NrOfBytesCurrentItr := 112;
            #BytesSent := #BytesSent + 112;
        ELSE
            #NrOfBytesCurrentItr := #TotalNrOfBytes
                                    - #BytesSent;
            #BytesSent := (#BytesSent)
                          + (#TotalNrOfBytes - #BytesSent);
        END_IF;
        #DoUpdate := NOT #DoUpdate;
    END_IF;

    // Writing total number of points to transfer
    POKE_BLK(area_src := 16#84,
             dbNumber_src := 14,
             byteOffset_src := 4,
             area_dest := 16#82,
             dbNumber_dest := 0,
             byteOffset_dest := 162,
             count := 2);
```

```
// Writing to profibus output
POKE_BLK(area_src := 16#84,
         dbNumber_src := 15,
         byteOffset_src := #SourceAddressCounter,
         area_dest := 16#82,
         dbNumber_dest := 0,
         byteOffset_dest := 166,
         count := #NrOfBytesCurrentItr);

// Set flag indicating send start
POKE_BOOL(area := 16#82,
          dbNumber := 0,
          byteOffset := 158,
          bitOffset := 0,
          value := TRUE);

// Get flag request compare coordinates from controller
#CompareRequested :=
PEEK_BOOL(area := 16#81,
          dbNumber := 0,
          byteOffset := 158,
          bitOffset := 0);

IF #CompareRequested THEN

    // Set flag indicating reset compare request
    POKE_BOOL(area := 16#82,
              dbNumber := 0,
              byteOffset := 158,
              bitOffset := 4,
              value := TRUE);

    // Update source address pointer,
    // calculate validation iterations, toggle update
    #SourceAddressCounter := #SourceAddressCounter
                           + #NrOfBytesCurrentItr;
    #CheckIterations :=
    DINT_TO_INT(#NrOfBytesCurrentItr / 4);

    #DoUpdate := NOT #DoUpdate;

    // Check that all read values match sent values
    // If not, flag error, exit and dont continue send
    FOR #CheckCounter := 0 TO #CheckIterations DO
        #ReadValue := PEEK_DWORD(area := 16#81,
```

```
                                        dbNumber := 0,
                                        byteOffset := 166 +
                                        (#CheckCounter * 4));
            #SentValue := PEEK_DWORD(area := 16#82,
                                        dbNumber := 0,
                                        byteOffset := 166 +
                                        (#CheckCounter * 4));

            // Exit if e.g. sent x and returned to be
            // checked x dont match
            IF #ReadValue <> #SentValue THEN
                #SendingError := TRUE;
                EXIT;
            END_IF;

            // Last iteration no errors compared points OK
            // If all sent, transfer complete, stop function
            IF #CheckCounter = #CheckIterations THEN
                POKE_BOOL(area := 16#82,
                          dbNumber := 0,
                          byteOffset := 158,
                          bitOffset := 1,
                          value := TRUE);

                IF (#BytesSent = #TotalNrOfBytes) THEN
                    "PalletPatternData"
                    .startSendingToController := false;
                    ;
                END_IF;
                ;
            END_IF;
        END_FOR;
    END_IF;
END_IF;
```

C

Appendix 3

RAPID programkod för kommunikation med PLC.

```
MODULE MainModule

    CONST num MAX_ARRAY_SIZE := 100;
    PERS robtarget points{MAX_ARRAY_SIZE};
    PERS num nrofentities := 0;
    PERS num currentarraysize := 0;
    PERS bool restartarray := TRUE;
    VAR bool startHsComplete := FALSE;

        PROC main()

        IF TestDI(DI_Init) AND startHsComplete = FALSE THEN
            SetDO DO_StartConfirmed, 1;
            startHsComplete := TRUE;
        ENDIF

        IF NOT TestDI(DI_Init) THEN
            SetDO DO_StartConfirmed, 0;
        ENDIF

        IF startHsComplete = TRUE THEN
            readpoints;
        ENDIF

        ENDPROC

    ! Function for reading positions from PLC
    ! Store all positions in array of robtargets
    PROC readpoints()

        ! Reset array with robtargets
        IF restartarray = TRUE THEN
            currentarraysize := 0;
            restartarray := FALSE;
            nrofentities := GI_NrOfEntities;
```

```
ENDIF

! "Send" back read values for validation
SetAO AO_Point1_X, AdjustValue(AI_Point1_X);
SetAO AO_Point1_Y, AdjustValue(AI_Point1_Y);
SetAO AO_Point1_Z, AdjustValue(AI_Point1_Z);
SetAO AO_Point1_R, AdjustValue(AI_Point1_R);

SetAO AO_Point2_X, AdjustValue(AI_Point2_X);
SetAO AO_Point2_Y, AdjustValue(AI_Point2_Y);
SetAO AO_Point2_Z, AdjustValue(AI_Point2_Z);
SetAO AO_Point2_R, AdjustValue(AI_Point2_R);

SetAO AO_Point3_X, AdjustValue(AI_Point3_X);
SetAO AO_Point3_Y, AdjustValue(AI_Point3_Y);
SetAO AO_Point3_Z, AdjustValue(AI_Point3_Z);
SetAO AO_Point3_R, AdjustValue(AI_Point3_R);

SetAO AO_Point4_X, AdjustValue(AI_Point4_X);
SetAO AO_Point4_Y, AdjustValue(AI_Point4_Y);
SetAO AO_Point4_Z, AdjustValue(AI_Point4_Z);
SetAO AO_Point4_R, AdjustValue(AI_Point4_R);

SetAO AO_Point5_X, AdjustValue(AI_Point5_X);
SetAO AO_Point5_Y, AdjustValue(AI_Point5_Y);
SetAO AO_Point5_Z, AdjustValue(AI_Point5_Z);
SetAO AO_Point5_R, AdjustValue(AI_Point5_R);

SetAO AO_Point6_X, AdjustValue(AI_Point6_X);
SetAO AO_Point6_Y, AdjustValue(AI_Point6_Y);
SetAO AO_Point6_Z, AdjustValue(AI_Point6_Z);
SetAO AO_Point6_R, AdjustValue(AI_Point6_R);

SetAO AO_Point7_X, AdjustValue(AI_Point7_X);
SetAO AO_Point7_Y, AdjustValue(AI_Point7_Y);
SetAO AO_Point7_Z, AdjustValue(AI_Point7_Z);
SetAO AO_Point7_R, AdjustValue(AI_Point7_R);

SetDO DO_RequestCompare, 1;
WaitDI DI_PointsOK, 1;
WaitDI DI_ResetCompare, 1;
SetDO DO_RequestCompare, 0;

! Adding coordinates to array
InsertPoint AI_Point1_X, AI_Point1_Y, AI_Point1_Z;
InsertPoint AI_Point2_X, AI_Point2_Y, AI_Point2_Z;
```

```
        InsertPoint AI_Point3_X, AI_Point3_Y, AI_Point3_Z;
        InsertPoint AI_Point4_X, AI_Point4_Y, AI_Point4_Z;
        InsertPoint AI_Point5_X, AI_Point5_Y, AI_Point5_Z;
        InsertPoint AI_Point6_X, AI_Point6_Y, AI_Point6_Z;
        InsertPoint AI_Point7_X, AI_Point7_Y, AI_Point7_Z;

    ENDPROC

    PROC InsertPoint(num x, num y, num z)
        ! Note, RAPID array index starts at 1
        Incr currentarraysize;
        IF currentarraysize <= nrofentities THEN
            points{currentarraysize} :=
            [[x,y,z],[0,0,0,0],[0,0,0,0],
            [9E+09,9E+09,9E+09,9E+09,0,0]];
        ENDIF
    ENDPROC

    ! Adjust value to counter scaling done by
    ! SetAO default instruction
    FUNC num AdjustValue(num val)
        IF val>0 THEN
            RETURN val+1;
        ELSEIF val<0 THEN
            RETURN val-1;
        ELSE
            RETURN val;
        ENDIF
    ENDFUNC

ENDMODULE
```

INSTITUTIONEN FÖR ELEKTROTEKNIK
CHALMERS TEKNISKA HÖGSKOLA
Göteborg, Sverige
www.chalmers.se

CHALMERS