Optimering av produktionslinje genom utveckling och implementering av ett automatiserat mellanlager

Av

Simon Sehr och Kristjan Jonsson

LUNDS UNIVERSITET
Lunds Tekniska Högskola

Department of Electrical and Information Technology Faculty of Engineering, LTH, Lund University SE-221 00 Lund, Sweden

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 2023

Sammanfattning

Detta arbete har gjorts hos TePe Munhygienprodukter AB och författarna studerar högskoleingenjör elektroteknik med automation vid Lunds Tekniska Högskola.

Projektet syftar till att utveckla och implementera ett automatiserat mellanlager för hantering av pallar med olika produkter. Systemet inkluderar en robot som smidigt och säkert hämtar och lämnar pallar från och till ett transportbandsystem. Systemet hanterar och flyttar pallar mellan olika transportband baserat på specifika kriterier som produkterna har. De ingående momenten för att bygga systemet har varit att konstruera och koppla ett elskåp, använda tekniker som PLC-programmering i CODESYS V3, samt använda ett overheadsystem för kommunikation mellan maskiner och robotar. Projektet har visat att det går att använda sig av ett mellanlager för att förbättra effektiviteten och produktiviteten inom produktionsprocessen. Vidare visar tester att det utvecklade systemet är kapabelt att hantera och flytta pallar på ett effektivt och säkert sätt, samtidigt som det möjliggör en hög grad av flexibilitet och anpassningsförmåga.

Nyckelord

Produktionslinje
Optimering
Mellanlager
Automation
PLC-programmering
Robotik

Abstract

This work has been carried out at TePe Oral Hygiene Products AB, and the authors are studying Electrical Engineering with Automation at Lund University of Technology.

The project aims to develop and implement an automated intermediate storage system for handling pallets with various products. The system includes a robot that efficiently and safely retrieves and delivers pallets to conveyor belt systems, which in turn manage and move pallets between different conveyor belts based on specific product criteria. This was made by constructing and connecting an electrical cabinet, using techniques such as PLC programming in CODESYS V3, and using an overhead system for communication between machines and robots. The project has demonstrated the feasibility of using an intermediate storage solution to improve efficiency and productivity within the production process. Tests show that the developed system is capable of handling and moving pallets effectively and securely, while offering a high degree of flexibility and adaptability.

Keywords

Production line
Optimization
Intermediate storage
Automation
PLC programming
Robotics

Förord

Denna rapport avslutar vår utbildning på Lunds Tekniska Högskola där vi har studerat elektroteknik med automation. Vi vill med detta förord passa på att tacka TePe Munhygienprodukter AB för att de har fått möjlighet att genomföra sina examensjobb hos dem. Vi riktar ett särskilt tack till Mats Lilja, Christian Nyberg från LTH samt Tove Mattsson, Albert Folkesson, Thomas Hansson, Bekim Osmani och Mats Larsson från TePe för deras stöd och hjälpsamhet under arbetets gång.

Innehållsförteckning

1. Inledning

För att bibehålla sin konkurrenskraft i dagens föränderliga, konkurrensutsatta och globala ekonomi måste alla företag kontinuerligt ifrågasätta, utveckla och förbättra sina produktionsprocesser för att fortsätta vara relevanta. Automatisering av produktionslinjer och produktionsprocesser har visat sig vara en nyckelfaktor för att uppnå detta.

Automation av produktionslinjer innebär utveckling av mjukvara, central styrning och övervakning av maskiner och processer för att minska behovet av mänsklig inblandning. Genom att minska den mänskliga inblandningen i produktionsprocessen minskar arbetskraftskostnader samtidigt som effektiviteten och produktiviteten ökar.

Andra fördelar med att minska mänsklig inblandning i industriella processer är även att kvaliteten blir jämnare och mer enhetlig samtidigt som risken för mänskliga fel minimeras. Vidare ger det en flexibilitet i att kunna snabbt anpassa produktionen efter förändrade marknadsförhållanden och kundbehov. Slutligen bidrar automation till en förbättrad arbetsmiljö då riskfyllda och repetitiva moment kan separeras från människor och risken för skador och olyckor minimeras [1].

Att bygga om redan befintlig utrustning är ett ekonomiskt och miljövänligt alternativ för att öka automatiseringsgraden och något som kommer spela en nyckelroll för de företag som har framförhållningen att förstå dess betydelse. Denna rapport kommer att stegvis gå igenom hur tidigare avvecklad utrustning hos TePe Munhygienprodukter AB genom ombyggnation och mjukvaruutveckling kan konverteras till ett automatiserat mellanlager för deras EasyPick-tandpetare.

1.1 Bakgrund

TePe Munhygienprodukter AB är ett svenskt företag som grundades 1965 och har för närvarande cirka 450 anställda. De är specialiserade på att tillverka munhygienprodukter såsom tandborstar, tandtråd och EasyPick tandpetare. Företaget har sitt huvudkontor och produktionsanläggning i Malmö, där de bedriver en avancerad och automatiserad produktionslinje [2].

I syfte att öka flexibiliteten i produktionen av EasyPick tandpetare och minimera stilleståndstider i produktionslinjen vill TePe optimera sina förpackningsmaskiner så att de är så effektiva som möjligt under bemannade arbetstimmar. En strategi för att uppnå detta är att omvandla befintliga, för närvarande oanvända, lådväxlare till mellanlager där färdigtillverkade men ännu inte förpackade produkter kan lagras temporärt.

Lådväxlare är en typ av transportbandssystem där maskiner placerar färdigbyggda komponenter direkt i lådor som sedan transporteras vidare i produktionsprocessen. Tidigare användes lådväxlare i produktionslinjen, men de har ersatts av mobila robotar som sköter transporten av förvaringslådorna. Nu, när lådväxlarna står oanvända, planerar TePe att återanvända dem för att skapa en buffert i produktionen av deras EasyPick tandpetare.

1.2 Syfte

Syftet med examensarbetet är att utvärdera om det går att skapa fungerande mellanlager för produktionen av EasyPick tandpetare. Det förväntade resultatet ska avgöra om det går att realisera idén om ett mellanlager som ska göra att delar av produktionslinjen står still så kort tid som möjligt.

1.3 Målformulering

Examensarbetet ska utvärdera ombyggnad av en lådväxlare för att se om det är lämpligt att konvertera oanvända sådana till ett litet mellanlager för att få en ökad flexibilitet i produktionen av EasyPick hos TePe.

1.4 Problemformulering

Inom detta examensarbete har sju frågor identifierats som särskilt intressanta att fördjupa sig kring och besvara. Det är sex delfrågor som gemensamt leder fram till fråga sju som representerar examensarbetets hela syfte och mål för både TePe och gruppen. Dessa frågor kommer att besvaras genom examensarbetet.

1. Hur fungerar befintliga lådväxlare och robotar idag?
2. Kan ett nytt PLC-program göras från grunden för styrning av motordrifter till banor och hiss?

3. Kan kommunikation mot mobil robot läggas till för att samverka med detta nya PLC-program?
4. Hur kan befintliga lådväxlare byggas om till ett mellanlager?
5. Hur kan säkerheten för en större cell med t.ex. fyra lådväxlare se ut?
6. Hur kan man i detta mellanlager säkerställa att produkter inte ligger för länge?
7. Är det lämpligt att konvertera de lådväxlare som inte används längre till ett mellanlager?

1.5 Motivering

Examensarbetets föreslagna omfattning passade väldigt väl inom ramen för examensarbetet samt att de ingående delarna matchade båda examensarbetarnas intressen inom programmering, mjukvarans koppling till hårdvara och automation. TePe är välkända för deras moderna produktionslinje och har skapat stora delar av den in-house vilket ger examensarbetarna vad de tror kan bli unika möjligheter för personlig utveckling hos dem.

TePe strävar efter att öka flexibiliteten i tillverkningen av EasyPick-tandpetare så att förpackningsmaskinerna kan stå stilla under en kortare tid för att sedan köras för fullt när de är bemannade. Därför ansåg examensarbetarna att det var ett utmärkt tillfälle att genomföra ett projekt som faktiskt var avsett att senare tas i bruk av företaget.

1.6 Avgränsningar

- Det ingår inte något automationstekniskt arbete för själva produktionen i fabriken.
- Det ingår inte litteraturstudier av teoretiska begrepp.
- Den skrivna PLC-programmeringen är inte menat att fungera på eller utveckla övriga maskiner eller delar av produktionslinjen.
- Det ingår inte att examensarbetarna ska bestämma vilken arbetsmiljö som styr-programmeringen ska skrivas i.
- Det ingår inte att examensarbetarna ska bestämma vilket språk som ska användas för PLC-programmeringen.
- Det ingår inte att konstruera den hårdvara som behövs för att implementera mellanlagret.

2. Teknisk bakgrund

I det här avsnittet kommer den tekniska bakgrunden presenteras som gäller för de olika komponenterna och systemen som är involverade i projektet. Produkten som är tänkt att lagras i mellanlagret, EasyPick, kommer att presenteras och den PLC som hanterar styrsignalerna kommer att beskrivas övergripligt. Vidare kommer det att beskrivas vilka komponenter som ingått i elskåpet, vad ett HMI är och vad det används till och en kort beskrivning av vad en asynkronmotor är. Avslutningsvis beskrivs overheadsystemet som sköter orderhanteringen i nätverket, teknisk data om transportbanden samt den programmeringsmiljö som har använts. Genom en grundlig förståelse för de tekniska delar som ingår i helheten och hur de samverkar kommer läsaren att enklare förstå den arbetsprocess som sedan kommer att beskrivas.

2.1 Produkten

I detta avsnitt så beskrivs produkten som arbetats med under projektet.

Figur 1. EasyPick tandpetare i olika storlekar.

Easypick från TePe är en slags tandpetare som är utvecklade för att rengöra mellan tänderna. De har en konisk form och en silikonbeläggning som gör dem effektiva och skonsamma mot tandköttet. De kan användas även vid tandställning, implantat och kronor. De kommer i tre storlekar (XS/S, M/L och XL) och i två förpackningsstorlekar (36-pack med fickfodral och 60-pack utan fickfodral) [3].

2.2 PLC

En PLC (Programmable Logic Controller) är en digital enhet som främst används inom industrin och utgör en central del av automatiserade styrsystem och maskiner. PLC:er är speciellt utformade för att vara exakta, driftsäkra, snabba och robusta eftersom de används i krävande miljöer med vibrationer, skiftande temperaturer och damm.

En PLC övervakar ingående signaler från givare, omvandlare och andra enheter som är anslutna till den för att sedan behandla dessa signaler enligt ett programmerat logiskt schema och genererar sedan utgångssignaler som styr aktuatorer, motorer och andra enheter.

PLC:er styrs av mjukvara som består av specialiserade programmeringsspråk som följer IEC 61131-3-standarder. Dessa programmeringsspråk är särskilt framtagna för att enkelt kunna användas av ingenjörer och tekniker som jobbar med automatisering. Exempel på programmeringsspråk enligt denna standard är Ladder Logic, Sequential Function Chart, Instruction List, Structured Text och Function Block Diagram. Inom detta projekt har Function Block Diagram och Structured Text använts för styrning av systemet, genom att kombinera dessa har ett lättförståeligt och anpassningsbart program skapats som möter projektets behov [4].

2.2.1 Strukturerad Text

Structured Text (ST) är ett av de fem språk som beskrivs av IEC 61131-3-standarden. ST är framtaget för textbaserad styrning och kontroll av en PLC. Det är ett högnivåspråk som är strukturerat i block och syntaxen påminner främst om Pascal men även till viss del C och Java. ST är utformat för att vara både lättläst och enkelt att förstå samtidigt som det är lämpligt att använda för komplexa algoritmer och funktioner [5].

2.2.2 Funktionsblocksdiagram

Function Block Diagram (FBD) är ett annat av IEC 61131-3-standardens programmeringsspråk. Det har en betydligt mer visuell natur än ST och används för att visuellt tydliggöra de block, anslutningar och kopplingar emellan som representerar funktioner och deras relationer. FBD är visuellt och intuitivt vilket gör det mycket enkelt att förstå och analysera ett program [6].

2.3 Komponenter till Elskåp

Elskåpet är en central del av lådväxlarens styrsystem och används för att förse de elektriska komponenterna med nödvändig spänning och ström samt skydda dem från potentiella skador orsakade av störningar i nätspänningen. Elskåpet innehåller flera viktiga komponenter som samverkar för att säkerställa en säker och effektiv drift av lådväxlaren.

Ingående komponenter i elskåpet:

- Matningsenhet: +24V / -24V
- Hybridstartare till asynkronmotorerna
- Beckhoff BK5151 busskopplare
- I/O-enheter av modell Beckhoff KL1808 & KL2809
- Skyddsrelä
- Nödstopp
- Säkringar
- Kopplingar mellan samtliga komponenter

2.4 HMI

HMI (Human-Machine Interface) är ett begrepp som beskriver en typ av gränssnitt som agerar brygga mellan människa och maskin. Det kan vara en instrumentpanel, gaspedaler, knappar och pekskärmar. Kortfattat så möjliggör det kontroll av en maskin baserat på indata från en människa, t.ex. att en hiss åker upp eller ner beroende på om en knapp trycks ner eller inte. HMI kan användas för att beskriva ett mjukvarumässigt användargränssnitt inom IT-system men är oftast ett begrepp som används inom industrin och särskilt i kontexten av en skärm där information och styrning görs tillgänglig [7].

Det HMI som använts i detta projekt är det som redan var monterat på lådväxlaren som består av fyra knappar och fyra omkopplare utan någon skärm vilket utnyttjades för manuell kontroll av motordrifter.

2.5 Asynkronmotor

En asynkronmotor, även känd som en induktionsmotor, är en typ av elektrisk motor som använder elektromagnetisk induktion för att omvandla elektrisk energi till mekanisk rörelse. Dessa motorer är vanliga inom industrin på grund av deras enkla konstruktion, pålitlighet och låga underhållskrav [8].

Märkdata för asynkronmotorer som använts i projektet [9]:

- Effekt: 120 W
- Frekvens: 50 Hz
- Spänning: 230/400 V (enfas/trefas)
- Ström: 0.81/0.47 A (enfas/trefas)
- Varvtal: 1350 rpm (varv per minut)
- Effektfaktor: Cos φ = 0.62

Denna asynkronmotor är konstruerad för att fungera vid en frekvens av 50 Hz, vilket är den vanligaste frekvensen för elektriska nät i många länder, bland annat Sverige. Motorn är ansluten antingen till en enfas eller en trefasspänning på 230 V respektive 400 V, vilket är standardspänningarna för elektriska nät i Sverige.

Motorns effekt är 120 watt, vilket innebär att den kan omvandla 120 watt elektrisk energi till mekanisk energi vid märklast. Motorns varvtal, 1350 rpm, är nära dess synkrona hastighet, vilket är den hastighet som motorns magnetiska fält roterar.

Effektfaktorn, Cos φ, är en viktig parameter som beskriver hur effektivt motorn omvandlar elektrisk energi till mekanisk energi. I detta fall är effektfaktorn 0,62, vilket indikerar att motorn har en relativt låg effektivitet. Högre värden på effektfaktorn innebär bättre energieffektivitet, och moderna motorer kan ha effektfaktorer på över 0,9.

Sammanfattningsvis är den beskrivna asynkronmotorn en pålitlig och robust komponent som kan användas inom industrin för att driva maskiner och system. Dess enkla konstruktion och låga underhållskrav gör den lämplig för en mängd olika applikationer och miljöer.

2.6 Overheadsystem

För att kunna hantera och styra de automatiska robotar som rör sig inom produktionsanläggningen finns ett övergripande orderhanteringssystem implementerat inom nätverket, ett s.k. overheadsystem. Detta system sköter kommunikationen mellan maskinerna och robotorna för att säkerställa en effektiv och smidig drift av produktionsprocesserna. Utöver att samordna kommunikationen så tillhandahåller overheadsystemet realtidsdata om var robotorna befinner sig och var de är på väg i en grafisk vy.

Maskiner och robotar tar emot och skickar instruktioner till overheadsystemet vars roll kan beskrivas som spindeln-i-nätet för hela produktionsprocessen. Ett fiktivt exempel på overheadsystemets arbetsgång kan beskrivas enligt följande:

1. Producerande maskin rapporterar att en pall med produkt är klar och redo för upphämtning till overheadsystemet
2. Overheadsystemet tar emot denna information och beställer en robot för upphämtning
3. Roboten anländer till maskinen och kopplar upp sig direkt till den och överlämningen av produkt från maskin till robot genomförs
4. Robot meddelar overheadsystemet att den har mottagit produkten
5. Overheadsystemet beordrar roboten att åka med produkten till det nya mellanlagret
6. Roboten anländer till mellanlagret och kopplar upp sig direkt till transportbandet för att gå igenom sekvens för överlämning av produkt
7. Mellanlagret meddelar overheadsystemet att produkten är mottagen och att transportband 1 är nu fullt.
8. Roboten som nu saknar aktiv arbetsorder åker och ställer sig på en laddningsplats

Overheadsystemet möjliggör central kontroll av de automatiserade processerna och kan genom lagring av data även användas för optimering av dessa. Eventuella problem och ineffektiviteter kan snabbt identifieras och åtgärdas.

2.7 Transportband

WCS-OB står för “Wide Chain conveyor System-Outfeed Buffer” och är en typ av rullband utan motordrift som kan användas som en buffertzon för att lagra och mata ut produkter från produktionen. Buffertzonen har en horisontell lutning som gör att pallar rör sig från ena sidan till den andra utan motordrift. I den ursprungliga lådväxlaren har det nedersta bandet varit av typen WCS-OB [10].

WCS-R står för “Wide Chain conveyor System-Roller” och är en typ av motordrivet transportband för att flytta produkter fram och tillbaka enligt styrning. Transportbandet består av ett PVC-band som klarar temperaturer mellan -10ºC och +70ºC. Transportbandet drivs av en elektronisk motor från Bonfiglioli som har en max lastkapacitet på 50 kg/m [11].

Båda banden är delar av Wemos modulbaserade materialhanteringssystem för hantering av lådor, backar och pallar i robotceller.

2.8 CODESYS V3

CODESYS V3 är en öppen och fri programmeringsmiljö som används för att utveckla och implementera PLC-kod i industriella automationssystem. CODESYS är en förkortning av Controller Development System.

CODESYS är specifikt utvecklat för att följa IEC 61131-3-standarden, vilket innebär att den stöder alla de fem programmeringsspråken som definieras av standarden: Ladder Diagram (LD), Function Block Diagram (FBD), Structured Text (ST), Instruction List (IL) och Sequential Function Chart (SFC). Detta ger utvecklare möjlighet att välja det programmeringsspråk som bäst passar deras behov och kompetens.

På grund av CODESYS öppenhet och standardisering så är kompatibilitet med de flesta olika PLC-märken och hårdvaruplattformar stödda antingen direkt i programvaran eller via bibliotek. Genom att möjliggöra import av externa bibliotek direkt från hårdvaruutvecklare och eftersom att programmet allmänt används inom industrin så finns det stöd för nästan samtlig utrustning på automationsmarknaden direkt i CODESYS. Dessutom så innehåller CODESYS V3 en rad funktioner och verktyg som kan hjälpa utvecklare att skapa, testa och felsöka PLC-kod, såsom simulering och visualisering av automatiserade arbetsflöden [12]. I projektet har CODESYS V 3.5 SP16 Patch 4 använts.

3. Metod

I en inledande fas av projektet påbörjades observationer av lådväxlaren för att få en bättre förståelse för vilka funktioner den hade samt vilka möjligheter som fanns för ombyggnad och modifiering. Genom att noggrant studera hur lådväxlaren fungerar och vilka givare, signaler och motorer som var inblandade så kunde examensarbetarna få en övergripande bild av dess tekniska egenskaper. Det noterades hur lådväxlaren reagerade på olika inmatningar och laster och hur signalerna från olika givare sammankopplas för att påverka dess rörelsemönster. Genom att få en bättre förståelse för dessa faktorer kunde det sedan avgöras vilka delar som var viktigast för att uppnå önskad prestanda. Det undersöktes också hur motorerna var monterade och vilka begränsningar som fanns i deras kapacitet. Vidare förs det samtal med personal som var involverade i drift och underhåll av lådväxlaren. Dessa samtal gav bättre förståelse för vad som var viktigt att tänka på när det planerades hur systemets olika delar skulle modifieras. Samtalen gav också bättre förståelse för vilka problem som ofta uppstod i samband med användning och drift av lådväxlarna.

Efter den inledande fasen så gjordes en noggrann och strukturerad planering som skulle ligga till grund för den kommande arbetsprocessen. Planeringen skulle vara en enkel och tydlig projektplan som skulle följas genom hela arbetet för att ha en bra överblick och kunna visualisera projektets framsteg. Regelbundna möten mellan både examensarbetarna och handledare på TePe säkerställde att examensarbetarna hela tiden låg på rätt spår tidsmässigt och för att snabbt kunna anpassa projektet efter nya idéer eller efter problem som uppstått.

3.1 Beskrivning av det befintliga transportbandsystemet

Lådväxlarsystemet består av fem långa transportband staplade vertikalt ovanpå varandra med en hiss i ena änden som åker mellan dem. Den nedersta våningen består av ett lutande rullband utan motordrift, Wemo WCS-OB, som fungerar som en utbuffert för material från systemet. Ovanför den nedersta våningen är fyra horisontella motordrivna transportband, Wemo WCS-R, staplade på varandra.

Hissen och de fyra horisontella transportbanden drivs av varsin Bonfigiloi Riduttori asynkronmotor som styr dess hastighet, rörelse och riktning. Banden är fyra meter långa och hissen som är en halv meter lång förflyttar sig mellan banden och är utrustad med ett

horisontellt transportband av typen WCS-R. Systemets totala längd är fyra och en halv meter långt medan höjden är två och en halv meter.

Varje band har givare på varsin sida och ett tryckluftsstyrt stoppsystem i änden mot hissen. Hissen är utrustad med en givare i mitten av dess transportband. Systemet är även utrustat med ett HMI bestående av fyra knappar och fyra omkopplare som har använts av operatörer för manuell styrning av transportbanden.

3.2 Översikt över ändringar som gjorts på systemet

I utvecklingen av den första prototypen av mellanlagret har den ursprungliga konstruktionen av lådväxlaren till stor del behållits men fler givare har installerats och metallskenor på främst hissens transportband har justerats. Genom att utöka antalet givare i systemet har en detaljerad styrning och en effektivare övervakning av in- och utlastningsprocessen gjorts möjlig. Det finns långtgående planer på att montera bort det nedersta icke motordrivna rullbandet som tidigare använts som utbuffert då det i det nya systemet står helt oanvänt men då det kräver en mer omfattande ombyggnation så gjordes inte det inom ramen för detta projektet.

Dockning för den automatiska roboten har installerats och hissens position har justerats så att den ligger i linje med robotens höjd. Givarna på hissen har dubblerats och flyttats om för att få till konsekvent placering av pallarna på transportbanden.

3.3 Installation av elskåp

För att få lådväxlaren att fungera efter önskad funktionalitet så behövdes ett elskåp med rätt ingående komponenter och rätt kopplingar mellan dessa komponenter. Till lådväxlarens ursprungliga utseende och funktionalitet så fanns ett tillhörande färdigkopplat elskåp. Detta elskåp ville examensarbetarna nu modifiera och byta ut en del komponenter samt att koppla ihop dem efter det nya kopplingsschemat som fanns för den nya tänkta funktionaliteten.

3.3.1 Modifiering av befintligt elskåp

För att anpassa det befintliga elskåpet till projektets krav och den nya funktionaliteten för lådväxlaren, genomfördes flera modifieringar. Dessa inkluderade installation av hybridstartare, matning, busskopplare, I/O-enheter, skyddsreläer och nödstopp. Frekvensomriktaren som varit installerad kopplades bort då det inte fanns ett behov av att styra motorernas varvtal.

3.3.2 Installation av hybridstartare och spänningsmatning

Elskåpet har en matningsenhet som transformerar ner den trefasiga växelströmmen till en stabil spänningsförsörjning på +24 V och -24 V till systemets olika komponenter. Sex hybridstartare installerades sedan i elskåpet, en för varje asynkronmotor. Dessa förbättrar motorernas start- och stopprestanda, förlänger livslängden och skyddar motorerna från skador som kan uppstå från plötsliga startmoment. Annan utrustning i elskåpet som det dragits matningsspänning till är bland annat en nätverksswitch, skyddsrelä, busskopplare och I/O-enheter.

3.3.3 Installation av I/O-enheter

För att säkerställa korrekt kommunikation och styrning av lådväxlaren installerades en busskopplare, Beckhoff BK5151, i elskåpet. Till denna anslöts I/O-enheter KL1808 och KL2809. Dessa enheter fungerar som en länk mellan PLC:n och de olika komponenterna i lådväxlaren, såsom motorer, givare och nödstopp. I/O-enheterna gör det möjligt att både samla in data från komponenterna och skicka styrkommandon till dem.

3.3.4 Installation av skyddsreläer och nödstopp

Skyddsreläer och nödstopp installerades i elskåpet i syfte att skydda både utrustning och personal. Om farliga situationer såsom överbelastning, kortslutning eller risk för skada uppstår kan strömmen brytas omedelbart både automatiskt och via de två nödstopp som installerats och finns placerade i varsin ände av systemet. Framförallt hissen och dess rörelse är en säkerhetsrisk då man enkelt skulle kunna fastna och klämmas vid automatisk drift då den i sitt grundutförande inte är inkapslad bakom någon typ av skydd.

3.3.5 Slutlig konfiguration och koppling av komponenter

Efter att de nya komponenterna installerats i elskåpet och det dragits spänningsmatning till samtliga behövde allting kopplas samman. Det innebar inkoppling av samtliga motorers in- och utsignaler, samtliga givare, skyddsreläer, säkringar och nödstopp in i I/O-enheterna. Dessa i sin tur behövde anslutas till PLC:n för övervakning och styrning. Inkopplingen skedde delvis efter ett kopplingsschema som tagits fram internt för ändamålet.

Med alla komponenter inkopplade följde ett flertal tester för att se att alla insignaler registrerades korrekt i systemet och att styrning från PLC genom utsignaler gav önskat resultat.

Korrekt inkopplat elskåp spelar en central del i mellanlagrets funktion då det är fundamentet som gör övriga delar av projektet möjligt. Det är därför av största vikt att det är korrekt inkopplat och att alla signaler hanteras så som förväntat. Det var endast efter att detta var gjort korrekt som projektet kunde gå vidare för att börja styra systemets tilltänkta funktioner.

3.4 Anslutning av HMI och motorer i elskåp för manuell kontroll

När komponenterna i elskåpet var korrekt inkopplade och signaler in och ut ur systemet fungerade enligt krav som det kunde börja förberedas för manuell kontroll av transportbanden och hissens motordrifter.

I lådväxlarens befintliga konstruktion fanns det ett enklare HMI-system bestående av fyra omkopplare och fyra knappar som var monterade på lådväxlaren. Genom att ansluta det in i det nya elskåpet gavs möjlighet att implementera enklare styrsystem för att vidare testa givarnas funktion i mer realistiska förhållanden. Testerna bekräftade motorernas funktion liksom att givarna gav utslag när det förväntades. Vid det här laget hade det säkerställts att den grundläggande funktionalitet som var nödvändig för att kunna utveckla mjukvaran uppfyllde de ställda kraven.

3.4.1 Undersökning av slitage på transportband

Syftet med undersökningen var att utvärdera och analysera slitage på transportbanden i lådväxlaren för att identifiera eventuella problem, optimera underhållsplaner och förlänga transportbandens livslängd.

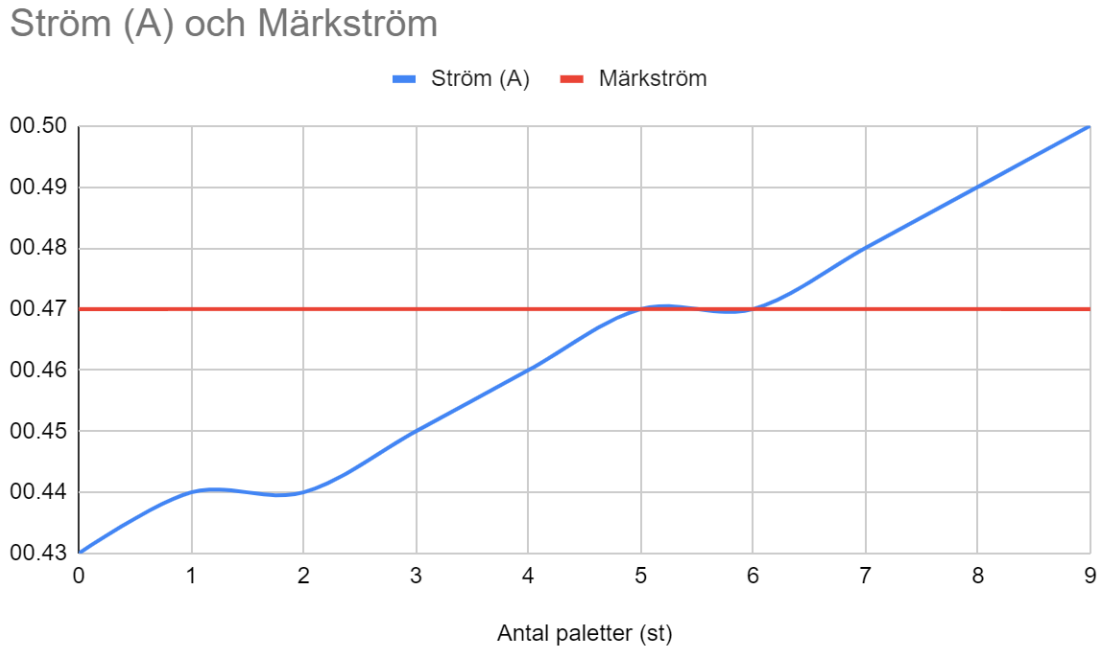
Först utfördes en visuell inspektion av transportbanden för att identifiera eventuella synliga tecken på slitage, såsom skador, sprickor eller deformationer. Efter detta undersöktes transportbandens tjocklek och bredd för att bedöma hur mycket material som slitits bort under användning. Detta var en viktig punkt att undersöka då det under de få gångerna som transportbanden använts fanns tendenser till att gummi-flisor lossnade från transportbanden. Detta konstaterades bero på att bandet inte var centralt monterat runt drivaxeln. Vidare så jämfördes dessa mätningar med tillverkarens rekommenderade minimivärden för att avgöra om transportbanden behövde bytas ut, vilket inte behövdes.

Mätning av valsmotstånd utfördes sedan på lådväxlaren. Valsmotståndet för varje transportband testades genom att mäta kraften som krävdes för att dra transportbandet över en given sträcka. Högt valsmotstånd kan tyda på slitna eller skadade valsar, vilket kan påverka transportbandets prestanda och livslängd.

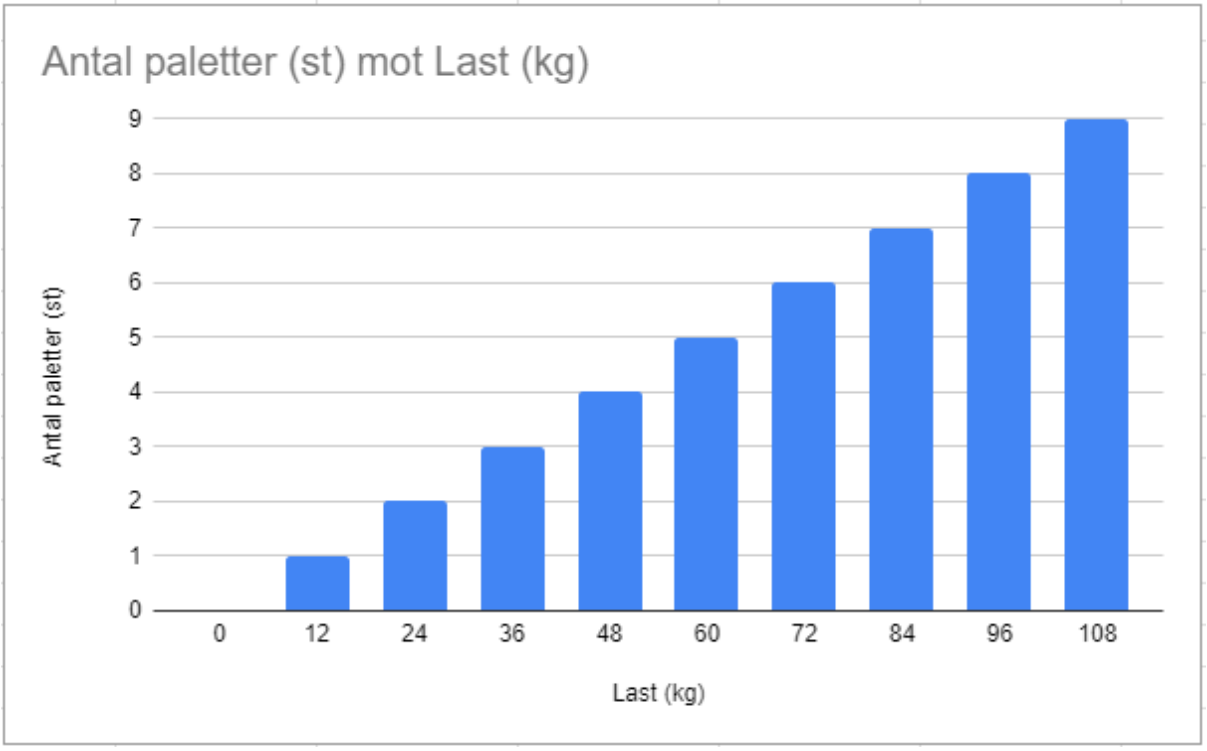
Genom att regelbundet genomföra en undersökning av transportbandens slitage kan man upptäcka och åtgärda problem i ett tidigt skede, vilket bidrar till att förbättra lådväxlarens driftsäkerhet och livslängd. Informationen från undersökningen kan även användas för att justera och optimera underhållsplaner, vilket kan leda till minskade driftskostnader och ökad produktivitet.

3.4.2 Undersökning av asynkronmotorernas ström, prestanda och effektivitet

I detta avsnitt så beskrivs den undersökning som gjordes på motorerna som användes under projektet.



Figur 2: Diagram ström i ampere mot antal pallar i st



Figur 3: Diagram antal pallar i st mot last i kg

Syftet med denna undersökning var att utvärdera asynkronmotorernas strömförbrukning vid olika laster för att säkerställa att strömmen inte överskred motorernas märkström och att

motorerna kunde hantera belastningen av ett transportband fyllt med fyllda pallar. Undersökningen syftade också till att identifiera eventuella problem med motorernas prestanda och effektivitet.

För att genomföra undersökningen monterades en strömtång runt varje asynkronmotors matarkabel. Strömtången användes för att mäta motorernas strömförbrukning vid olika belastningar. Flera olika lastprofiler valdes för att testa motorernas prestanda under olika belastningar. Profilerna inkluderade tomma transportband, transportband fyllda med fyllda pallar och transportband med varierande belastning.

Under testerna registrerades strömvärdena för varje motor vid de olika lastprofilerna. Dessa värden jämfördes sedan med motorernas märkström, som anges av tillverkaren, för att bedöma motorernas förmåga att hantera belastningen. Resultaten från datainsamlingen analyserades för att identifiera eventuella avvikelser eller problem i motorernas prestanda. Om strömförbrukningen för någon motor överskred märkströmmen med en betydande marginal, kunde det tyda på ett problem med motorns belastningskapacitet eller effektivitet.

Som diagrammet i figur 1 visar så varierade strömmen från 0.43 A vid tomt transportband till 0.50 A vid 9 st pallar på transportbandet. Den nominella strömmen för asynkronmotorn är 0.47 A enligt märkdata. Detta betyder att den rekommenderade strömmen överskreds när antalet pallar blev 7 st eller fler vilket är ekvivalent till när lastens tyngd i kg blev större än 84kg som visas i figur 2.

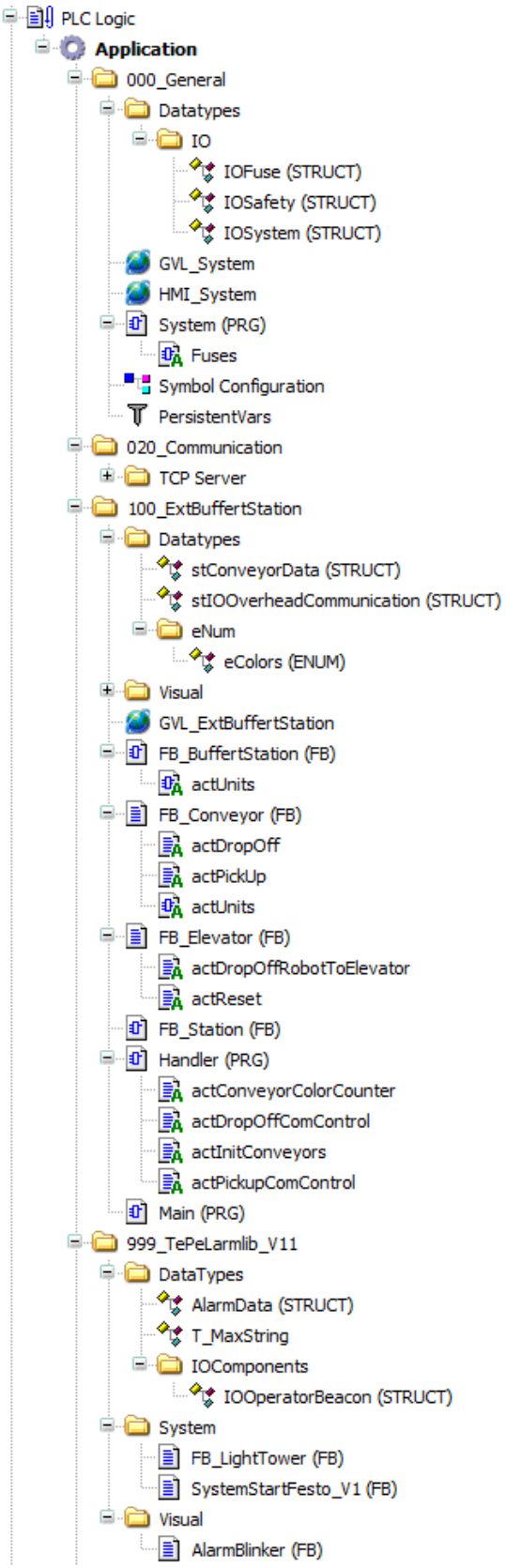
En elektrisk motor kan belastas med mer än märkström. Detta ger en ökad termisk belastning på isoleringen av ledarna som förkortar motorns livslängd och möjligheten bör utnyttjas så sparsamt som möjligt.

Att strömmen för tyngre last överskrider märkströmmen anses inte som något problem av ett par olika anledningar. För det första så är ökningen inte stor och motorn hade klarat av att hålla en ström på 0.49 A ett långt tag innan något problem hade uppstått. För det andra så är det inte speciellt ofta som det faktiskt kommer att ligga fler än sju pallar på transportbandet vilket innebär att strömmen sällan kommer gå över märkström på 0.47 A. För det sista så är tiden som det tar att lasta på eller lasta av en pall från eller till transportbanden endast fåtal

sekunder. Vilket tillsammans med föregående anledningar innebär att tiden som motorn kör på för hög ström kan försummas. [2]

3.5 Programmering av PLC

I detta avsnitt så beskrivs alla ingående POUs som programmet består av.



Figur 4: Ingående POUs

Mjukvaran som har tagits fram för att styra systemet har utgått ifrån en objektorienterad designfilosofi med moduläritet i åtanke. Det innebär att koden är uppdelad i mindre funktionsblock eller moduler som hanterar de ingående delarna i systemet. Dessa moduler är sedan sammankopplade för att bygga upp en större helhet. Fördelen med en modulär design är bland annat ökad återanvändbarhet, enklare underhåll och skalbarhet samt lättare förståelse och dokumentation av koden.

Funktionsblocken som styr lådväxlaren har delats upp i ett för transportband och ett för hissen. Dessa kan sedan instansieras i önskat antal vilket gör skalbarheten för systemet väldigt enkel och leder till att det inte är någon mjukvarumässig skillnad oavsett hur många transportband eller hissar som är aktiva inom systemet.

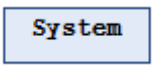
Kraven som ställs på koden är därför att den inte kan vara unik för ett särskilt transportband utan måste vara lika för alla vilket ställer en del krav på hur koden är skriven. Figur 4 visar hela PLC-programmets struktur och ingående POUs (Program Organization Unit).

3.5.1 Programmet Main

Main är huvudprogrammet som körs som Main Task på PLC:n. Det innebär att det är Main som körs i PLC:ns programcykel. Main-programmet innehåller deklarationer av konstanter, instanser, arrays med element och felmeddelanden som sedan används i samtliga övriga program. Main är det översta lagret i programmet och samtlig övrig kod körs i block som ligger nästlade i main. De programblock som körs i Main är “System”, “Handler” och en instans av “FB_BuffertStation”.

3.5.2 Programmet System

I detta avsnitt så beskrivs programmet System.



Figur 5: Programmet System

Programmet "System" fungerar som ett bevakande block som hanterar och koordinerar olika signaler och händelser på hårdvarunivå. Som figur 5 visar ovan så saknar programmet både in- och utvariabler. Säkringar, nödstopp eller manuellt operatörsstopp av lådväxlaren är exempel på händelser som “System”-programmet hanterar. “System” ansvarar även för att visuellt vidarebefordra dessa händelser till den ljusfyr som är installerad på elskåpet. Ljusfyren lyser rött vid systemkritiska fel kallat Error, blått för att uppmärksamma operatörer att något inte stämmer men där lådväxlarens grundfunktion är opåverkad, kallat Message och grönt när inga problem är aktiva. Exempel på ett Error är att nödstopp är aktivt, exempel på ett Message kan vara om transportbandens räknare av någon anledning skulle få ett värde som visar mindre än 0.

Variablerna inuti programmet "System" har följande funktioner:

1. LightTower: En instans av funktionsblocket FB_LightTower, som hanterar ljusfyrens funktion och färgkodning. Ljusfyren används för att visuellt kommunicera systemets tillstånd, såsom normal drift, felmeddelanden eller alarm till operatörerna.

2. SystemStart: En instans av funktionsblocket SystemStartFesto_V1, som ansvarar för att initiera och starta systemet. Detta block tar emot startsignaler, övervakar systemets status och säkerställer att all nödvändig utrustning är redo för drift.

3. ResetIndicationBlink: En instans av funktionsblocket AlarmBlinker, som används för att styra blinkningen av ljusfyren när systemet behöver återställas eller när det finns en indikation som kräver operatörens uppmärksamhet.

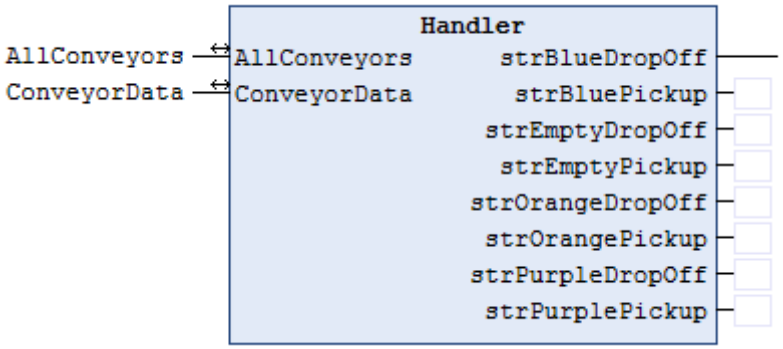
4. FuseAlarmActive: En bool-variabel som indikerar om det finns ett säkringslarm aktiverat i systemet. Om denna variabel är sann (TRUE), innebär det att det finns ett problem med en säkring som behöver åtgärdas.

5. bMessageActive: En BOOL-variabel som indikerar om det finns ett aktivt felmeddelande eller en varning i systemet. Om denna variabel är sann (TRUE), innebär det att det finns en händelse som kräver operatörens uppmärksamhet.

Programmet "System" fungerar genom att samordna de olika funktionsblocken och variablerna för att övervaka systemets status och agera på eventuella problem eller händelser. Om ett fel uppstår, kan programmet ändra ljusfyrens färg och blinkmönster för att informera operatören om situationen. Programmet säkerställer också att systemet startas korrekt och att alla komponenter fungerar som de ska. Detta bidrar till en effektiv och säker drift av lådväxlaren och underlättar för operatörerna att fatta korrekta beslut vid olika situationer.

3.5.3 Programmet Handler

I detta avsnitt så beskrivs programmet Handler.



Figur 6: Programmet Handler

Funktionsblocket Handler är utformat för att hantera och övervaka olika transportband som transporterar pallar med olika färger mellan stationer. Det gör det genom att använda en kombination av intern logik, in- och utvariabler samt övervakning av givare och motorstyrning. Handler är exklusivt mjukvarubaserat och har ingen direkt koppling till varken komponenter eller hårdvara.

Invariabler:
AllConveyors: Detta är en tvådimensionell array som innehåller FB_Conveyor-instanser för varje station och transportband.

ConveyorData: Detta är en tvådimensionell array som innehåller stConveyorData-strukturer för varje station och transportband. stConveyorData innehåller realtidsvärden av variabler från alla instanser av FB_Conveyor såsom antal pallar på transportbanden, om transportbandet håller på att tömmas med mera.

Genom att skicka in arrays där alla instanser av alla transportband är representerade som element in i blocket Handler kan operationer och loopar utföras inom programblocket.

Utvariabler:
strBlueDropOff, strBluePickup, strEmptyDropOff, strEmptyPickup, strOrangeDropOff, strOrangePickup, strPurpleDropOff, strPurplePickup:
Dessa string-variabler används för att enkelt åskådliggöra vilket band som just nu är aktivt för in- och urlastning. Exempel på en sådan utsignal är om strBlueDropOff är "13", innebär det att vid aktuell tidpunkt är transportband 3 från station 1 det aktiva transportbandet utvalt för inlastning av blå pallar.

Huvudlogiken för funktionsblocket är uppbyggd kring en "CASE"-sats eller tillståndsstruktur för att bestämma vilket skede av hanteringen av pallarna på transportbanden som den för närvarande befinner sig i.

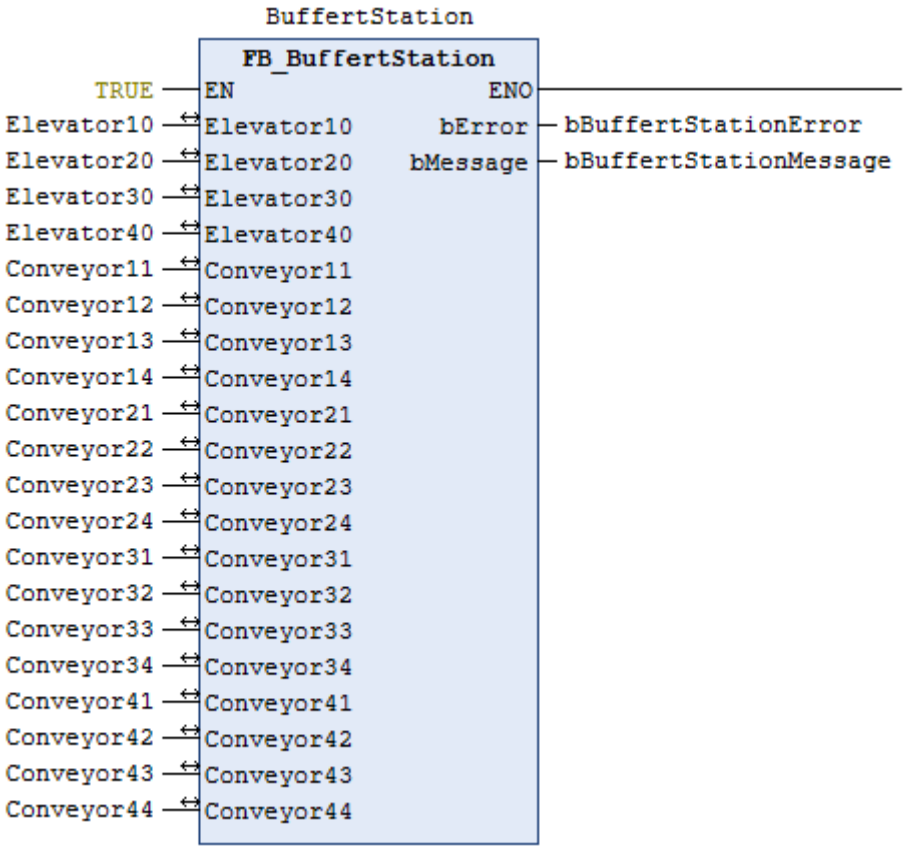
Baserat på det aktuella steget i hanteringen av färgpallar på transportbanden, kommer olika uppsättningar av instruktioner att utföras. Detta kan innebära att skanna alla transportband för att hitta ett aktivt band med plats för en pall, övervaka det aktiva bandet för att kontrollera om det är fullt eller om det börjar bli tomt, samt vid behov identifiera och aktivera ett nytt transportband för pallar.

Sammantaget är Handler utformad för att styra och övervaka olika transportband som transporterar pallar med olika färger mellan stationer och det gör det genom att använda en kombination av intern logik, in- och utvariabler, samt övervakning av givare och motorstyrning.

Under avsnitt 3.10 har programmet Handler förklarats vidare och hur det har använts för att dynamiskt välja vilka band som är aktiva i vilket läge.

3.5.4 Funktionsblocket FB_BuffertStation

I detta avsnitt beskrivs funktionsblocket FB_Buffertstation.



Figur 7: Instans av funktionsblocket FB_BuffertStation

FB_BuffertStation är ett funktionsblock som fungerar som en samlande enhet för en buffertstation, där flera stationer och deras komponenter (hissar och transportband) är grupperade tillsammans. Det är utformat för att ge en tydligare struktur och översikt över systemet genom att koppla samman flera stationer och deras respektive komponenter samt för att underlätta förståelsen och underhållet av programmet.

In/utvariabler:

Elevator10, Elevator20, Elevator30, Elevator40:

Instanser av FB_Elevator som representerar hissarna för varje station inom buffertstationen.

Conveyor11, Conveyor12, Conveyor13, Conveyor14:

Instanser av FB_Conveyor som representerar transportbanden för station 1.

Conveyor21, Conveyor22, Conveyor23, Conveyor24:

Instanser av FB_Conveyor som representerar transportbanden för station 2.

Conveyor31, Conveyor32, Conveyor33, Conveyor34:
Instanser av FB_Conveyor som representerar transportbanden för station 3.

Conveyor41, Conveyor42, Conveyor43, Conveyor44:
Instanser av FB_Conveyor som representerar transportbanden för station 4.

Utvariabler:
bError: En BOOL-variabel som indikerar om något fel har inträffat i någon av buffertstationens komponenter.

bMessage: En BOOL-variabel som används för att skicka meddelanden till andra delar av systemet vid behov.
Internvariabler:

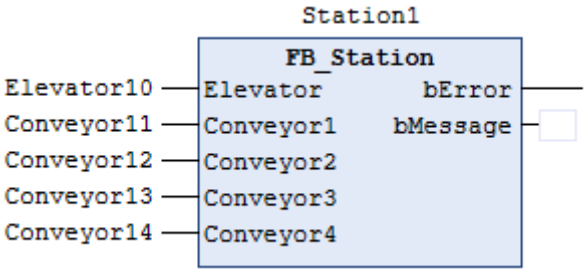
Station1, Station2, Station3, Station4: Instanser av FB_Station som representerar de fyra stationerna inom buffertstationen.

Kod:
FB_BuffertStation innehåller action "actUnits" där de faktiska instanserna av Conveyor och Elevator skapas. Dessa instanser kopplas sedan till de respektive Station1, Station2, Station3 och Station4 inom buffertstationen. På så sätt kan en övergripande status för buffertstationen erhållas genom att övervaka bError och bMessage-variablerna.

Funktionsblocket FB_BuffertStation syftar till att ge en tydligare struktur och översikt över systemet genom att sammanföra flera stationer och deras komponenter på en buffertstationsnivå. Det underlättar förståelsen och underhållet av programmet och hjälper till att säkerställa en effektiv och korrekt kommunikation mellan olika delar av systemet.

3.5.5 Funktionsblocket FB_Station

I detta avsnitt så beskrivs funktionsblocket FB_Station.



Figur 8: Instans av funktionsblocket FB_Station

FB_Station har i huvudsak en visuell funktion för att organisera och tydliggöra vilka band och vilken hiss som gemensamt utgör en station i verkligheten. Detta block implementerades för att tydligt illustrera hur ett fel från ett transportband sprids uppåt i programstrukturen. Det gör det möjligt att följa felorsaken steg för steg, från huvudprogrammet ända ned till dess ursprung. FB_Station ger också en enkel överblick över vilka komponenter som utgör en station. Inom blocket utförs inga operationer annat än felindikering.

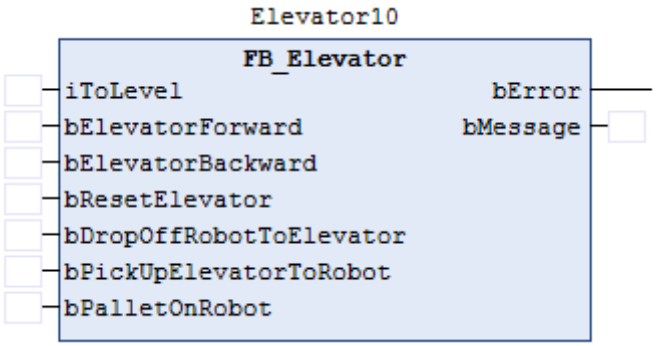
Invariabler:
Elevator: En instans av FB_Elevator som är kopplad till stationen.
Conveyor1, Conveyor2, Conveyor3, Conveyor4:
Instanser av FB_Conveyor som representerar de fyra transportbanden som finns på varje station.

Utvariabler:
bError: En bool-variabel som indikerar om något fel har inträffat i någon av stationens komponenter.
bMessage: En bool-variabel som används för att skicka meddelanden till andra delar av systemet vid behov.

Kod:
Den enda kod som körs Inom FB_Station representeras av två logiska OR-grindar för att sammanföra information om fel (bError) och meddelanden (bMessage) från de olika transportbandens instanser. Om fel uppstår på något av de fyra banden eller hissen leder det till att stationen indikerar att ett fel finns.

3.5.6 Funktionsblocket FB_Elevator

I detta avsnitt så beskrivs funktionsblocket FB_Elevator.



Figur 9: Instans av funktionsblocket FB_Elevator

FB_Elevator hanterar hissens styrning och motordrift av hissens eget transportband. FB_Elevator instansieras och kopplas sedan samman med det antal instanser av transportband som tillsammans utgör en station. Det är ett block som är beroende av insignaler skickade från andra delar av systemet för att utföra sin programlogik och är i praktiken underställt och en förlängning av FB_Conveyor och dess funktion som beskrivs i asvnitt 3.5.6.

Insignaler:
- **iToLevel**: Detta är en variabel som meddelar den interna logiken vilken våning eller transportband hissen ska förflytta sig till.

- **bElevatorForward, bElevatorBackward, bResetElevator, bDropOffRobotToElevator, bPickUpElevatorToRobot, bPalletOnRobot**: Dessa variabler fungerar som kontrollsignaler för hissen och dess anslutning till den mobila roboten.

-bResetElevator:
Om signalen blir hög på denna har en särskild programsekvens tagits fram för att tvinga hissen att återgå till sin startposition. Exempel på när detta är relevant kan vara efter ett nödstopp eller annan händelse som har låst hissen i en position.

Utsignaler:
- **bError**: En bool-variabel som indikerar om något fel har inträffat i hissens funktion.

- **bMessage**: En bool-variabel som används för att skicka meddelanden som kräver uppmärksamhet men inte omedelbar åtgärd.

Funktionsblockets huvudlogik är uppbyggd kring en “CASE”-sats som beroende på insignalen iToLevel stegar genom processen att röra hissen mot rätt våning och ta emot eller lämna av en pall.

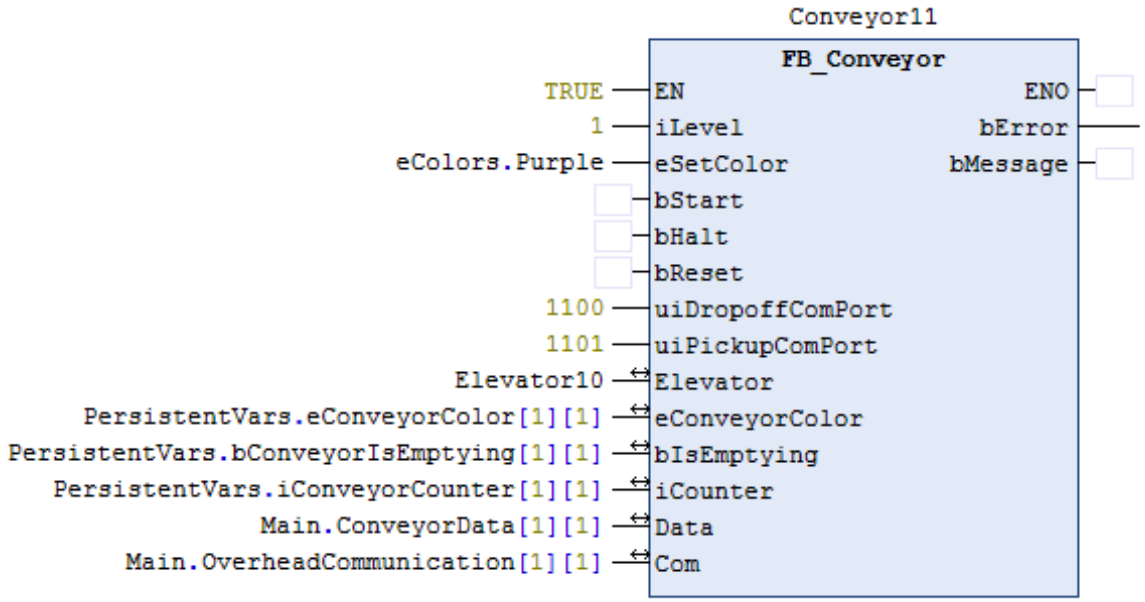
För varje steg i “CASE”-satsen utförs en uppsättning instruktioner som är avsedda för att styra hissen för att nå målet. Det innebär att aktivera bElevatorUp eller bElevatorDown för att flytta hissen uppåt eller nedåt baserat på var hissen befinner sig och var hissen ska, samt kontinuerlig övervakning av de signaler som givarna placerade på hissen skickar ut.

När hissen har nått målet kommer hissen att efter klartecken från övriga sekvenser, återgå till sin ursprungsvåning och hissen kommer att gå in i ett vänteläge i väntan på ytterligare instruktioner.

Att hissens funktionalitet ska finnas i ett eget funktionsblock är inte något som inom ramen för kravspecifikation egentligen hade behövts. Hissen kunde ha realiserats antingen som en förlängning av FB_Conveyor eller som en separat datastruktur kopplad till ett annat funktionsblock. Det valdes dock att göra den till ett separat, instantiserat funktionsblock. Detta beslut gjordes för att tydligare koppla ihop den med andra delkomponenter som utgör en station, vilket leder till ökad förståelse och förenklad felhantering.

3.5.7 Funktionsblocket FB_Conveyor

I detta avsnitt så beskrivs funktionsblocket FB_Conveyor.



Figur 10: Instans av funktionsblocket FB_Conveyor

Funktionsblocket FB_Conveyor är skapat för att hantera och styra transportsbandens rörelse och funktion. Funktionsblocket är också utformat för att kunna kommunicera med FB_Elevator för att manövreringen mellan transportbandet på hissen och övriga transportband ska ske smidigt och på önskvärt sätt. FB_Conveyor ska kunna hantera två olika typer av händelser vilka kallas för drop-off och pick-up. För att enklare få förståelse för de olika händelserna så har två actions skapats. Dessa två actions skapar och beskriver den önskade sekvensen för respektive händelse.

FB_conveyor har också en tredje action som heter actUnits. Denna action instansierar kommunikationsblocken PickupComServer och DropoffComServer. Dessa block används för att systemet ska veta vilket transportband som de inkommande eller hämtade pallarna ska lämnas på eller hämtas från. På detta vis blir koden mer kortfattad och kommunikationskonflikter mellan lådväxlare, overheadsystem och robot undviks men det innebär att varje enskilt transportband måste ha sina egna in och ut-portar. Detta innebär också att utifrån och från robotens ståndpunkt så ser varje transportband ut som en egen station, vilket är något som gör hela buffertsystemet modulärt och enkelt att justera storleksmässigt.

Funktionsblocket FB_Conveyor, alla in och ut-signaler samt blockets två Actions, actDropOff och actPickUp beskrivs mer i detalj nedan:

FB_Conveyor fungerar tillsammans med hissen (FB_Elevator) för att hantera avlämning och upphämtning av pallar på ett transportband. Det tar emot information från både hissen och robotarna som ansluter till transportbanden. FB_Conveyor har flera ingångar, utgångar och variabler som används för att övervaka och styra de olika sekvenserna och transportbandens funktion.

Funktionsblocket har flera ingångar och utgångar som definieras i VAR_INPUT, VAR_OUTPUT och VAR_IN_OUT sektionerna. Ingångarna inkluderar information om transportbandets nivå, dess färg och olika signaler för start, stopp och återställning. Utgångarna ger information om fel i systemet och meddelanden som indikerar att något måste åtgärdas men som inte är akut. VAR_IN_OUT sektionen innehåller variabler som delas med andra funktionsblock såsom FB_Elevator och kommunikationsstrukturen stIOOverHeadCommunication.

FB_Conveyor använder två huvudsakliga "CASE"-strukturer för att kontrollera avlämnings- och upphämtningssituationer. Varje "CASE" är uppdelad i flera steg som hanterar olika aspekter av processen.

FB_Conveyor använder actDropOff och actPickUp för att kontrollera situationer som avlämning och upphämtning. Båda actions är skrivna i strukturerad text som bygger en sekvens av händelser för att få önskad rörelse av maskinen och önskad förflyttning av pallarna i systemet. Varje action bygger på en “CASE”-struktur som innebär att maskinen väntar vid ett tillstånd tills vissa villkor uppfylls för att då utföra specifika uppgifter och sedan gå vidare till nästa steg.

Action actDropOff hanterar avlämning av pallar från robot till hissen och sedan till rätt transportband. Det sker i följande steg:

1. Inledningsvis står hissen i vänteläge och väntar tills en robot anländer och bekräftar att den är ansluten och redo att lämna av en pall.
2. Det aktuella transportbandet kontrollerar om hissen är ledig och om så, går den vidare till nästa steg.
3. Pallen överförs från roboten till hissen och sekvensen för avlämning är då startad.
4. Hissen åker till den angivna nivån iLevel och när den är i rätt position så startas bandet på hissen för att förflytta pallen mot transportbandet.
5. När pallen är förflyttad till rätt position på rätt transportband, fortsätter hissen tillbaka ner till vänteläge som i systemet kallas för nivå 5 och INT-variabeln iCounter på transportbandet ökar med 1.

Action actPickUp hanterar upphämtning av pallar från transportören till robotarna. Det sker i följande steg:

1. Inledningsvis står hissen i vänteläge och väntar tills en robot anländer och bekräftar att den är ansluten och redo att hämta en pall.
2. Det aktuella transportbandet kontrollerar om hissen är ledig och om så, går den vidare till nästa steg.
3. Hissen går till den angivna nivån iLevel och transportbandet aktiveras och förflyttar sig framåt tills pallen som ska hämtas är i rätt position på hissen.
4. När pallen är i position, hanteras den upp av hissen och transportbandet stannar.
5. Hissen åker tillbaka till nivå 5 och överför pallen till roboten.
6. Antalet pallar på det aktuella transportbandet och värdet på variabeln iCounter minskar med 1.

Under processen kommunicerar FB_Conveyor med FB_Elevator och robotarna genom att skicka och ta emot data från dessa enheter. Den uppdaterar också en datastruktur (stConveyorData) med relevant information om transportbandets status och kommunikation.

Funktionsblocket FB_Conveyor innehåller även en del felhantering och tidsgränser för att säkerställa att processen fungerar korrekt och för att förhindra att transportbandet blir överbelastad med pallar.

Sammanfattningsvis fungerar FB_Conveyor med actUnits, actDropOff och actPickUp som en huvudkontrollenhet för att hantera och styra transportbanden i systemet. Den ansvarar för att instansiera och initialisera det individuella kommunikationsblocket för varje transportband och underlättar koordination, kommunikation, felhantering och diagnostik för hela systemet.

3.6 Persistent variables

Persistent variables är en särskilt typ av variabler som kan användas när man programmerar styrsystem för att lagra data även vid händelse att systemet stängs av, startas om eller att strömmen bryts. De tar upp minnesplats permanent och ska generellt användas endast för det som faktiskt har behov av denna funktionalitet [13].

I detta projekt har det använts tre olika persistent variables som det bedömdes att systemet behöver ha tillgång till och som behöver stämma vid alla tidpunkter och även om systemet skulle stängas ner av någon anledning.

iConveyorCounter: Varje transportband har en räknare av antalet pallar som just nu står på bandet. Det finns en övre gräns bestående av 8 pallar per band och på grund av att denna siffra kontinuerligt kommuniceras mot overheadsystemet och utgör basen för programmet Handlers funktion liksom vilket band roboten ska ansluta till så är det viktigt att den alltid stämmer.

bConveyorIsEmptying: I kravspecifikationen för systemet, för att undvika att en pall med produkt står för länge, så är det bestämt att om ett transportband börjar tömmas, då ska det fortsätta tömmas tills det är tomt. Ett tömmande band definieras som att en utlämning har skett från det bandet. Endast ett transportband per färg ska ha denna variabel aktiv samtidigt.

eConveyorColor: Alla transportbandens färger. För projektet har det diskuterats om färgen per transportband ska hårdställas vilket är fallet i prototyplösningen som presenterats eller om den dynamiskt ska sättas efter behov. Programmet är utformat så att det kan hantera båda varianterna.

Dessa tre persistent variables är satta för varje instans i en tvådimensionell array enligt en konsekvent struktur. iConveyorCounter[2][4], bConveyorIsEmptying[2][4] och

eConveyorColor[2][4] hänvisar alla till samma instans av ett transportband, i det här fallet transportband 4 i station 2.

Genom att använda persistent variables på det här sättet säkerställs att viktig information om systemet inte går förlorad vid avbrott eller omstart vilket gör att normal drift kan återupptas snabbt och smidigt när det finns behov för det.

4. Genomförande

4.1 Programmering av transportbandets rörelse och sekvenskontroll

Den automatiserade process som hanterar transport, in- och utlagring samt lyft av pallar utgörs av blocken FB_Conveyor och FB_Elevator. I övervakning och felhanteringssyfte har transportbandens och hissens funktioner i så stor utsträckning som möjligt separarerats, det innebär att hissens rörelse styrs av logiken inom FB_Elevator medan transportbandens motordrift styrs av sekvenser inom FB_Conveyor. Båda blockens sekvenser är beroende av signaler ifrån varandra och programmet stannar i sina sekvenser innan det andra blocket skickar klartecken att gå vidare.

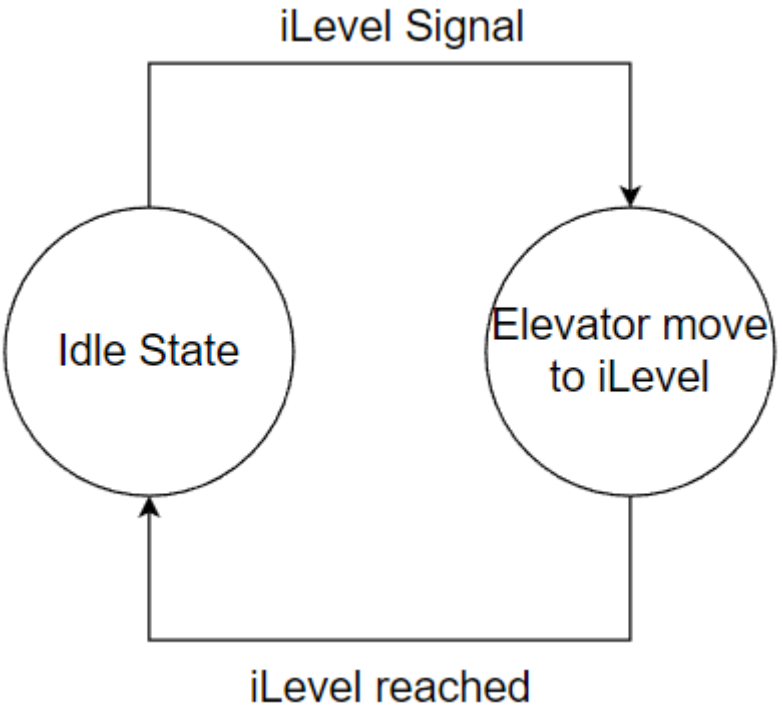
De viktigaste funktionerna som utförs i respektive block för en upphämtnings- eller avlämningssekvens är följande:

FB_Conveyor:

- Anslutning mot overheadsystem och robot.
- Styrning av transportbandens motordrift och riktning.
- Detektering av objekt på transportbandet.
- Övervakning av transportbandets status, såsom stopp, start eller fel.
- Signalering till FB_Elevator för att utföra ytterligare uppgifter.
- Räknare som håller koll på antal pallar på transportband.
- Rapport till overheadsystem om den fullständiga sekvensen är klar.

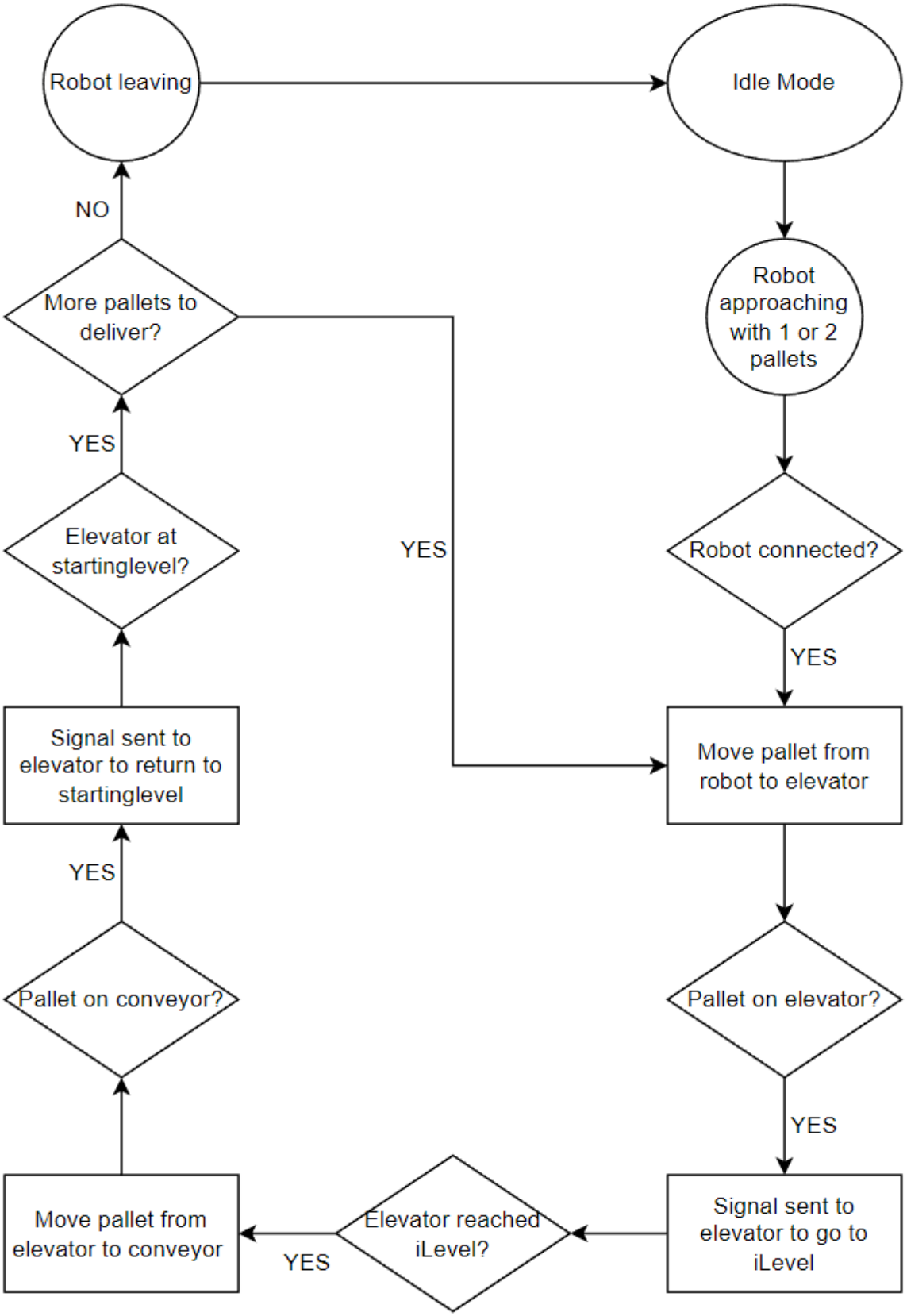
FB_Elevator:

- Styrning av hissens position och riktning.
- Detektering av objekt på hissens transportband.
- Övervakning av hissens status, såsom stopp, start eller fel.
- Signalering till FB_Conveyor om status av pågående transport.



Figur 11: Flödesschema för kontroll av hissens position

Styrningen av hissens rörelse i vertikal led hanteras uteslutande i funktionsblocket FB_Elevator. Flödesschema för hanteringen av hissens rörelse visas i figur 11 ovan.



Figur 12: Flödesschema för en DropOff-sekvens

Sekvensen mellan FB_Conveyor och FB_Elevator för en avlämning av en produktpall ser ut enligt figur 12 ovan och kan beskrivas med ord enligt följande:

1. Overheadsystemet tar emot information från en producerande maskin att det finns en pall med producerad produkt att hämta.
2. Overheadsystemet skickar en robot att hämta produkten från produktionen. På grund av löpande kommunikation från varje transportband till overheadsystemet så finns det ett band redo för avlämning. Roboten plockar upp pallen från produktionen.
3. När roboten anländer till stationen och ansluter till det enda transportband som just nu är aktivt för avlämning påbörjas sekvensen.
4. Transportbandets anslutning och uppkoppling mot roboten initialiserar hissens transportband att börja rulla. Roboten börjar samtidigt att mata ut pallen från sig själv och över till hissen.
5. När hissens givare detekterar att pallen har lämnat roboten och är nu på hissen så skickas information om detta till transportbandet som i sin tur meddelar detta till roboten, här avslutas robotens uppkoppling till transportbandet och den är fri att fortsätta sitt arbete på annan plats.
6. Transportbandet som vet vilken våning den själv befinner sig på skickar därefter signal till hissen att röra sig till denna våning.
7. Hissen går genom sekvensen som styr den till rätt våning.
8. När hissens positionsgivare för rätt våning ger utslag påbörjas sekvensen att transportera pallen från hissen till transportbandet.
9. Hiss- och transportband har givare placerade på ett sätt så att överlämningen fortgår tills dess att hela pallen har lämnat hissen och istället är på transportbandet.
10. Transportbandet skickar klartecken till hissen att den är klar och hissen kan återgå till sin bottenvåning och avvakta tills den får nya instruktioner.

Genom att samordna och kommunicera mellan FB_conveyor och FB_elevator på detta sätt säkerställs en effektiv och automatiserad process för att hantera transport och lyft av pallar. Denna sammansvetsning av funktionsblocken ger en enkel och flexibel lösning som kan anpassas efter olika krav och applikationer där fel och problem isoleras till det block där de uppstått.

4.2 Testning och utveckling av sekvens

Projektets kravspecifikation nämnde specifikt att styrprogrammet skulle utvecklas från grunden. Det har funnits tillgång till den interna kodbasen som inspiration men skulle skapa ett eget program. Det innebar flera olika utmaningar som behövde hanteras.

Första steget var att etablera en övergripande planering där idéer diskuterades, definierade sekvenser, försökte identifiera hinder och utmaningar till sekvenserna och utvecklade en sekvensskiss.

Därefter inleddes projektet med att konstruera en enklare simuleringsmiljö med hjälp av CODESYS visualeringsverktyg där det kunde testas grundläggande antaganden och de sekvenskedjor som diskuterats. Genom att följa den simulerade sekvensen steg för steg och notera hur givare och tillstånd behövde förändras kunde det etableras ett sekvensskelett som det senare i projektet fanns stor nytta av. Särskilt viktigt var att etablera hur samverkan mellan hiss och transportband skulle fungera.

Tidiga maskintester visade dock att simuleringen inte gick att överföra till verkliga förhållanden. Under de första försöken att realisera sekvenserna upplevdes det problem i den mjukvarumässiga kommunikationen mellan hiss och transportband, en opålitlighet i givarsignalerna och att sekvensen stundtals fastnade i sina tillstånd.

Det kunde tillfälligt arbetas runt dessa problem genom användning av timer-funktioner i koden och i mjukvaran hårdställda givarsignaler, men det var uppenbart att det inte var en lösning som kunde användas för annat än de mest grundläggande sekvenstester.

Under projektets planeringsstadie enades examensarbetarna om att det skulle användas funktionsblock för att isolera och styra transportbandens samt hissens funktion. Dock hade det inte tagits hänsyn till ett av funktionsblockens mest fundamentala strukturella regler. Variabler inom ett funktionsblock kan inte modifieras utanför funktionsblocket. Det innebär att det inte går att ändra värdet på en variabel från en annan del av programmet. Det gick därför inte att utbyta information effektivt mellan funktionsblocken för transportband och hiss på ett sätt enligt den struktur påbörjats.

Istället för att börja om valdes det att försöka lösa problemet på olika sätt. Det övervägdes flera lösningar för att hantera problemet. Dessa innefattade användning av invariabler till blocken istället för variabler inne i blocken, att skapa ytterligare ett funktionsblock som skulle sköta samverkan, samt att bygga ihop både transportbanden och hissens funktionsblock till ett enda block. Det undersöktes också om det fanns möjlighet att utveckla separata datastrukturer som kunde agera brygga mellan funktionsblocken. Alla dessa alternativ testades i sökningen efter den bästa lösningen.

Samtliga av dessa förslag löste problemet som fanns men medförde egna följdproblem. Kravet om en tydlig visuell kodstruktur gjorde att styrning med hjälp av invariabler valdes bort på grund av den visuella rörighet som det gav upphov till. Ytterligare ett funktionsblock som sköter samverkan mellan blocken skulle bryta mot hur den planerade kodens struktur där stationens ingående komponenter är isolerade i sina egna funktionsblock. Att sammanfoga transportbanden och hissen i ett enda funktionsblock skulle leda till att skalbarheten och modulariteten i systemet skulle försvinna.

Flera av dessa lösningar och försök utvecklades till problem med dålig inkapsling av funktioner inom hela kodbasen och ledde till så kallade “race condition”-problem, vilket innebär att flera processer kommer åt och ändrar data samtidigt. Genom sekvensutvecklingen hade det skapats problem som examensarbetarna försökte undvika från första början genom uppdelningen i separata funktionsblock. Därför behövdes det grundligt gå igenom rad för rad i samtliga sekvenser för att säkerställa att det inte skrevs olika värden till samma variabel samtidigt.

Efter denna noggranna genomgång enades det om att utveckla en separat datastruktur för transportbanden som gjorde det möjligt att skriva till transportbandens interna variabler utifrån samt att ändra hissens sekvensstruktur så att den enbart skulle styras via dess invariabler. Strukturen stConveyorData skapades och hissens vertikala rörelse styrs nu enbart av sin insignal iToLevel.

Utvecklingen möjliggjorde skapandet av separata sekvenser för transportband och hiss som var beroende av varandra, kunde kommunicera med varandra och fortsatt bestå av separata funktionsblock. Under tester av sekvenserna framkom det att de ledde till en inkonsekvent

pallplacering på transportbanden, något som berodde på för få givarsignaler att basera sekvenserna på. Genom att öka antalet givare samt flytta dem som redan fanns, blev det till sist möjligt att skapa sekvenser som konsekvent ledde till samma pallplacering.

Den ombyggda lådväxlaren är utvecklad för att integreras i ett redan automatiserat produktionsflöde. Det är därför viktigt att alla tillstånd i samtliga sekvenser har alternativa vägar ut så att processen inte fastnar i något tillstånd. Genom att mäta den tid som sekvensen bör befinna sig i varje tillstånd under optimala förhållanden kunde gränsvärden fastställas för när något hade gått fel. Dessa gränsvärden användes för att sätta upp tidsgränser, som markerar när sekvensen har befunnit sig i ett tillstånd för länge. Om sekvensen fastnar i ett tillstånd, vidtas åtgärder baserat på det specifika tillståndet. Det kan antingen innebära att återgå till ett tidigare tillstånd eller övergå till ett 'error'-tillstånd. Det senare kräver ingripande av en operatör.

Tester av olika varianter på systemet har bidragit till att det nu finns en stabil sekvens för att kunna hantera pallarna på ett pålitligt sätt. Som det illustreras i figur 12 så kommer det alltid att hållas koll på var den aktuella pallen befinner sig och se till så att den alltid hamnar på rätt plats. Skulle något mot förmodan hända under sekvensen så finns det bra hantering av dessa problem som sedan kan åtgärdas på ett smidigt sätt. Dessutom har det hittats en bra viktpunkt för funktionsblockens och programmets struktur där systemet är modulärt men där det också är möjligt att nå samtliga viktiga variabler och signaler får huvudblocken i programmet.

4.3 Nätverksanslutning och integration av robot

Inom produktionen hos TePe används idag mobila robotar som transporterar pallar med produkter till och från olika maskiner. För att den ombyggda lådväxlaren ska kunna användas inom detta större system behövdes det därför eb integration av en lösning för att dessa robotar även skulle kunna ansluta till lådväxlaren. Initialt betraktades hela lådväxlaren som en enhet för roboten att ansluta till innan övergången skedde till att hantera varje transportband istället som anslutningspunkt. Det gör helheten mer modulär och möjliggör en mer detaljerad styrning på transportbandsnivå istället för på lådväxlarnivå.

För att hantera kommunikationen mellan robotar och transportband i systemet används en datastruktur skapad av TePe som kallas DUT_RobotDataExchange. Denna struktur innehåller

flera statusbitar och kommandobitar som är relaterade till robotens och transportbandens funktioner och tillstånd. Strukturen gör det möjligt att övervaka och styra robotarnas och transportbandens aktiviteter och att fatta beslut baserat på deras aktuella status.

Hela systemet oavsett antal lådväxlare eller band tilldelas en fast lokal ip-adress som alla transportband delar på och varje transportband identifieras i sin tur av två separata portar. Dessa portar leder till två TCP-servrar, en för avlämning (Dropoff) och en för upphämtning (Pickup). FB_Conveyor-funktionsblocket innehåller variabler för att hantera dessa TCP-servrar och strukturerna DUT_RobotDataExchange för både avlämning och upphämtning.

Ett exempel på hur DUT_RobotDataExchange används är i programmets övervakningstillstånd, där systemet passivt väntar på att en robot ska anlända till hissen och koppla upp sig mot transportbandet. När en robot anländer och dess statusbitar är uppdaterade, skickas kommandon och statusbitar mellan roboten och transportbandet för att initiera och slutföra avlämningsprocessen.

För att roboten ska veta vilket transportband den ska ansluta till måste även kommunikation med overheadsystemet implementeras. Overheadsystemet skickar ut arbetsordrar till robotar för upphämtning eller avlämning och i varje arbetsorder är det specificerat till vilket transportband (dvs port) som roboten ska ansluta till. För att göra detta möjligt används en struktur som kallas stIOOverheadCommunication. Denna struktur innehåller både ingångs- och utgångsvariabler för att hantera kommunikationen mellan transportbandet och overheadsystemet. Genom att använda denna struktur kan systemet övervaka och styra materialflödet och robotarnas rörelser i enlighet med produktionsbehoven.

För att summera så är nätverksanslutning och robotintegration kritiska aspekter för att lådväxlaren ska kunna användas inom den existerande produktionen. Genom att använda strukturer som DUT_RobotDataExchange och stIOOverheadCommunication, samt TCP-servrar för kommunikation, kan systemet övervaka och styra robotarnas och transportbandens aktiviteter och fatta beslut baserat på deras aktuella status.

4.4 Mjukvarulösning för dynamiskt val av transportband

Med målsättningen att optimera lagringsprocessen utvecklades en mjukvarulösning där varje transportband agerar som en egen separat server i systemet. Baserat på data som produktfärg och antal pallar som redan står på bandet ställs en variabel som kommuniceras till ordersystem om bandet är tillgängligt för ut- och inleverans. För varje färg av EasyPick som produceras ska ett band ta emot och ett band skicka ut produkter. När transportbandet för inleverans blir fullt eller transportbandet för utleverans blir tomt ska ett nytt band automatiskt väljas ut och kommuniceras till ordersystemet.

För att realisera denna funktionalitet i PLC-programmet har ett separat funktionsblock utvecklats som räknar antal pallar per band, räknar antal pallar per färg och regelbundet går igenom samtliga band för att identifiera det mest lämpliga transportbandet för in- och utleverans. När banden har identifierats skickas deras tillgänglighet till orderhanteringssystemet medan samtliga övriga band kommuniceras som icke-tillgängliga. Vid varje tidpunkt finns det bara ett transportband aktivt för inleverans och ett transportband aktivt för utleverans per färg.

Detta funktionsblock ligger i bakgrunden och körs i varje programcykel för att omedelbart hantera och välja ut aktiva transportband. Ett antal variabler från varje instans av varje transportband har behövts göras tillgängliga i en separat datastruktur som kontinuerligt uppdateras i realtid för att göra detta möjligt.

Vid tester av detta nya kodblock som fick namnet Handler, upptäcktes en viss risk för en överskridning av programcykelns längd och att all kod och alla beräkningar inte hinner utföras inom den givna cykeltiden. Detta på grund av att det är ett beräkningstungt block som regelbundet stegar igenom flera tvådimensionella arrays och jämför flera olika värden löpande. Detta kan i värsta fall leda till att operationen inte hinner utföras och att systemet inte fungerar korrekt. För att säkerställa funktionalitet har cykeltiden för hela PLC-programmet höjts från 20 ms till 50 ms. Att höja cykeltiden leder till en långsammare process och påverkar precisionen i systemet negativt men då systemet i sin helhet inte är beroende av varken hög precision eller en snabb process så är riskerna med för låg cykeltid större än riskerna med en högre cykeltid.

4.5 Säkerhet

I lådväxlarens ursprungliga konfiguration har den tömts och fyllts manuellt av operatörer, vilket har medfört att konstruktionen varit mycket öppen och givit enkel fysisk åtkomst till samtliga komponenter. Denna öppenhet var nödvändig för lådväxlarens tidigare användningsområde, men nu när den har konverterats till en automatisk drift innebär detta betydande risker för personalens säkerhet. Eftersom den ombyggda lådväxlaren befinner sig i en produktionshall där personal kontinuerligt rör sig runt maskinen, är det av yttersta vikt att överväga en omfattande säkerhetslösning för att minimera risken för personskador.

För att utveckla en sådan säkerhetslösning har befintliga säkerhetssystem i andra produktionsmaskiner studerats, och samråd har förts med personal på plats. Detta har bidragit till att skapa en grund för att utforma en lämplig säkerhetsåtgärd.

Vid denna rapports publicering har ännu ingen slutgiltig säkerhetslösning kunnat bestämmas då den kommer bero på mellanlagrets placering i den produktionshall som är under uppbyggnad. Alternativ som diskuterats är ljusvakter, inkapsling i en slussliknande struktur eller tillbyggnad av ytterligare transportband som separarerar mellanlagrets funktion ytterligare från robotens dockningsstation.

4.6 Framtida utvecklingsmöjligheter

Systemet som har tagits fram omfattar fyra transportband och en hiss. Sedan starten av projektet då programmets grundstruktur tagits fram så har det utformats för att kunna styra ett större system med många fler transportband och hissar. Detta medförde att mjukvaran redan från början formades med en modulär arkitektur som skulle vara enkel och skalbar. Att utforma det på detta sättet redan från projektets start har lett till att det finns god potential att bygga vidare på det uppnådda resultatet. En uppskalning från fyra transportband till exempelvis sexton bedöms vara möjlig redan idag med den utvecklade strukturen. Det enda behovet en sådan uppskalning medför är elektrisk koppling och mappning av I/O-signaler på den PLC som ska styra systemet.

Ett programblock som aldrig lämnade planeringsstadiet men som både är fullt genomförbart och även önskvärt i en vidareutveckling av detta system är att hantera varje enskild produktpall separat. Likt FB_Elevator skulle ett sådant block kunna utvecklas som underordnat FB_Conveyor, alternativt som en en datastruktur som länkas till transportbanden på samma sätt som kommunikationsstrukturerna. Detta block skulle kunna innehålla exempelvis tid som den enskilda pallen har lagrats, var den kom ifrån, när den tillverkades och andra intressanta variabler.

En sådan vidareutveckling skulle ge en mer granulär kontroll och programmet Handler skulle då kunna ta hänsyn till pallarnas enskilda data när transportband väljs för ut- och intransport istället för det nuvarande systemet då ingen hänsyn tas till hur länge en pall har stått på bandet. Att göra på detta sätt hade även gett tydlig åtkomst till palldata som står på transportbandet, exempelvis i arrayformat. Diskussioner om en tredimensinell array som specifikt pekar på en pall och dess data har förts och skulle då resultera exempelvis i en datastruktur där en persistent variabel PalletData[1][3][4] skulle ge specifik information om pallen som står på station 1, transportband 3, plats 4.

För att kunna hantera artiklar enskilt på detta sätt och se till att den som lagrats längst skulle lämna systemet först så skulle en intern sorteringsfunktion även behöva implementeras. Även det är föremål för framtida utveckling men leder då till att systemet inte är tillgängligt för ut- och inleverans under denna sorteringstid.

Den grundläggande prototyplösning som har tagits fram är på grund av sin modulära struktur lämpad att utvecklas vidare på detta sätt alternativt efter annan kravspecifikation. Med en ökad komplexitet på det beskrivna sättet ökar också risken för fel och problem vilket skulle behöva tas hänsyn till och hanteras om vidareutveckling av projektet sker.

5. Resultat

5.1 Projektets resultat i helhet

Under projektets gång har ett fungerande helhetssystem tagits fram och implementerats, vilket realiserar ett mellanlager enligt de önskemål och krav som sattes innan arbetet påbörjades. De automatiska robotar som redan är aktiva i produktionsprocessen kan hämta och lämna pallar från det utvecklade systemet på ett säkert och smidigt sätt.
Den ombyggda lådväxlaren är utrustad med en intelligent styrning som kan flytta pallar mellan olika transportband samt till och från hissen som i sin tur kan transportera dessa till och från roboten. Givet kriterier såsom vilken färg på tillverkad produkt som kommer med roboten kan systemet avgöra vilket band som pallen ska hämtas från eller lämnas till. Genom användning av asynkronmotorer och hybridstartare har en smidig och konsekvent motordrift till transportbanden uppnåtts vilket har förbättrat effektiviteten och flexibiliteten.

Kommunikation mellan transportband, robot och overheadsystem har blivit effektivt implementerat och genom elskåpets inkopplade komponenter har en pålitlig intern och extern kommunikation och styrning av systemets olika delkomponenter säkerställts. Utförliga tester har verifierat att systemet fungerar enligt uppsatta krav och mål och de har visat att systemet är kapabelt att hantera den uppgift som det konstruerats för.

Sammanfattningsvis har ett framgångsrikt prototypsystem utvecklats som kan agera mellanlager mellan produktion och förpackning i en automatiserad industriell miljö och detta har åstadkommits med modulära komponenter vilket innebär att systemet snabbt kan skalas upp. Det finns tydliga idéer för hur den ombyggda lådväxlaren kan utvecklas vidare för att stärka dess förmåga och möjlighet att ha en effektiviserande påverkan för hela produktionsanläggningen.

5.2 Funktionsblock i styrprogrammet

Under projektets gång har flera funktionsblock utvecklats från grunden som skapat styrprogrammet. I korthet är dessa:

FB_Conveyor: Det viktigaste och största blocket. Hanterar anslutning till nätverk, tar emot och skickar signaler och styrning inom den ombyggda lådväxlaren och är det block som ansvarar för flest funktioner.

FB_Elevator: Underställt FB_Conveyor och är skapat för att separarera hissens funktion inom systemet och isolera det till ett block. Har säkerställt jämn och pålitlig vertikal förflyttning mellan de olika transportbanden.

FB_Station: Har tillhandahållit en grafisk översikt över en ombyggd lådväxlarens ingående komponenter. Har skapats för enkel överblick och förståelse av var och när fel uppstår.

FB_BuffertStation: Liknande funktion som FB_Station men ännu mer övergripande. Ger en grafisk översikt över samtliga komponenter och instanser som slutligen ingår i helheten.

FB_Handler: Det största blocket vad gäller ren mjukvara. Ansvarar för att välja vilka transportband som har vilka uppgifter. Sammanställer data från varje enskilt transportband och tar beslut baserat på denna. Har skapat förutsättningar för en dynamisk hantering av systemet och sätter relevanta variabler i rätt läge för kommunikation med nätverket. Utför huvuddelen av de mjukvarumässiga beräkningar som ingår i systemet.

Dessa funktionsblock har gjort det möjligt att skapa ett styrprogram som uppfyller de krav på automatisering och effektivitet som har legat till grund för projektets arbete. Styrprogrammet i sin helhet har visat sig vara pålitligt under tester och bekräftar att den föreslagna ombyggnationen av avvecklade lådväxlare är möjligt och framgångsrikt kan genomföras.

6. Slutsats

Detta projekt har visat att det går att framgångsrikt bygga om avvecklad industriell utrustning till ett automatiserat mellanlager som kan integreras sömlöst i redan existerande processer. Ett noggrant förarbete har lagts för att förstå systemets tekniska egenskaper och driftskrav, vilket har lett till identifiering och modifiering av de komponenter som har behövts för att skapa förutsättningar för detta nya ändamål.

Den utvecklade mjukvaran har resulterat i ett styrprogram som uppfyller de krav på automatisering och effektivitet som har efterfrågats och tester har bekräftat att systemet är pålitligt och effektivt. Det har bekräftats att det här är ett möjligt sätt att återanvända utrustning för att skapa förutsättningar för en förbättrad effektivitet.

Slutligen så har projektet visat att det är en möjlighet att inom industriell produktion finna nya användningsområden för avvecklad utrustning som annars skulle kasseras. Det har dessutom visats att det är något som kan höja effektivitetsgraden i redan existerande processer och hur innovativa lösningar gällande utrustning och material som redan finns på plats är ett kostnadseffektivt sätt att skapa konkurrensfördelar. Projektet har uppnått de mål och krav som ställdes genom att implementera en automatiserad lagerlösning där tidigare maskiner inte har kunnat köras för full effekt på grund av flaskhalsar i produktionen.

6.1 Besvarande av problemformuleringar

I detta avsnitt så besvaras de frågeformuleringar som projektet utgått ifrån att undersöka.

6.1.1 Befintliga lådväxlare och robotar

Befintliga lådväxlare och robotar fungerar idag effektivt och autonomt för att hantera och transportera pallar inom en produktionsmiljö. De är programmerade för att utföra uppgifter som att hämta och lämna pallar på förutbestämda platser samt att kommunicera med andra enheter i systemet.

6.1.2 Utveckling av ett nytt PLC-program

Projektet har framgångsrikt lyckats utveckla ett nytt PLC-program från grunden för styrning av motordrifter till banor och hiss. Detta har visat sig vara fullt möjligt och har gett möjligheten att anpassa styrningen efter specifika behov och krav.

6.1.3 Kommunikation med mobil robot

En framgångsrik implementering av kommunikation mellan de mobila robotar och det overheadsystem som styr dem har uppnåtts, vilket nu samverkar effektivt med det nya PLC-programmet. Detta har möjliggjort en effektiv och smidig samordning mellan lådväxlaren och roboten, vilket har bidragit till en ökad effektivitet och flexibilitet i systemet.

6.1.4 Ombyggnad av befintliga lådväxlare till mellanlager

Under projektets gång har möjligheten att bygga om befintliga lådväxlare till ett mellanlager undersökts. Genom att kombinera flera lådväxlare och anpassa deras styrning och kommunikation har ett effektivt mellanlager som kan hantera och lagra pallar på ett flexibelt sätt skapats.

6.1.5 Säkerhet för en större cell

För att säkerställa säkerheten i en större cell med fyra lådstaplare har en säkerhetslösning som inkluderar skyddsrelä, nödstopp och säkerhetsbarriärer utvecklats. Dessa åtgärder har bidragit till att minimera riskerna och säkerställa en säker arbetsmiljö för personal och utrustning.

6.1.6 Säkerställa att produkter inte ligger för länge

För att säkerställa att produkter inte ligger för länge i mellanlagret har en tidsbaserad övervakning och styrning i PLC-programmet implementerats. Denna funktion övervakar lagringstiden för varje pall och ger prioritet till de pallar som har legat längst i mellanlagret för att undvika onödiga förseningar och förbättra produktionsflödet.

6.1.7 Lämplighet att konvertera lådväxlare till mellanlager

Efter utvärdering och analys av resultaten från projektet bedöms det som lämpligt att konvertera några stycken lådväxlare till ett mellanlager. Detta ger möjlighet att skapa ett

flexibelt och effektivt system för hantering och lagring av pallar, vilket kan bidra till en ökad produktivitet och en bättre anpassning till förändrade produktionsbehov.

7. Etiska aspekter

Automatisering av processer och system inom automationsteknik kan ge upphov till ett antal etiska frågeställningar. I samband med det projekt som denna rapport beskriver är det viktigt att överväga både de positiva och negativa konsekvenserna av det automatiserade mellanlager som har skapats.

En av de mest uppenbara och viktiga frågorna kring automatiseringen är dess påverkan på arbetssituationen inom företaget. Genom att automatisera processer kan manuella och repetitiva arbetsuppgifter elimineras, vilket i sin tur innebär att vissa yrkesroller kan försvinna eller förändras. Detta kan resultera i arbetslöshet för vissa individer och kan därför påverka bland annat någons ekonomiska situation och/eller deras välbefinnande. Exempel på andra aspekter som hade kunnat diskuteras kan vara arbetsmarknadsförändringar, säkerhet och ansvar, miljöpåverkan och integritet och övervakning. Projektets resultat bedöms dock inte påverka dessa områden i någon större utsträckning.

8. Källförteckning

[1] Encyclopedia Britannica. (2023). Advantages and disadvantages of automation [Elektronisk resurs]. Tillgänglig: https://www.britannica.com/technology/automation/Advantages-and-disadvantages-of-automation. Hämtad: 16 maj 2023.

[2] Tepe. "Om Tepe." [Elektronisk resurs]. Tillgänglig: https://www.tepe.com/sv/om-tepe/. Hämtad: 8 maj 2023.

[3] TePe, “TePe EasyPick™,” [Elektronisk resurs]. Tillgänglig: https://www.tepe.com/sv/produkter/tandpetare/easypick/. Hämtad: 16 maj 2023

[4] W. Bolton, Programmable Logic Controllers, 6th ed., Oxford, UK: Newnes, 2015, kap. 1, pp. 1–18.

[5] W. Bolton, Programmable Logic Controllers, 6th ed., Oxford, UK: Newnes, 2015, kap. 5, pp. 115–136.

[6] W. Bolton, Programmable Logic Controllers, 6th ed., Oxford, UK: Newnes, 2015, kap. 6, pp. 167–182

[7] Automation Community, “What is a Human Machine Interface (HMI)? Types, Advantages,” [Elektronisk resurs]. Tillgänglig: https://automationcommunity.com/human-machine-interface/. Hämtad: 16 maj 2023.

[8] L. Harnefors and H.-P. Nee, *Elektriska maskiner*, 2nd ed. Stockholm: Liber, 2015, kap. 10, “Asynkronmaskinen”.

[9] Bonfiglioli, “BN Series Asynchronous Motors,” [Elektronisk resurs]. Tillgänglig: https://www.bonfiglioli.com/_default_upload_bucket/BR_CAT_BNEX_STD_ENG_R05_0.pdf. Hämtad: 16 maj 2023

[10] Wemo Group, “WCS - OB” [Elektronisk resurs]. Tillgänglig: https://www.wemogroup.com/media/1316/wcs-ob.pdf. Hämtad: 16 maj 2023

[11] Wemo Group, “WCS-R” [Elektronisk resurs]. Tillgänglig: https://www.wemogroup.com/media/1317/wcs-r.pdf. Hämtad: 16 maj 2023.

[12] CODESYS Group, “CODESYS,” [Elektronisk resurs]. Tillgänglig: https://www.codesys.com/the-system.html. Hämtad: 16 maj 2023

[13] CODESYS Help, “Retain and Persistent Variables,” [Elektronisk resurs]. Tillgänglig: https://help.codesys.com/api-content/2/codesys/3.5.12.0/en/_cds_vartypes_retain_persistent/. Hämtad: 16 maj 2023

9. Appendix

9.1 Main

```
PROGRAM Main
VAR CONSTANT
(*Konstanter som används i övriga programmet. Ändra dessa endast här om antalet
stationer/band förändras. *)

c_STATIONS : INT    := 4;    //Number of stations active
c_CONVEYORS_PER_STATION : INT := 4;          //Number of conveyors active per station
c_PALLETS_PER_CONVEYOR : INT := 8;           //Number of pallets before a conveyor is
considered full. Default 8, tested and OK up to 9.

c_TOTAL_CONVEYORS : INT := (c_STATIONS * c_CONVEYORS_PER_STATION) ;
//Total numbers of all conveyors from all stations combined.
END_VAR
VAR
        Elevator10: FB_Elevator;
        Elevator20: FB_Elevator;
        Elevator30: FB_Elevator;
        Elevator40: FB_Elevator;

        Conveyor11: FB_Conveyor;
        Conveyor12: FB_Conveyor;
        Conveyor13: FB_Conveyor;
        Conveyor14: FB_Conveyor;

        Conveyor21: FB_Conveyor;
        Conveyor22: FB_Conveyor;
        Conveyor23: FB_Conveyor;
        Conveyor24: FB_Conveyor;
```

```
        Conveyor31: FB_Conveyor;
        Conveyor32: FB_Conveyor;
        Conveyor33: FB_Conveyor;
        Conveyor34: FB_Conveyor;

    Conveyor41: FB_Conveyor;
    Conveyor42: FB_Conveyor;
    Conveyor43: FB_Conveyor;
    Conveyor44: FB_Conveyor;

        bBuffertStationError : BOOL;
        bBuffertStationMessage : BOOL;

        (* ConveyorData, variabler från FB_Conveyor av intresse utanför FB_Conveyor *)
        ConveyorData:          ARRAY[1..c_STATIONS]     OF ARRAY
[1..c_CONVEYORS_PER_STATION] OF stConveyorData;

        (* AllConveyors, alla conveyors i [station][conveyor] *)
        AllConveyors:                ARRAY[1..c_STATIONS]     OF ARRAY
[1..c_CONVEYORS_PER_STATION] OF FB_Conveyor;

        (* OverHeadCommunication *)
        OverheadCommunication: ARRAY [1..c_STATIONS] OF ARRAY
[1..c_CONVEYORS_PER_STATION] OF stIOOverheadCommunication;

        BuffertStation: FB_BuffertStation;
END_VAR
```

9.2 FB_BuffertStation

```
FUNCTION_BLOCK FB_BuffertStation
VAR_INPUT
END_VAR
```

```
VAR_IN_OUT

        Elevator10: FB_Elevator;
        Elevator20: FB_Elevator;
        Elevator30: FB_Elevator;
        Elevator40: FB_Elevator;

        Conveyor11: FB_Conveyor;
        Conveyor12: FB_Conveyor;
        Conveyor13: FB_Conveyor;
        Conveyor14: FB_Conveyor;

        Conveyor21: FB_Conveyor;
        Conveyor22: FB_Conveyor;
        Conveyor23: FB_Conveyor;
        Conveyor24: FB_Conveyor;

        Conveyor31: FB_Conveyor;
        Conveyor32: FB_Conveyor;
        Conveyor33: FB_Conveyor;
        Conveyor34: FB_Conveyor;

        Conveyor41: FB_Conveyor;
        Conveyor42: FB_Conveyor;
        Conveyor43: FB_Conveyor;
        Conveyor44: FB_Conveyor;
END_VAR
VAR_OUTPUT
        bError: BOOL;
        bMessage: BOOL;
END_VAR
VAR
        Station1 : FB_Station;
```

```
        Station2 : FB_Station;
        Station3 : FB_Station;
        Station4 : FB_Station;

END_VAR
```

9.3 FB_Station

```
FUNCTION_BLOCK FB_Station
VAR_INPUT
   Elevator : FB_Elevator;
        Conveyor1 : FB_Conveyor;
        Conveyor2 : FB_Conveyor;
        Conveyor3 : FB_Conveyor;
        Conveyor4 : FB_Conveyor;
END_VAR

VAR_IN_OUT

END_VAR

VAR_OUTPUT
        bError : BOOL;
        bMessage : BOOL;
END_VAR
```

9.4 FB_Conveyor

```
FUNCTION_BLOCK FB_Conveyor
VAR_INPUT
        EN : BOOL;
        iLevel : INT; // Vilken nivå transportbandet befinner sig på i verkligeheten
        eSetColor : eColors; //Vilken av de fyra färgerna transportbandet behandlar
```

```
        bStart : BOOL;
        bHalt : BOOL;
        bReset : BOOL;
        uiDropoffComPort : UINT; //Kommunikationsport för dropoff
        uiPickupComPort : UINT; //Kommunikationsport för pickup

END_VAR
VAR_OUTPUT
        ENO : BOOL;
        bError : BOOL; // Blir hög om fel i systemet uppstår
        bMessage : BOOL; // Blir hög om felmeddelande i systemet uppstår
END_VAR
VAR_IN_OUT
        Elevator : FB_Elevator; //Instans av den hiss som det aktuella bandet tillhör
        eConveyorColor : eColors;
        bIsEmptying: BOOL; //Hög om tömning av bandet påbörjats
        iCounter : INT; //Räknar antalet pallar på bandet
        Data : stConveyorData; //Instans av den struct som samlar information om bandet
        Com: stIOOverHeadCommunication; // Instans av den struct som behandlar
kommunikationssignalerna till overheaden.
END_VAR
VAR



        DropOffComServer : FB_TCP_SERVER;
        DropoffOutputToRobot: DUT_RobotDataExchange;
        DropoffInputFromRobot: DUT_RobotDataExchange;

        PickupComServer : FB_TCP_SERVER;
        PickupOutputToRobot: DUT_RobotDataExchange;
        PickupInputFromRobot: DUT_RobotDataExchange;

        emptyDUT                        : DUT_RobotDataExchange;
```

```
        bConveyorForward : BOOL; //Kör band fram
        bConveyorBackward : BOOL; //Kör band bak
        iDropOff : INT; //Sekvensvariabel
        iPickUp : INT; //Sekvensvariabel
        tConveyorTimeOut : TIME := T#40S; //Timer
        DropOffDone: BOOL; //Checkvariabel

        bConveyorFull : BOOL; //Signal från en givare som blir hög när bandet är fullt
        bBoxInPosition : BOOL; // Givare som håller koll på när en pall kommit in helt på
bandet.
        bConveyorGo: BOOL; //Knapp för manuell kontroll

        iNbrOfPalletToDeliver : UINT; // Anger om de är 1 eller 2 pallar i beställningen.
        iNbrOfPalletDelivered : UINT; // Anger hur många, 1 eller 2 pallar, som levererats.
        bPalletOnRobot : BOOL; // Blir hög när pallen kommit helt på roboten

        bStartTCPServer: BOOL;
END_VAR
```

9.5 FB_Elevator

```
FUNCTION_BLOCK FB_Elevator
VAR_INPUT
   iToLevel : INT;
        bElevatorForward : BOOL;
        bElevatorBackward : BOOL;
        bResetElevator : BOOL;
        bDropOffRobotToElevator : BOOL;
        bPickUpElevatorToRobot : BOOL;
        bPalletOnRobot : BOOL;
```

```
END_VAR
VAR_OUTPUT
	bError : BOOL;
	bMessage : BOOL;

END_VAR

VAR
	bBusy : BOOL;
	iElevatorAtLevel : INT;
	(* Control elevator motor *)
	bElevatorUp : BOOL;
	bElevatorDown : BOOL;

	(* Elevator sensors *)
	bElevatorAtConveyor1 : BOOL;
	bElevatorAtConveyor2 : BOOL;
	bElevatorAtConveyor3 : BOOL;
	bElevatorAtConveyor4 : BOOL;
	bElevatorAtRobot: BOOL;
	bElevatorHasPalletOuter : BOOL;
	bElevatorHasPalletInner : BOOL;

	(* HMI *)
	bSwitchDirection : BOOL;
	bElevatorButtonUp: BOOL;
	bElevatorButtonDown: BOOL;
	bElevatorGo: BOOL;

	tElevatorTimeOut : TIME := T#15S;

	iElevator: INT;
	iReset: INT;
```

```
        iDropOffRobotToElevator: INT;
        iPickUpElevatorToRobot: INT;
        bPalletPastOuter: BOOL;
END_VAR
```

9.6 Robot

```
FUNCTION_BLOCK FB_Robot_1
VAR_INPUT
        EN:BOOL:=TRUE;
        bRobotWaiting: BOOL;
        sWaitingStatus: STRING;
        InputFromStation : DUT_RobotDataExchange;
        bArclReady : BOOL;
        bArclDone : BOOL;
        bEStopActive : BOOL;
END_VAR
VAR_OUTPUT
        ENO:BOOL;
        bError: BOOL;
        bWaitCancel: BOOL;
        bLocalizeOut: BOOL;
        sLocalizeMacro: STRING;
        bShutDown: BOOL;
        OutputToStation : DUT_RobotDataExchange;
        bSay : BOOL;
        sStringToSay : STRING;
END_VAR
VAR_IN_OUT
        stIO: IORobotConveyor;
END_VAR
VAR
        iState :INT;
        Motor : FB_RollerMotor;
```

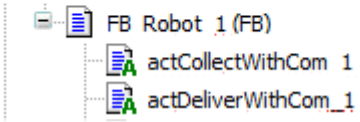
```
(* Timers *)
ton_DeliverBoxes: TON;
ton_RetryElevator: TON;
ton_AlignLoad: TON;
ton_ShutDown: TON;
ton_DelayAlignLoad: TON;

(* Variables for communication, general *)
bPairingIn: BOOL;           (* INPUT *)
bPairingOut: BOOL;          (* OUTPUT *)

(* Variables for communication, mode collect/deliver with com*)
bStationReadyForBoxes: BOOL;    (* INPUT, used by deliver only *)
bStationConveyorStarted: BOOL;  (* INPUT *)
bStationDeliverDone: BOOL;          (* INPUT, used by collect only *)
bRobotReadyForBoxes: BOOL;          (* OUTPUT, used by collect only *)
bRobotConveyorStarted: BOOL;    (* OUTPUT *)
bRobotDeliverDone: BOOL;        (* OUTPUT, used by deliver only *)

FB_TCP_Client:      FB_TCP_Client;
emptyDUT: DUT_RobotDataExchange;
rtrig_Estop : R_TRIG;

END_VAR
```



Figur 13: FB_Robot_1