Universitetet i Stavanger

DET TEKNISK-NATURVITENSKAPELIGE FAKULTET

BACHELOROPPGAVE

Studieprogram/spesialisering: Bachelor i ingeniørfag / Automatisering og elektronikkdesign	Vårsemesteret 2021 Åpen
Forfattere: Sondre Holmestuen, Edvin Benjaminsen, Marie Bø-Sande	
Fagansvarlig: John Breiland Veileder: Karl Skretting	
Tittel på bacheloroppgaven: Automatisere en produksjonslinje for fylling av 5-liters flasker Engelsk tittel: Automation of a production line for filling of 5-liter bottles	
Studiepoeng: 20	
Emneord: PLS, HMI, OPC, automatisering, flaskefylling	Sidetall: 65 + vedlegg/annet: 97 Stavanger 15. mai 2021

Sammendrag

Norwegian Oil Supply er en bedrift lokalisert på Forus. De leverer utstyr, tjenester og kjemikalier til oljeleverandører i Norge. I dag har bedriften fire operative produksjonslinjer for fylling av kjemikalier. Linjene er delvis automatisert, men de ønsker å se på muligheten til å videreutvikle produksjonen.

Rapporten tar for seg hvordan bedriften kan videreutvikle en av disse produksjonslinjene. Det blir først gjennomgått hvordan nåværende systemet fungerer og hvilke komponenter som blir brukt. Virkemåten til systemet blir vist i form av tilstandsdiagrammer, og deretter implementert som ladderkode i Sysmac Studio.

Videre i rapporten kommer det forslag til videreutvikling av systemet. Forslagene til oppgraderingen innebærer utbytting av PLS, måling av påfyllingsmengde ved bruk av veiecelle, OPC-kommunikasjon til PC, utbytting av det manuelle brukergrensesnittet til et grafisk og automatisk pakking av flasker ved bruk av en robotarm.

Simulering av fyllesystem er vist i video:
`https://www.youtube.com/watch?v=OO8jHTafjYo`.

Simulering av robotarm er vist i video:
`https://www.youtube.com/watch?v=5EOzuQ4PZGk`.

Forord

Forfatterne til denne rapporten består av tre studenter; Marie Bø-Sande, Edvin Benjaminsen og Sondre Holmestuen. Samtlige studerer automatisering og elektronikkdesign på UiS, hvor en også har fagbrev som Automatiker fra Mongstad, Equinor.

I løpet av de fem siste månedene har vi jobbet med å automatisere en fyllelinje for flasker, samt simulering av fylle- og pakkeprosessen. I sammenheng med oppgaven har vi fått hjelp av ansatte ved UIS, og vi vil gjerne rette en takk til:

- **Karl Skretting** for god oppfølging og veiledning
- **John Breilad** for kontaktperson for bedriften
- **Romuald Karol Bernacki** for gode råd og lån av laboratorieutstyr
- **Ståle Freyer** for hjelp med Robotstudio
- **Didrik Efjestad Fjereide** for hjelp og råd med Sysmac Studio

Innhold

Kapittel 1

Introduksjon

1.1 Problemstilling

Industrien innenfor automatisering er i hurtig utvikling og teknologien blir stadig bedre. Denne utviklingen kan være veldig gunstig for mange aktører og man har muligheten til automatisere produksjonen i større grad enn tidligere. Med implementeringen av industri 4.0 vil standardiseringen av kommunikasjonen mellom maskiner gi muligheten til å implementere flere systemer inn i samme løsning.

Med økende funksjonalitet vil også kompetansen som kreves bli forandret. Programvarer og maskinvarer som før var vanlige å bruke kan bli utdatert som gjør at videreutdannelse for automatikere er nødvendig. Ved å holde produksjonslinjene oppdatert vil det også være lettere å holde følge med andre aktører.

Et godt eksempel på dette er Norwegian Oilfield Supply. For å fremtidssikre seg ønsker de å se på muligheten til å oppgradere to av sine eldre produksjonslinjer. Konsekvensene ved å ikke gjøre dette kan være at systemet blir foreldet, og at ekspertisen i fagfeltet har gått over til nyere systemer. Andre ulemper er mangelfull overvåking og rapportering av produksjonslinjen som kan føre til sen oppdagelse av feil i systemet. Ved å oppdatere produksjonslinjen vil det være høyere initiale kostnader, men med et nytt system er

det mulig å øke produksjonsvolum, redusere variable kostnader og unngå inaktivitet i form av systemfeil.

1.2 Oppgavebeskrivelse

Bacheloroppgaven er gitt i samarbeid med Norwegian Oilfield Supply AS. Oppgaven går ut på å kartlegge en av de fire eksisterende produksjonslinjene til selskapet, og se på muligheten for å oppgradere den. Nåværende PLS'en skal bli erstattet med en nyere modell, og systemet skal få nye funksjonaliteter; som OPC-UA[1] kommunikasjon, et enklere og digitalt brukergrensesnitt og høyere grad av pålitelighet. I siste del av oppgaven vil det bli sett på muligheten til å implementert en industriell robotarm.

Beskrivelse av oppgaven, hentet fra Digital studentekspedisjon:

"[...] En maskin for fylling av 5-liters dunker skal oppgraderes med ny PLS og HMI. Maskinen består av transportbånd, 4 fyllehoder som fyller 4 flasker samtidig, pneumatiske sylindre og stempler som fylles med ønsket mengde væske for hver dunk. Første del av arbeidet består i å få en fullstendig oversikt over systemet slik det er nå. Deretter vise hvordan all funksjonalitet kan implementeres ved hjelp av en PLS. Til slutt kan en på UiS laboratorium E458 bruke ABB robot for å plassere enkle objekter omkring, på en måte som gir noe sammenheng med å plassere flasker i esker og esker på pall. Sammenheng er gjerne gitt ved at styringen av prosessen, det vil si samkjøringen av roboten(e) og eventuelt transportbånd fra HMI og PLS, på laboratoriet er så lik som mulig styring av produksjonslinje i bedrift. Nedenfor er en punktliste som på stikkordsform lister (mulige) oppgaver.

- Utarbeide oversikt over elektriske komponenter som sensorer, ventiler, motorer etc.
- Utarbeide oversikt over innganger og utganger på eksisterende PLS og hva disse brukes til (IO-liste)
- Foreslå en prosjektplan for gjennomføring av oppgradering

[1] OPC-UA - Open Platform Communications-United Architecture

- Lage tilstandsdiagram og realisere det i Sysmac
- Vise status på all IO på HMI.
- Når ny ordre ankommer til maskin via OPC linken fra klient på PC blinker fysisk varsellys.
- Fra HMI grensesnitt kunne starte utførelse av ny ordre.
- Vise status på ordre og andre relevante data i PLS på PC (OPC-klient).
- Vise enkel animasjon (simulering) av maskinen på PC.
- Legg til ekstra task for å simulere maskin. Evt. bruk en ekstra PLS for simulering og koble sammen IO.
- Motta og behandle alarmer og hendelser fra OPC.
- Utvikle nødvendige algoritmer i RobotStudio med simulering.
- Skrive program for samkjøringen av roboten(e) og eventuelt transportbånd for PLS og HMI i Sysmac.
- Program skal kunne rapportere status til server via OPC.

[...]"

1.3 Revidert oppgavebeskrivelse

I februar ble det i samråd med veiledrene bestemt å oppdatere oppgaven. Den ene grunnen var manglende tegninger og dokumentasjon av PLS program på det eksisterende systemet. En annen grunn var at etter et besøk hos bedriften kom det fram at det var ønskelig med detektering av fyllenivået i flaskene under produksjonen. Det kom også frem at bedriften ikke ønsket å styre systemet med en PC. Derfor utgikk punktet med ordreoverføring fra PC.

Oppgaven gikk også ut på å bruke robotarmen hos UiS, men i samråd med veileder ble det bestemt i April at kun simuleringen skulle bli prioritert. Dette fordi simuleringen er såpass lik den virkelige prosessen.

1.3 Revidert oppgavebeskrivelse

Etter revideringen ble følgende punkter prioritert. De oppdaterte punktene er skrevet med fet skrift:

- Utarbeide oversikt over innganger og utganger på eksisterende PLS og lage en I/O liste over de.
- Oppgradere brukergrensesnittet til en digital HMI skjerm.
- Lage tilstandsdiagram og realisere det i Sysmac.
- Vise status på all IO på HMI.
- Fra HMI grensesnitt kunne starte utførelse av ny ordre.
- Lag enkel animasjon (simulering) av maskinen på PC.
- Motta og behandle alarmer og hendelser fra OPC.
- Forslag til kode for styring av PLS.
- Utvikle nødvendige algoritmer i RobotStudio med simulering.
- **Simulering i Robotstudio i stedet for lab-test**.
- **Se på muligheter til å kontrollere om en flaske har fått riktig mengde påfyll** .

Kapittel 2

Bakgrunn

Med tanke på at det nye systemet skal bygges på det gamle er det hensiktsmessig å se på oppbygningen til det eksisterende systemet før man kan ta for seg en løsning. I dette kapittelet utdypes det hvordan eksisterende system fungerer, PLS´en oppbygning og programvarer.

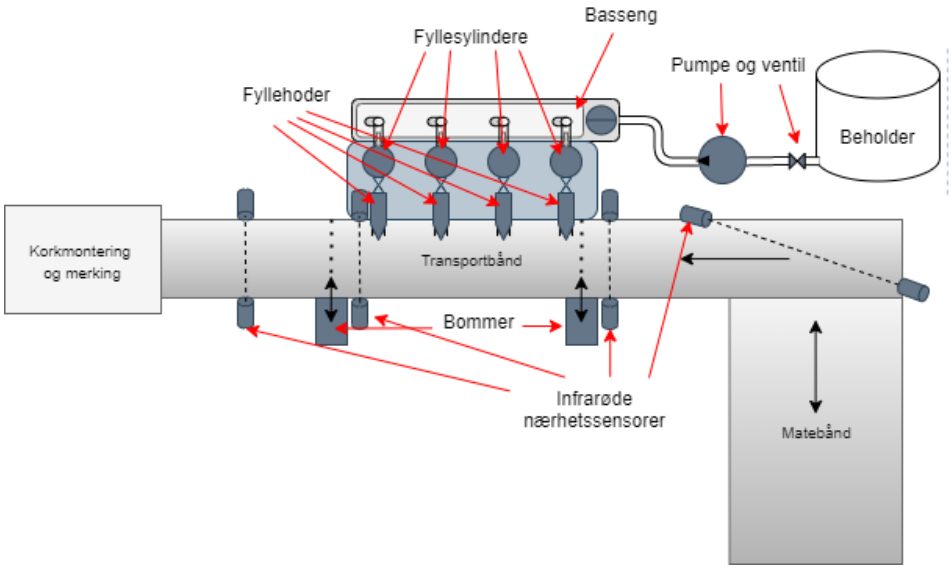
2.1 Beskrivelse av eksisterende system

Produksjonslinjen er i dag delvis automatisert og består av følgende deler:

- Et mateband
- Et transportband
- Fire fyllehoder med ventil
- Fire fyllesylindere
- Fire pneumatiske stempel
- To pneumatiske bommer
- Fire infrarøde nærhetssensorer

- Et basseng med nivåmåler
- En pumpe og en ventil til å fylle bassenget

Figur 2.1 og beskrivelsen under er basert observasjoner fra bedriftsbesøket og illustrerer det nåværende systemet.



Figur 2.1: Illustrasjon av det eksisterende systemet

Bassenget er en buffer for beholderen og er ansvarlig for å ha tilstrekkelig med væske til fyllesylinderne. Den regulerer væskeinnholdet når det synker ved at den åpner en ventil og pumper mer væske fra beholderen. Bedriften har mange beholdere med forskjellig innhold, og skiftes manuelt med en truck avhengig av hva som skal fylles.

Flaskene blir manuelt lagt på matebåndet. De blir plassert fire i bredden med en orienteringen slik at flaskehalsen er på venstre side, relativt til figur 2.1. De blir så ført videre til transportbåndet. Matebåndet vil stanse når en sensor registrerer flasker og starte igjen når samme sensor ikke registrerer flasker. Transportbåndet går kontinuerlig så lenge systemet er på.

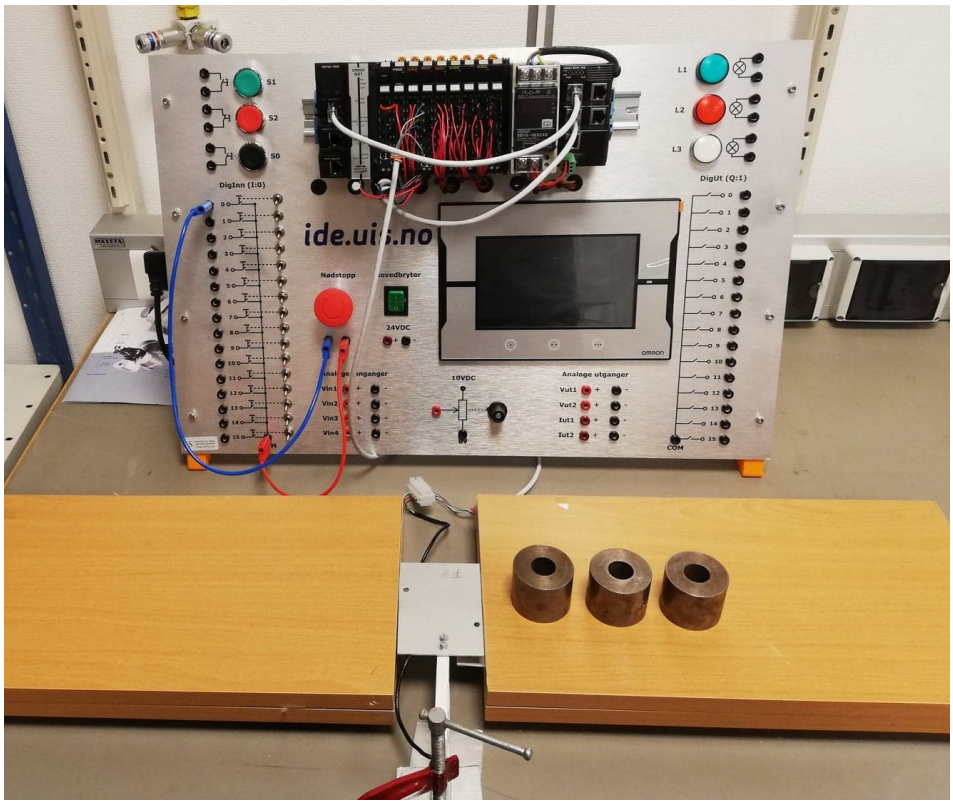
For å forhindre at det kommer flere enn fire flasker under fyllehodene om gangen, brukes det bommer. Først er den fremste bommen (bommen til høyre i figur 2.1) åpen, og den bakre bommen (bommen til venstre i figur 2.1) lukket. Når det har blitt samlet fire flasker, lukker den fremre bommen seg og isolerer de fire flaskene. Videre blir fyllehodene senket ned i flaskene og fyllesylinderne begynner å fylle flaskene. Fyllesylinderne gjør to påfyllinger på 2.5 liter hver, og det kan manuelt justeres hvor mye de skal romme. I det eksisterende systemet er de stilt inn til 2.5 liter. Deretter heves så fyllehodene, og bommen bak åpnes. Flaskene blir så ført videre til korkmontering, merking og deretter pakking. Når sensoren ved bommen bak ikke lenger registrerer flasker, lukkes bakre bom og fremre bom åpnes. Deretter er systemet klar til å ta imot nye flasker.

Produksjonslinjen er utstyrt med manuelle brytere som styrer prosessen, se figur 2.2. Skjermen har ulike funksjonaliteter som blant annet å velge mellom auto- og manuellprogram, regulere hastigheten og lamper som lyser når det skjer enkelte feil i systemet.

Figur 2.2: Brukerpanelet til det eksisterende systemet.

2.2 Maskinvare

I den foreslåtte oppgraderingen av det eksisterende systemet har det blitt brukt en PLS av typen Omron NX102-1000. I tillegg brukes en HMI[1]-skjerm av typen Omron NA5-7W001-V1. Figur 2.3 viser et bilde av arbeidsstasjonen som er brukt for å realisere oppgaven. For mer spesifikk info om NX-102-1000 kan man lese i manualen[3]. For mer info om NA5-7W001-V1 kan man lese i manualen[1].



Figur 2.3: Arbeidsstasjon med kontroller, HMI-skjerm og veiecelle.

[1]HMI - Human-Machine Interface

2.2.1 Oppbygningen av PLS'en

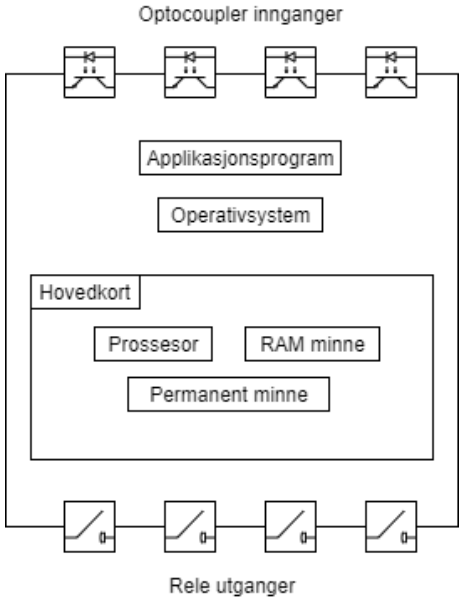
En PLS er en kontroller og står for Programmerbar Logisk Styring. Den kan sammenlignes med en vanlig datamaskin, fordi den er bygd opp av mange av de samme komponentene.

Hovedkomponenter i en PLS:

- Hovedkort
- Prosessor
- Ram(Random accsess memory)
- Permanent minne
- Innganger og utganger

Den største forskjellen mellom en PLS og en datamaskin er at førstnevnte ikke bruker skjerm, tastatur og mus for å kommunisere med omverden. Den kommuniserer i stede med elektriske signaler gjennom innganger og utganger.

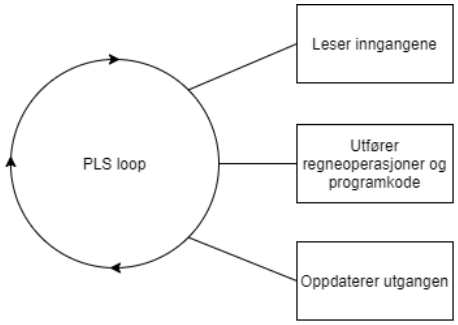
Figur 2.4 viser en typisk oppbygging av en PLS. På inngangene brukes det optokoblere for å beskytte PLS'en mot store spenninger og strømmer. En optokobler består av en lysdiode og en fototransistor. Når et signal går inn på en inngang går det gjennom lysdioden. Det får dioden til å lyse, og som fanges opp av fototransistoren. Når lyset treffer fototransistoren blir det generert en liten strøm som går videre til PLS'en.



Figur 2.4: En typisk oppbygging av en PLS.

På utgangene er det ofte relé- eller transistorutganger. De fleste leverandører tilbyr begge typene. En transistorutgang kan håndtere liten til middels strøm. De fleste transistorutgangene kan kun håndtere likestrøm opp til strømforsyningsspenningen til PLS'en (dvs. 12VDC eller 24VDC). Et relé består av en spole og magnetiske brytere. Når en strøm går igjennom spolen genereres det et magnetfelt som trekker til seg bryterne og endrer bryterkontakten. Fordelen med dette er at man kan bruke en krets til å styre en annen krets som er helt avskilt. Hvis man trenger å styre noe som går på vekselspenning eller noe som bruker mer strøm en det PLS'en kan tilføre, må man bruke reléutgang.

En PLS overvåker ikke inngangene og utgangene kontinuerlig. Den kjører en løkke som er vist i figur 2.5. Den leser først inngangene og oppdaterer disse til en inngangstabell. Deretter kjører den programkoden og eventuelt regneoperasjoner. Ut ifra hva prorammet har gjort oppdaterer den utgangstabellen. I siste steg setter den utgangene. Hvis en inngang endrer seg etter at syklusen har lest inngangene, blir ikke inngangen oppdatert før neste syklus.



Figur 2.5: PLS'ens operasjonssykelus.

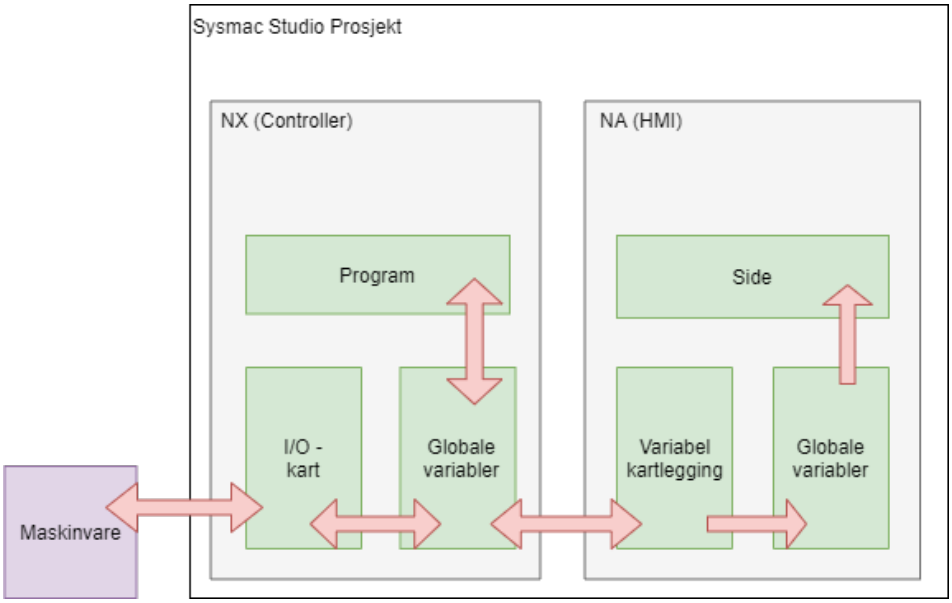
2.3 Programvare

Programvarene som blir brukt er hovedsaklig Sysmac Studio for programmeringen, simuleringen av kontrolleren og brukergrensesnittet. Robotstudio er programvaren som blir brukt til robotarmen.

2.3.1 Sysmac Studio

Sysmac Studio er programvaren som Omron leverer til sine nyeste PLS'er. Det brukes for å sette opp PLS'en, lage programkoden, utforme HMI-skjermen, sette opp OPC-UA og mer. For å installere og sette opp Sysmac Studio kan man følge instruksjonen vist i kapittel 2 i manualen[4].

I Sysmac Studio er det implementert et digitalt brukergrensesnitt som kan kommunisere med kontrolleren. Ved å dele variablene fra kontrolleren med en HMI-applikasjon er det mulig å styre programmet fra skjermen, se figur 2.6. HMI er et grafisk brukergrensesnitt mellom en bruker og en maskin. I industriell sammenheng blir det ofte brukt et dataprogram til å styre en automatisert prosess. Dette dataprogrammet er et grafisk brukergrensesnitt som gjør det mulig for brukeren å samhandle med maskinen gjennom en skjerm. Det vil si at det er mulig å få grafisk informasjon om prosessen og samtidig styre deler av den ved å trykke på ulike programmerbare knapper.



Figur 2.6: Kommunikasjon mellom kontroller og HMI.

OPC-UA

OPC-UA[2] er en universell arkitektur for kommunikasjon mellom industrielle maskiner. Den tillater brukeren å sende informasjon fra en server til en klient via et nettverk. Kommunikasjonen kan for eksempel være mellom en mikrokontroller og en datamaskin.

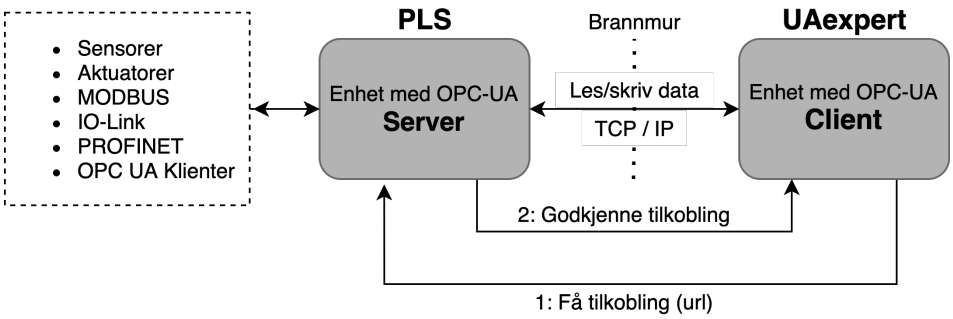
OPC-standarden er en serie spesifikasjoner som er utviklet i samarbeid med leverandører, brukere og programvareutviklere. Standarden definerer grensesnittet mellom klienter og servere, samt servere og servere. Gjennom OPC vil man få tilgang til sanntidsdata, overvåking av alarmer og hendelser, tilgang til historiske data og andre funksjonaliteter.

OPC-UA kom i 2008 og kombinerer de tidligere arkitekturene fra OPC til en felles, som gir flere fordeler. Før OPC-UA måtte klienten bli kjørt på en maskin med Windows OS og flere servere måtte kjøres for å oppnå samme funksjonalitet. Den universelle arkitekturen (UA) gjør kommunikasjonen

[2]OPC-UA - Open Platform Communications United Architecture

mer standardisert og tillater derfor brukeren å benytte andre systemer; som Raspberry Pi og Linux OS. Nettverket er ikke begrenset til en klient per server.

Figuren 2.7 viser en oversikt over et OPC-UA nettverk. Det består av en server og minimum en klient. Serveren er en maskinvare med OPC-UA som for eksempel en datamaskin, en smartkomponent eller i våres tilfelle en PLS. Klienten som skal kommunisere med serveren er som tidligere nevnt svært fleksibel. I denne rapporten blir det brukt en programvare (UA Expert) kjørt på en datamaskin med Windows OS. Ved initialiseringen av kommunikasjonen mellom enhetene må det først bli opprettet en forbindelse(1, se figur 2.7). Dette gjøres ved å gi klienten en URL og ett portnummer fra serveren. Deretter må man opperette ett sertifikat som lager en bruker-id med adgangsnivå og passord(2).



Figur 2.7: Oversikt over kommunikasjonen mellom OPC-UA server og klient.

2.3.2 Robot Studio

Robot Studio er en programvare utviklet av ABB-gruppen, og brukes for programmering og simulering av ABB sine industrielle roboter. Programmet kan brukes uten å være tilkoblet en fysisk robot som tillater brukeren å prøve nye løsninger uten å påvirke produksjonen. En viktig del av Robot Studio er muligheten til å bruke en virtuell kontroller til simulering. Den virtuelle kontrolleren er en eksakt kopi av programvaren som kjører på den fysiske roboten, som sikrer at programmet fungerer i praksis. Det er også mulig å simulere fysiske deler av produksjonslinjen; som sensorer, transportband og laging/fjerning av objekter. Disse delene er kalt smartobjekter, og

er hovedsaklig for visualisering og testing. De blir derfor ikke overført til roboten brukt i selve programmet.

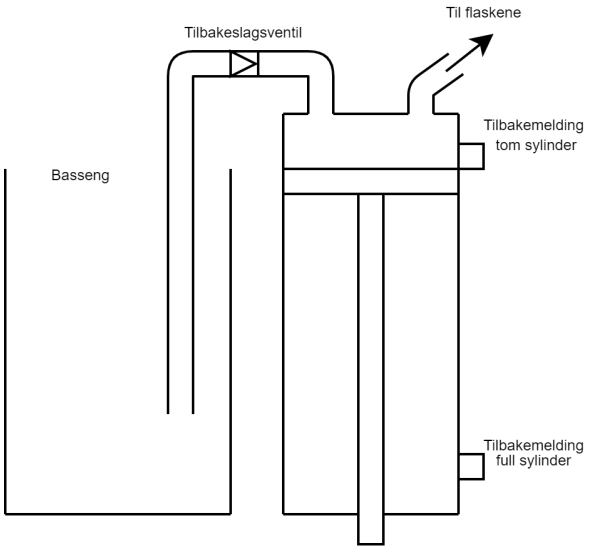
2.3.3 Fusion 360

Fusion 360 blir brukt for å lage CAD-tegninger av komponenter som er brukt i simuleringen i Sysmac Studio.

2.4 Annen relevant bakgrunn

Fyllesylinder

Det er fire fyllesylindere på maskinen. Hver sylinder har til hensikt å fylle hver sin flaske. Mengden som fylles i sylinderen blir manuelt justert. I figur 2.8 finner man en prinsipiell skisse av fyllesylinderen. I enden av røret som går til flaskene er det en ventil som styres av PLS'en. På det røret som går til bassenget er det en tilbakeslagsventil som hindrer væsken i å gå tilbake i bassenget.



Figur 2.8: Prinsipiell skisse av fyllesylinder. Når stemplet går ned dannes det et vakuumtrykk som suger væsken fra bassenget inn i sylinderen. Når stemplet kommer helt ned, kommer tilbakemeldingen full sylinder på. Før stemplet går opp åpnes ventilen på utgangsrøret. Når stemplet går opp fylles flaskene. Tilbakeslagsventilen stopper væsken i å renne tilbake i bassenget.

Kapittel 3

Løsning av eksisterende system

Det er brukt mye tid på å studere nåværende system. Både under bedriftsbesøket, og video som ble tatt av maskinen under besøket. Prosessen skal i hovedsak fungere som før. Løsningen i dette kapittelet er derfor basert på hvordan nåværende system ser ut til å fungere. Videreutvikling av systemet blir først gjennomgått i neste kapittel.

I/O-tabeller og flytskjema er et viktig grunnlag når PLS programmet skal lages. Siden de ikke er tilgjengelige for det eksisterende systemet er de laget ut ifra våre observasjoner.

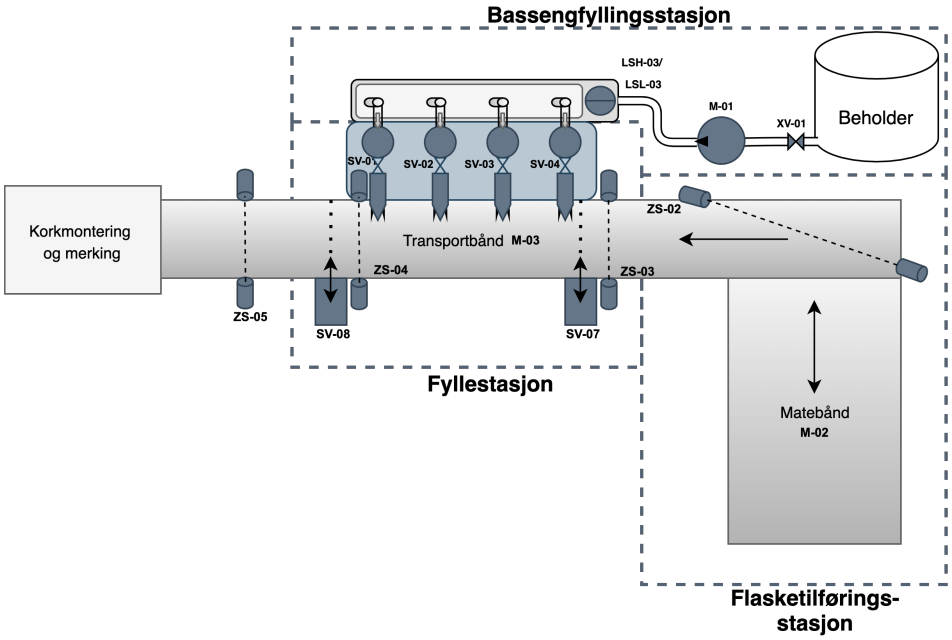
3.1 Overordnet beskrivelse av systemet

I/O-tabellen som har blitt laget finnes under vedlegg B.1. Tabellen viser inngangene og utgangene brukes i systemet.

En illustrasjon av systemet er vist i figur 3.1. Systemet er delt opp i tre hovedprosesser som gjør det enklere å beskrive prosessen videre i oppgaven. Prosessene er ikke direkte avhengig av hverandre og det er derfor laget tilstandsdiagram for hver prosess.

De tre hovedprosessene er som følger:

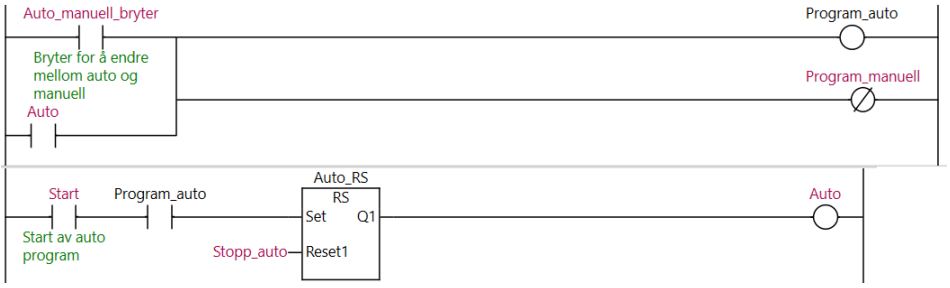
- Bassengfyllingsstasjon
- Flasketilføringsstasjon
- Fyllestasjon



Figur 3.1: Tegning av systemet med variabelnavn.

3.1.1 Start/stopp av program

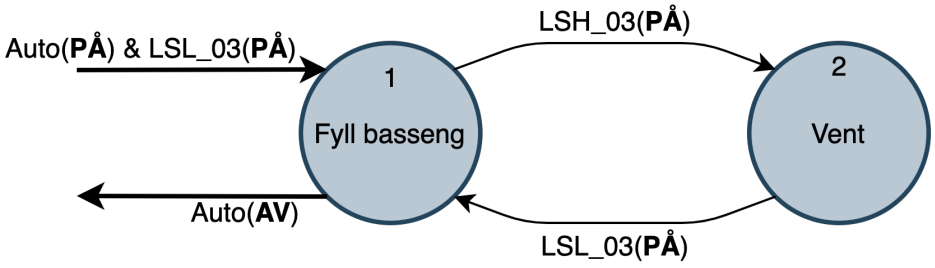
Hensikten med dette programmet er å velge om systemet skal bli kjørt automatisk eller manuelt. Om man velger å kjøre systemet automatisk, kan man trykke på startknappen for å sette *Auto* høy. Når man trykker stopp vil *Auto* gå lav, og systemet stanser.



Figur 3.2: Utdrag av kode. Når bryter *Auto-manuell-bryter* er på, er programmet i auto. Hvis *Auto-manuell-bryter* er av, er programmet i manuell.

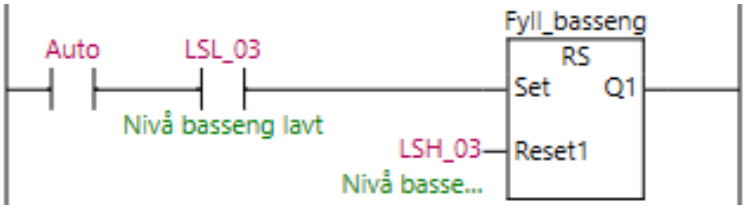
3.1.2 Bassengfyllingsstasjon

Bassengfyllingsstasjon har til hensikt å alltid etterfylle bassenget med ønsket væske. Figur 3.3 viser tilstandsdiagrammet over fyllingen av bassenget. Det er ingen kontinuerlig nivåmåling, men en nivåmåler med to endebrytere montert internt som styrer fylleprosessen. Endebryterne kan justeres ut ifra hvor det er ønsket at høyt og lavt nivå skal være. Tilstandsdiagrammene er nummerert for å enklere referere til dem.



Figur 3.3: Tilstandsdiagram over fylling av basseng.

- For at sekvensen skal starte må *Auto*-programmet være aktivert[1]. Når nivået i bassenget treffer den nedre endebryteren *(LSL-03)*[2] åpnes en ventil og en pumpe starter.
- Når nivået treffer den øvre endebryteren(*LSH-03*)[3] stopper pumpen og ventilen lukkes. Sekvensen vil fortsette så lenge *Auto* er aktiv.



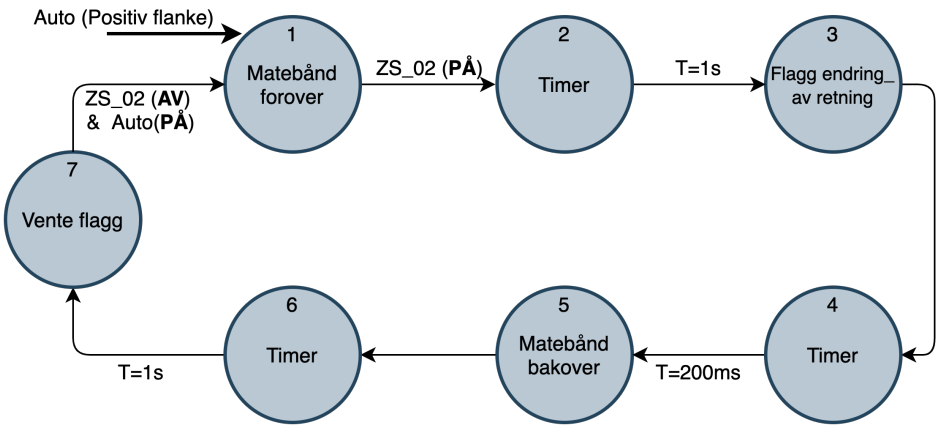
Figur 3.4: Utdrag av koden som regulerer bassengfyllingen.

Figur 3.5: Utdrag av program som styrer utgangene for pumpen og ventilen.

[1] **PÅ** - variabelen settes høy, **AV** - variabelen settes lav
[2] LSL - Limit Switch Low
[3] LSH - Limit Switch High

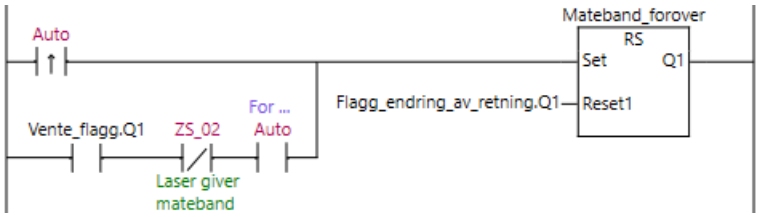
3.1.3 Flasketilføringsstasjon

Flasketilføringsstasjonen har til hensikt å levere flasker til fyllestasjonen. Dette blir i dag utført ved at et stort matebånd fører fire flasker om gangen over på et transportbånd. En sensor(*ZS-02*)[4] brukes for å registrere flaskene. Transportbåndet fører flaskene fra matebåndet til fyllingen, og videre til korkingmontering og merking. Transportbåndet går så lenge prosessen er aktiv. Figur 3.6 illustrerer sekvensen til matebåndet.



Figur 3.6: Tilstandsdiagram av sekvensen til matebåndet.

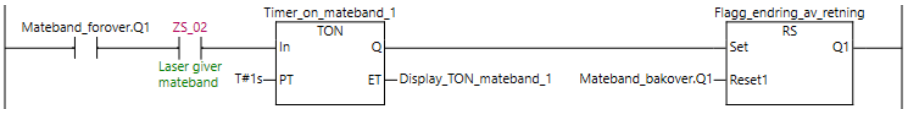
- Når *Auto*-programmet aktiveres går programmet i tilstand 1. Matebåndet begynner dermed å føre flakser inn på transportbåndet.



Figur 3.7: Utdrag av koden som starter matebåndet.

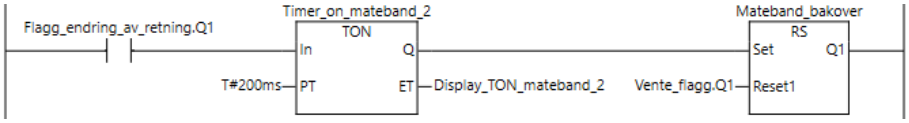
[4]ZS - Proximity Sensor

- Det tidspunktet flaskene blir registrert av sensor *ZS-02* vil sensoren gå høy og en timer på 1 sekund starter(tilstand 2). Dette er for å sikre at flasken blir ført fullstendig over til transportbåndet. Når timeren er ferdig vil prosessen gå over i tilstand 3 som stopper matebåndet.



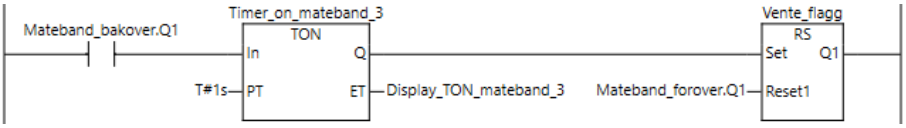
Figur 3.8: Utdrag av koden som stopper matebåndet.

- Programmet går dermed i tilstand 4 og en timer på 200ms starter. Dette blir gjort for å unngå at motoren går direkte i revers. Motoren vil få en stor belastning når dreieretnigne endres og vil derfor unngås.



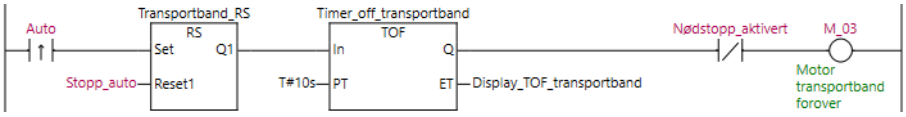
Figur 3.9: Utdrag av koden som starter matbåndet i revers.

- Når timeren er ferdig vil prosessen gå i tilstand 5 og matebåndet begynner å gå bakover. Tilstand 6 går høy og en timer på 1 sekund starter.



Figur 3.10: Utdrag av kode som stopper matebåndet fra å bli kjørt i revers etter 1 sekund.

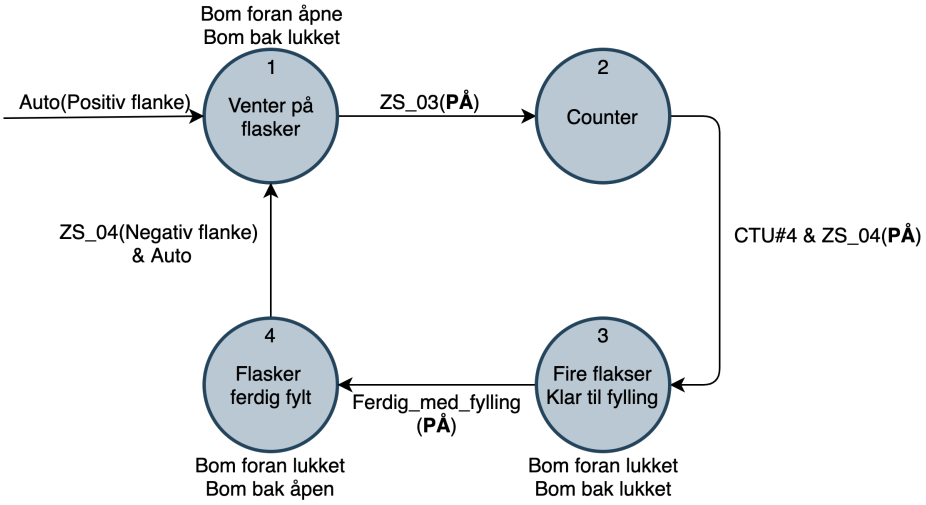
- Når timeren har kjørt ferdig vil prosessen være i tilstand 7 hvor motoren står i ro mens den venter på at sensor *ZS-02* blir lav. Når sensoren går lav er transportbåndet klar til ny tilføring av flasker og sekvensen starter forfra.



Figur 3.11: Utdrag av koden som styrer transportbåndet. Det går forover så lenge *Auto* er aktiv, og fortsetter å gå forover i 10s etter *Auto* har blitt slått av.

3.1.4 Fyllestasjon

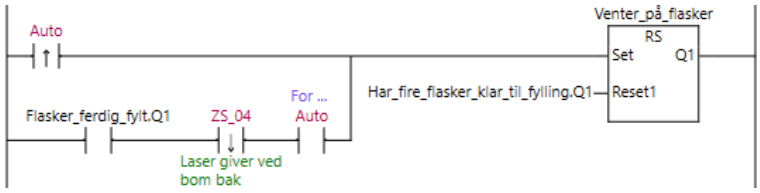
Fyllestasjon har til hensikt å isolere fire flasker, for så å fylle de med ønsket innhold. Fyllestasjonen består av to prosesser, bomstyring og fylling. Sekvensene er illustrert henholdsvis i figur 3.12 og 3.17.



Figur 3.12: Tilstandsdiagram av bomstyringen.

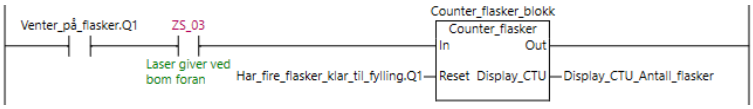
Bomstyringssekvensen skjer på følgende måte:

- Når *Auto* aktiveres går programmet i tilstand 1. I denne tilstanden er fremre bom åpen, mens bakre bom er lukket. Tilstanden forblir uendret frem til sensor *ZS-03*(sensor ved fremre bom) blir høy.



Figur 3.13: Utdrag av kode som setter tilstanden til 1 i flytskjema.

- Når sensor *ZS-03* kommer inn, begynner *Counter*-blokken å telle (tilstand 2). Måten denne blokken fungerer er beskrevet i delkapittel 3.2.1.

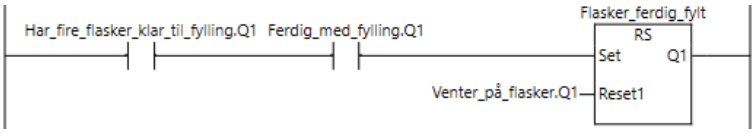


Figur 3.14: Utdrag av koden som teller flasker. *Counter-flaske-blokk* er en egendefinert funksjon som teller antall flasker som passerer *ZS-03*.

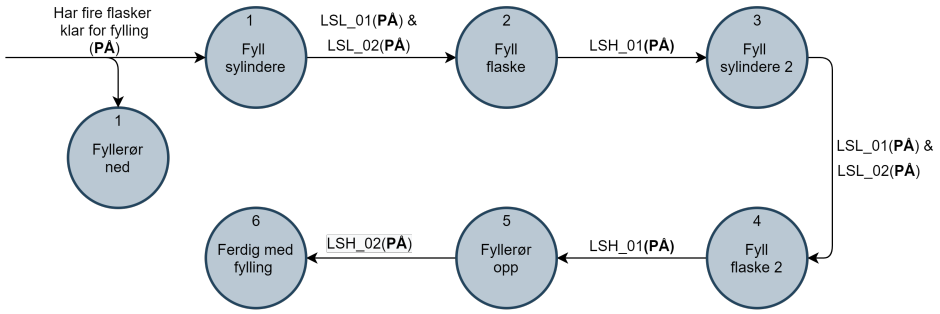
- Når *Counter*-blokken har telt fire flasker og sensor *ZS-04* (sensor ved bakre bom) har blitt høy, går programmet videre til tilstand 3. I denne tilstanden er begge bommen lukket. *Fire-flasker-klar-til-fylling* blir satt høy som indikerer at fire flasker er isolert og fyllesekvesen kan starte.

Figur 3.15: Utdrag av kode. *Har-fire-flasker-klar-til-fylling* er tilstanden som starter fyllesekvensen.

- Når fyllesekvensen er ferdig går variabelen *Ferdig-med-fylling* høy og bomsekvensen videre i tilstand 4. I denne tilstanden er bommen foran lukket, mens bommen bak er åpen. Når alle flaskene har passert *ZS-04* starter sekvensen på nytt.



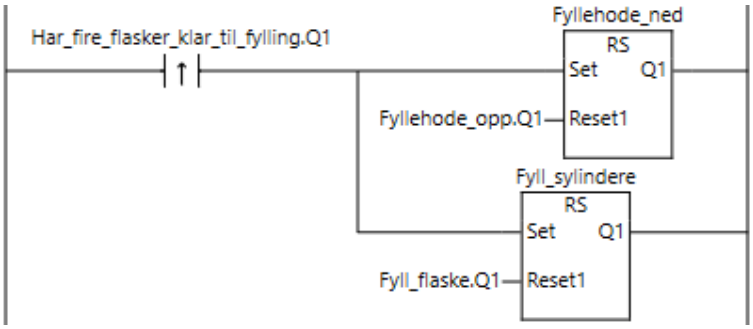
Figur 3.16: Utdrag av kode som viser når flaskene er ferdig fylt.



Figur 3.17: Tilstandsdiagram over fyllesekvensen

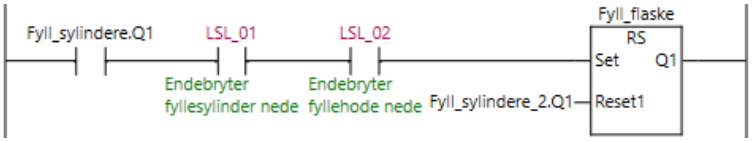
Flaskefyllingsekvensen skjer på følgende måte:

- For at denne sekvensen skal begynne må variabelen *Har-fire-flasker-klar-til-fylling* være høy. Når denne betingelsen er oppfylt går fyllerørene ned i flasken, samtidig som den begynner å fylle sylinderne (tilstand 1).



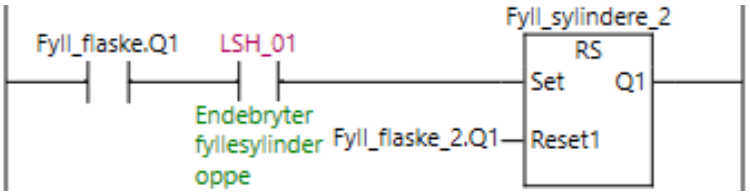
Figur 3.18: Utdrag av kode for tilstand 1.

- Når endebryteren *LSL-02* blir høy, befinner fyllerørene seg i flaskene og er klare til påfylling. I tillegg må endebryteren *LSL-01* som viser at sylinderne er fulle, bli høy. Når dette skjer går den over i tilstand 2 og begynner å fylle flaskene.



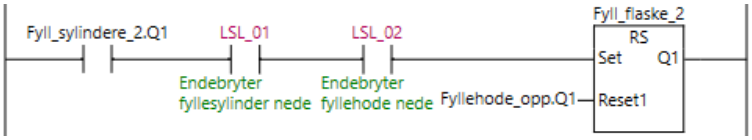
Figur 3.19: Utdrag av kode som viser hvilke variabler som må være aktiv for å starte fylling av flasker.

- Når sylinderne er tomme vil *LSH-01* bli høy. Sekvensen går da videre i tilstand 3 hvor den fyllingen av sylinderene starter på nytt.



Figur 3.20: Utdrag av kode for tilstand 3.

- I tilstand 3 forblir fyllerørene i samme posisjon som fra tilstand 2. Når sylinderene er fulle blir *LSL-01* høy og sekvensen går videre til tilstand 4. Her begynner fyllingen av flaskene for andre gang.

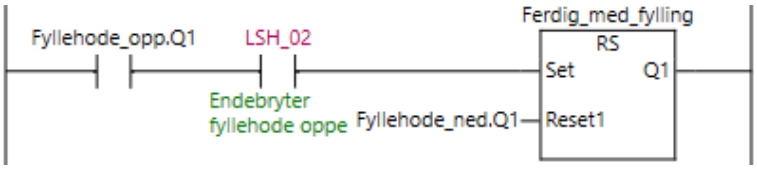


Figur 3.21: Utdrag av kode som viser hvilke variabler som må være aktiv for å starte fylling av flasker for andre gang.

- Når sylinderne er tomme for andre gang vil *LSH-01* bli høy igjen. Sekvensen går da til tilstand 5 og fyllerørene går opp.

Figur 3.22: Utdrag av kode som får fyllerørene til å gå opp.

- Når fyllerørene har kommet helt opp vil *LSH-02* bli høy. Fyllingen er da ferdig og variabelen *Ferdig-med-fylling* settes høy. Sekvensen er da over og begynner forfra når systemet har fire nye flasker til fylling.



Figur 3.23: Utdrag av kode som ferdigstiller sekvensen.

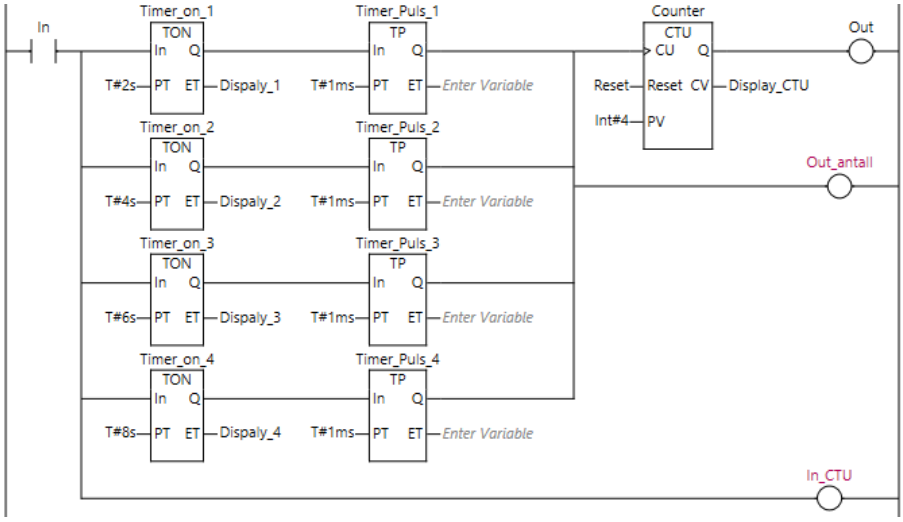
3.2 Tilleggsprogrammer

3.2.1 Counter blokk

Det er antatt at en flaske bruker 2 sekunder på å passere en nærhetssensor. Dette er en tid som må måles i et ekte anlegg, da den vil variere utifra flaske type, fart på bånd, og andre faktorer.

Counter blokken er vist i figur 3.24. Den er bygd opp av fire TON[5]-blokker på 2,4,6 og 8 sekunder. Disse fire TON-blokkene har hver sin puls-blokk koblet på seg. Til slutt er det en CTU[6]-blokk som teller alle pulsene.

[5]TON - On-Delay Timer
[6]CTU - Counter Uppwards

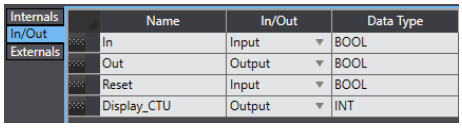


Figur 3.24: Oppbyggning av Counter blokk

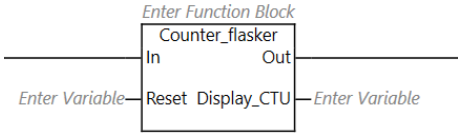
Virkemåten til funksjonsblokken kan best forklares med følgende eksempler:

Eksempel 1: Hvis én flaske kommer vil øverste TON-blokken aktiveres som får CTU-blokken til å telle en puls.

Eksempel 2: Hvis det kommer fire flasker vil den første TON-blokken aktiveres etter 2 sekund, den andre etter 4, og videre. Hver av disse sender en puls som CTU-blokken teller. Når den har nådd fire flasker sette utgangen høy.



Figur 3.25: Alle innganger og utganger som er synlig fra utsiden av blokken.



Figur 3.26: Hvordan Counter blokk ser ut i Sysmac Studio.

Kapittel 4

Videreutvikling av systemet

I forrige kapittel ble det gjennomgått hvordan sekvensene i systemet kan bli realisert. Videre vil det bli tatt for seg mulige videreutviklinger av systemet. Følgende oppgraderinger blir foreslått:

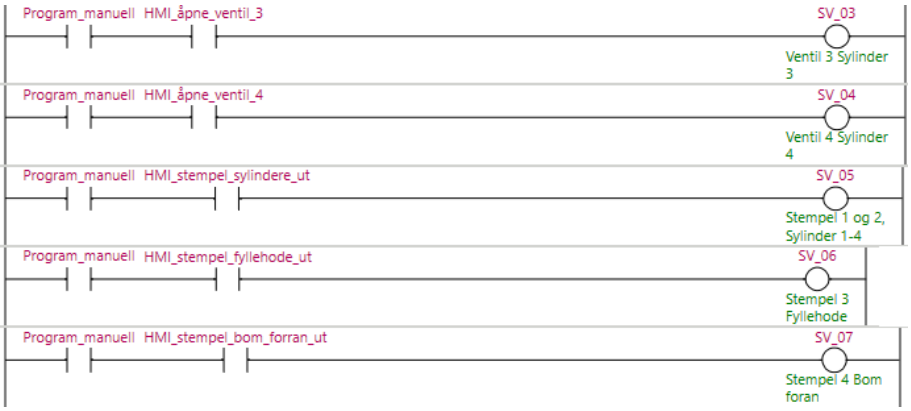
- Manuell kjøring av systemet
- Kontroll på antall fylte flasker
- Alarmer ved feil i systemet
- Kontroll av påfyllingsmengde
- OPC-UA kommunikasjon til en PC
- Digitalt brukergrensesnitt
- Automatisert pakking av flasker

4.1 Nye funksjonaliteter for PLS

I denne seksjonen blir det gjennomgått nye funksjonaliteter som er mulig å realisere ved bruk av Sysmac Studio.

4.1.1 Manuell

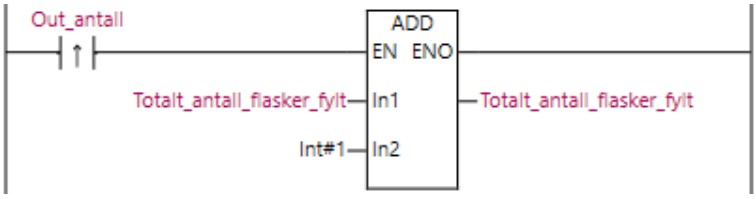
Det manuelle programmet er laget for å enkelt kunne feilsøke og teste komponenter. Figur 4.1 viser et utdrag av det manuelle programmet. Så lenge *Program_manuell* er aktiv, kan man aktivere de forskjellige utgangen med å trykke på knappene i brukergrensesnittet. Den relevante skjermen fra brukergrensesnittet er vist i figur 4.19.



Figur 4.1: Utdrag av manuell program

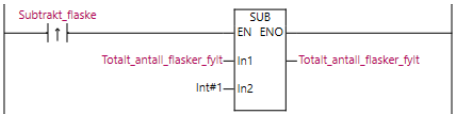
4.1.2 Flasketelling

Programmet i figur 4.2 holder tellingen på antall flasker som er fylt. Antall fylte flasker vises til operatørene via brukergrensesnittet og sendes til en PC via OPC-UA.

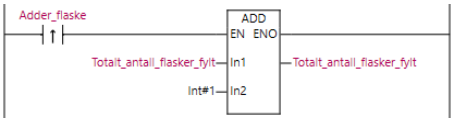


Figur 4.2: Hver gang *Out_ antall* fra *Counter*-blokk(3.2.1) får en høy flanke legger blokken til +1 i variablen *Toltalt_ antall_ flasker_ fylt*.

På hovedsiden av brukergrensesnittet er det et minustegn og et plusstegn ved siden av totalt antall flasker, se figur 4.18. Ved å trykke på minustegnet trekker man fra -1 i variabelen *Toltalt_ antall_ flasker_ fylt*, og plusstegnet vil legge til +1 i variabelen.

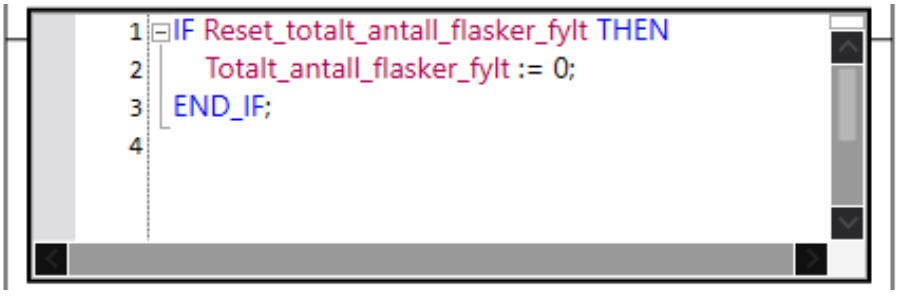


Figur 4.3: Trekker fra flasker



Figur 4.4: Legger til flasker

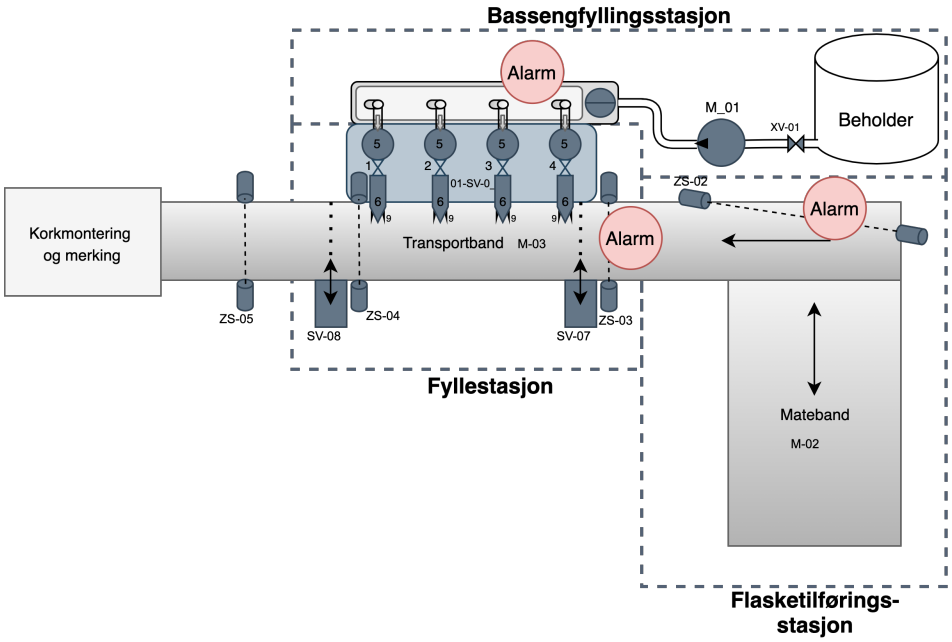
Reset_ totalt_ antall_ flasker_ fylt er definert i strukturert tekst. Når den går høy vil variabelen *Totalt_ antall_ flasker_ fylt* bli satt til 0, se figur 4.5.



Figur 4.5: Reseter variabelen til 0.

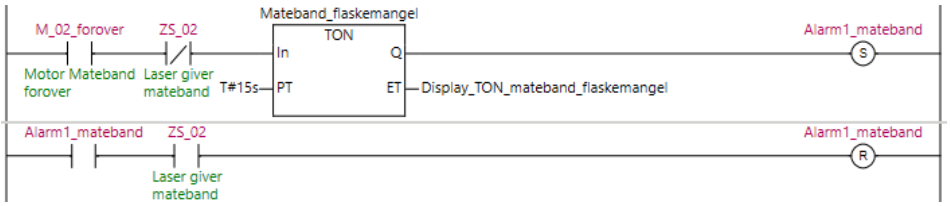
4.1.3 Alarmer

Alarmer er viktig å ha med i et system. Hvis det oppstår en feil mens programmet kjører vil en alarm gjøre operatøren oppmerksom på problemet. Alle alarmene er koblet opp mot brukergrensesnittet, samt tatt med i en alarmlogg. Alarmloggen er vist i figur 4.21. En overordnet illustrasjon over alarmene er vist i figur 4.6.



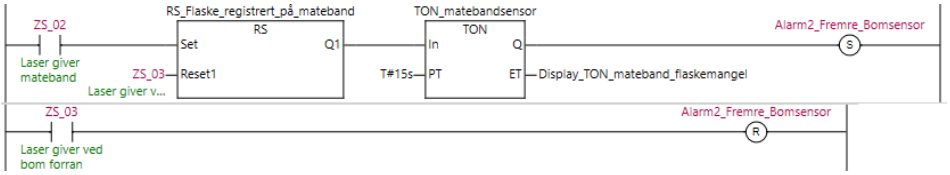
Figur 4.6: Figur som viser systemet og hvor alarmene blir aktive.

Alarmen som er vist i figur 4.7 har som hensikt å opplyse brukeren om at det ikke er registrert flasker på matebåndet. Alarmen blir satt høy etter 15 sekunder og blir resatt når en flaske er detektert. Det er verdt å merke seg at alarmen vil gå høy ved sensorfeil eller om flasker ikke blir tilført i tide.



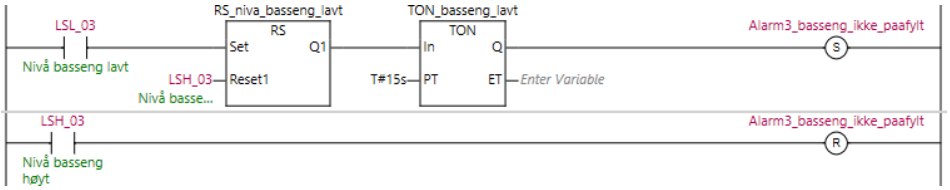
Figur 4.7: Utdrag av kode for alarm på matebånd.

Figur 4.8 viser en alarm som aktiveres dersom flaskene bruker lengre tid enn 5 sekunder fra sensoren på matebåndet til fremre sensor.



Figur 4.8: Utdrag av kode for alarm for sensor ved fremre bom.

Hvis bassenget har lite væskeinnhold skal en pumpe starte. Pumpen er antatt å fylle bassenget på under 15 sekunder. Hvis bassenget ikke er fylt etter de 15 sekundene skal en alarm settes høy, som forteller operatøren at bassenget ikke blir påfylt som forventet. Mulige årsaker til at alarmen aktiveres er tom beholder, feil på pumpen eller sensorfeil. Utdrag for implementeringen av alarmen er vist i figur 4.9.



Figur 4.9: Utdrag av kode for alarm til bassengfylling.

4.1.4 Kontroll av påfyllingsmengde

Et ønske fra bedriften var å finne en løsning for å kontrollere at det ble fylt riktig mengde i flaskene. I seksjon 5.3 er det diskutert hvorfor det er valgt å bruke en veiecelle til å måle påfyllingsmengden.

Per dags dato er det ingen måling på mengde fylt på flaskene, men det er mulig å justere inn ønsket mengde manuelt på fyllesylinderen. Siden det ikke er noe måling er det ingen mulighet for å sjekket at ønsket mengde faktisk er fylt.

Det ble brukt en veiecelle av typen AL6B i denne rapporten. Det er ikke tenkt at denne skal brukes av bedriften, men kun for å vise at prinsippet fungerer. Veiecellen er koblet til et modulskort av typen NX-RS1201. Hvordan veiecellen blir satt opp er forklart i manual[7] og brukermanualen for modulskortet er i manual[8].

Modulskortet har innebygde filter for å filtrere signalet fra veiecellen. Det er ett lavpass-filter og to MA[1]-filter.

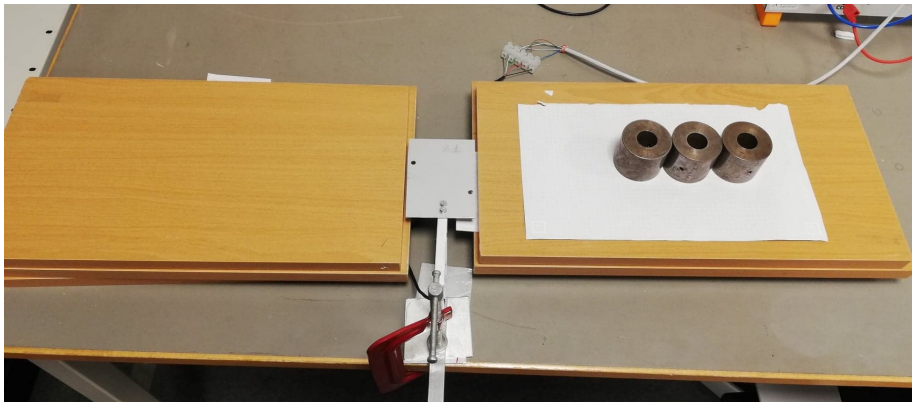
Digital Low- pass Filter Cutoff Frequency/...	80	×0.1Hz
Filter 1 Moving Average Count/Ch1 Filter...	160	times
Filter 2 Moving Average Count/Ch1 Filter...	133	times

Figur 4.10: Filter verdiene som ble brukt. Disse er også standard-verdiene når man setter opp modulskortet.

Det er også en innebygd kalibreringsfunksjon i modulkortet. Den kan brukes til å kalibrere vekten gjennom Sysmac Studio. I manual[8] s.140-168 er det beskrevet hvordan man kalibrer veiecellen med bruk av ladder kode. Koden som er vist i manualen blir brukt i vår løsning.

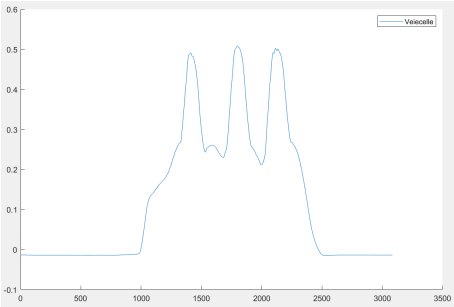
For å representere flaskene ble det brukt tre sylindere av jern, med en vekt på omtrent 500 gram. Det ble brukt et ark for å representere transportbåndet. Jernsylinderene ble lagt på arket og ført over veiecellen. Oppsettet er vist i figur 4.11.

[1] MA - Moving Average

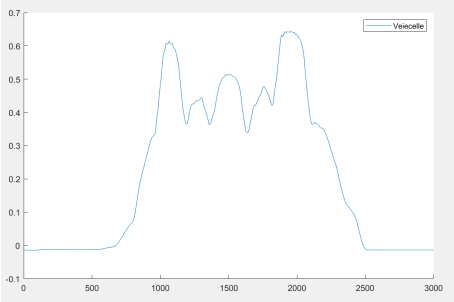


Figur 4.11: Oppsett av veiecelle på lab.

I figur 4.12 og 4.13 er det vist hvordan lasten på veiecellen varierer mens arket med jernsylinderene blir dratt over veiecellen. Figur 4.12 viser hvordan det vil se ut når alle flaskene har samme innhold. I figur 4.13 er det vist hvordan det vil se ut når den fremste og bakre flasken er full, mens flasken i midten mangler 100g (tilnærmet 1dL).



Figur 4.12: Graf over last på veiecelle. Alle tre jernsylinderene er ca. 500g.



Figur 4.13: Graf over last på veiecelle. To av jernsylinderene er ca. 600g, mens den ene er ca. 500g.

Det hadde vært ønskelig å ha et program for å detektere avvik i toppunkter på grafen. Dette blir ikke gjort i denne oppgaven. Derfor er det heller ikke laget et program for å varsle om en eller flere flasker ikke har ønsket mengde væske. Hvorfor dette ikke er laget er forklart mer i seksjon 5.3.

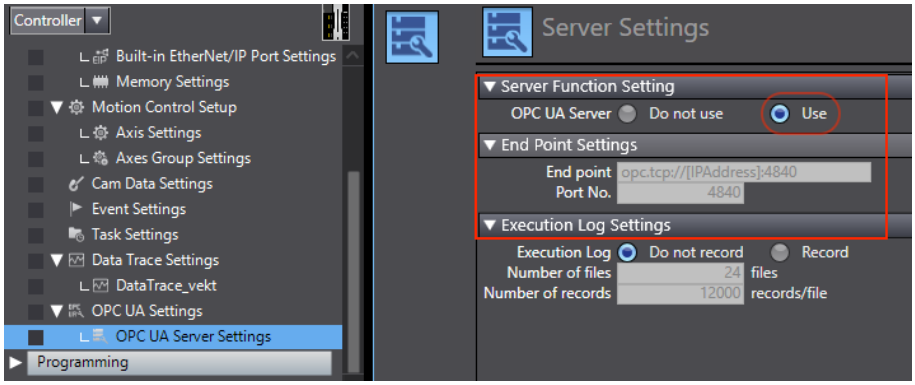
4.2 Bruk av OPC-UA

Kontrolleren som blir brukt i denne oppgaven er som nevnt tidligere kompatibel med OPC-UA. Ved å sette opp kontrolleren som en OPC-UA server kan man kommunisere med kontrolleren via en klient. Illustrasjon av oppsettet er vist i figur 2.7.

4.2.1 Oppsett av OPC-UA server

Oppsettet av serveren gjøres i Sysmac Studio. Fullstendig manual over oppsettet finner man i manual[5] på side 43, kapittel 3.

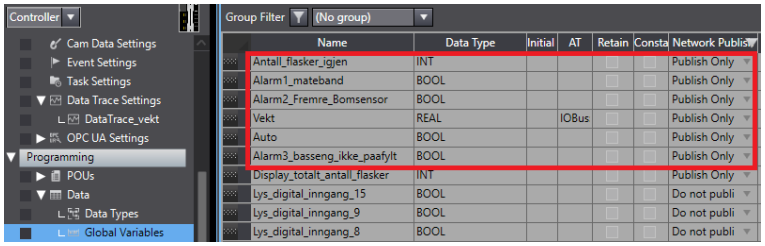
For å sette opp serveren må den først aktiveres under kontroller-innstillinger i Sysmac Studio. Dette er vist i figur 4.14. For å unngå at uønskede personer skal få tilgang til serveren burde brukernavn og passord opprettes. Dette gjøres ved å høyreklikke på <OPC UA Server Settings> og gå inn på <Security Settings>. Deretter velger man brukernavn og passord, samt krypteringsgrad. For å teste at serveren fungerer kan serverens status bli funnet ved å høyreklikke på <OPC UA Server Settings>, deretter <Server Status>.



Figur 4.14: Server-innstillinger for kontroller i Sysmac Studio.
End pointer serverens url og portnummer som klienten må koble til.

4.2.2 UaExpert - OPC-UA klient

Når serveren er aktiv kan klienten kobles til. I denne rapporten blir programmet UaExpert brukt. Installasjon og oppsett er vist i video[6]. Det er viktig å forsikre seg om at riktig server-url er brukt. Server-url er navngitt 'End point' og finnes på <Server Status> eller under server instillinger som vist i figur 4.14. Hvis man oppretter bruker og passord i Sysmac Studio må man bruke det samme i UaExpert. Når klienten er tilkoblet serveren vil den tilgjengelige informasjonen fra kontrolleren komme opp under <Adresse Space>. For å gi klienten mulighet til å lese variabler må disse gjøres tilgjengelige. Tilgangen settes enten som 'Input', 'Output' eller 'Publish Only'. I dette tilfelle er det ikke ønskelig at prosessen kan bli styrt fra andre steder enn produksjonsområdet, derfor settes variablene til 'Publish Only'. Variablene som er gjort tilgjengelig er vist i figur 4.15.



Figur 4.15: Globale variabler som har blitt gjort tilgjengelige for OPC-UA klienten.

Etter man har koblet til serveren på nytt vil de nye variablene komme opp i mappen <Controller/GlobalVars>. For å se verdien på variablene må de flyttes inn i <Data Access View>-vinduet. Det er også mulig å se annen informasjon som statussen og modussen til kontrolleren. Disse finner man under <Devise Status>-mappen i <Address Space>. Overvåkningsvinduet til klienten er vist i figur 4.16.

Data Access View

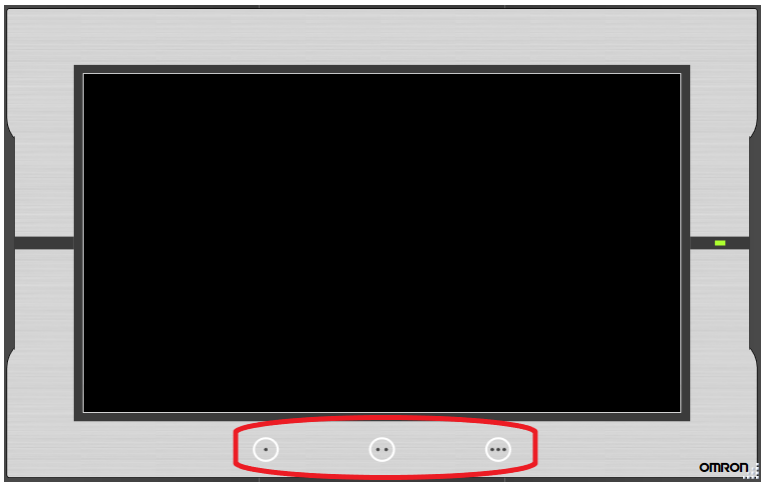
#	Server	Display Name	Value	Node Id	Datatype
1	NxOpcUaSe...	Mode	RUN	NS4|String|DeviceStatus.Mode	String
2	NxOpcUaSe...	ErrorStatus	NoError	NS4|String|DeviceStatus.ErrorStatus	String
3	NxOpcUaSe...	NumOfValues	7	NS4|String|NumOfValues	Int32
4	NxOpcUaSe...	Alarm1_mateband	false	NS4|String|Alarm1_mateband	Boolean
5	NxOpcUaSe...	Alarm2_Fremre_Bomsensor	false	NS4|String|Alarm2_Fremre_Bomse...	Boolean
6	NxOpcUaSe...	Alarm3_bassenq_ikke_paafylt	false	NS4|String|Alarm3_bassenq_ikke_p...	Boolean
7	NxOpcUaSe...	Auto	true	NS4|String|Auto	Boolean
8	NxOpcUaSe...	Display_totalt_antall_flasker	6	NS4|String|Display_totalt_antall_fla...	Int16
9	NxOpcUaSe...	Vekt	-0.0125985	NS4|String|Vekt	Float
10	NxOpcUaSe...	Antall_flasker_igjen	4	NS4|String|Antall_flasker_igjen	Int16

Figur 4.16: Variabler som blir overvåket av OPC-UA klienten. Skjermbildet er tatt i UaExpert.

4.3 Beskrivelse av HMI-program

I dag er produksjonslinjen utstyrt med et manuelt brukergrensesnitt, som vist i figur 2.2. Ved å implementere et digitalt brukergrensesnitt (HMI) vil operatøren ha tilgang til mer detaljert informasjon. Dette resulterer i et brukergrensesnitt som er mer brukervennlig og enklere å tilpasse. Ved å bruke en digital HMI kan man tilpasse skjermen med spesifikke funksjonaliteter som for eksempel forandring av ordre eller alarmer. Hvis det oppstår en feil i programmet vil det komme alarmer opp på skjermen. Det er også mulig å se hvor i prosessen programmet er ved å implementere bit-lamper som lyser når tilstanden er aktiv. Skjermen som er brukt i denne rapporten er NA5 som er en del av NA-serien som er implementert i Sysmac Studio.

HMI-skjermen har tre funksjonstaster som kan programmeres, som vist i figur 4.17. Den første tasten ble programmert til å gå tilbake til forrige side. Neste tast blir brukt til å komme til hovedsiden, og den siste tasten brukes til å logge seg inn som en administrativ bruker. Måten funksjonstastene er programmert vises på side 151 i manualen til HMI'en [1].

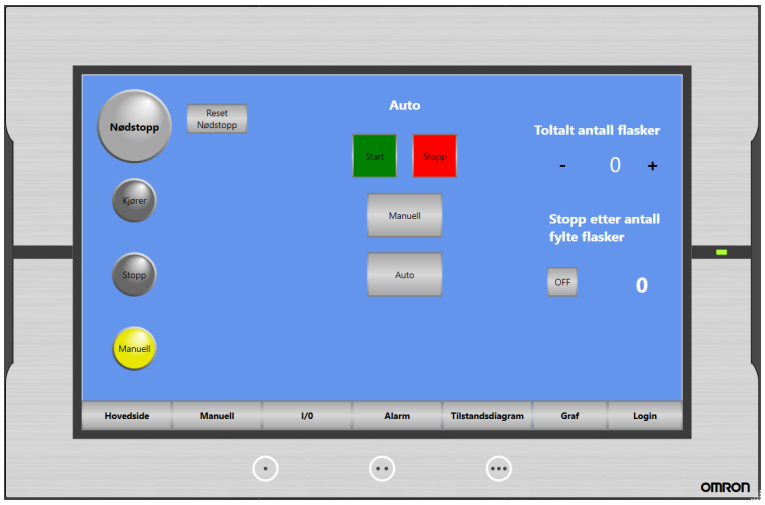


Figur 4.17: Programmerbare knapper på skjermen.

4.3 Beskrivelse av HMI-program

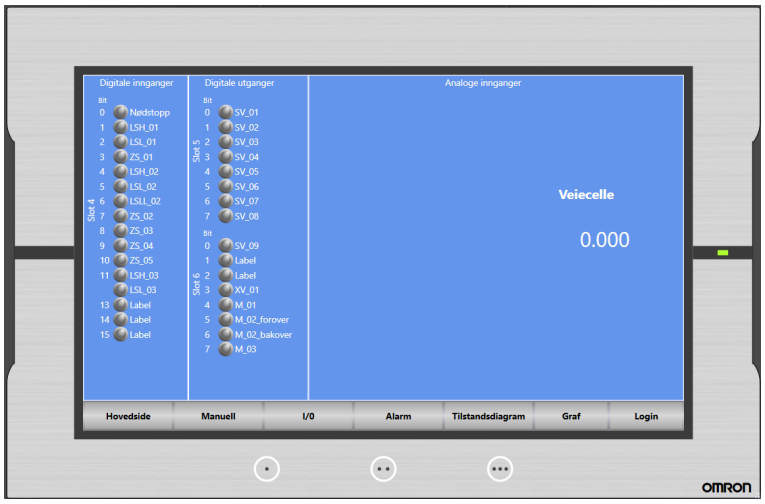
I figur 4.18 vises hovedsiden til HMI-programmet. Dette vil være den første siden som dukker opp på skjermen. Alle rektangulære figurer er knapper som setter en variabel høy, mens runde figurer er bitlamper som viser når en tilstand er aktiv. Nederst på skjermen er det lagt inn knapper som skifter til de ulike sidene.

Videre blir sidene til HMI-skjermen vist, med en kort forklaring i figurtekst.

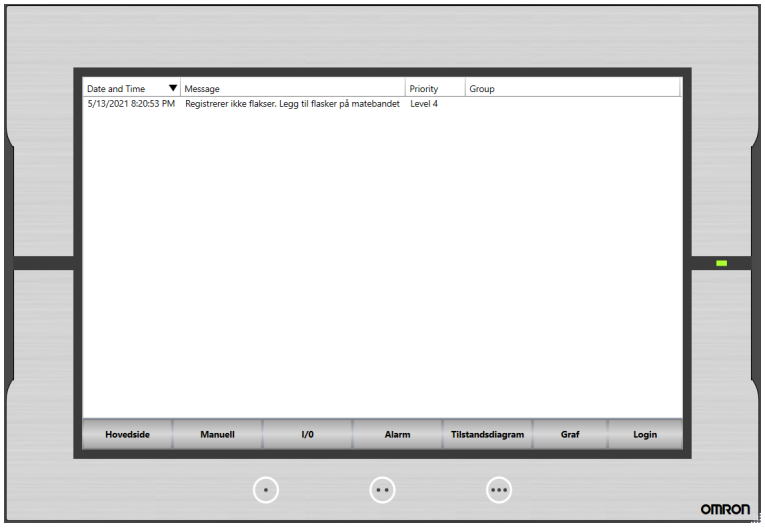


Figur 4.18: Brukerene kan velge om de ønsker å kjøre programmet i auto eller manuell. Bit-lamper viser hvilken tilstand programmet er i og når programmet kjører i auto. Ved nødstopp vil en bitlampe blinke til den er nullstilt. Det er mulig å stille inn antall flasker som skal fylles, og programmet vil stoppe når det er fullført. Hvis flasker går i svinn kan de trekkes fra med minustegnet på totaltantall flasker. Plusstegnet legger til en flaske i antallet om det er ønskelig.

Figur 4.19: Denne siden brukes til manuellkjøring. Den grønne bit-lampen viser at programmet står i manuell. Hensikten er å kunne testkjøre og feilsøke enkelte komponenter.

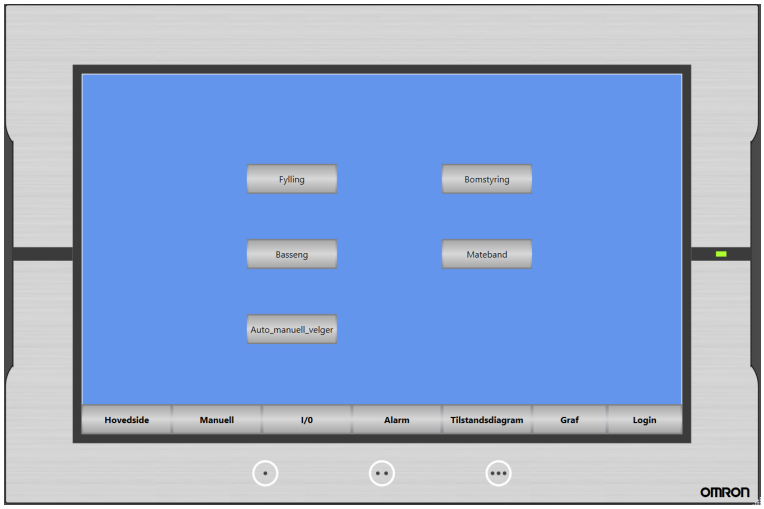


Figur 4.20: Alle digitale innganger og utganger er koblet opp mot en bitlampe som lyser når de er høye. Veiecellen er en analog inngang. På skjermen vises belastningen på veiecellen til en hver tid.

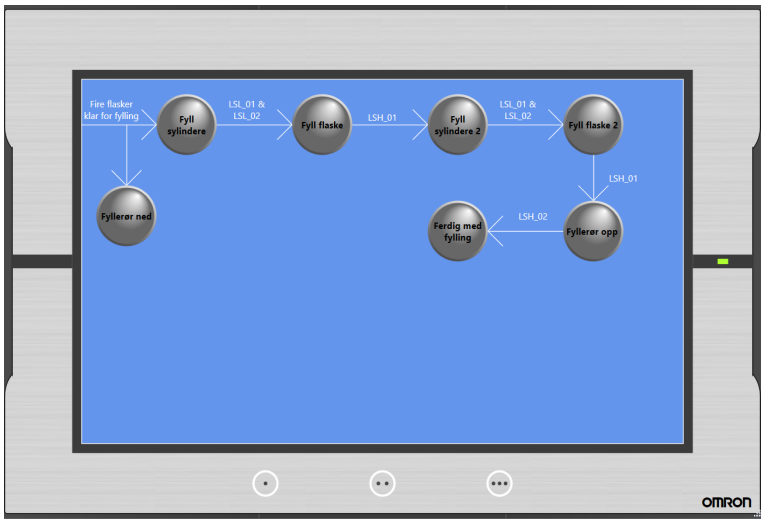


Figur 4.21: Her logges alamer som har skjedd. Figuren viser alarm på matebåndet, se figur 4.6.

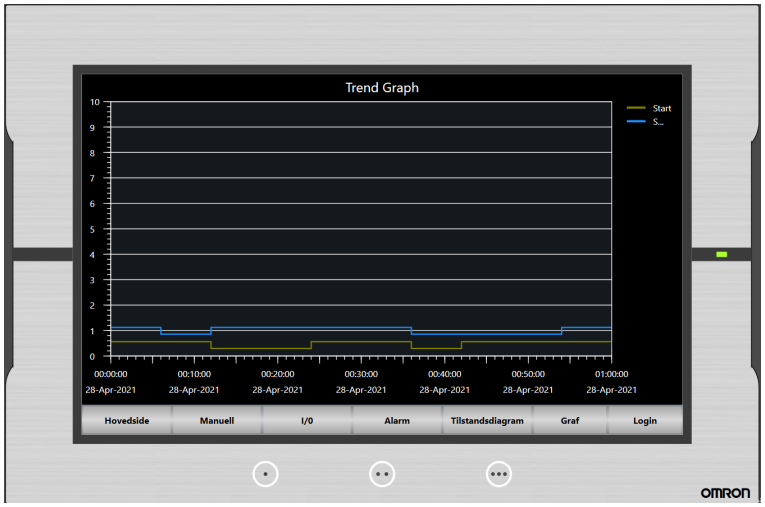
Figur 4.22: Alvorlige alarmer vil dukke opp som en pop-up skjerm uavhengig av hvilken side som vises.



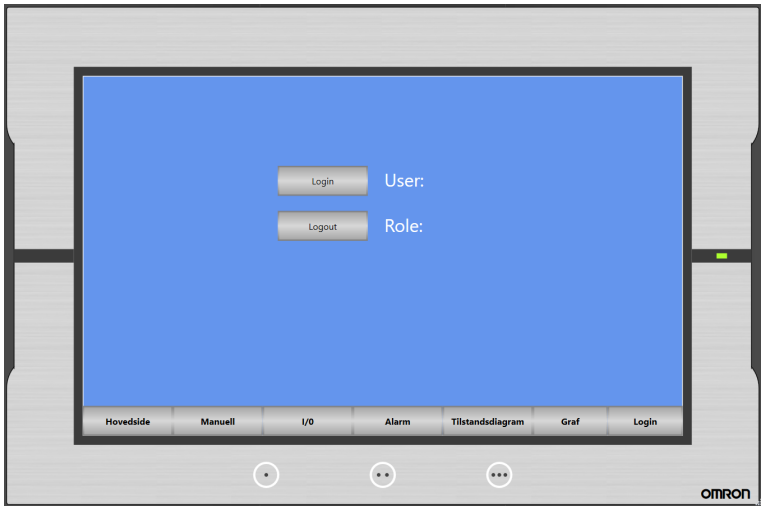
Figur 4.23: Tilstandsdiagramsiden viser alle de ulike prosessene som skjer i programmet. Det er mulig å trykke inn på de forskjellige og få opp det relevante tilstandsdiagrammet, som vist for fylleprogrammet i figur 4.24.



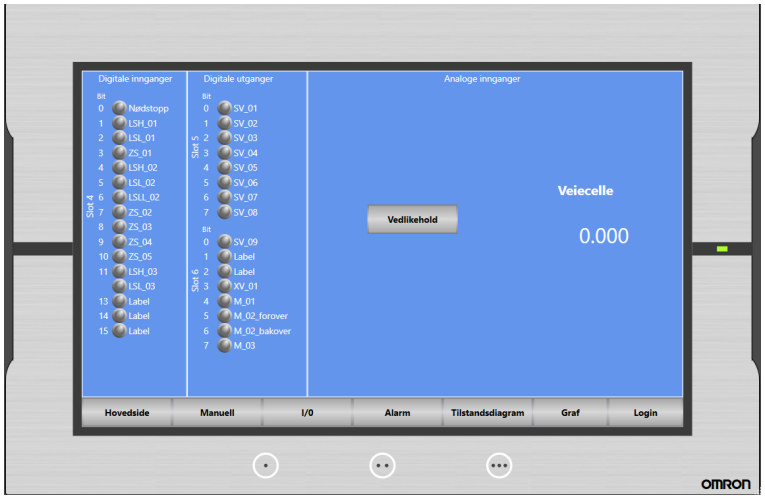
Figur 4.24: Tilstandene er koblet til en bitlampe. Når de er høye vil de lyse grønt. Det er beskrevet hvilke betingelser som skal til for at neste tilstand skal bli aktiv. Ved å ha dette på HMI-skjermen vil det være lettere å se hvor i prosessen man er og feilsøke om det har stoppet.



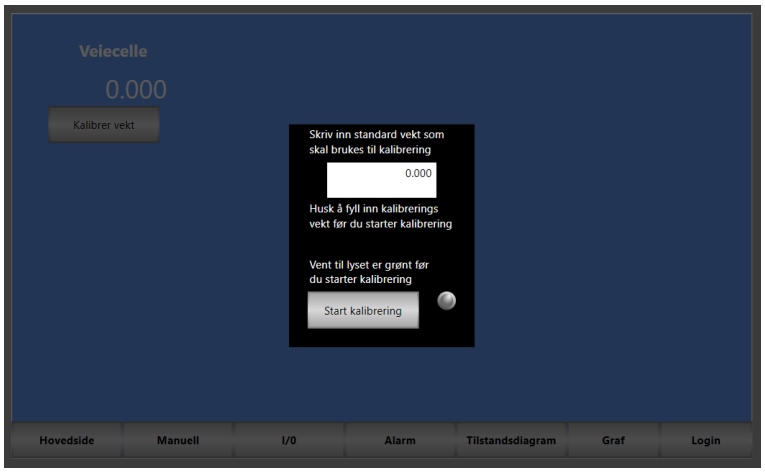
Figur 4.25: Der er mulig å logge variabler i en datalogg og få vist det på skjermen som en trend.



Figur 4.26: Innlogging for å få økt tilgang til systemet. Når en administrativ bruker har logget seg inn vil det komme en ekstra knapp på I/O-siden hvor veiecellen kan kalibreres.



Figur 4.27: Vedlikeholdknappen er bare synlig etter å ha logget inn.



Figur 4.28: Starter kalibrering av veiecellen ved å trykke på *Start Kalibrering*.

4.4 Beskrivelse av robotarm-program

Denne delen av rapporten tar for seg hvordan en robotarm kan bli brukt til å frakte flasker fra et samlebånd til en eske. For å demonstrere programmet blir det brukt en virtuell kontroller i Robot Studio.

Den nåværende pakkeprosessen er ikke automatisert. Etter korkmontering og merking blir flaskene sendt til et rundt bord som roterer ved hjelp av en motor. Dette er tidkrevende og lite effektivt da de ansatte må flytte flaskene fra bordet til eskene. Det er ønskelig å automatisere denne delen av prosessen. Ved å gjør dette vil det ikke bare avlaste de ansatte med tungt arbeid, men også frigjøre de til å prioritere andre oppgaver.

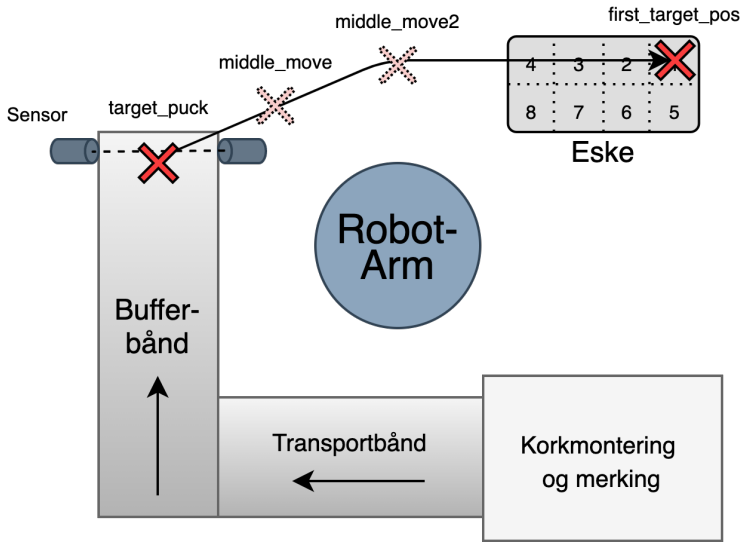
4.4.1 Sikkerhet

Bruk av robotarmer krever strenge sikkerhetstiltak. Personalet eller objekter risikerer å komme i klem som kan føre til store skader, avhengig av styrken til robotarmen. Vanlige HMS tiltak er å avsperre operasjonområdet til roboten med bruk av gitter eller markering. En annen måte å sikre området på er bruk av kamera og sensorer. De kan detektere om personer kommer innenfor operasjonsområdet og dermed stoppe prosessen øyeblikkelig. Denne løsningen kan redusere kravet til avsperringen av det relevante området. Elsikkerheten er også en stor faktor når man jobber med roboter. Det er nødvendig med en nødstopp som er lett tilgjengelig.

4.4.2 Løsning

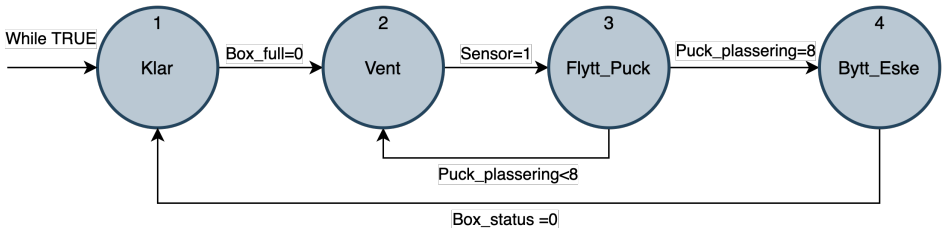
I denne oppgaven blir det brukt Robotstudio til å simulere pakkeprosessen ved bruk av en ABB IRB 140 robotarm med en griper. I simulering er det brukt puck der det skulle vært flaske.

Det er tenkt å bruke en robotarm til å frakte flaskene fra bufferbåndet til en eske. For å unngå opphopingen av flasker blir det i denne rapporten foreslått å legge til et bufferbånd som ligger 90 grader på hovedbåndet. Dette konseptet er illustrert i figur 4.29. Flaskene vil vende retning slik at de bruker mindre plass. På enden av bufferbåndet er det en sensor som detekterer når en flaske er klar til å bli hentet av robotarmen. Når sensoren går høy får robotarmen klarsignal, som aktiverer henteprosessen. Robotarmen vil plukke opp flasken og fører den videre til første posisjon i esken. Denne prosessen fortsetter frem til alle plassene i esken er fylt, og esken må bli byttet ut. Etter esken blir byttet ut må operatøren sende et signal til roboten og prosessen starter på nytt.



Figur 4.29: Figur over robotarmens bevegelse og oppsett av bufferbåndet.

Figur 4.30 inneholder en oversikt over tilstandene i programmet. Videre er det ved hjelp av tilstandsdiagrammet og koden fra figur 4.31 og figur 4.32 forklart hvordan prosessen er realisert.



Figur 4.30: Figur viser et tilstandsdiagram over programmets sekvens.

- Når koden går inn i while-løkken flytter griperen seg til et mål 200mm over *target_puck*. Dette målet befinner seg rett over puckens endeposisjon på bufferbåndet. Programmet sjekker om esken er tom(*Box_Full*=0).
- Om variabelen *Box_Full* er lav går prosessen i tilstand 2. Her startes en for-løkke som flytter puckene når de blir detektert av sensoren(tilstand 3) som befinner seg i enden av bufferbåndet.
- Etter alle puckene er flyttet blir utgangen *DO_Box_full_status* satt høy som vil si at esken er full(tilstand 4). Variabelen må setter lav før roboten kan flytte nye pucker.

RAPID-kode

```
1   MODULE Module1
2
3       ! variable target
4       VAR robtarget target_pos;
5
6       ! starting from first_target_pos
7       PROC Path_10()
8           WHILE TRUE DO
9               ! moves to target_puck and wait for a empty box
10              MoveJ Offs(Target_puck, 0, 0, 200),v1000,z100,tGripper\WObj:=wobjTableN;
11              WaitDO DO_Box_full_status,0;
12
13              ! wait for puck to be detected by sensor
14              ! moves the Puck to 2x4 targets with (105.2,74.25,0) offset for each iteration
15              ! from the first target position
16              FOR x FROM 0 TO 1 DO
17                  FOR y FROM 0 TO 3 DO
18                      WaitDI DI_sensor_status,1;
19                      movePuck(Offs(first_target_pos,x*105.2,-y*74.25,0));
20                  ENDFOR
21              ENDFOR
22
23              ! setting the box as full
24              setDO DO_Box_full_status,1;
25
26          ENDWHILE
27      ENDPROC
```

Figur 4.31: Kode i Modul1 som utfører flyttingen av pucker som vist i tilstandsdiagram 4.30

I figur 4.32 vises hovedmodellen. Her defineres de nødvendige workObjects som brukes for å referere til koordinatene for bordet og esken, samt mål for flytting av robotarmen. Grunnet store avstander ved flyttingen er det lagt til to mellommål. Hvis roboten skulle gått direkte fra hovedbåndet til esken hadde ruten blitt for lineær og aksene til roboten hadde ikke vært i stand til utføre bevegelsen.

```
1  MODULE MainModule
2
3
4      ! defines workobjects
5      TASK PERS wobjdata wobjTableN:=[FALSE,TRUE,"",[[150,-500,8],[0.707106781,0,0,-0.707106781]],[[0,0,0],[1,0,0,0]]];
6      TASK PERS wobjdata wobj_A4:=[FALSE,TRUE,"",[[298.5,-395,8.2],[0,0,0,1]],[[0,0,0],[1,0,0,0]]];
7
8
9      ! robtarget convayer
10     CONST robtarget Target_puck:=[[-1075.811563016,266.865117271,-82.34],[0,0.258819045,0.965925826,0],[0,0,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
11
12     ! assistance moves
13     CONST robtarget middle_move:=[[-415.738401754,499.94536929,92],[0,1,0,0],[-1,0,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
14     CONST robtarget middle_move_2:=[[-215.738401754,349.94536929,92],[0,1,0,0],[-2,0,-1,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
15
16     ! robtarget A4 Sheet
17     CONST robtarget first_target_pos:=[[-52.7,111.375,8.2],[0,0.707106781,0.707106781,0],[-1,0,-1,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
18
19
20     ! gets a puck controlled
21     PROC getPuck(robtarget pos)
22         MoveJ Offs(pos, 0, 0, 200),v1000,z200,tGripper\WObj:=wobjTableN;
23         MoveJ Offs(pos, 0, 0, 50),v200,z10,tGripper\WObj:=wobjTableN;
24         MoveL Offs(pos, 0, 0, 10),v50,fine,tGripper\WObj:=wobjTableN;
25         closeGripper(TRUE);
26         MoveL Offs(pos, 0, 0, 50),v50,z10,tGripper\WObj:=wobjTableN;
27         MoveJ Offs(pos, 0, 0, 200),v200,z50,tGripper\WObj:=wobjTableN;
28     ENDPROC
29
30     ! puts a puck back controlled
31     PROC putPuck(robtarget pos)
32         MoveJ Offs(pos, 0, 0, 200),v1000,z200,tGripper\WObj:=wobjTableN;
33         MoveJ Offs(pos, 0, 0, 50),v200,z10,tGripper\WObj:=wobjTableN;
34         MoveL Offs(pos, 0, 0, 10),v50,fine,tGripper\WObj:=wobjTableN;
35         closeGripper(FALSE);
36         MoveL Offs(pos, 0, 0, 50),v50,z10,tGripper\WObj:=wobjTableN;
37         MoveJ Offs(pos, 0, 0, 200),v200,z50,tGripper\WObj:=wobjTableN;
38     ENDPROC
39
40
41     ! moves a Puck from (target_Puck) to (pos)
42     PROC movePuck(robtarget pos)
43         getPuck(target_puck);
44         MoveJ middle_move,v1000,z200,tGripper\WObj:=wobjTableN;
45         MoveJ middle_move_2,v1000,z200,tGripper\WObj:=wobjTableN;
46         putPuck(pos);
47         MoveJ middle_move_2,v1000,z200,tGripper\WObj:=wobjTableN;
48         MoveJ middle_move,v1000,z200,tGripper\WObj:=wobjTableN;
49         MoveJ Offs(target_puck, 0, 0, 200),v1000,z200,tGripper\WObj:=wobjTableN;
50     ENDPROC
51
52     ! main program
53     PROC main()
54         Path_10;
55     ENDPROC
56
57 ENDMODULE
```

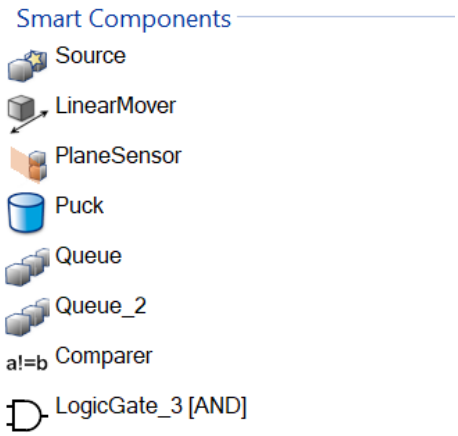
Figur 4.32: Hovedmodulen til programmet. Her defineres prosedyrer som brukes i programmet fra Module1 som flytter pucker.

På linje 21 i figur 4.32 er det definert en prosedyre som henter pucken med en kontrollert bevegelse, deretter lukkes griperen og pucken løftes. Linje 30 har en lignende prosedyre for å slippe pucken.

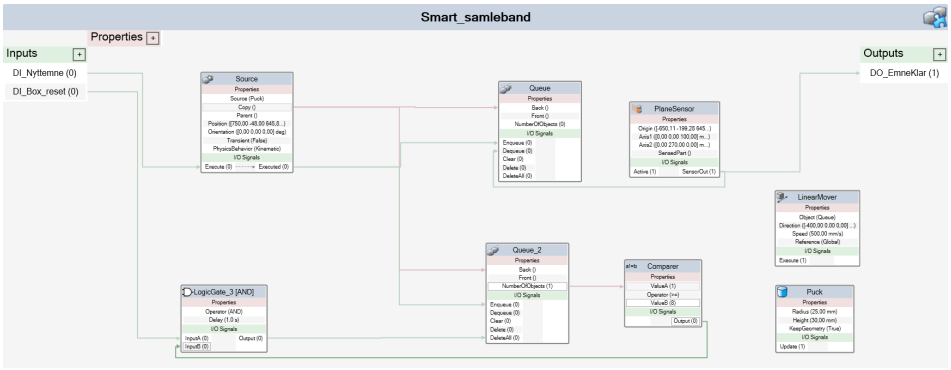
I movePuck prosedyren på linje 41 kombineres de tidligere nevnte prosedyrene til en ny prosedyre. MovePuck henter pucken fra bufferbåndet og flytter den via middle_move og middle_move2 til et valgfritt mål. Når bevegelsen er ferdig blir griperen flyttet tilbake slik at den er klar for neste puck.

Smartkomponenter

En smartkomponent er et RobotStudio-objekt som kan implementere en tilpasset atferd, og den kan kobles opp mot andre smartkomponenter for å få en mer kompleks oppførsel. Dette blir brukt for simulering av bevegelse til puckene og koble den opp mot en sensor. Smartkomponenetene som har blitt bruket vises i figur 4.33. De ulike smartkomponentene er koblet sammen som vist i figur 4.34. Mer informasjon om smartkomponenter kan leses på side 238 i manual [12] fra RobotStudio.



Figur 4.33: Smartkomponentene som blir brukt.

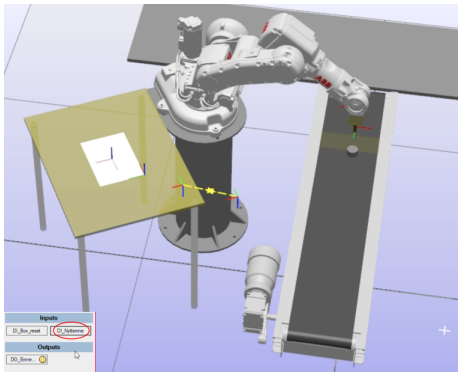


Figur 4.34: *Source*-komponenten lager nye pucker som *Queue* tar inn og legger inn i en kø. *LinearMover* tar inn puckene som står i kø, og beveger dem på bufferbåndet. Når puckene treffer *PlaneSensor* går sensoren høy og fjerner pucken fra køen, som da stopper puckens bevegelse. *Queue_2* tar inn pucker som blir laget av Source, og dens verdi går deretter inn i en *Compare*-blokk som sammenligner to verdier. *ValueA* representerer hvor mange pucker som har blitt laget og *ValueB* har verdien 8 som er antallet posisjoner i esken. Hvis *ValueA* er lik eller større enn *ValueB* skal utgangen gå høy. *LogicGate_3* er en AND-gate som tar inn verdien fra *Compare*-blokken og variabelen *DI_Nyttemne* som representerer nye pucker som ankommer bufferbåndet. For å gi brukeren kontroll over simulasjonen blir pucktilføringen styrt av brukeren ved å sette *DI_Nyttemne* høy. Hvis begge signalene er høye blir også utgangen satt høy og deretter slettes alle puckene som ligger i køen.

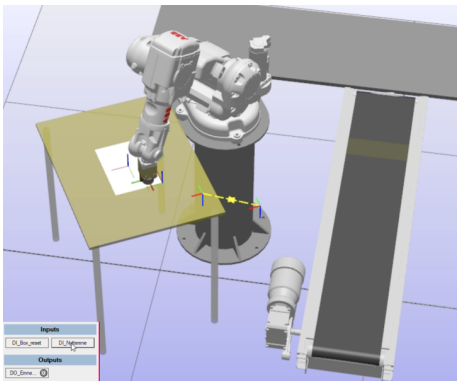
4.4.3 Simulering

Det har blitt tatt stillebilder av simuleringen for å vise funksjonaliteten.

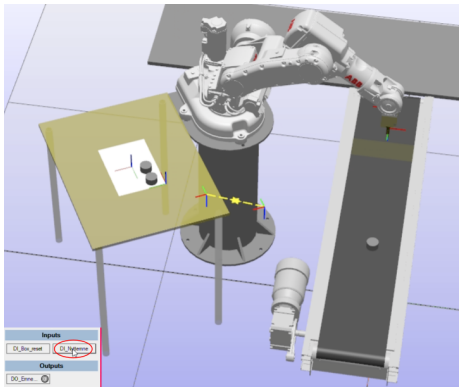
Video av simuleringen finnes på `https://youtu.be/5EOzuQ4PZGk`.



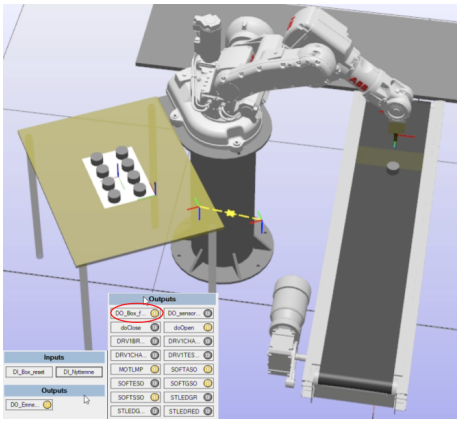
Figur 4.35: DI_Nyttemne er aktivert som lager en puck. Sensor er høy og roboten er i ferd med å flytte pucken.



Figur 4.36: Når pucken er plassert i esken vil roboten gjøre seg klar for å hente en ny puck.



Figur 4.37: DI_Nyttemne er aktivert og roboten venter på at pucken skal nå sensoren før flytteprosessen starter.



Figur 4.38: DO_Box_full_status er høy som hindrer roboten i å flytte flere pucker til variabelen blir resatt.

4.5 Simulering i Sysmac Studio

For å teste PLS programmet blir det simulert i Sysmac Studio. Simuleringen gjør det enkelt å testet at sekvensene fungerer som ønsket. For å simulere i Sysmac Studio må man sette opp Application Manager, som er en 3D-utvidelse i programmet . Hvordan man setter opp Application Manager og bygger simuleringen er beskrevet på side 32-156 i manual[2]: Sysmac Studio 3D Simulation. Når Application Manager er satt opp er det mulig å legge til forskjellige figurer, komponenter og sensorer inn i simuleringen. Man kan også legge til egendefinerte figurer.

Simuleringsmodellen er bygd opp av sensorer fra application manager, egenlagde figurer og tre figurer fra Grabcad[2][9][10][11].

Link til video av simuleringen: `https://www.youtube.com/watch?v=OO8jHTafjYo`

I Sysmac Studio er simuleringen bygd i C sharp. Sysmac Studio generer mye av C koden selv. I figur 4.39, 4.40 og 4.41, er kode som ikke er generert av Sysmac Studio.

```
112         private DateTime previousTime;
113         /// <summary>
114         /// Called to render the Shape Script object
115         /// </summary>
116         /// <returns>The list of shapes to render</returns>
117         public override void Render(IEnumerable<ShapeRenderInfo> renderInfoList) {
118             int speed = 700;
119             DateTime currentTime = this.GetCurrentControllerTime("Controller");
120             var MoveBelt = this.GetBoolVariable("Controller", "M_03");
121             var MoveBandforover = this.GetBoolVariable("Controller", "M_02");
122             var MoveBandbakover = this.GetBoolVariable("Controller", "M_22");
123             var Reset = this.GetBoolVariable("Controller", "Reset_simulator");
124             ICadData cadData = (ICadData) ace["/ApplicationManager0/Flaske_4"];
```

Figur 4.39: Utdrag av C sharp kode: Variabler som blir hentet fra kontrolleren.

[2]Grabcad er en nettside som tilbyr gratis skybasert samarbeidsmiljø som hjelper ingeniørteam med å administrere, se og dele CAD-filer.

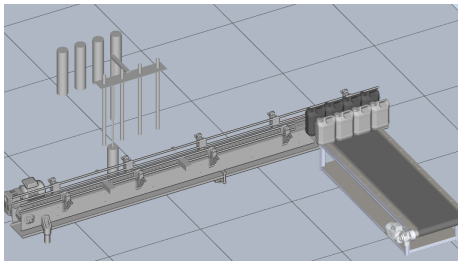
```
131             if (MoveBelt & this.IsInitialized & (cadData.OffsetFromParent.DY >= -415)) {
132                 var delta = currentTime - this.previousTime;
133                 var distance = delta.TotalMilliseconds * speed / 1000;
134                 cadData.OffsetFromParent = new Transform3D(
135                     cadData.OffsetFromParent.DX - distance,
136                     cadData.OffsetFromParent.DY,
137                     cadData.OffsetFromParent.DZ);
138             }
```

Figur 4.40: Utdrag av C sharp kode: Koden som får flaskene til å bevege seg på transportbandet.

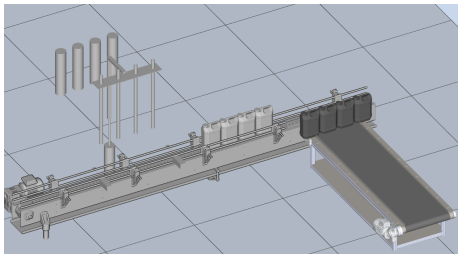
```
156             if (MoveBandbakover & this.IsInitialized & (cadData.OffsetFromParent.DY <= -415)) {
157                 var delta = currentTime - this.previousTime;
158                 var distance = delta.TotalMilliseconds * speed / 1000;
159                 cadData.OffsetFromParent = new Transform3D(
160                     cadData.OffsetFromParent.DX,
161                     cadData.OffsetFromParent.DY - distance,
162                     cadData.OffsetFromParent.DZ);
163             }
164
165             if (MoveBandforover & this.IsInitialized & (cadData1.OffsetFromParent.DY <= -415)) {
166             var delta = currentTime - this.previousTime;
167             var distance = delta.TotalMilliseconds * speed / 1000;
168             cadData1.OffsetFromParent = new Transform3D(
169                 cadData1.OffsetFromParent.DX,
170                 cadData1.OffsetFromParent.DY + distance,
171                 cadData1.OffsetFromParent.DZ);
172             }
```

Figur 4.41: Utdrag av C sharp kode: Koden som får flaskene til å bevege seg på matebandet.

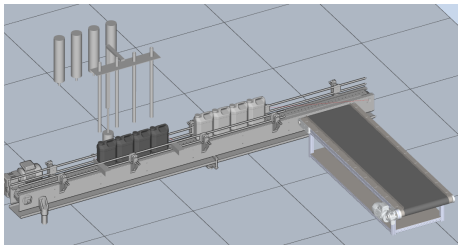
Stillbilder av simuleringen. Flaskene går fra å være hvite til svarte når de blir oppdaget av en sensor.



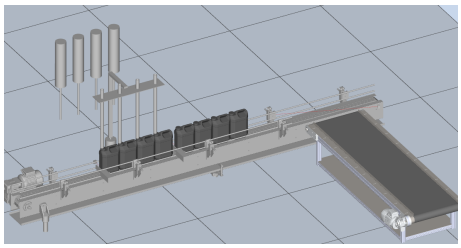
Figur 4.42: De fremste flaskene blir dyttet over på transportbåndet, mens de bakerste blir stående å vente.



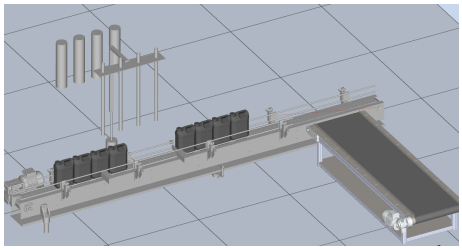
Figur 4.43: De første flaksene har gått ut av sensorens deteksjonsområde. De bakerste flaskene blir dyttet over på transportbåndet.



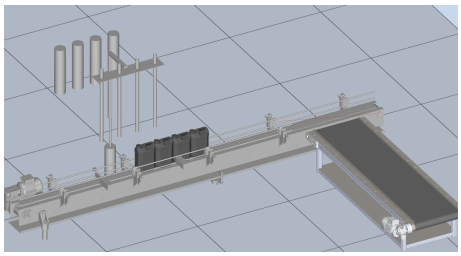
Figur 4.44: De første flaskene blir isolert av bommene.



Figur 4.45: Systemet begynner å fylle flaskene.



Figur 4.46: Fyllingen er ferdig, og de fremste flaskene går videre til korking og merking.



Figur 4.47: Neste parti med flasker på vei inn for å bli isolert av bommene.

Kapittel 5

Diskusjon

Oppgaven gikk ut på å kartlegge og videreutvikle systemet til Norwegian Oil Supply AS. Det var først antatt at mesteparten av tiden skulle brukes til videreutvikling, men kartleggingen og gjenskapingen tok lenger tid enn forventet. Vi fikk liten innsikt i hvordan produksjonslinjen fungerte fra oppgavebeskrivelsen som gjorde det vanskelig å kartlegge oppgaven før det andre bedriftsbesøket sent i februar. Kartleggingen ble gjort ved at vi observerte utstyret som ble brukt i tillegg til at produksjonslinjen ble demonstrert.

I vår løsning har vi ikke sett på muligheten til å tilføre nye flasker til matebåndet med en robot. Denne delen var ikke foreslått i oppgavebeskrivelsen, men for å oppnå bedre effektivitet kan det være aktuelt for bedriften å se på muligheten for å automatisere denne delen. Hadde dette blitt gjort hadde man frigjort en operatør som kunne prioritert andre oppgaver. Å flytte ferdigefylte esker over på pall var heller ikke en del av oppgaven, men er en annen del bedriften kan vurdere å automatisere.

5.1 Drøfting av PLS oppgradering

Rapporten tar utgangspunkt i at bedriften skal oppgradere sin PLS til en nyere modell. Argumenter for oppgraderingen er at bedriften vil kunne implementere OPC-UA kommunikasjon til en klient, brukergrensesnittet kan

bli byttet ut og man vil unngå at kontrolleren blir defekt og produksjonen stopper uventet. Kontrolleren som brukes i dag er utdatert og det kan derfor være mangel på fagekspertise hvis programmet skal bli oppdatert. Motargumentet for oppgraderingen vil være at det er unødvendig å investere i en ny PLS hvis den eksisterende fungerer etter hensikt.

Produksjonslinje på en felles PLS

Produksjonslinjen består i dag av to kontrollere. Kontrolleren som er relevant for denne oppgaven er den som styrer flaskefyllingen. Den andre kontrolleren blir brukt til korkmontering og merking av flaskene. Den nye kontrolleren som er brukt, har mulighet for å koble 32 tileggsmoduler. Det er derfor mulig å implementere begge programmene i samme kontroller. Fordelen med det er at man får alt på samme kontroller.

Bruk av OPC-UA

I oppgavebeskrivelsen var det nevnt følgende mulige oppgaver som gjelder OPC:

- Når ny ordre ankommer til maskin via OPC-UA linken fra klient på PC blinker fysisk varsellys.
- Vise status på ordre og andre relevante data i PLS på PC (OPC-klient).
- Motta og behandle alarmer og hendelser fra OPC-UA.
- Program skal kunne rapportere status til server via OPC-UA.

Det kom frem i bedriftsbesøket at de ikke ønsket muligheten til å legge inn nye ordre fra en OPC-UA klient. Vi har derfor ikke prioritert denne løsningen. Bedriften bruker et annet system for håndtering av ordre. OPC-UA klienten er allerede koblet opp til serveren, så om bedriften ønsker å legge til denne funksjonaliteten vil det lett kunne implementeres.

I vår løsning vises kontrollerens modus, feilstatus, alarmer, programmets modus(Auto), totalt antall flasker som skal fylles, antall gjenværende flasker som skal fylles og belastningen på veiecellen. Variablene vises i figur 4.16. Vi er fornøyd med vår løsning for implementeringen av OPC-UA.

5.2 Justeringer til HMI

Et grafisk brukergrensesnitt skal være enkel og intuitiv for en bruker. Det vil være forskjellig behov ettersom hvilken type bedrift som skal bruke det. Derfor er det viktig at operatørene ikke føler det mangler noe essensielt.

I dette tilfelle var oppgaven å lage det nye brukergrensesnittet slik at den opererer tilnærmet likt som den gjør i dag med ekstra funksjonaliteter. Det var ønsket å vise status på all I/O, starte utførelse av ny ordre og vise på skjermen hvor programmet er i sekvensene.

I HMI-programmet er det laget syv hovedsider, men å implementere alle sidene i det nye systemet er antageligvis ikke nødvendig. Det kan være en idé å ha noen av sidene bare tilgjengelig for en administrativ bruker. På den måten kan skjermen som operatørene ser være helt enkel og bare til styring, mens fagpersoner kan logge seg inn og for å få tilgang til mer informasjon om prosessen.

På det nåværende grensesnittet er det mulig å regulere hastigheten på transportbåndet. Da det ikke er dokumentert hvordan det fungerer eller hvordan det blir brukt, er dette er noe som ikke er blitt prioritert i løsningen på det nye grensesnittet.

Oppgaven opplyste at det var ønskelig å starte utførelse av ny ordre fra HMI-grensesnittet. I løsningen har dette blitt gjort ved å manuelt legge inn hvor mange flasker som skal fylles, og programmet vil stoppe når fyllingen er ferdig.

5.3 Valg av mengdemåling

Måling av væskenivået i flasker kan blir gjort på flere måter. Vi var innom flere måleprinsipp før vi endte med veiecelle. Kamera eller lyssensor kunne ikke brukes på flaskene fordi de er lys-ugjennomtrengelige. På grunn av liten og kortvarige væskestrømmer ble ultralyd- og turbinmåling ugunstig. Trykkmåler ble utelatt på grunn av at det fylles forskjellige medier med forskjellige tettheter. Man ville også trenge fire trykkmålere, en på hver fyllesylinder. Det endte derfor på veiecelle. Tanken er å måle alle fire flaskene etter hverandre og sammenligne de. Da trenger man ikke å tenke på tettheten til mediet siden man kun sammenligner de med hverandre. Vi har vist på lab at det er mulig å sammenligne vekten på fire objekter som kommer rett etter hverandre. Om denne løsningen fungerer optimalt for bedriften er vanskelig å si. Det kan være uønsket støy som vi ikke har tatt hensyn til. I tilegg har vi ikke alt utstyr som trengs for å simulere veiecellen som tenkt. Tanken er å montere en veiecelle under et rullebånd. Når flaskene ruller over rullebåndet veies de en etter en. Så brukes det en algoritme for å finne ut om de er like. Hvis de ikke er like, går det en alarm. Siden vi ikke har hatt tilgang til et rullebånd er det ikke sikkert løsningen vil fungere som tenkt.

Veiecelle

Kontroll av mengde i flasker var ikke opprinnelig en del av oppgaven. Når det ble spurt om vi kunne se på det, var vi godt i gang med de andre punktene i oppgaven. Vi prioriterte derfor dette punktet sist. Når vi begynte å se på lastcellen tok det lengre tid å sette seg inn i virkemåte og oppsett enn vi hadde planlagt. Det ble ikke tid til å lage en algoritme for å sammenligne vekten på flaskene. Derfor ble det heller ikke laget et PLS-program for å varsle om feil nivå på flaskeinnholdet. Vi tror bedriften kunne ha ønsket en bedre oversikt over feilfylling av falsker. Det er derfor noe vi anbefaler bedriften å realisere.

5.4 Robotarm

Avlasting av operatører

En automatisert pakkeprosess har en stor innvirkning på å avlaste de ansatte for slitasjeskader. De ansatte må fremdeles manuelt fylle på med tomme esker og stable eskene etter teiping av de ferdigfylte eskene. Selv om det hadde vært optimalt å få hele prosessen automatisert vil de ansatte slippe å løfte flerfoldige 5 liters flasker om dagen. I tillegg kan arbeidskapasiteten brukes til andre oppgaver.

Alternativer for å unngå opphopning av flasker

Vi har valgt å bruke ett bufferbånd for å kontrollere opphopingen av flasker mens de venter på å bli fraktet av robotarmen. For å unngå opphopingen var vi innom flere alternativer. De alternative løsningene vi var innom var å øke kapasiteten til armen ved å bruke en klype som kan hente flere flasker om gangen, øke hastigheten til armen, minske avstanden armen må forflytte flaskene eller øke antallet robotarmer. I dette tilfellet er de nevnte løsningene ikke praktiske.

Valg av robotarm

En av de foreslåtte punktene i oppgavebeskrivelsen var å teste robotarmen i praksis. Vi fant ut i april at dette ikke var nødvendig grunnet at simuleringen i Robot Studio var såpass nær det virkelige systemet. Valget av robotarmen var basert på at vi hadde den tilgjengelig på UiS. Siden den praktiske delen utgikk hadde det vært fordelsfullt å valgt en robotarm som var kompatibel med OPC-UA. Vi føler likevell at programmet for robotarmen fungerer etter hensikt. Den er ikke direkte avhengig av å kommunisere med kontrolleren.

5.5 Siumulering i Sysmac Studio

Simuleringsutvidelsen i Sysmac Studio er relativt nytt. Fordi utvidelsen er så ny, var det lite informasjon og eksempler på nettet om hvordan man lager og bygger en simulering. Det ble derfor brukt lenger tid å sette seg inn i utvidelsen en antatt. Manualen for 3D simuleringen bruker et eksempel som ikke var relevant for oss når manualen gikk i gjennom oppsettet. Vi måtte derfor plukke ut den informasjonen som var nyttig for oss, og resten måtte vi finne ut selv.

Kapittel 6

Konklusjon

I konklusjonen blir det sammenlignet hvordan vi har utført de ulike punktene fra den reviderte oppgavebeskrivelsen. Etter hvert punkt følger det en utdypelse.

- *Utarbeide oversikt over innganger og utganger på eksisterende PLS og lage en I/O liste over de.*

  I/O-tabellen som ble laget er ikke direkte basert på det eksisterende systemet. Det ble tatt utgangspunk i den nye kontrolleren og hvordan inngangene og utgangene kunne blitt koblet opp. Det er lagt til en veiecelle som ikke fantes i det eksisterende systemet.

- *Oppgradere brukergrensesnittet til en digital HMI skjerm.*

  Brukergrensesnittet ble oppgradert til et grafisk brukergrensesnitt med foreslåtte sider og funksjonaliteter som kan være relevante for bedriften.

- *Lage tilstandsdiagram og realisere det i Sysmac.*

  Tilstandsdiagrammet som ble laget er basert på observasjoner fra bedriftsbesøket. Det kan avvike noe fra det eksisterende systemet, men programmet oppnår det samme resultatet, i tillegg til nye funksjonaliteter.

- *Vise status på all IO på HMI.*

  Alle innganger og utganger som blir brukt i programmet vises på det nye brukergrensesnittet. De er koblet til bitlamper som vil lyse når den relevante inngangen eller utgangen er aktiv. Det analoge inngangssignalet til veiecellen blir vist på brukergrensesnittet i sanntid.

- *Fra HMI grensesnitt kunne starte utførelse av ny ordre.*

  Brukergrensesnittet tillater operatøren å manuelt legge inn antall flasker som skal fylles. Det er mulig å trekke fra og legge til antallet i ettertid. Når valgt antall flasker er fylt vil prosessen stoppe. Denne funksjonen kan deaktiveres ved en knapp på skjermen. Da vil prosessen fortsette til den blir stoppet.

- *Lag enkel animasjon (simulering) av maskinen på PC.*

  Det ble laget en simulering av systemet i Sysmac Studio. Simuleringen viste at vår løsning på PLS-kode fungerte som tenkt.

- *Motta og behandle alarmer og hendelser fra OPC.*

  OPC-UA har blitt brukt til å kommunisere mellom kontrolleren og en PC med UaExpert. Alarmer og de relevante variablene vises i UaExpert. Det var kun ønskelig å styre systemet via HMI-skjermen, så behandlingen av alarmer fra PC ble ikke implementert.

- *Forslag til kode for styring av PLS.*

  Koden som er utviklet for styring av systemet er basert på tilstandsdiagrammene som har blitt laget. Programmet fungerer etter hensikt som er vist i simuleringen. Programmet kan avvike fra programmet bedriften bruker i det eksisterende systemet, men det oppnår samme funksjon.

- *Utvikle nødvendige algoritmer i RobotStudio med simulering.*

  Programmet for robotarmen ble utviklet og simulert i Robot Studio. Roboten henter pucker fra et bufferbånd og flytter de til en eske. Hensikten med programmet er å vise prinsippet bak hvordan flytteprosessen kan bli utført. Programmet fungerte etter hensikt.

- *Se på muligheter til å kontrollere om en flaske har fått riktig mengde påfyll .*

  En veiecelle er implementert i systemet. Det er vist hvordan den analoge inngangen til veiecellen ser ut etter filtrering. Ferdig program for å detektere feilfylling ble ikke laget.

Tillegg A

Vedlegg

Bibliografi

[1] Omron: NA-series Programmable Terminal Software User's Manual, https://assets.omron.eu/downloads/manual/en/v13/v118_na_series_programmable_terminal_software_users_manual_en.pdf

[2] Omron: Sysmac studio 3D simulation operation manual https://assets.omron.eu/downloads/manual/en/v1/w618_sysmac_studio_-_3d_simulation_function_operation_manual_en.pdf

[3] Omron: NX-102 Hardware user manual https://assets.omron.eu/downloads/manual/en/v6/w593_nx102_cpu_unit_technical_manual_en.pdf

[4] Omron: Sysmac studio operation manual http://products.omron.us/Asset/Sysmac_Studio_Ver1_18_OperationManual_en_201704_W504-E1-20.pdf

[5] Omron: NJ/NX-Series OPC-UA Manual https://assets.omron.eu/downloads/manual/en/v9/w588_nj_nx-series_opc_ua_cpu_units_users_manual_en.pdf

[6] Omron Youtube video: Connecting NX102 to UaExpert client. https://www.youtube.com/watch?v=JiJK1pU9XMU

[7] Omron: Load Cell Input Unit Startup Guide for Weight Measurement, https://assets.omron.eu/downloads/manual/en/v1/p104_nx_series_load_cell_input_unit_setup_manual_en.pdf

[8] Omron: NX-series Load Cell Input Unit user manual, https://assets.omron.eu/downloads/manual/en/v3/w565_nx-series_load_cell_input_unit_users_manual_en.pdf

[9] Dindin Saepudin: Table Top Conveyor
`https://grabcad.com/library/table-top-conveyor`

[10] Denis Augusto Freitas Zutin: Conveyor
`https://grabcad.com/library/conveyor-217`

[11] Jegor Glushenkov: 5l canister
`https://grabcad.com/library/5l-canister`

[12] RobotStudio: Operating manual
`https://library.e.abb.com/public/c5872e80fa697e76c1257b440052854c/3HAC032104-001_revC_en.pdf`

Tillegg B

Programlisting

I/O						
Position	Type	Beskrivelse	R/W	Data type	Variabel	Variabel berskrivelse
2	NX-AD3603	Analog inngang 1	R	INT		
		Analog inngang 2	R	INT		
		Analog inngang 3	R	INT		
		Analog inngang 4	R	INT		
3	NX-RS1201	Deteksjonstatus	R	WORD		Samlede data for deteksjonsstatus for Ch1
		Utførelsesstatus	R	WORD		Samlede data for kjøringsstatus for Ch1
		Bruttovektverdi	R	REAL	Vekt	Bruttovektverdi for Ch1
		Kalibreringskommando svar SID	R	UINT	Calibration_Command_Response_SID	
		Kalibreringskommando svar	R	WORD	Calibration_Command_Response	
		Operasjonskommando	W	WORD		
		Kalibreringskommando	W	UINT	Calibration_Command_SID	
		Kalibreringskommando	W	WORD	Calibration_Command	
		kalibreringsdata	W	REAL	Calibration_Data	
4	NX-ID5442	Digital inngang 0	R	BOOL	Nødstopp	Nødstopp
		Digital inngang 1	R	BOOL	LSH_01	Endebryter fyllesylinder oppe
		Digital inngang 2	R	BOOL	LSL_01	Endebryter fyllesylinder nede
		Digital inngang 3	R	BOOL	LSH_02	Endebryter fyllerør oppe
		Digital inngang 4	R	BOOL	LSL_02	Endebryter fyllerør nede
		Digital inngang 5	R	BOOL	LSH_03	Nivå basseng høyt
		Digital inngang 6	R	BOOL	LSL_03	Nivå basseng lavt
		Digital inngang 7	R	BOOL	ZS_01	Fyllehode ute av posisjon
		Digital inngang 8	R	BOOL	ZS_02	Kapasitiv giver mateband
		Digital inngang 9	R	BOOL	ZS_03	Kapasitiv giver bom foran
		Digital inngang 10	R	BOOL	ZS_04	Kapasitiv giver bom bak
		Digital inngang 11	R	BOOL	ZS_05	Kapasitiv giver etter bom
		Digital inngang 12	R	BOOL		
		Digital inngang 13	R	BOOL		
		Digital inngang 14	R	BOOL		
		Digital inngang 15	R	BOOL		
5	NX-OC4633	Digital utgang 0	W	BOOL	SV_01	Ventil 1 Sylinder 1
		Digital utgang 1	W	BOOL	SV_02	Ventil 2 Sylinder 2
		Digital utgang 2	W	BOOL	SV_03	Ventil 3 Sylinder 3
		Digital utgang 3	W	BOOL	SV_04	Ventil 4 Sylinder 4
		Digital utgang 4	W	BOOL	SV_05	Stempel 1 og 2, Sylinder 1-4
		Digital utgang 5	W	BOOL	SV_06	Stempel 3 Fyllerør
		Digital utgang 6	W	BOOL	SV_07	Stempel 4 Bom forran
		Digital utgang 7	W	BOOL	SV_08	Stempel 5 Bom bak
6	NX-OC4633	Digital utgang 0	W	BOOL	SV_09	Stempel 6 Flaskeholder
		Digital utgang 1	W	BOOL		
		Digital utgang 2	W	BOOL		
		Digital utgang 3	W	BOOL	XV_01	Ventil tilførsel basseng
		Digital utgang 4	W	BOOL	M_01	Pumpe basseng
		Digital utgang 5	W	BOOL	M_02_forover	Mateband forover
		Digital utgang 6	W	BOOL	M_02_bakover	Mateband bakover
		Digital utgang 7	W	BOOL	M_03	Transportband forover

Figur B.1: I/O liste

B.1 Utdrag fra kode av hele Sysmac Studio

Table of Contents

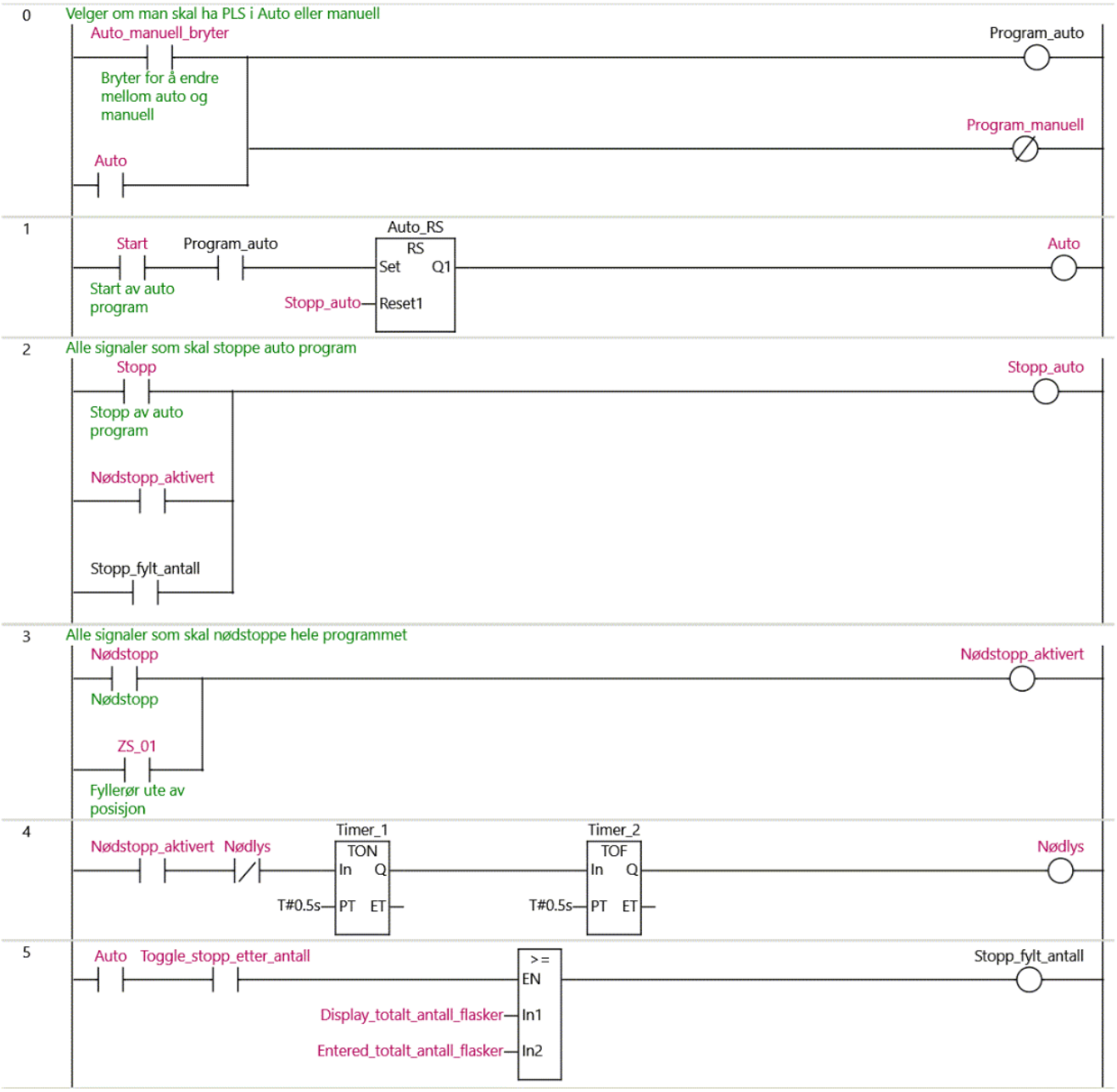
1.Controller

1-9.POUs

1-9-1.Programs

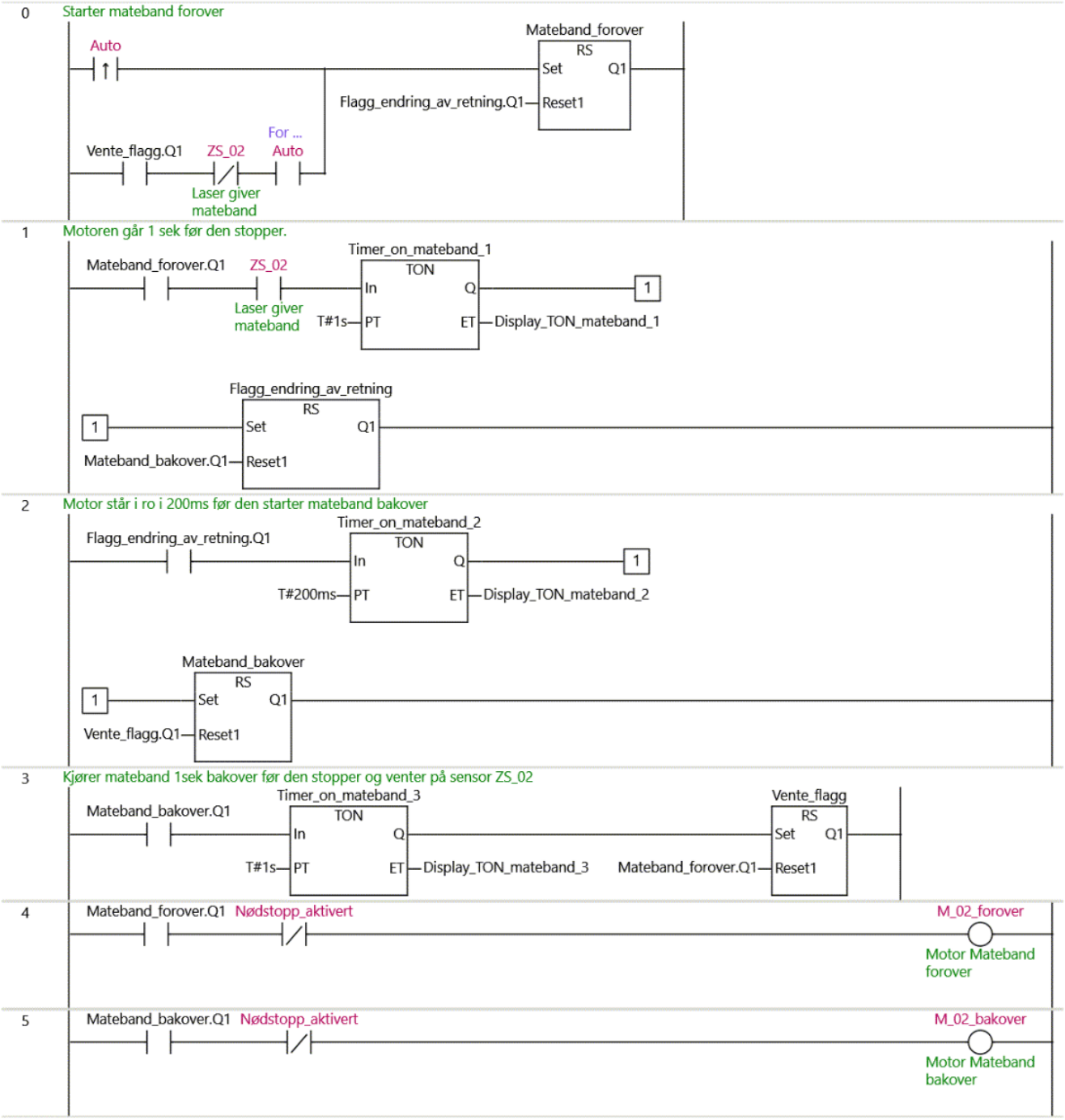
1-9-1-1.Auto_Manuell_program

1-9-1-1-2.Auto_manuell_velger



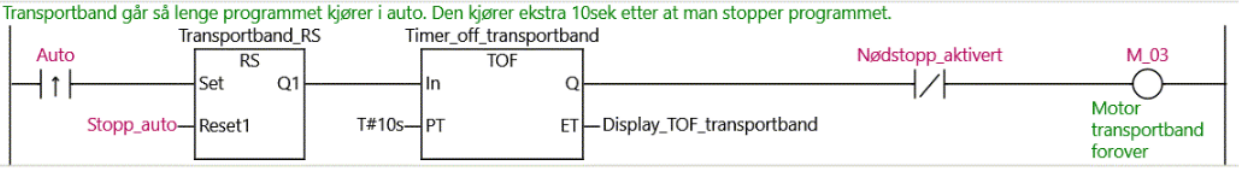
1-9-1-2.Auto_program

1-9-1-2-2.S_Mateband

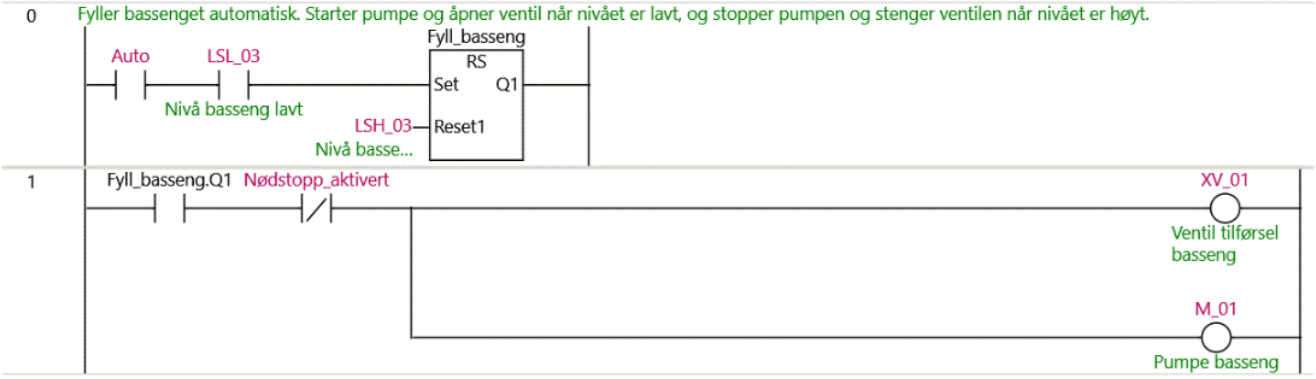


1-9-1-2-3.S_Transportband

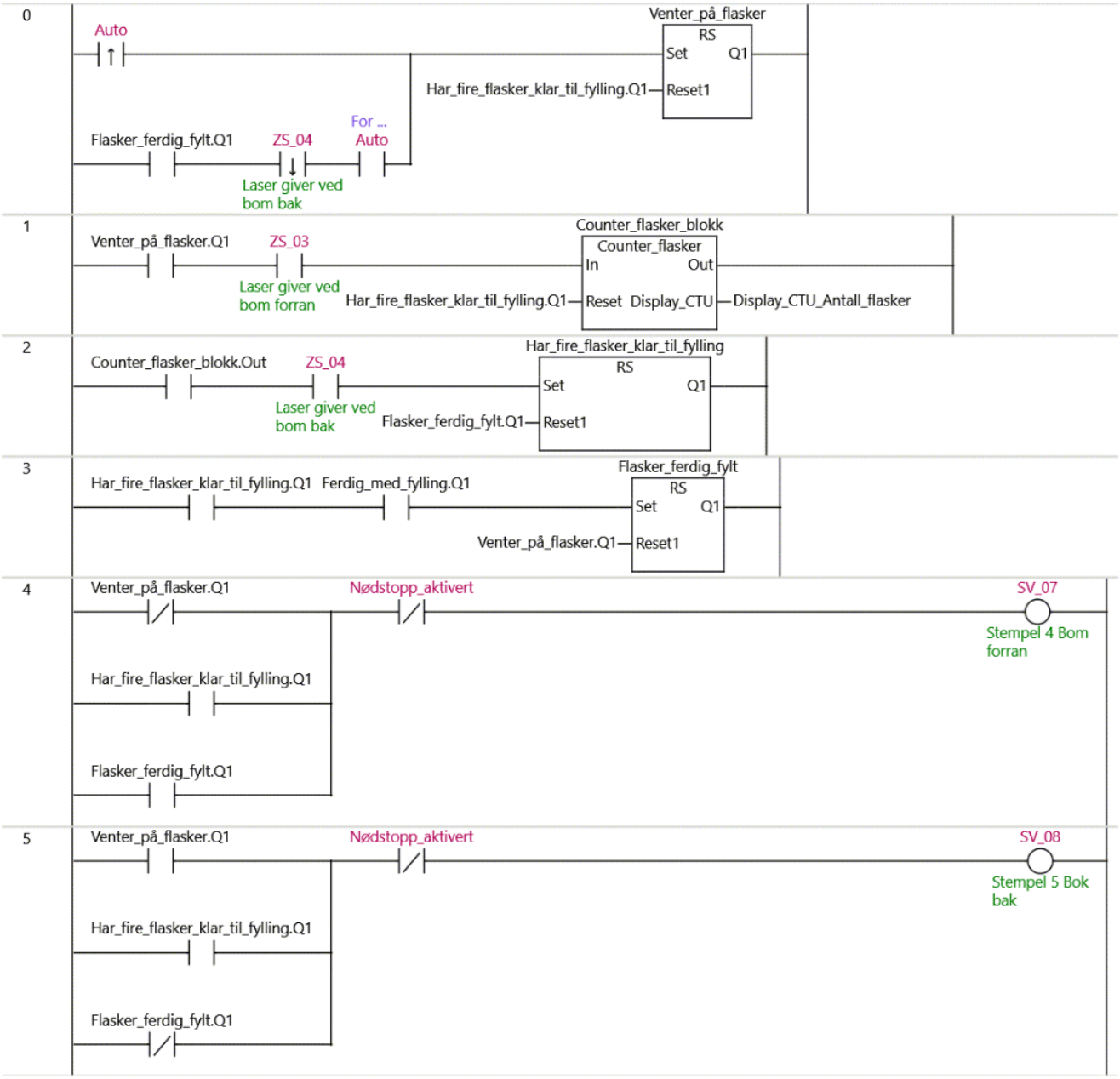
0 Transportband går så lenge programmet kjører i auto. Den kjører ekstra 10sek etter at man stopper programmet.



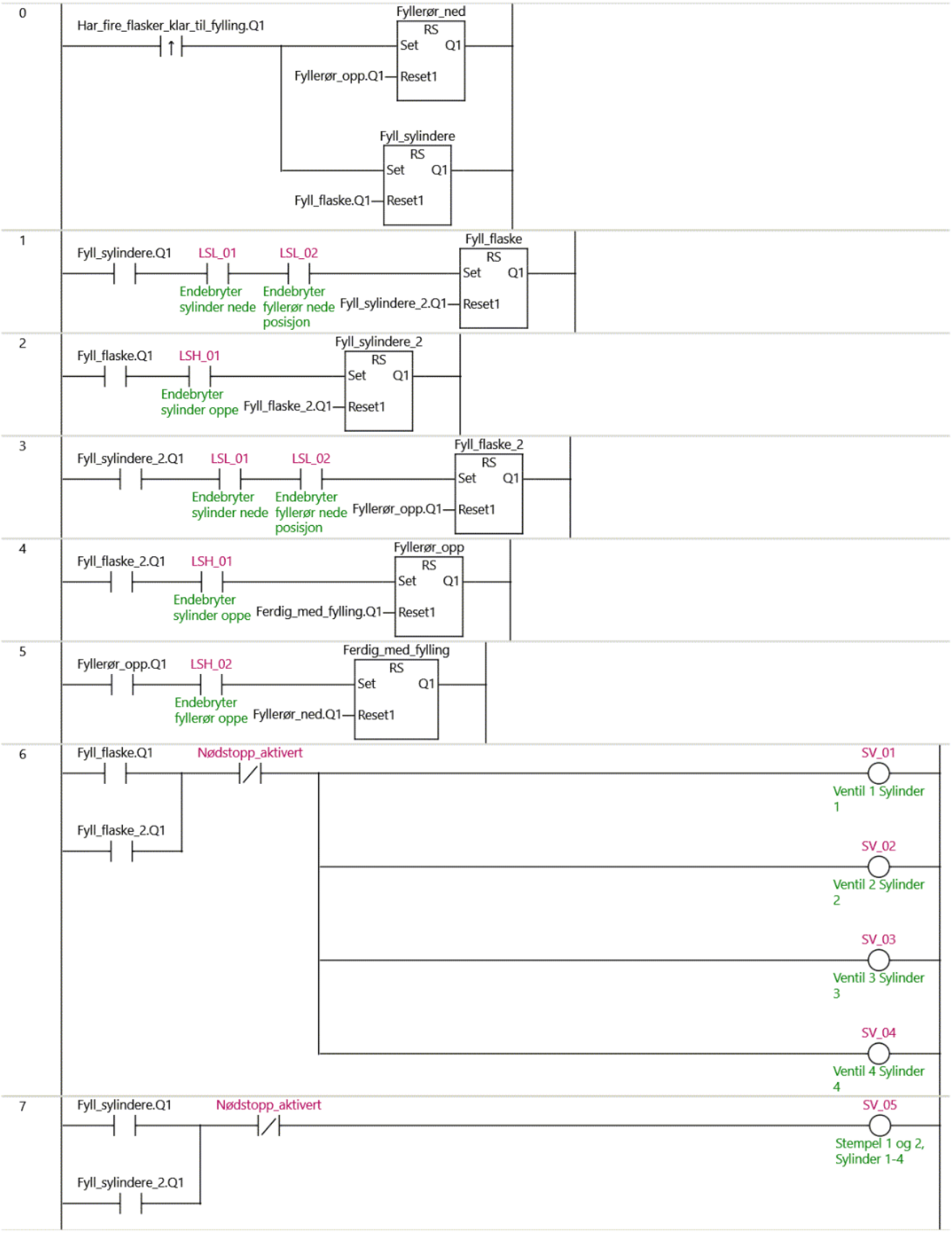
1-9-1-2-4.S_Basseng

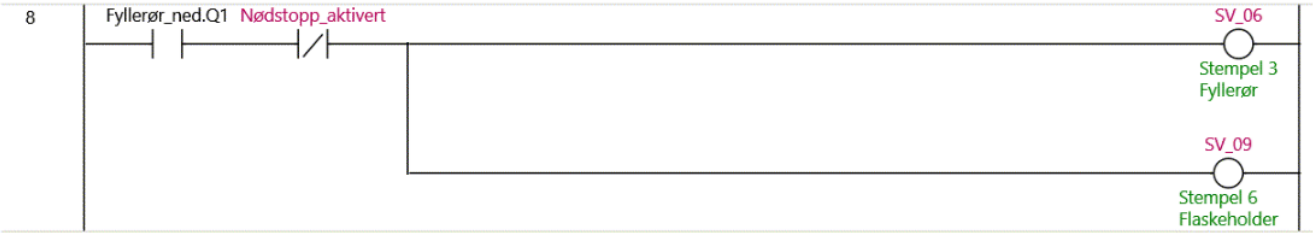


1-9-1-2-5.S_Bomstyring

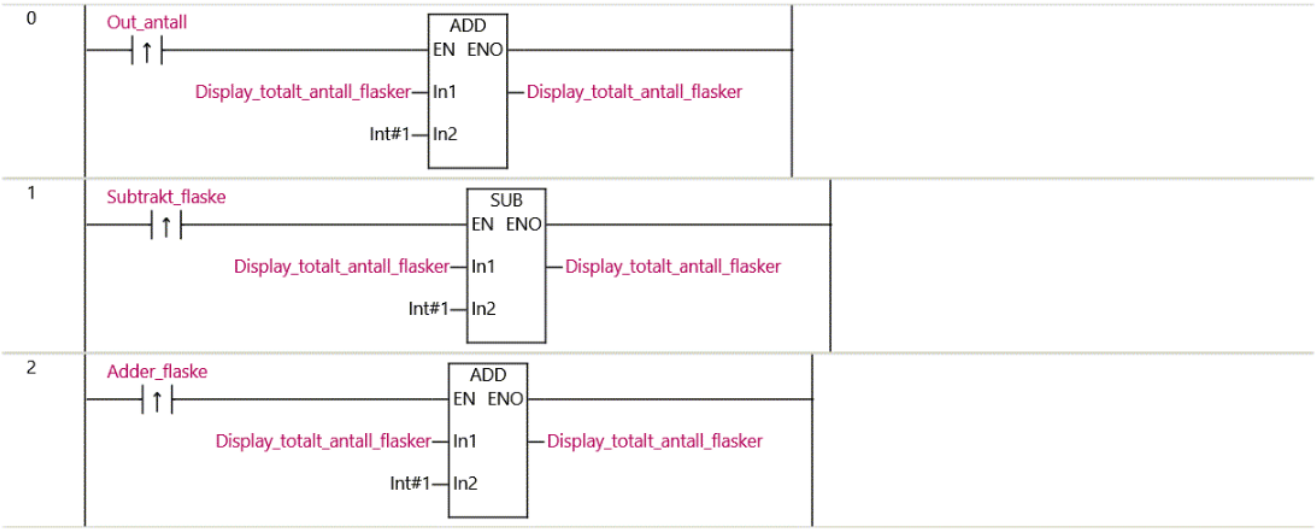


1-9-1-2-6.S_Fylling

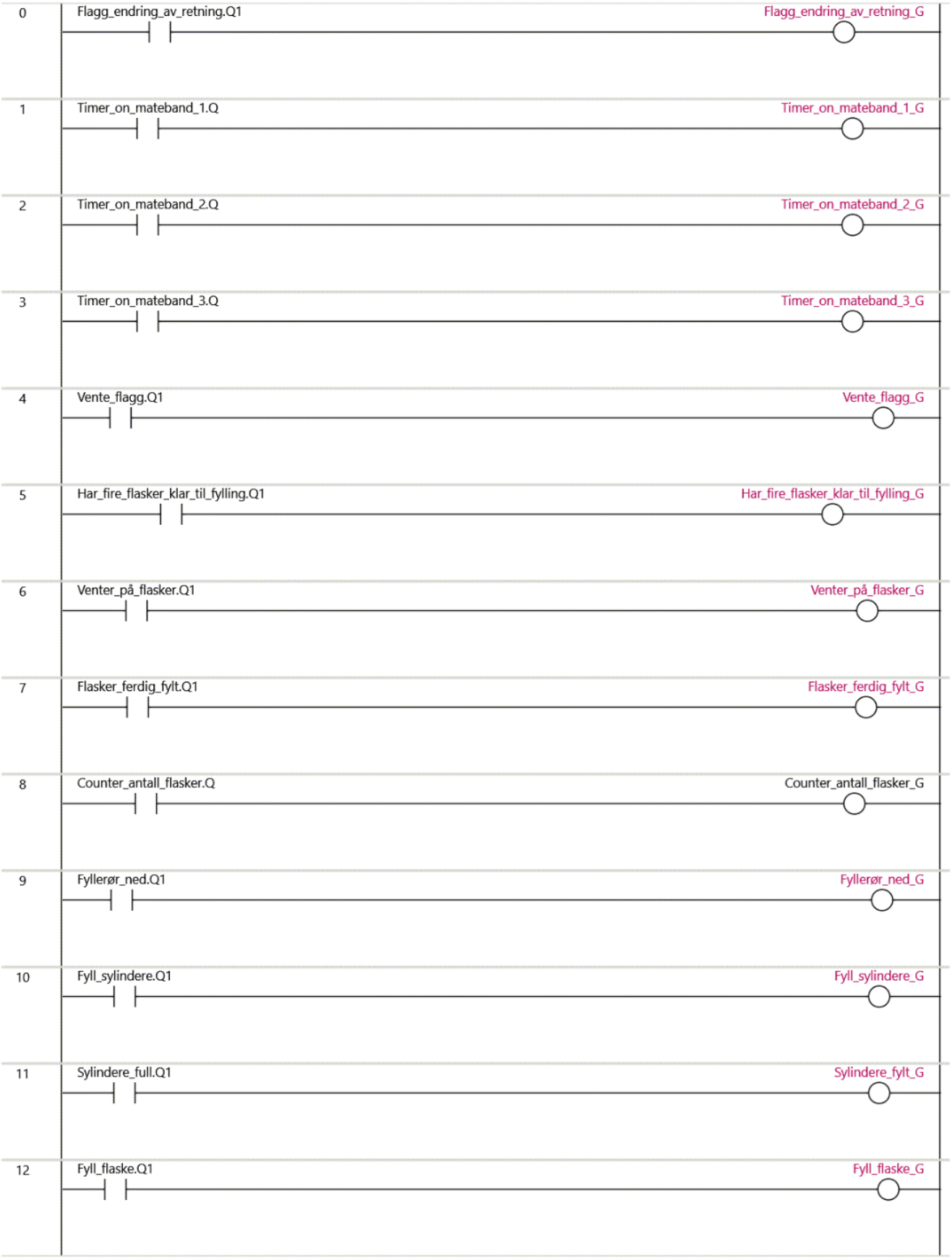



8
Fyllerør_ned.Q1
Nødstopp_aktivert
SV_06
Stempel 3
Fyllerør
SV_09
Stempel 6
Flaskeholder

1-9-1-2-7.S_Totalt_antall_flasker



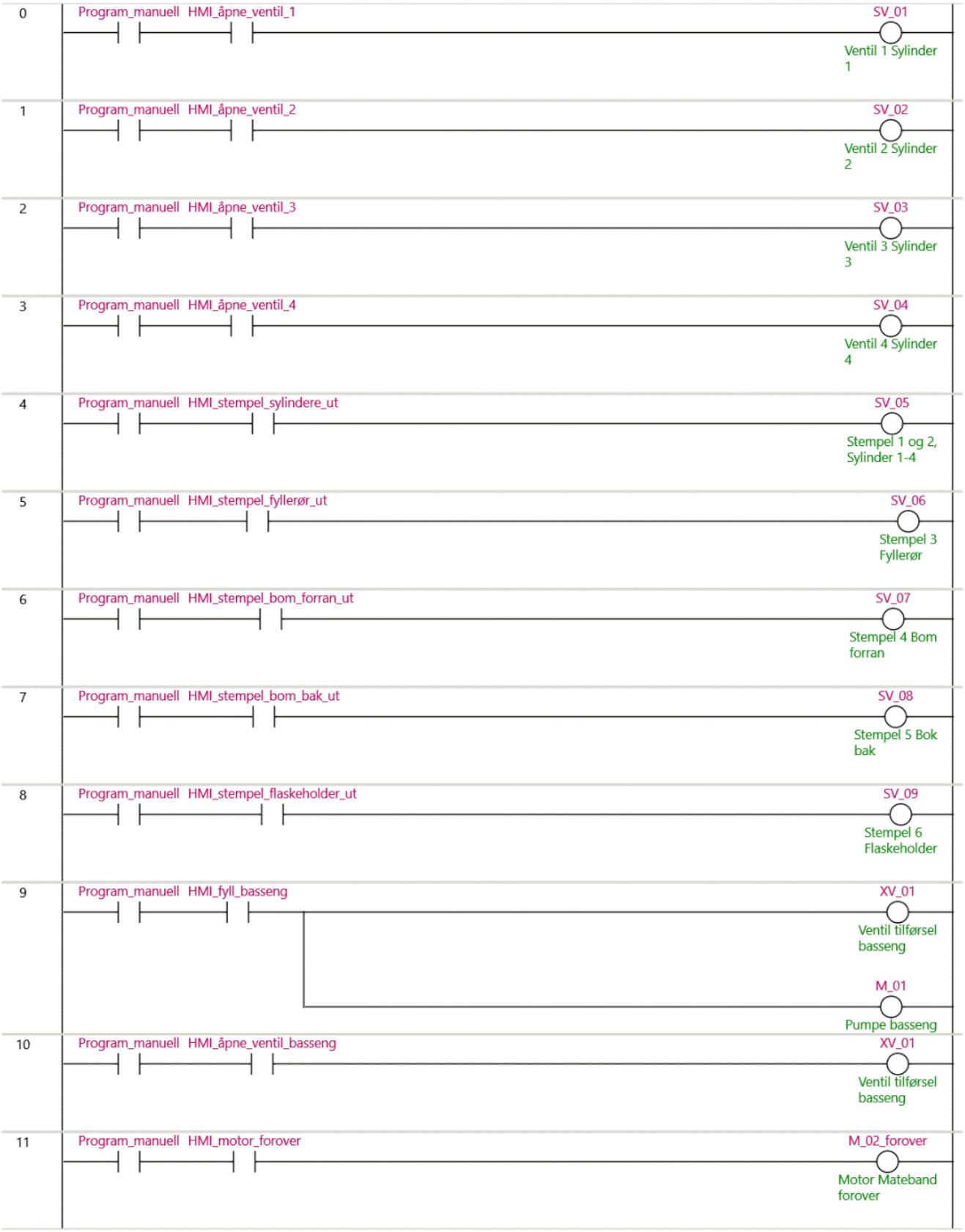
1-9-1-2-8.HMI


0
Flagg_endring_av_retning.Q1
Flagg_endring_av_retning_G
1
Timer_on_mateband_1.Q
Timer_on_mateband_1_G
2
Timer_on_mateband_2.Q
Timer_on_mateband_2_G
3
Timer_on_mateband_3.Q
Timer_on_mateband_3_G
4
Vente_flagg.Q1
Vente_flagg_G
5
Har_fire_flasker_klar_til_fylling.Q1
Har_fire_flasker_klar_til_fylling_G
6
Venter_på_flasker.Q1
Venter_på_flasker_G
7
Flasker_ferdig_fylt.Q1
Flasker_ferdig_fylt_G
8
Counter_antall_flasker.Q
Counter_antall_flasker_G
9
Fyllerør_ned.Q1
Fyllerør_ned_G
10
Fyll_sylindere.Q1
Fyll_sylindere_G
11
Sylindere_full.Q1
Sylindere_fylt_G
12
Fyll_flaske.Q1
Fyll_flaske_G


13
Sylinder_tom.Q1
Sylinder_tom_G
14
Fyllerør_opp.Q1
Fyllerør_opp_G
15
Fyll_sylindere_2.Q1
Fyll_sylindere_2_G
16
Sylindere_full_2.Q1
Sylindere_fylt_2_G
17
Fyll_flaske_2.Q1
Fyll_flaske_2_G
18
Sylinder_tom_2.Q1
Sylinder_tom_2_G
19
Ferdig_med_fylling.Q1
Ferdig_med_fylling_G
20
Stopp_auto
Stopp_lys_RS
RS
Set
Q1
Reset_stopp_lys
Reset1
Stopp_lys
21
Start
Reset_stopp_lys
Start av auto program
Program_manuell

1-9-1-3.Manuell_program

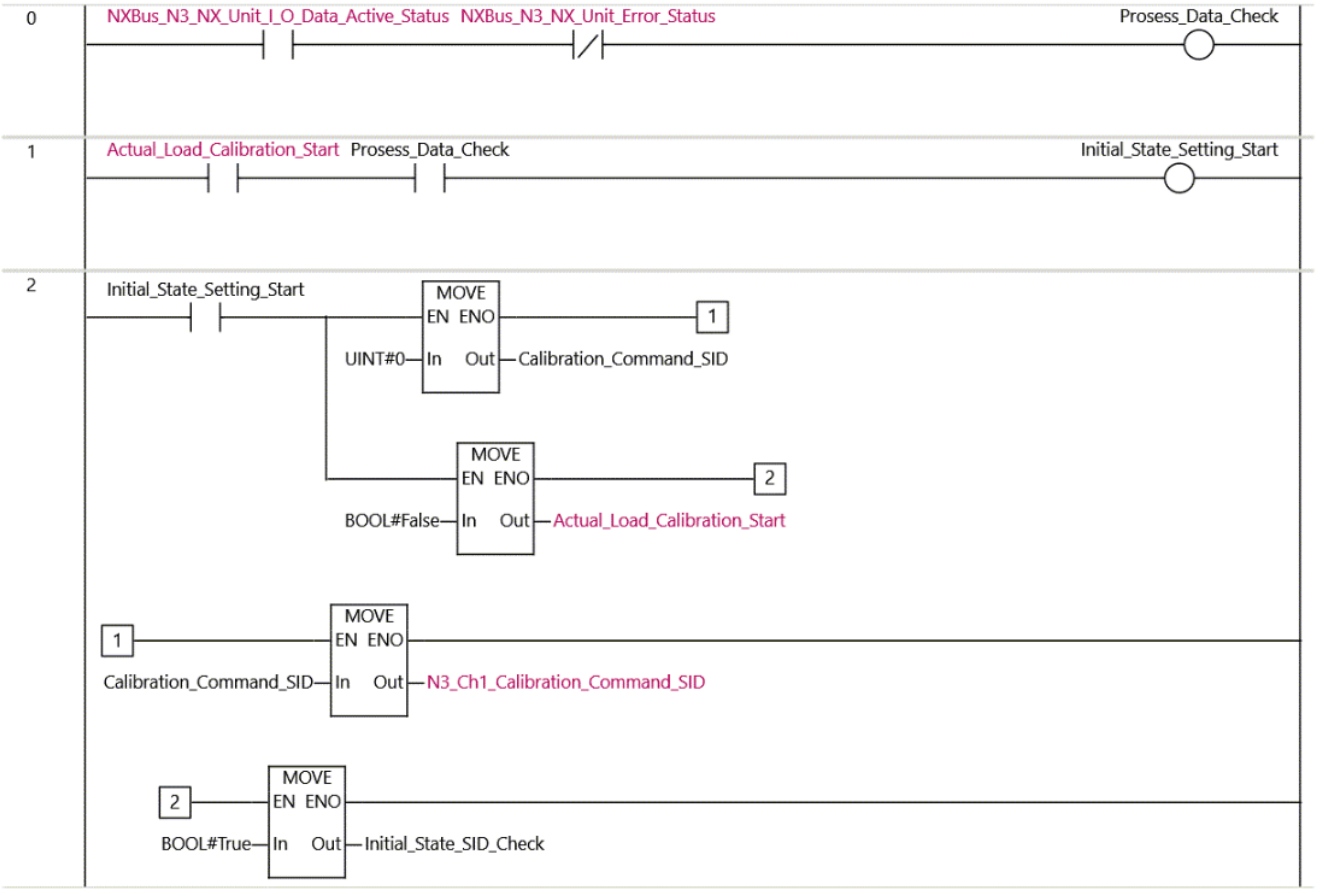
1-9-1-3-2.S_Manuell



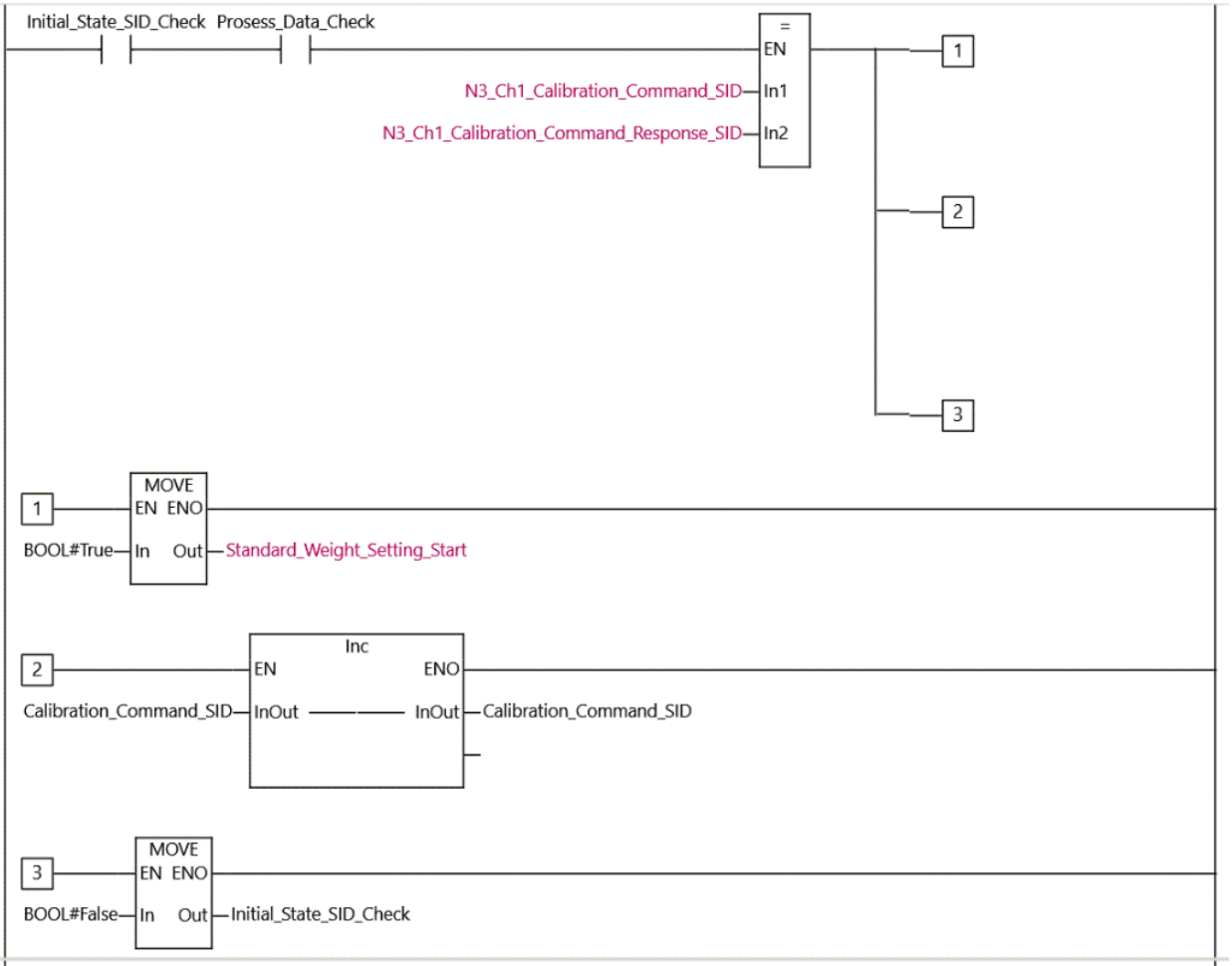

12
Program_manuell HMI_motor_bakover
M_02_bakover
Motor Mateband bakover
13
Program_manuell HMI_motor_tranportband_forover
M_03
Motor transportband forover

1-9-1-4.Veiecelle_program

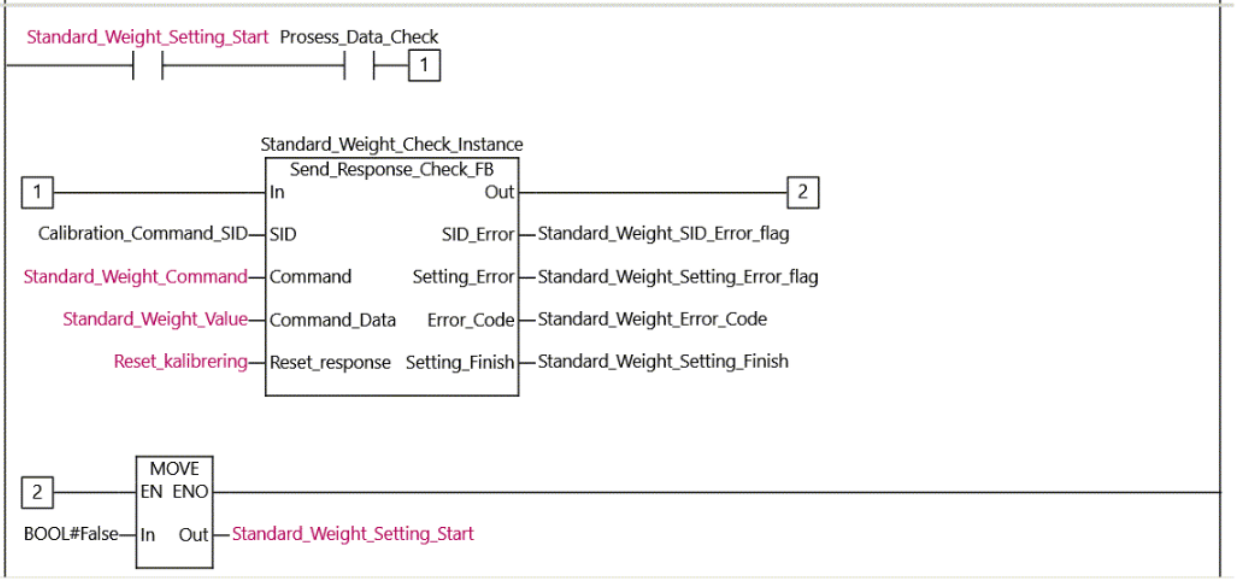
1-9-1-4-2.S_kalibrere_vekt

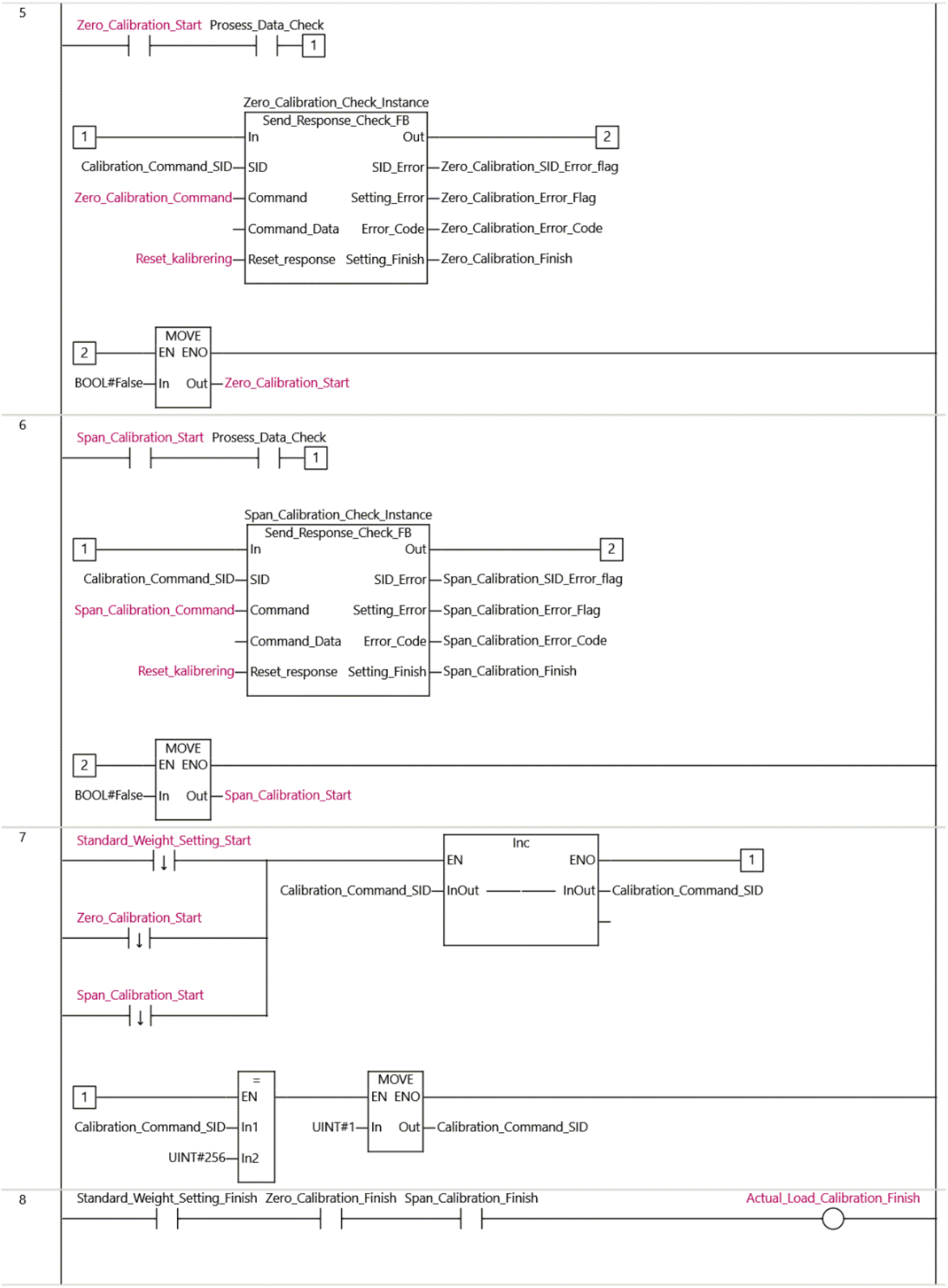


3


Initial_State_SID_Check
Prosess_Data_Check
=
EN
In1
In2
N3_Ch1_Calibration_Command_SID
N3_Ch1_Calibration_Command_Response_SID
1
2
3
MOVE
EN ENO
In Out
BOOL#True
Standard_Weight_Setting_Start
Inc
EN ENO
InOut InOut
Calibration_Command_SID
MOVE
EN ENO
In Out
BOOL#False
Initial_State_SID_Check

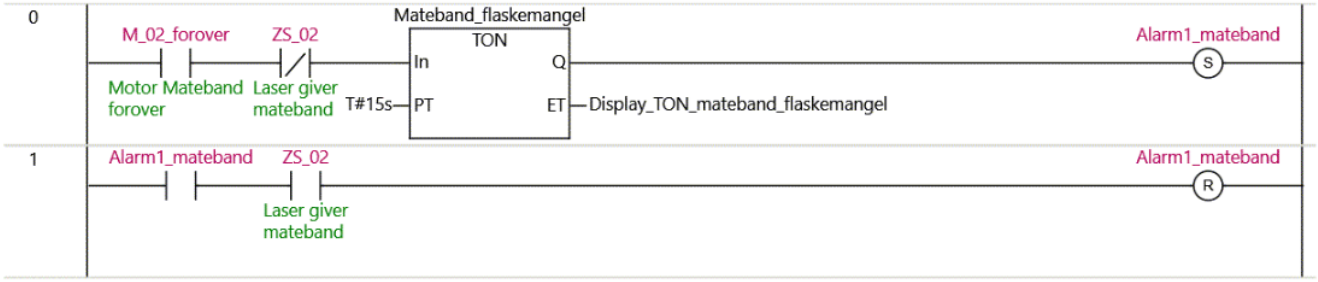
4


Standard_Weight_Setting_Start
Prosess_Data_Check
1
Standard_Weight_Check_Instance
Send_Response_Check_FB
In Out
SID SID_Error
Command Setting_Error
Command_Data Error_Code
Reset_response Setting_Finish
Calibration_Command_SID
Standard_Weight_Command
Standard_Weight_Value
Reset_kalibrering
Standard_Weight_SID_Error_flag
Standard_Weight_Setting_Error_flag
Standard_Weight_Error_Code
Standard_Weight_Setting_Finish
2
MOVE
EN ENO
In Out
BOOL#False
Standard_Weight_Setting_Start

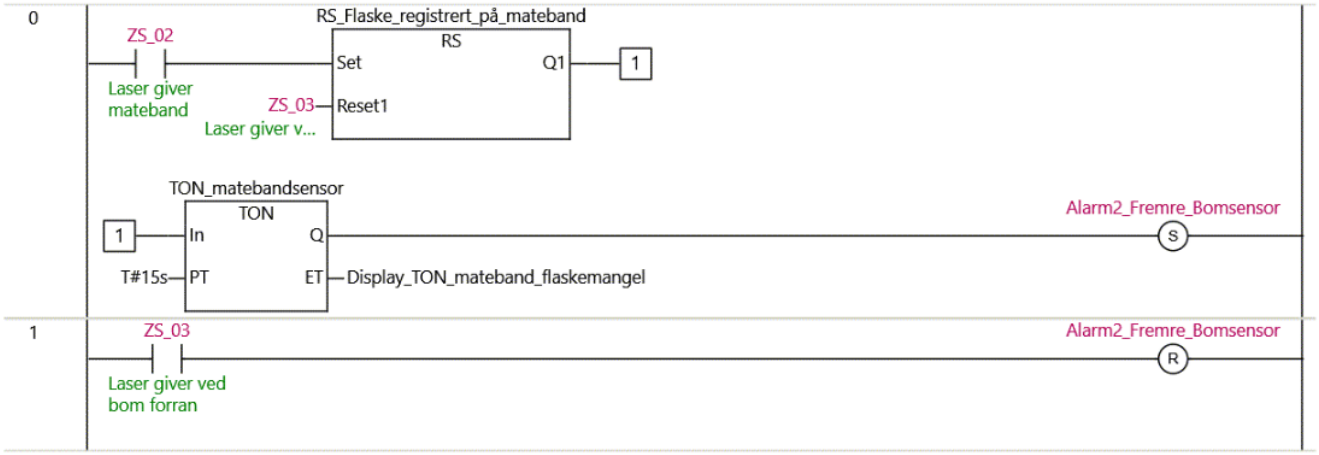

5
Zero_Calibration_Start
Prosess_Data_Check
1
Zero_Calibration_Check_Instance
Send_Response_Check_FB
1
In
Out
2
Calibration_Command_SID
SID
SID_Error
Zero_Calibration_SID_Error_flag
Zero_Calibration_Command
Command
Setting_Error
Zero_Calibration_Error_Flag
Command_Data
Error_Code
Zero_Calibration_Error_Code
Reset_kalibrering
Reset_response
Setting_Finish
Zero_Calibration_Finish
2
MOVE
EN ENO
BOOL#False
In
Out
Zero_Calibration_Start
6
Span_Calibration_Start
Prosess_Data_Check
1
Span_Calibration_Check_Instance
Send_Response_Check_FB
1
In
Out
2
Calibration_Command_SID
SID
SID_Error
Span_Calibration_SID_Error_flag
Span_Calibration_Command
Command
Setting_Error
Span_Calibration_Error_Flag
Command_Data
Error_Code
Span_Calibration_Error_Code
Reset_kalibrering
Reset_response
Setting_Finish
Span_Calibration_Finish
2
MOVE
EN ENO
BOOL#False
In
Out
Span_Calibration_Start
7
Standard_Weight_Setting_Start
Inc
EN
ENO
1
Calibration_Command_SID
InOut
InOut
Calibration_Command_SID
Zero_Calibration_Start
Span_Calibration_Start
1
=
EN
Calibration_Command_SID
In1
UINT#256
In2
MOVE
EN ENO
UINT#1
In
Out
Calibration_Command_SID
8
Standard_Weight_Setting_Finish
Zero_Calibration_Finish
Span_Calibration_Finish
Actual_Load_Calibration_Finish

1-9-1-5.Alarmer

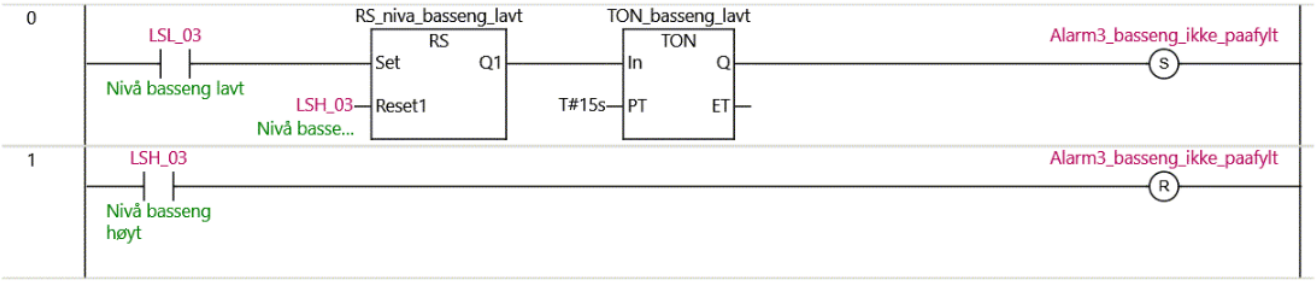
1-9-1-5-2.Alarm_Mateband



1-9-1-5-3.Alarm_Fremre_Bom



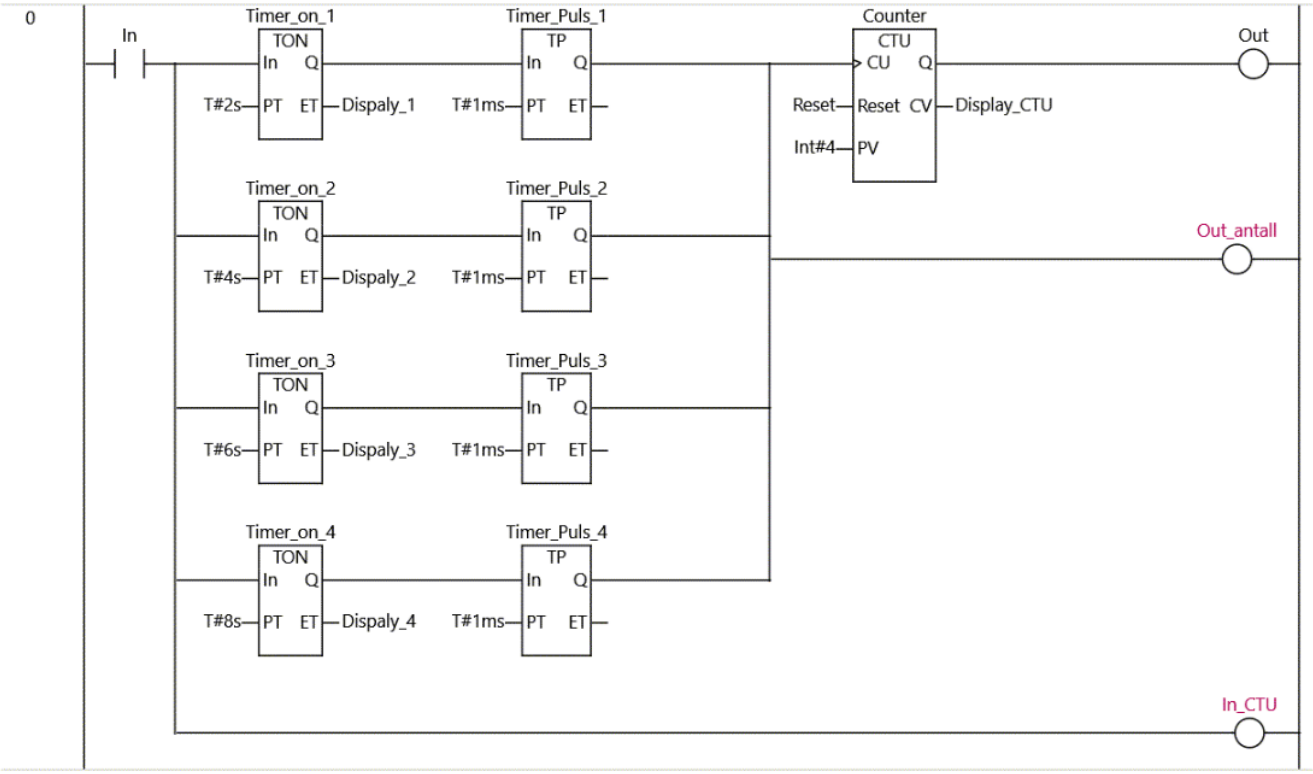
1-9-1-5-4.Alarm_Basseng



1-9-2.Function Blocks

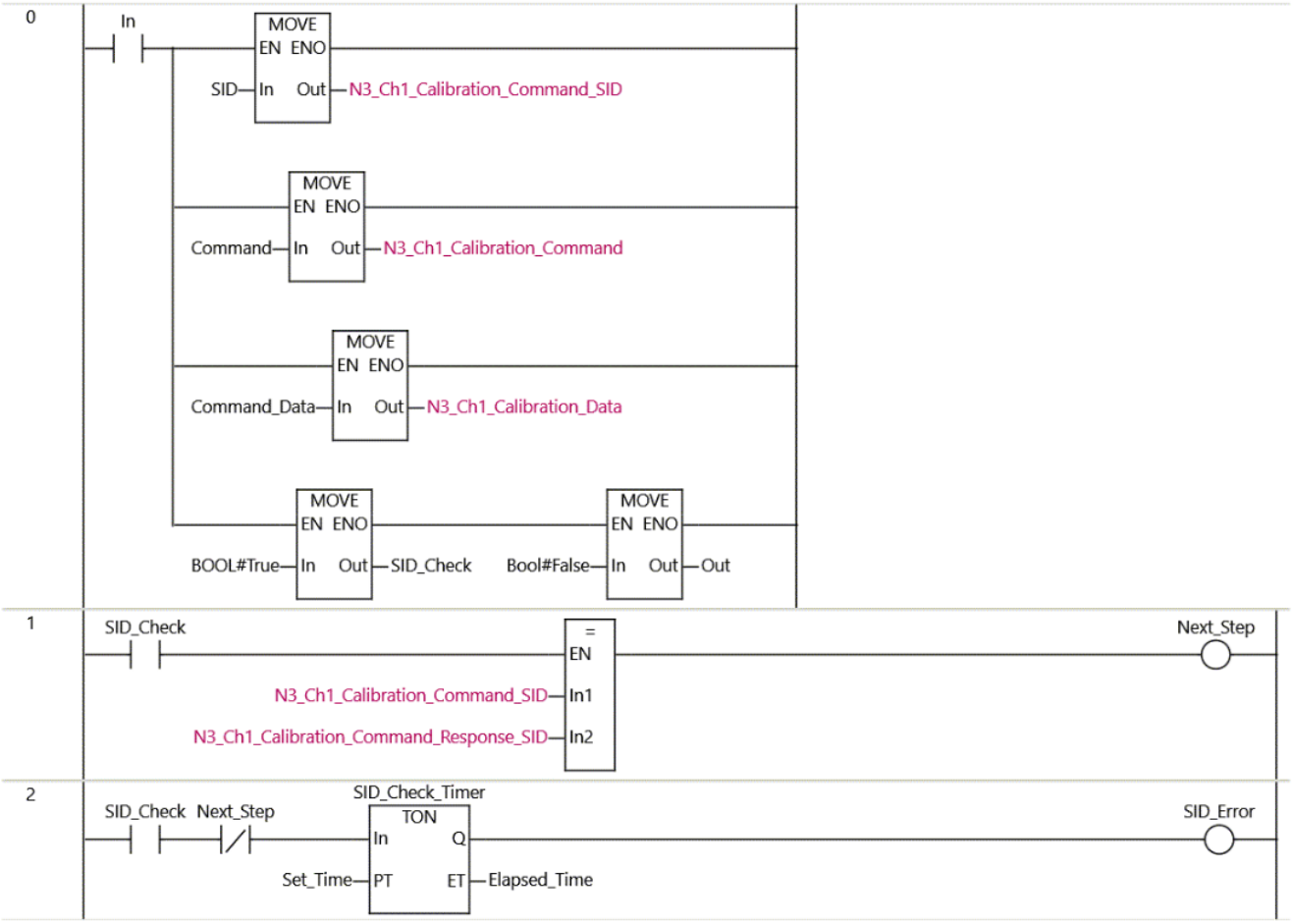
1-9-2-1.Counter_flasker

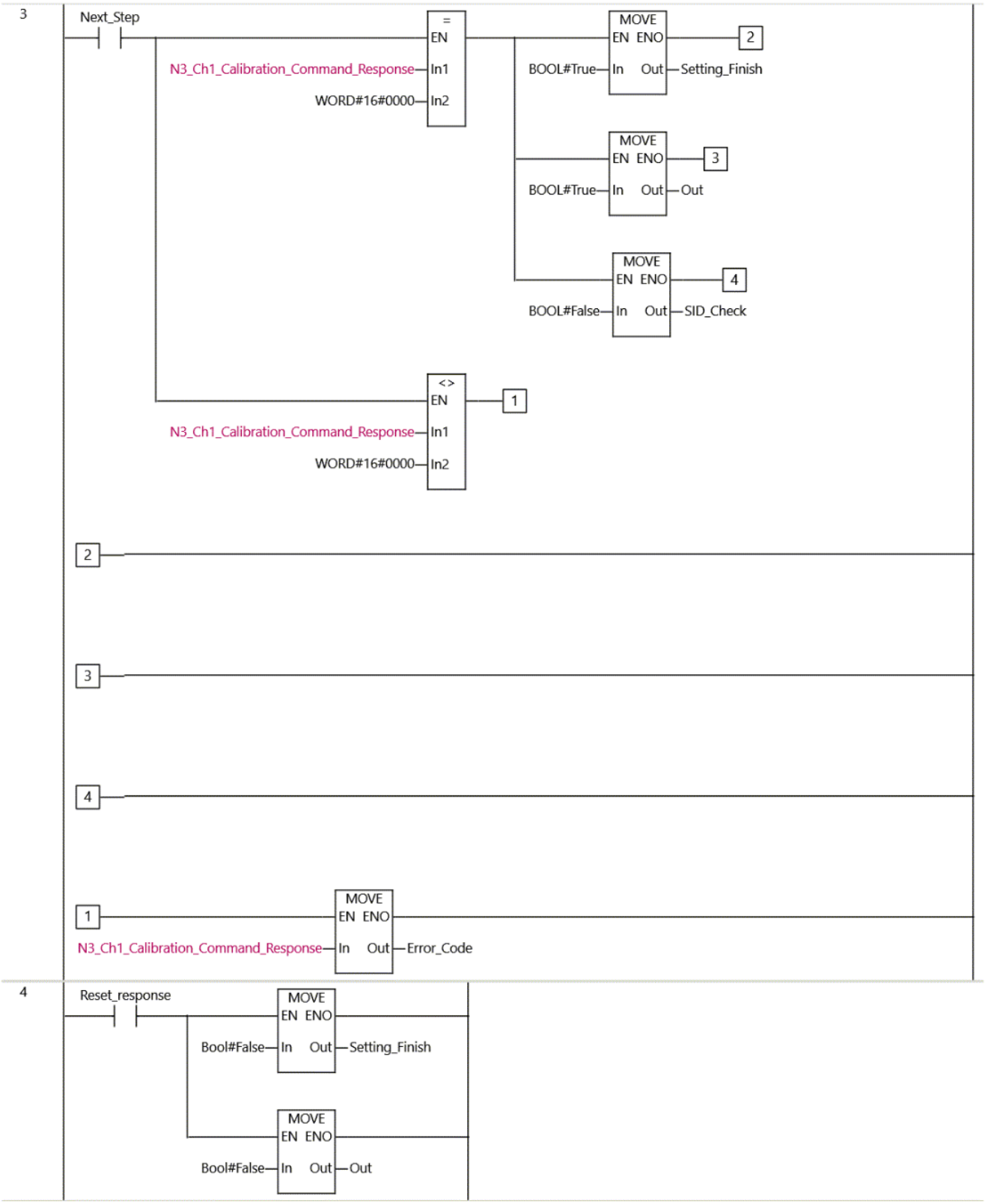
1-9-2-1-2.LadderBody



1-9-2-2.Send_Response_Check_FB

1-9-2-2-2.LadderBody




3
Next_Step
=
EN
N3_Ch1_Calibration_Command_Response
In1
WORD#16#0000
In2
MOVE
EN ENO
2
BOOL#True
In Out
Setting_Finish
MOVE
EN ENO
3
BOOL#True
In Out
Out
MOVE
EN ENO
4
BOOL#False
In Out
SID_Check
<>
EN
1
N3_Ch1_Calibration_Command_Response
In1
WORD#16#0000
In2
2
3
4
1
MOVE
EN ENO
N3_Ch1_Calibration_Command_Response
In Out
Error_Code
4
Reset_response
MOVE
EN ENO
Bool#False
In Out
Setting_Finish
MOVE
EN ENO
Bool#False
In Out
Out