med svart bakgrunn dette blir gjort med hjelp av pyplot som er en klasse fra matplotlib som er en klasse i python biblioteket. Resultatet kan variere basert på mange forskjellige grunner som hvor nært ansiktet er kamera og belysning.

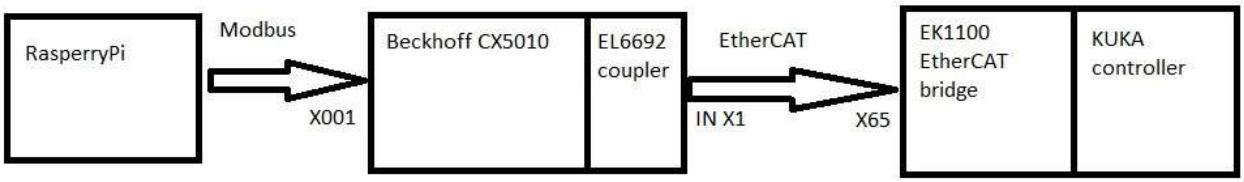
Eksempel på ferdig prosessert bilde

3.3.1.6 Lage G-kode

For å kunne lage koordinater som skal videresendes til roboten har vi laget en egen klasse. I koden blir hver pixel sjekket for farge, og bare hvite pixler blir lagret. X og Y koordinatene til pixlene blir så lagret i separate "hashmap". Så blir det sjekket for kontinuerlige pixler i Y retning som blir lagret som vektorer for å redusere antall koordinater som må sendes til robot. For å gjøre vektor delen av koden enklere lastet vi ned et eget bibliotek for java som inneholder en type Map "LinkedMap()" som kan hente ut første og siste "entry" i mapet.

3.3.2 Kommunikasjon

Topologi:



3.3.2.1 Modbus i RaspberryPi

For å kunne skrive og lese data mellom Raspberry Pi og Beckhoff PLS'en valgte vi å importere et java bibliotek som heter EasyModbusTCP. Dette biblioteket har ferdige metoder for read/write funksjoner med mer. Dette gjorde at modbus koden fikk en fin og lett forståelig struktur. Koden leser en variabel fra PLS'en og dersom den har riktig verdi, sender den en x og en y koordinat som to separate variabler og en variabel som skal indikere at koordinatene er sendt. I java koden leser vi register fra PLS'en der vi må angi hvilken adresse i PLS'en vi skal lese og hvor mange register. For å skrive en verdi til PLS'en må vi også angi modbus adressen og en integer verdi som skal sendes.

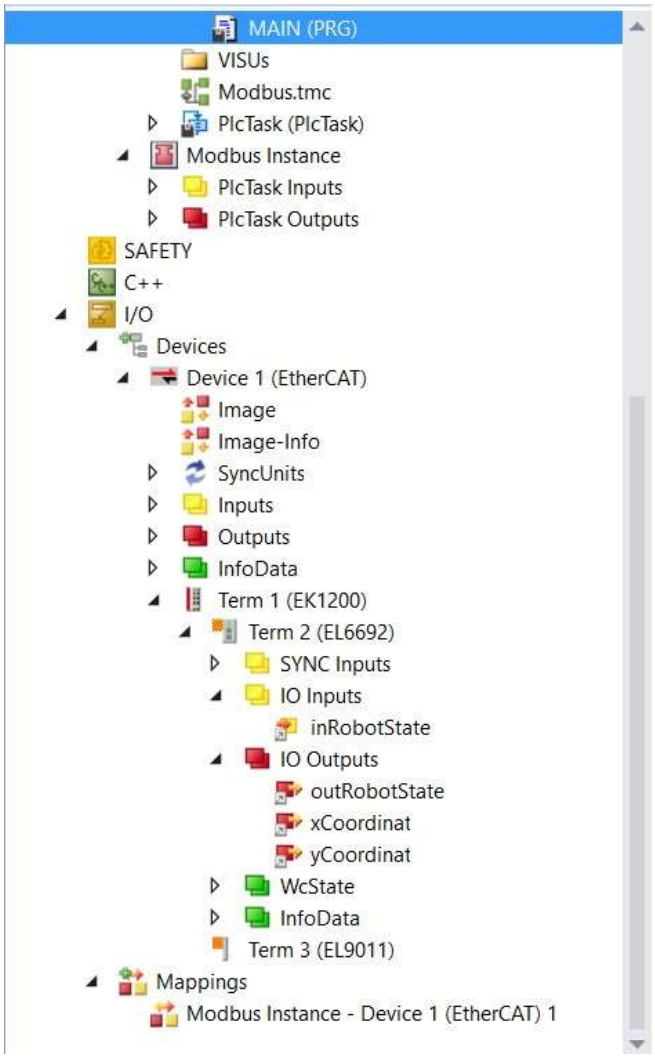
3.3.2.2 Modbus i Beckhoff PLS

Beckhoff pls støtter ikke modbus/TCP som standard. Derfor måtte vi installere et tillegg i PLS'en som støttet bruk av modbus. Dette tillegget fant vi på beckhoff sine sider på internett, lastet den ned og deretter førte det over på minnekortet på PLS'en og installerte det. Dette gjør at du kan deklarere variablene i twinCAT3 som modbus variabler, type %MW0*. Da kan disse refereres til en start adresse i modbus standard, 12288. Disse variablene er mulig å sette inn i fra java koden i Raspberry Pi, samtidig som Raspberry Pi kan lese verdiene fra variabler som blir definert i pls koden. Vi mottok integer som datatype fra Raspberry Pi. I twinCAT3 er der en ferdig definert metode INT_TO_BYTE som gjør om integer verdiene til byte verdier, slik at vi kan sende de til roboten. På grunn av at koordinatsystemet vårt ikke overskrider 255 i verdi, slipper vi å dele opp integer verdiene i flere byte. Da blir integer verdiene lagret som samme tallverdien, men som byte.

3.3.2.3 EtherCat i Beckhoff PLS

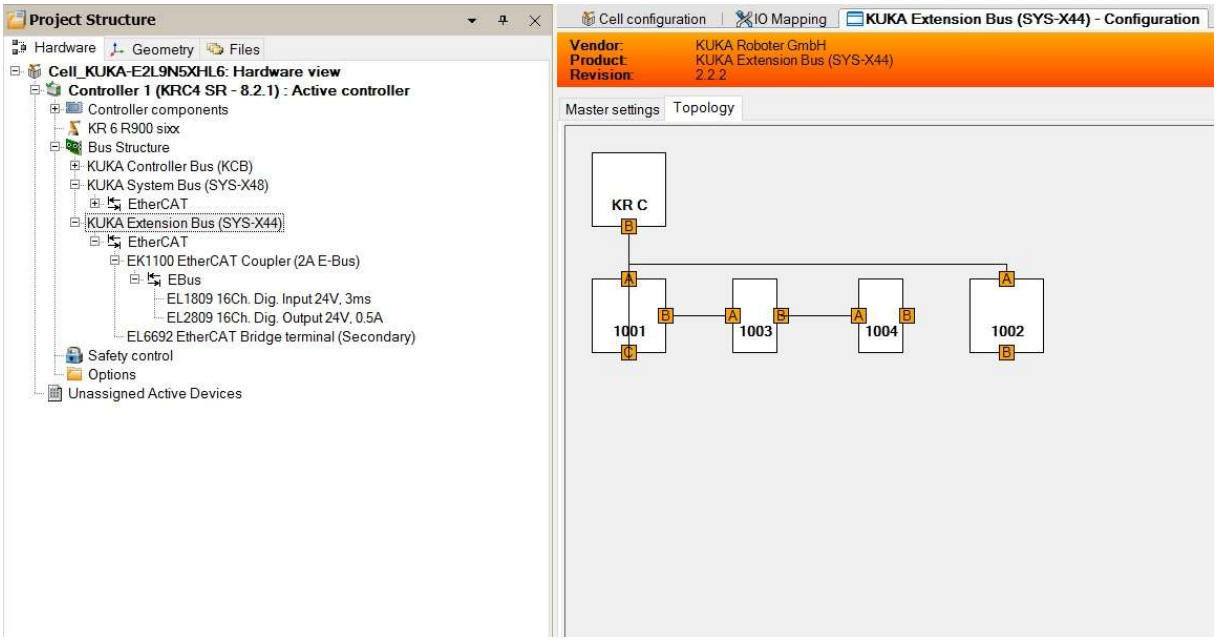
I PLS'en kjører vi to typer kommunikasjonsprotokoller. PLS'en kommuniserer med Raspberry Pi gjennom modbus, men til KUKA robot blir det brukt ethercat kommunikasjon. Grunnen til dette er at ethercat overfører data raskt, og at robot kontrolleren er satt opp for ethercat kommunikasjon som standard. På PLS'en er det koblet på en kobler(EK1200), en bro (EL6692) og en endeplate (EL9011). Dette hardware oppsettet er du nødt til å mappe i twinCAT3 slik at PLS'en vet hvordan systemet fysisk er satt opp. Du kan mappe dette på to forskjellige måter. Du kan sette det opp manuelt, eller kjøre en "scan". En scan resulterer i at PLS'en leter etter hardware som er koblet opp. Da vil device 1 komme opp. Denne må du legge til, og deretter kjøre en "scan bridge" for å få opp undermappingene (Term 1, Term 2, Term 3 i bilde under). Når dette er satt opp kan du linke variabler under EL6692 til variabler som er deklarert som %I*

og %Q* i programmet ditt. Da er disse mappet ut på ethercat bussen og er lesbare/skrivbare i robot programmet.

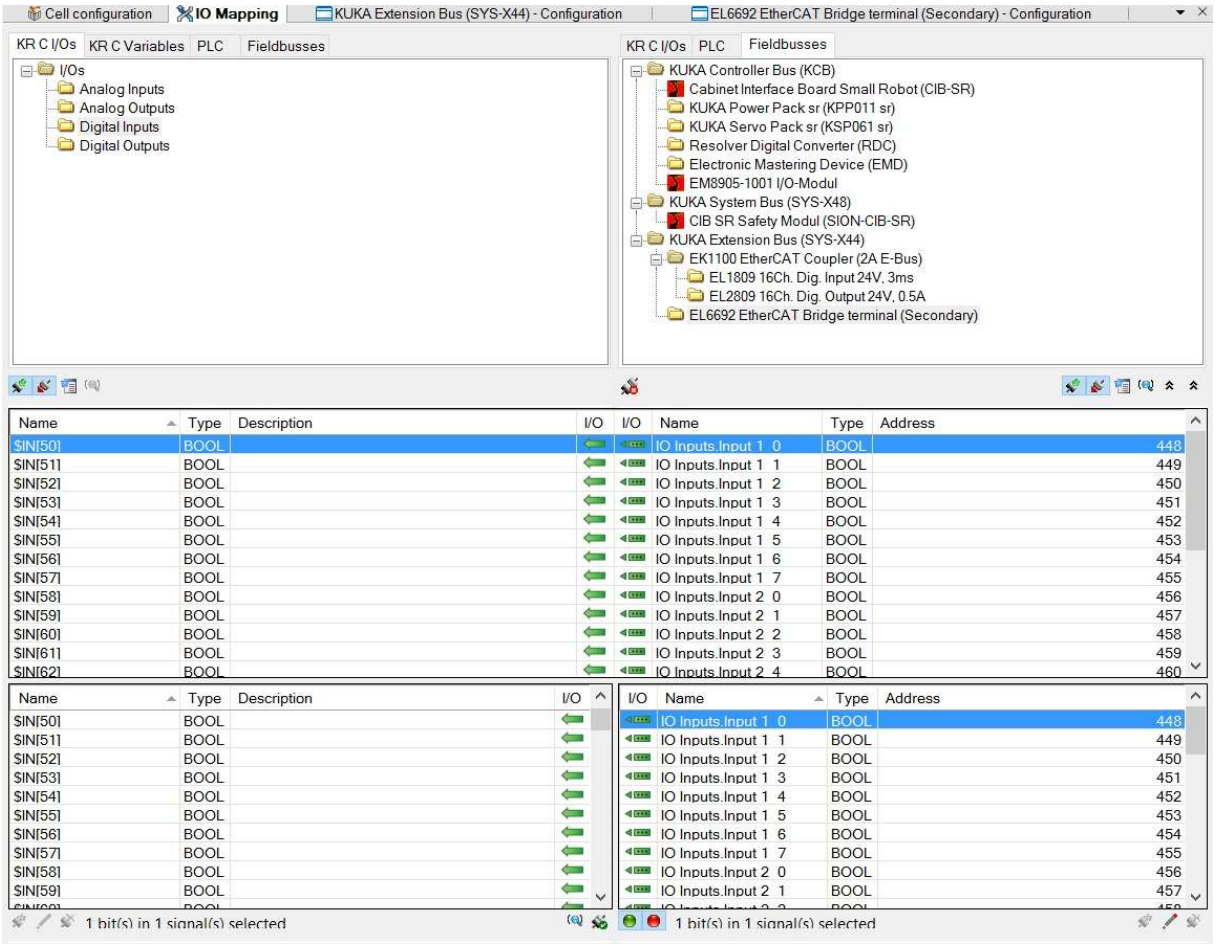


3.3.2.4 Ethercat i KUKA robot

For at KUKA roboten skal registrere tilkoblet coupler og bridge må dette bli konfigurert i WorkVisual med lik topologi som det er fysisk oppkoblet. For å gjøre dette må en ha lagt til riktig coupler og bridge fra DTM Catalogs inn i ekstern buss (x44). Her bør en laste ned nye/oppdaterte device descriptions for coupler og bridge som kan finnes på beckhoff sine sider (må importeres med å gå på File-import/export-import device description). Må og legge til andre komponenter som roboten er koblet til som kontroll komponenter og andre fieldbus som roboten bruker.



I/O mapping er en viktig del som må bli gjort i WorkVisual, for at roboten skal vite hva innganger og utganger roboten skal starte og slutte å lese på. Dette må gjøres for å kunne samhandle variabler i KUKA koden med variabler i PLS'en, som skal jobbe opp mot kvarandre. Her må en splitte opp signalene fra inngangene fra EL 6692. Her må en ha deklarere hvor mange innganger og utganger den har i bridge komponenten under eksterne busser (høgre klikk på komponenten configuration, så inn på fanen slave settings). For å utføre I/O-mapping må en inn på I/O-mapping vinduet og finne fram EL 6692 under fieldbusses. Her kan en splitte opp signalene mellom KUKA og bridge ved å velge den inngangen/utgangen som skal splitte signalet og kobles opp mot robotens innganger/utganger.



Når man er ferdig i WorkVisual må man føre over prosjektet og aktivere det i robotkontrolleren. Det gjøres ved å koble til kontrolleren med ethernet kabel på port x65. Dette vil gjøre at roboten kan lese av verdiene sendt over ethercat fra PLS'en. Variablene som blir lest av inngang/utgang må bli deklarert enten i robotens config fil eller i robot koden, dette gjøres ved å skrive SIGNAL variabelNavnInn $IN[start inngang] TO $IN[slutt inngang] for innganger og SIGNAL variabelNavnOut $OUT[start inngang] TO $OUT[slutt inngang] for utganger.

3.3.3 Simulering

Simuleringen er laget med å lage et GUI vindu i java som lager et bilde ut av hva koordinatene som skal bli sendt til KUKA. Dette er gjort med å lage en label som holder og oppdaterer bilde etter kvar piksel som blir forandret. Pikslene på bilde blir forandret med å bruke lin funksjonen til Graphics2D klassen som er en del av java biblioteket. Denne funksjonen trenger to sett av y og x koordinater, slik som vi har det i robot koden. Dette vil simulere hvordan bilde blir når roboten skal utføre oppgaven sin.

Modbus TCP fungerer akkurat som modbus, med identiske kommandoer og responser, med unntak av at modbus TCP ikke inkluderer en checksum i meldingene, da den stoler på TCP for å garantere integriteten av meldingene.

2.2.2 EtherCAT

EtherCAT er en ethernet basert feltbuss, som er utviklet av Beckhoff automasjon. EtherCAT bruker egne moduler eller enheter for å sende data. Ene siden av enheten (for.eks koblet på PLS) blir satt opp som primær, og andre siden (f.eks. robot kontroller) blir satt opp som sekundær. Når man skal sende data, må begge sidene (primær og sekundær) ha nøyaktig likt oppsett for inn og utganger. Om utgangene ikke stemmer overens, vil man få feilmelding, og ingen av dataene vil kunne sendes/mottas. Når IO'ene er satt opp riktig, vil man kunne lese og skrive data på registrene.

Den generelle måten etherCAT fungerer på, er at dataen prosesseres av EtherCAT slaven samtidig som pakken mottas. Dette i stedet for at hele datapakken må være levert før dataen skal prosesseres. På grunn av dette blir forsinkelsen minimal, og syklustider kommer typisk under 100 mikrosekund.

En av «fordelene» til EtherCAT med forskjelle mastere, er at sidene ikke har matchende adresse for innganger. På primær siden begynner input registeret på for eksempel 39, 40, ... (som er det CX5010, EL6692 bruker), mens på sekundær siden kan output registeret (som blir input for primær) starte med f.eks 12756 (som er det KUKA bruker i sitt register). I stedet for å sette matchende registeradresse, så må man altså passe på signalene har samme datatypene, og står i riktig rekkefølge på begge sidene av enheten. (Vedlegg 4)

2.2.3 TCP/IP

TCP er en transportprotokoll bygd på IP protokollen. Formålet med TCP er å abstraktere vekk den underliggende transporten av data, slik alt applikasjonen ser er en "tunell" som sender en strøm av data inn og ut. TCP garanterer at data sendt kommer fram til mottaker, og i samme rekkefølge.

2.3 KUKA programvare

2.3.1 Sunrise Workbench

Sunrise workbench er verktøyet som blir brukt for å programmere applikasjonen som skal styre LBR iiwa roboten. I workbench kan diverse sikkerhets konfigurasjoner og det generelle oppsettet til roboten gjøres. Det er i workbench funksjonaliteten til roboten blir programmert. Java blir brukt for å programmere applikasjonen som roboten skal bruke. KUKA har laget et eget Java bibliotek som implementerer funksjonene som styrer roboten. I tillegg til dette biblioteket er også de vanlige funksjonene i Java tilgjengelige. (Vedlegg 1)

2.3.2 WorkVisual

WorkVisual er KUKA sitt verktøy for å programmere roboter, men ved bruk av LBR iiwa så blir workvisual kun brukt for feltbuss konfigurering. Det er også i workvisual IO kan bli opprettet for bruk i sunrise workbench. Når en IO er opprettet kan den kobles sammen med en feltbuss adresse. Når IO og feltbuss konfigurering er ferdig så eksporteres dette til sunrise workbench slik applikasjonen har tilgang til IO.

(Vedlegg 2) (KUKA, 2017)

3.3 Raspberry PI 3

Raspberry PI 3 er en personlig datamaskin bygd rundt en firekjerners ARM-prosessor. Den kjører vanligvis en skreddersydd linux-distribusjon og brukes av mange som en billig og allsidig kontroller for prosjekter. Den har både integrert bluetooth, WiFi, Ethernet, HDMI, 2 kamera-kontakter og GPIO-pins som kan brukes for en rekke formål.

3.4 Robotiq Adaptive Gripper, C-Model

3.5 LBR iiwa 7 R800

KUKA LBR iiwa roboten er en fleksibel og lett robot som er designet for utføre nøyaktige og sensitive operasjoner. Roboten har 7 ledd som gjør den veldig fleksibel og kan derfor utføre krevende operasjoner på plassbegrensede plasser. LBR iiwa har også en rekke sensorer som føler på momentet til vært av leddene. Om roboten skulle registrere en motstandskraft som overgår en maks grense vil den stoppe. Dette åpner opp muligheten for samarbeid mellom roboten og operatøren for å utføre operasjoner sammen. Designet til roboten minsker også klemfaren mellom leddene. For enkel programmering av roboten og dens funksjoner blir roboten programmert i Java som er et objekt orientert programmeringsspråk som blir brukt verden over. (KUKA, 2017)

3.6 SmartPAD

KUKA smartPAD er roboten sitt kontrollpanel og er en vital del i oppstart testing av LBR iiwa. Ved bruk av smartPAD har operatør mulighet for å håndguide robotarmen til ønskede posisjoner som kan lagres som punkt. Punktene kan så overføres til selve applikasjonen senere. SmartPAD har ikke mulighet til å programmere eller endre applikasjonen som styrer roboten. Det er kun ved bruk av smartPAD utvikler har mulighet for å kalibrere roboten til et ønsket koordinatsystem. For å starte en applikasjon er det smartPAD som blir brukt, dette kan gjøres i forskjellige moduser. Den modusen som egner seg best til testing er T1. I denne modusen blir farten på roboten redusert og operatør må holde inne en play knapp på kontrollpanelet under kjøring. Den andre modusen er AUT, i denne modusen så blir applikasjonen startet og kan utføres i full hastighet uten at operatør gjør noe på smartPAD etter initialisering av applikasjonen. For å opprettholde sikkerheten har også smartPAD en innebygd nødstopp som kan stoppe roboten. (Vedlegg 1) (KUKA, 2017)

3.7 Stykkliste

KUKA LBR iiwa 7 R800
KUKA Sunrise.Controller
KUKA SmartPAD
Robotiq gripper Model-C
Raspberry PI 3
Raspberry PI 7'' Touchskjerm
Cisco SG110D-05 switch
Beckhoff CX5010 PLS
Beckhoff EL6692 EtherrCAT bridge
Wago 787-602 power supply
Wago 787-1002 power supply
Wago 750-8100 PLS
Wago 750-600 endemodul
Telemecanique ABL7 RE2405 power supply
Ethernet kabel 4 stykk

DIN skinne 50 CM
Sponplate 80*120 CM

4 RESULTATER

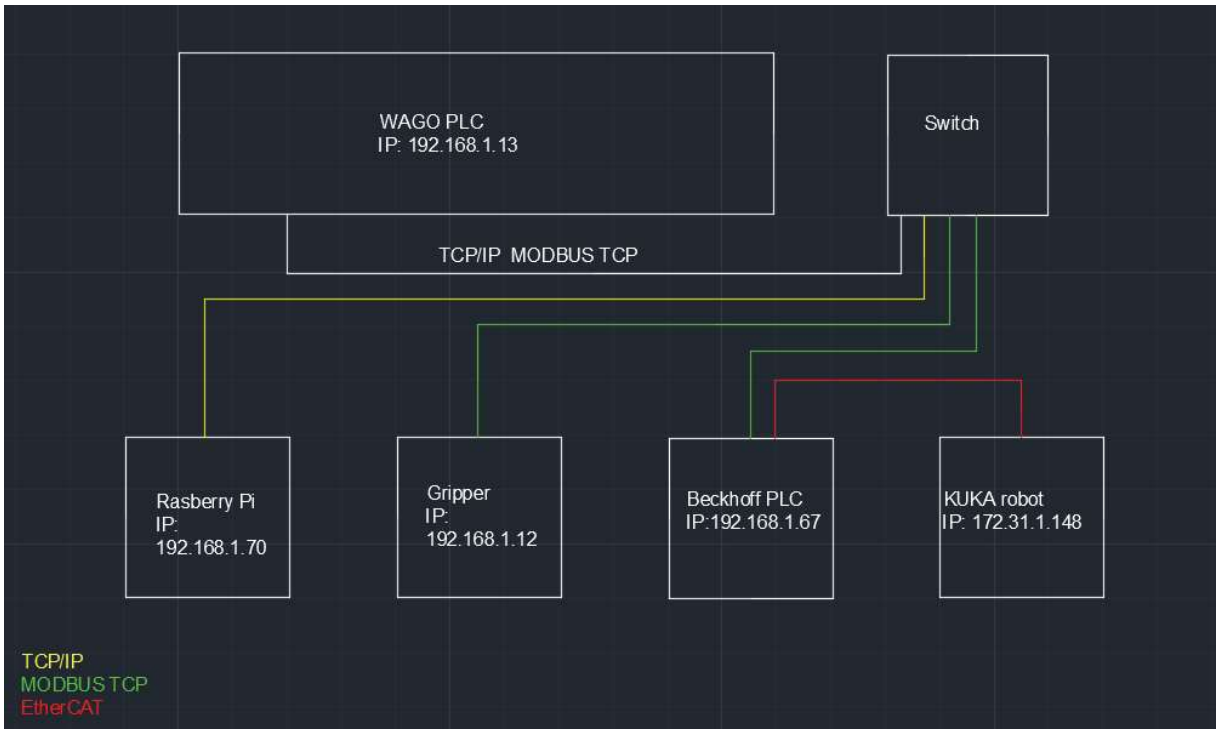


Fig 2: Topologi av kommunikasjonen i dette prosjektet

4.1 Wago 750-8100

Vi bruker en Wago 750-8100 til å styre all logikk som roboten skal utføre. Hovedprogrammet er satt opp i SFC. Det gjør at koden blir oversiktlig, og deler opp utfordringer i mindre oppgaver, som igjen gjør det lett å jobbe med koden.

Kommunikasjonen mellom WAGO'en og Beckhoff'en er MODBUS TCP. Dette konfigureres med e!cockpit i device structure menyen. Beckhoff PLS ble satt opp som en generic-slave. Dette gjør at Wagoen blir en MODBUS-master. Når det var gjort kunne vi legge til variabler og adressere dem. Vi hadde problemer med adresseringen, men dette løste seg ved å bruke absolutt adresser. Bildet under kan du se hvilke adresser som skal brukes:
Lese adresser er fra 0-31999 (0x0000-0x7CFF).
Lese skrive adresser er fra 32000-63999 (0x7D00-0x79FF).(Vedlegg 16)

9 MODBUS – *e!RUNTIME*

9.1 MODBUS Address Overview

	MODBUS Register Access	MODBUS Bit Access
PFC-OUT MODBUS-IN Size: 32000 registers	0x0000	0x0000
		Only read access FC1, FC2
		0x7FFF
	Only read access FC3, FC4, FC23, FC66	
	0x7CFF	
PFC-IN MODBUS-OUT Size: 32000 registers	0x7D00	0x8000
		Read and write access FC1, FC2, FC5, FC15
		0xFFFF
	Read and write access FC3, FC4, FC6, FC16, FC23, FC66	
	0xF9FF	
MODBUS Special registers Size: 1536 registers	0xFA00	
	Read and write access FC3, FC4, FC6, FC16, FC23, FC66	
	0xFFFF	

Figure 43: MODBUS Address Overview

Fig 3: Profibus adresse oversikt

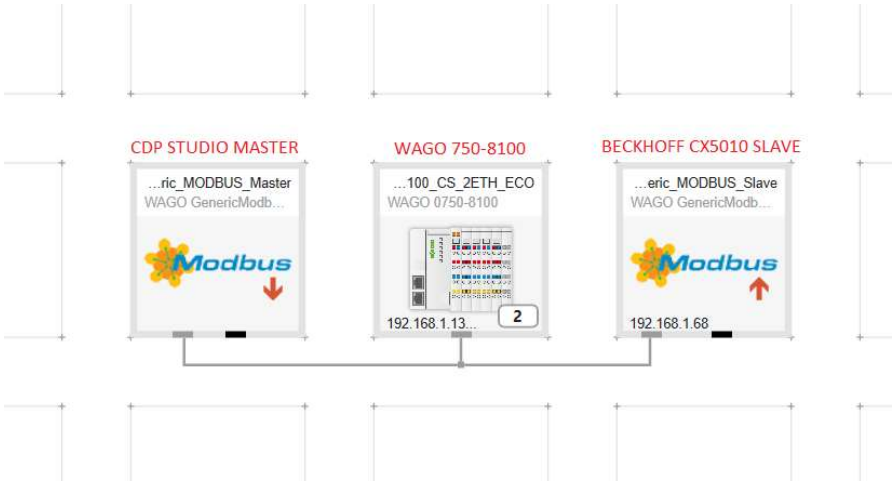


Fig 4: Modbus tilkoblingene i oversikten til e!Cockpit.

Til Raspberry PI har vi brukt TCP/IP til kommunikasjon. Her bruker vi et bibliotek fra e!cockpit som heter FbTcpServerSingleConnect.

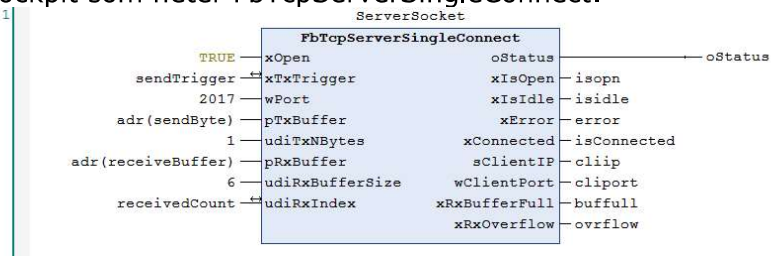


Fig 5: FbTcpServerSingleConnect

Griperen bruker modbus TCP. Her har vi brukt et bibliotek i e!cockpit som heter FbTcpClient.

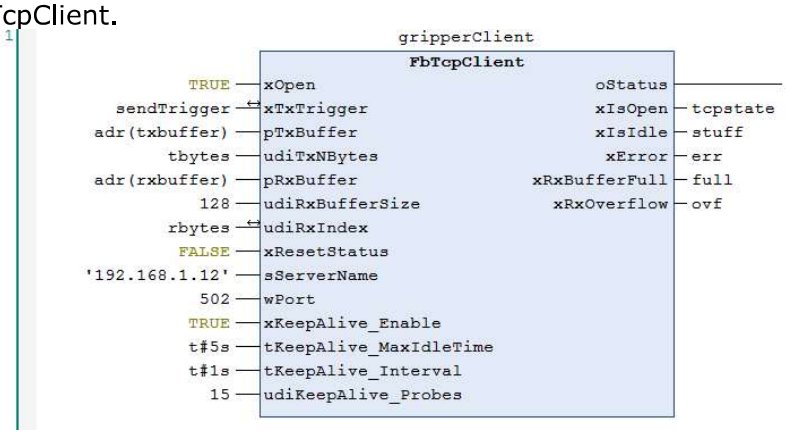


Fig 6: FbTcpClient.

Bilder av oppsettet i SFC av hovedprogrammet:

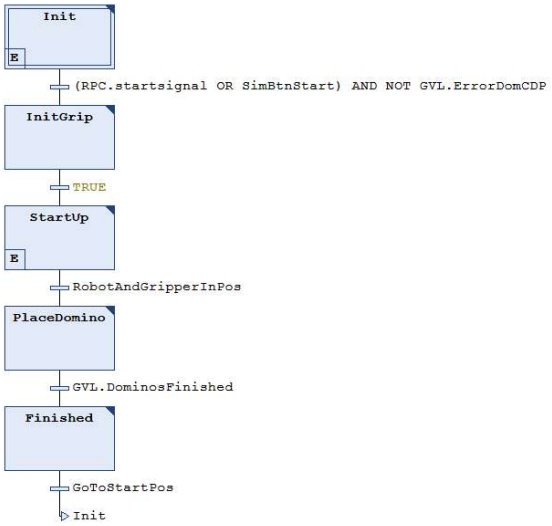


Fig 7: SFC Hovedprogram

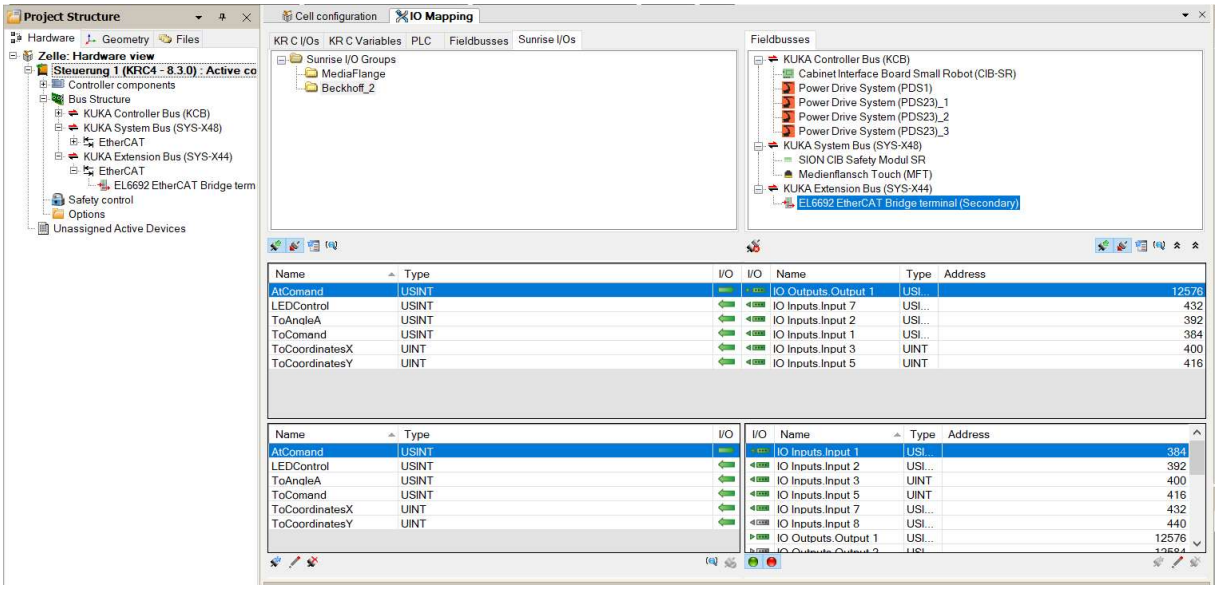


Fig 12: IO mapping i workvisual

4.2.2 Sunrise workbench

Workbench ble brukt for programmering av applikasjonen som LBR iiwa kjører. Selv om vi har noe kunnskap om java programmering, så var der mye nytt av funksjoner i biblioteket til roboten. Det krevdes derfor mye tid for å bli kjent med funksjonene og hvordan vi skulle få roboten skulle bevege seg på ønskelig måte. Det lå noen eksempel applikasjoner på roboten fra før av. Disse eksemplene gjorde det lettere å komme i gang, siden vi fikk en referanse på hvordan funksjonene fungerte.

I manualen til workbench står det beskrevet hvordan de forskjellige bevegelsesfunksjonene fungerer og hvordan de kan implementeres. Vi bestemte oss for bruk av Point To Point(PTP) og Linear(LIN) bevegelser.

Ved bruk av PTP så vil roboten rotere alle ledd for å ta den mest effektive veien til slutt punktet sitt, men ikke den korteste. Ved bruk av LIN så vil roboten bevege seg linjert fra nåværende punkt til nytt punkt, men dette kan ta lenger tid, siden den ikke bruker naturlige bevegelser.

Det er fire måter å sette sluttpunktet til roboten:

- Sette koordinater for X, Y, Z og vinker for A, B og C i en bevegelses funksjon.
- Sette vinkel for hvert ledd A1, A2, …, A7 i en bevegelses funksjon.
- Sette koordinater for X, Y, Z og vinker for A, B og C og opprette et punkt i workbench. Dette punktet kan da brukes i funksjoner.
- Bruke smartPAD for å lagre koordinatene der roboten er. Dette krever også synkronisering mellom smartPAD og workbench slik punktet blir kjent for workbench.

```
public boolean moveToTable()
{
    gripper.getFrame("/TCP1").move(ptp(getApplicationData().getFrame("/Table/P2")).setJointVelocityRel(0.60));
return true;
}
```

Fig 13: PTP bevegelse ved bruk av et ferdig satt punkt.

Roboten har ingen logikk for styringen, den utfører bare den operasjonen som kommandoen tilsier. (Vedlegg 1)

```
93          else if(IO.getToComand() == 2 && IO.getAtComand() != 2){
94              moving();
95
96              if(moveToTable())
97                  {
98                  IO.setAtComand(2);
99                  }
100         }
101         else if(IO.getToComand() == 3 && IO.getAtComand() != 3){
102             moving();
103
104             if(moveToStorage())
105                 {
106                 IO.setAtComand(3);
107                 }
108         }
109         else if(IO.getToComand() == 4 && IO.getAtComand() != 4){
110             moving();
111
112             gripper.getFrame("TCP1").move(ptp(getApplicationData().getFrame("/Table/StorageP4/P1")).setJointVelocityRel(0.30));
113
114             if(moveToCoordinates(getXCoordinates(), getYCoordinates(), getAngleA())){
115                 moveToZDrop();
116                 IO.setAtComand(4);
117             }
118         }
119
120         else if(IO.getToComand() == 5 && IO.getAtComand() != 5){
121             moving();
122
123             if(moveToZWork()){
124                 IO.setAtComand(5);
125             }
126         }
```

Fig 16: Kommandoene til roboten

4.2.3 KUKA smartPAD

SmartPAD'en er et verktøy for testing og oppstart av LBR iiwa roboten. Vi har brukt smartPAD for å kalibrere koordinatsystemet til roboten og for å lage punkt som er brukt i applikasjonen. Ved kalibrering av koordinatsystemet «table» så flyttet vi roboten manuelt til hjørnet av bordet vårt. Dette punket ble origo til «table». Fra dette punktet definerte vi positiv X akse og positiv Y akse. Dette koordinatsystemet bruker vi for å bevege roboten. Det sparer oss arbeid, siden vi får bruke vårt egendefinerte koordinatsystem.

Under testing av roboten var det praktisk at vi kunne bruke smartPAD for å se verdiene til EtherCAT IO. På denne måten kunne vi kontrollere at roboten fikk riktig signal, og at den returnerte riktig feedback i forhold til posisjonen sin (Vedlegg 1).

4.3 Beckhoff

4.3.1 CX5010

Beckhoff Cx5010 er satt opp som et mellomledd for å kunne kommunisere mellom Wago750-8100 og kontrolleren til KUKA roboten. For å kommunisere med KUKA iiwa må vi bruke EtherCAT som kommunikasjon, og vi ble bedt om å bruke Beckhoff CX5010en for det. For å kommunisere med Wago 750-8100 har vi valgt å bruke modbus-TCP.

TwinCAT 3 er IDE'en vi bruker for å kode på CX5010. CX5010'en legges inn i TwinCAT ved å velge IP-adressen dens (192.168.1.68) som target system. Når enheten var lagt til, kunne vi bruke «scan» funksjonen for å detektere tilkoblede enheter. EL6692, som er EtherCAT modulen vi har koblet til, ble da oppdaget. (Vedlegg 3)

4.3.2 Setup EtherCAT

Å konfigurere EtherCAT virket som det skulle være greit, men å få det til å fungere viste seg i stedet å bli et av de større problemene med prosjektet. Å legge til EtherCAT IO på EL6692 ble gjort ved å høyreklikke på IO og velge «add item» og deretter datatype. IOene ble da automatisk tildelt adresse i EtherCAT registeret til enheten. Etter at at IOene ble lagt til på EL6692 enheten, så kunne de knyttes opp mot IOene i GVL'en. I første omgang, ble IO listen til EtherCATen seende slik ut:

Name	Type	Size	>Ad...	In/O...	User...	Linked to
Sync Mode	BIT2	0.2	39.0	Input	0	
TxPDO toggle	BIT	0.1	40.3	Input	0	
TxPDO state	BIT	0.1	40.4	Input	0	
Control valu...	BIT	0.1	40.5	Input	0	
Timestamp u...	BIT	0.1	40.6	Input	0	
External devi...	BIT	0.1	40.7	Input	0	

Fig 17: EtherCAT Sync

Name		Type	Size	>Address	In/Out	User...	Linked to
GVL.Kuka_input_1	X	BIT	0.1	41.0	Input	0	GVL.Kuka_input_1 . PlcTask Inputs . Kuka_PLC Instance . Kuka_PLC
GVL.Kuka_input_2	X	BIT	0.1	41.1	Input	0	GVL.Kuka_input_2 . PlcTask Inputs . Kuka_PLC Instance . Kuka_PLC
GVL.Kuka_input_3	X	BIT	0.1	41.2	Input	0	GVL.Kuka_input_3 . PlcTask Inputs . Kuka_PLC Instance . Kuka_PLC
GVL.Kuka_input_4	X	BIT	0.1	41.3	Input	0	GVL.Kuka_input_4 . PlcTask Inputs . Kuka_PLC Instance . Kuka_PLC
GVL.Kuka_input_5	X	BIT	0.1	41.4	Input	0	GVL.Kuka_input_5 . PlcTask Inputs . Kuka_PLC Instance . Kuka_PLC
GVL.Kuka_input_6	X	BIT	0.1	41.5	Input	0	GVL.Kuka_input_6 . PlcTask Inputs . Kuka_PLC Instance . Kuka_PLC
GVL.Kuka_input_7	X	BIT	0.1	41.6	Input	0	GVL.Kuka_input_7 . PlcTask Inputs . Kuka_PLC Instance . Kuka_PLC
GVL.Kuka_input_8	X	BIT	0.1	41.7	Input	0	GVL.Kuka_input_8 . PlcTask Inputs . Kuka_PLC Instance . Kuka_PLC

Fig 18: Beckhoff EtherCAT Input

Name		Type	Size	>Address	In/Out	User...	Linked to
GVL.Kuka_output_1	X	BIT	0.1	39.0	Output	0	GVL.Kuka_output_1 . PlcTask Outputs . Kuka_PLC Instance . Kuka_PLC
GVL.Kuka_output_2	X	BIT	0.1	39.1	Output	0	GVL.Kuka_output_2 . PlcTask Outputs . Kuka_PLC Instance . Kuka_PLC
GVL.Kuka_output_3	X	BIT	0.1	39.2	Output	0	GVL.Kuka_output_3 . PlcTask Outputs . Kuka_PLC Instance . Kuka_PLC
GVL.Kuka_output_4	X	BIT	0.1	39.3	Output	0	GVL.Kuka_output_4 . PlcTask Outputs . Kuka_PLC Instance . Kuka_PLC
GVL.Kuka_output_5	X	BIT	0.1	39.4	Output	0	GVL.Kuka_output_5 . PlcTask Outputs . Kuka_PLC Instance . Kuka_PLC
GVL.Kuka_output_6	X	BIT	0.1	39.5	Output	0	GVL.Kuka_output_6 . PlcTask Outputs . Kuka_PLC Instance . Kuka_PLC
GVL.Kuka_output_7	X	BIT	0.1	39.6	Output	0	GVL.Kuka_output_7 . PlcTask Outputs . Kuka_PLC Instance . Kuka_PLC
GVL.Kuka_output_8	X	BIT	0.1	39.7	Output	0	GVL.Kuka_output_8 . PlcTask Outputs . Kuka_PLC Instance . Kuka_PLC
GVL.Kuka_output_9	X	WORD	2.0	41.0	Output	0	GVL.Kuka_output_9 . PlcTask Outputs . Kuka_PLC Instance . Kuka_PLC
GVL.Kuka_output_10	X	WORD	2.0	43.0	Output	0	GVL.Kuka_output_10 . PlcTask Outputs . Kuka_PLC Instance . Kuka_PLC

Fig 19: Beckhoff EtherCAT Output

For at EtherCAT forbindelsen skulle fungere, måtte IOene være identisk i oppsett på begge sidene av EtherCAT enheten (primær/sekundær). Det vil si at inputen på primær siden er identisk til outputen på sekundær siden, og vice versa. Problemet vårt var at selv om oppsettet var identisk, så fikk vi en «invalid IO configuration» feilmelding fra KUKA smartpadden, og ingen av IOene kunne leses. Oppsettet på sekundær/KUKA siden av EL6692 så slik ut.

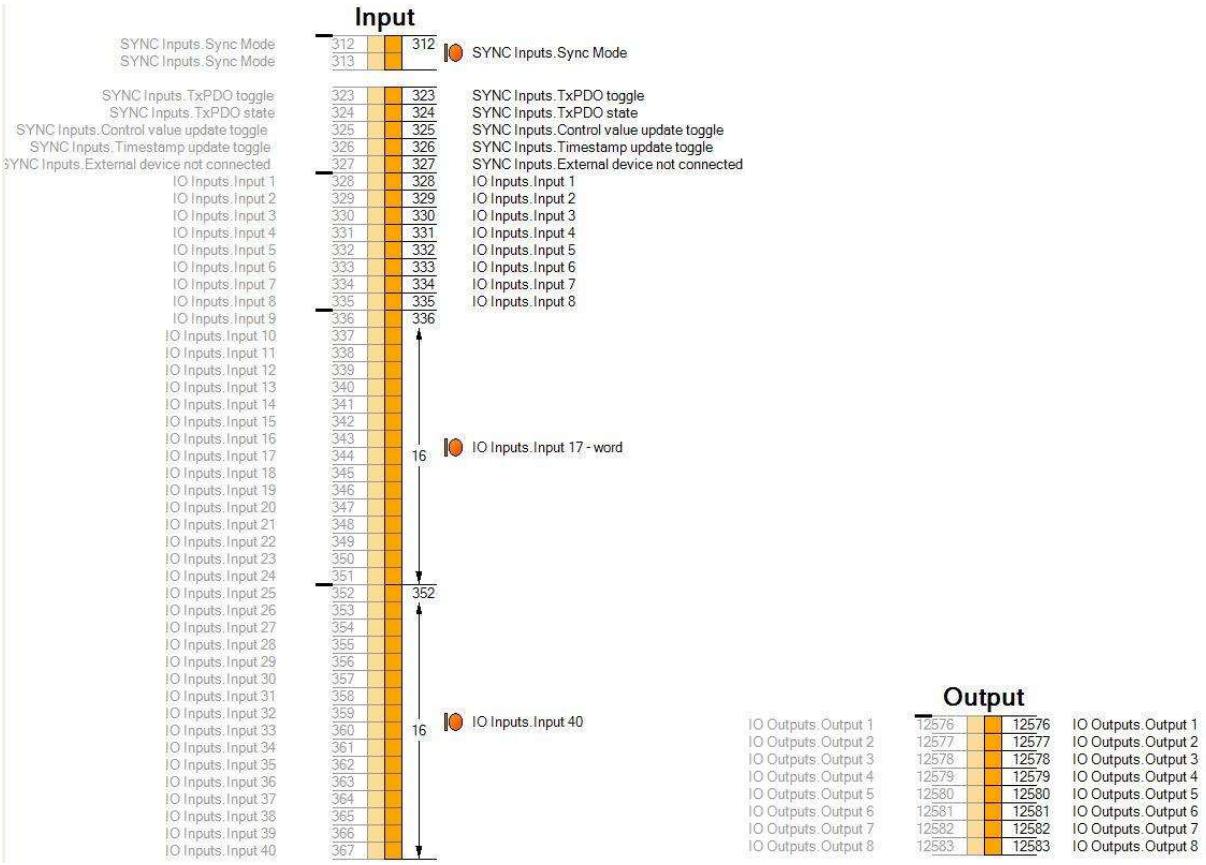


Fig 20: WorkVisual EtherCAT IO mapping.

Etter å ha kontaktet beckhoff support, og fått kontrollert ved bruk av teamviewer at oppsettet var riktig (selv om vi hadde feilmelding), så var vi veldig usikre på hva som ga feil. Til slutt endte det opp med at vi prøvde en anbefaling vi fikk av support for generell bruk av EtherCAT. Anbefalingen var å sende alle signaler som Unsigned integer (UINT / USINT). For å bruke andre datatyper, må de da oversettes i koden. For å bruke UINT/USINT, så endret vi bool verdiene våre til en command verdi, av typen USINT. Siden kun en av kommandoene skulle være aktiv av gangen, så passet det bra å gjøre det slik.

Etter at vi endret datatype, så forsvant feilmeldingen. Da var kommunikasjonen med KUKA roboten i orden. Den endelige IO listen for EtherCAT i TwinCAT ble da seende slik ut. (Vedlegg 4)

Name		Type	Size	>Ad...	In/Out	User...	Linked to
GVL.Kuka_I_WorkStatus	X	USINT	1.0	41.0	Input	0	GVL.Kuka_I_WorkStatus . PlcTask Inputs . Kuka_PLC Instance . Kuka_PLC
Unused_2		USINT	1.0	42.0	Input	0	
Unused_3		USINT	1.0	43.0	Input	0	
Unused_4		USINT	1.0	44.0	Input	0	
Unused_5		USINT	1.0	45.0	Input	0	
Unused_6		USINT	1.0	46.0	Input	0	
Unused_7		USINT	1.0	47.0	Input	0	
Unused_8		USINT	1.0	48.0	Input	0	

Fig 21: Beckhoff EtherCAT input

Name		Type	Size	>Ad...	In/Out	User...	Linked to
GVL.Kuka_O_Command	X	USINT	1.0	39.0	Output	0	GVL.Kuka_O_Command . PlcTask Outputs . Kuka_PLC Instance . Kuka_PLC
GVL.Kuka_O_ToAngleA	X	USINT	1.0	40.0	Output	0	GVL.Kuka_O_ToAngleA . PlcTask Outputs . Kuka_PLC Instance . Kuka_PLC
GVL.Kuka_O_ToCoordin...	X	UINT	2.0	41.0	Output	0	GVL.Kuka_O_ToCoordinatesX . PlcTask Outputs . Kuka_PLC Instance . Kuka_PLC
GVL.Kuka_O_ToCoordin...	X	UINT	2.0	43.0	Output	0	GVL.Kuka_O_ToCoordinatesY . PlcTask Outputs . Kuka_PLC Instance . Kuka_PLC
GVL.Kuka_O_Lights	X	USINT	1.0	45.0	Output	0	GVL.Kuka_O_Lights . PlcTask Outputs . Kuka_PLC Instance . Kuka_PLC
Unused		USINT	1.0	46.0	Output	0	

Fig 22: Beckhoff EtherCAT output

4.3.3 Modbus TCP – Beckhoff

TwinCAT 3 har lite ekstra innebygde ekstrafunksjoner, men har i stedet utvidelser for så godt som alt. For modbus TCP måtte vi installere «TF6250 | modbus TCP». Utvidelsen tillater setup av modbus tilkobling eller modbus server på CX5010. Den fungerer ved at man kan linke en modbus adresse til lokalminnet på enheten.

Når «TF6250» utvidelsen var installert, så ble 3 mapper for videre installasjon på forskjellige enheter lagt inn lokalt på PC driven. CE-ARMV4I for modbus på ARM baserte enheter, Win32 for modbus på Windows, og CE-x86 for modbus på Beckhoffs egne x86 chipset enheter.

CX5010 kan kobles til skjerm og tastatur/mus for setup. Når modbus utvidelsesfilen lå på enheten, så installerte vi den ved å kjøre den via RUN/CMD. TF6250/modbus ble installert under «harddisk/TwinCAT/functions/TF6250» og kjøres nå når enheten startes opp. Siste steget for setup av modbus var konfigurering av TF6250.xml filen for setup av register adresser. TF6250.xml kjøres også via RUN/CMD på selve enheten. (Vedlegg 3)

Vi leser og skriver fra read/write registeret, som er på modbus adresse 0x8000. Denne adressen linket vi opp mot index gruppe 16416, som da vil tilsvarer PLC minnet området %M. Det defineres på denne måten i TF6250.XML filen.

```
15      <OutputRegisters>
16          <MappingInfo>
17              <AdsPort>851</AdsPort>
18              <!-- Modbus Address 32768 = 0x8000  -->
19              <StartAddress>0</StartAddress>
20                  <EndAddress>639</EndAddress>
21                  <!-- IndexGroup 16416 = 0x4020 -> plc memory area %M -->
22                  <IndexGroup>16416</IndexGroup>
23                  <IndexOffset>0</IndexOffset>
24              </MappingInfo>
25      <MappingInfo>
```

Fig 23: TF6250 Modbus XML config på CX5010

Programmet i TwinCAT brukes kun for å sende data videre. Modbus TCP brukes for kommunikasjon med Wago PLSen. Mens EtherCAT brukes for å kommunisere med Kuka roboten. Vi laget også et testprogram for å lese alle inn verdier, og for å kunne manuelt sette out verdier, for test funksjonalitet. IOene for EtherCAT settes opp som vanlige IO og linkes til utganger, mens modbus IOene skrives rett i registeret. IOene ser slik ut:

```
VAR_GLOBAL
//Kuka Input variables
Kuka_I_WorkStatus AT %I* : USINT;

//Kuka Output variables
Kuka_O_Command AT %Q* : USINT;
Kuka_O_ToAngleA  AT %Q*: USINT;
Kuka_O_ToCoordinatesX AT %Q* : UINT;
Kuka_O_ToCoordinatesY AT %Q* : UINT;
Kuka_O_Lights AT %Q* : USINT;

//Wago output variables
Wago_O_KukaStatus AT %MW0 : USINT;

//Wago Input Variables
Wago_I_Command AT %MW1 : USINT;
Wago_I_ToCoordinatesX AT %MW2 : UINT;
Wago_I_toCoordinatesY AT %MW4 : UINT;
Wago_I_ToAngleA AT %MW6 : USINT;
wago_I_Lights AT %MW8 : USINT;

END_VAR
```

Fig 23: Beckhoff GVL

4.4 Griper

4.4.1 Robotiq gripper

Griperen som er festet på enden av robotarmen brukes for å gripe domino, den gir også feedback på om den holder en domino eller ikke. Hvor mye griperen skal åpne/lukke seg blir styrt fra wago PLS ved bruk av modbus TCP kommunikasjon. I tillegg kan hastigheten på lukkingen og åpningen justeres, samt hvor stor kraft griperen skal bruke før den stanser. På den måten kan vi justere hvor hardt den skal gripe.

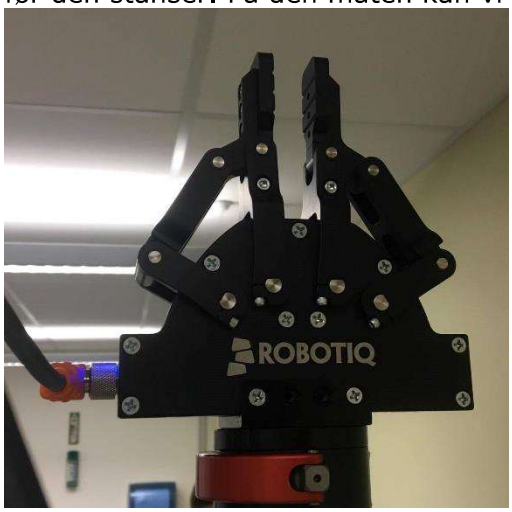


Fig 24: Bilde av robotiq griper model-C

4.4.2 Registere

Robotiq-griperen styres ved å lese og skrive til 3 registere på griperen ved hjelp av modbus TCP protokollen. Registerene, som er på 16 bit hver, blir brukt som 2 separate

bytes med forskjellig funksjon. Vi kaller da register *n* sin lavere byte for *n*.L og øvre byte for *n*.H. (0.L, 0.H, 1.L, …)

Noen av registerene kan leses og skrives til av både kontrolleren og av griperen. Vi skiller mellom funksjonen til registeret når det leses fra og når det skrives til. Dette er funksjonen til registeret når det blir skrevet til:

Register	**Funksjon**
0.L	"Action request", her kan flagg settes for å aktivere griperen og for å trigge en bevegelse.
0.H	Listet som reservert i dokumentasjonen
1.L	Listet som reservert i dokumentasjonen
1.H	Posisjon, her skrives et tall mellom 0 og 255 som spesifiserer hvilken posisjon griperen skal gå til
2.L	Fart, her skrives et tall mellom 0 og 255 som spesifiserer hvor fort griperen skal bevege seg
2.H	Kraft, her skrives et tall mellom 0 og 255 som sier hvor hardt griperen skal lukke/åpne, lavere tall vil få griperen til å stanse ved lavere motstand

Griperen kan også gi informasjon tilbake til kontrolleren ved å lese registerene, registerene vil da ha følgende funksjon/informasjon:

Register	**Funksjon**
0.L	Status-register, her blir flagg satt for å indikere om griperen er aktivert, om den kjører eller står stille, og om den møtte motstand under forrige åpning/lukking. Dette brukes for å teste om griperen har grepet en domino eller ikke.
0.H	Listet som reservert i dokumentasjonen
1.L	Feilmelding, her kan feilstatus leses, blir satt hvis griperen ikke er aktivert når den mottar en kommando, eller om andre feil har oppstått.
1.H	Posisjonen griperen har blitt bedt om å gå til
2.L	Den faktiske posisjonen til griperen
2.H	Hvor mye strøm som for øyeblikket er i bruk for å drive motorene i griperen, gitt i centiampere

4.4.3 Styring

Vi støtte på problemer når vi prøvde å bruke mer enn en link med standardimplementasjonen av modbus TCP. Derfor implementerte vi et skjelett av modbus TCP-protokollen, som lot oss skrive og lese de tre registerene beskrevet over. Dette er implementert som en TCP-blokk som sender et array som fylles med data fra en av to funksjonsblokker. En blokk fyller den med data som skal skrives, og den andre fyller den med en kommando som ber om å få lese registerene. Her kan vi se hvordan skrive-kommandoen blir fylt i den ene blokken~~, vi~~. Vi setter de to første bytene, som er en meldings-ID, med et pseudo-tilfeldig tall, og resten er hardkodet med unntak av det