Høgskulen på Vestlandet

BACHELOROPPGAVE

Standardisering på tvers av PLS-leverandører

Standardization across PLC-providers

Jo Martin Slåtten

Nikolai Abrahamsen Urne

Automatiseringsteknikk med robotikk
FIN/Førde/AUT
Joar Sande
21. mai 2024

Vi bekrefter at arbeidet er selvstendig utarbeidet, og at referanser/kildehenvisninger til alle kilder som er brukt i arbeidet er oppgitt, *jf. Forskrift om studium og eksamen ved Høgskulen på Vestlandet, § 12-1.*

Dokumentkontroll

Rapportens tittel:	*Dato/Versjon*
BO24EF-04 Standardisering på tvers av PLS-leverandører BO24EF-04 Standardization across PLC-providers	20. mai. 2024/0.20
	Rapportnummer: BO24EF-04
Forfatter(e): Jo Martin Slåtten Nikolai Abrahamsen Urne	*Studieretning:* AUT24
	Antall sider m/vedlegg 57
Høgskolens veileder: Joar Sande	*Gradering:* Åpen
Eventuelle Merknader: Vi tillater at oppgaven kan publiseres.	

Oppdragsgiver:	*Oppdragsgivers referanse:*
Goodtech ASA	Johannes Møgster
Oppdragsgivers kontaktperson(er) (inklusiv kontaktinformasjon): *Johannes Møgster, telefon: 995 53 866, e-post: Johannes.mogster@goodtech.no*	

Revisjon	**Dato**	**Status**	**Utført av**
0.10	09.03	Oppretting av bachelorrapport.	Jo Martin og Nikolai
0.11	20.03	Revidert utkast	Jo Martin og Nikolai
0.12	25.03	Skrevet på sammendrag, strukturering og progresjon og utforming av mulige løsninger	Jo Martin og Nikolai
0.13	11.04	Skrevet punkt 6. Analyse av problemet	Jo Martin og Nikolai
0.14	22.04	Skrevet punkt 6. Realisering av valgt løsning. Laget avsnittet 7. Testing. Laget avsnittet 8. Diskusjon. Skrevet mye om 5. Fundamentale forskjeller.	Jo Martin og Nikolai
0.15	23.04	Klargjort rapporten for innsending til veileder. B.2 Prosjektorganisering laget	Jo Martin og Nikolai
0.16	25.04	Skrevet siste avsnitt sammenheng med kravspesifikasjonen, GitHub Runner.	Jo Martin og Nikolai
0.17	08.05	Ferdigstilt og klar for siste gjennomgang av veileder.	Jo Martin og Nikolai
0.18	10.05	Rettskriving og skrevet siste avsnitt om LineIDs Laget brukermanualer, Appendiks C for GitHub Runner og brukergrensesnittet.	Jo Martin og Nikolai
0.19	19.05	Justering av figurer og kilder. Sørget for at alle samsvarer med den angivelige teksten. Rettskriving av dokument. Lagt til Appendiks A (ordforklaringer).	Jo Martin og Nikolai
0.20	20.05	Endelig versjon av rapport	Jo Martin og Nikolai

Forord

Denne rapporten dokumenterer vårt bachelorprosjekt. Oppgaven er levert av Goodtech, og laget for studenter som går Automatiseringsteknikk med robotikk på Høgskulen på Vestlandet.

Dette prosjektet har vært en lærerik reise, som har gitt oss dypere forståelse innen programmering, samarbeid og problemløsning. Vi har utført prosjektplanlegging ved hjelp av arbeidsplaner og fremdriftsplaner, som har vært til stor nytte for organisering av arbeidshverdagen. Vi har fått erfare hvordan det er å samarbeide med en arbeidsgiver og bruke deres erfaringer for å oppnå et best mulig resultat.

Vi vil gjerne takke Olav Sande for all hjelp han har bistått med. Hjelpen har vært til stor nytte til oppbygging av prosjektet og hva som skal prioriteres. Han kommer med mye nyttig informasjon og erfaringer som vi kan bruke og ta med oss videre.

Ikke minst en stor takk til vår oppdragsgiver, Johannes Møgster, som har laget en slik utfordrende, men spennende oppgave. Kommunikasjonen har vært veldig god. Vi utvekslet forbedringer, ulike prosjektideer og planlagt møter. Tilgjengeligheten har vært til stor hjelp når vi har spørsmål eller sitter fast med ulike problemer. Han har bistått med problemløsing utenfor arbeidstider, noe som har vært til stor fordel for progresjonen av prosjektet.

Sammendrag

I våren 2024 valgte vi å gå inn i et samarbeid med Goodtech AS for å utvikle en applikasjon knyttet til vedlikehold av programmeringsbiblioteker for Programmerbar Logisk Styring (PLS) som samsvarer til IEC 61131-3 standarden. Å automatisere konverteringsprosessen på tvers av leverandører, vil resultere i besparelser i tid og ressurser for Goodtech AS.

Etter en nøye vurdering av generelle forskjeller mellom PLS-systemene, med utgangspunkt i de eksisterende Monitoring Binary (MB) og Switch Binary Electric (SBE) blokkene, er målet å adressere forskjellene for å videre muliggjøre kompatibilitet på tvers av systemene med så lite tilpasning som mulig. Python ble brukt for å utvikle automatiserte verktøy som videre kunne konvertere Siemens blokkene, til kode og filformater kompatible med TwinCAT.

Vi utviklet en kjørbar exe fil som kunne automatisere konverteringsprosessen og videre en funksjon som integrerer prosessen i GitHub. På denne måten vil en kunne konvertere funksjonsblokkene direkte i GitHub, eller i en skrivebords applikasjon som kan bidra til en betydelig reduksjon i tidsbruk relatert til vedlikehold av PLS-biblioteker.

1 Innhold

Figurliste

1 Innledning

1.1 Oppdragsgiver

Figur 1. Goodtech logo

Goodtech er et norsk teknologiselskap, som spesialiserer seg innenfor robotisering, automatisering og digitalisering. De tilbyr høy kompetanse innen automasjon, robotikk, SCADA, MES, emballasjesystemer og høykompetanse for kraftdistribusjon for bedrifter innen kjemi-, metall- og gruveindustrien [1]. Hovedkontoret befinner seg i Oslo, men har kontorer blant annet i Førde, Bergen og i Sverige. Goodtech sine ansatte består av ingeniører og fagspesialister. Årsrapport fra 2022 viser at antallet tilsette er på 290. Goodtech ble etablert i Drammen i 1913 av to brødre som forutså at fremtiden ville bli mer elektrifisert og automatisert [2]. De jobber med å optimalisere og digitalisere industrien, og kombinerer prosess- og løsningskompetanse med ny programvare, teknologi og optimeringskompetanse [3].

1.2 Problemstilling

Goodtech bruker hovedsakelig Siemens Totally Integrated Automation (TIA) Portal som verktøy for å lage systemene ønsket fra kunde. Noen ganger er det behov for systemer som funker for andre leverandører, som for eksempel Beckhoff. Begge disse leverandørene forholder seg til IEC 61131-3 standarden, [4] [5] men kode kan tolkes av PLS-systemer på ulike måter. Det kan derfor oppstå feil om en bare skulle kopiert koden fra TIA. Problemet er dermed at visst man ønsker å kjøre Siemens TIA Portal basert kode på en Beckhoff-PLS, må dette konverteres manuelt. Denne prosessen er tidkrevende, derfor ønsker vi å lage et konvertering-skript i Python eller C# som automatisk konverterer koden. Med en automatisk prosess kan Goodtech spare både tid og ressurser.

1.3 IEC 61131-3

IEC 61131-3 er en internasjonal standard for programmeringsspråk i industriell automatisering. Ved bruken av en slik standard reduseres arbeidskostnader og tidsbruken betydelig gjennom alle faser av en programvares livssyklus, inkludert utvikling, testing, installasjon, drift og vedlikehold. Standarden støtter flere programmeringsspråk innenfor samme kontrollprogram. Dette medfører at brukeren kan benytte seg av det språket som passer seg best for oppgaven. Her er en liste over språk som inngår i IEC 61131-3, [6]:

- Ladder diagram (LD)
- Sequential Function Charts (SFC)
- Function Block Diagram (FBD)
- Structured Text (ST)
- Instruction List (IL)

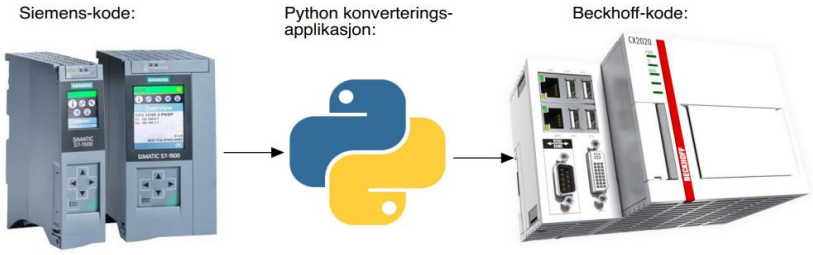
I denne oppgaven vil vi benytte oss av funksjonsblokkdiagram (FBD) og ST (strukturert tekst) for testing og konvertering. Et FDB program er et visuelt programmeringsspråk som bruker blokker/funksjoner for å illustrere dataflyten mellom ulike hendelser. FBD gjør det mulig å bygge et program ved å dra/slippe og koble sammen forskjellige funksjonsblokker og variabler [7]. ST er et høy-nivå programmeringsspråk som direkte kan lese inndata, produsere utdata og gi tilgang til hver del inni en PLS [8].

1.4 De utleverte funksjonsblokkene

Blokkene vi skal ta hensyn til er MB- og SBE-blokker. Disse er laget og levert fra oppdragsgiver selv, og skal være funksjonsblokker som forholder seg til IEC 61331-3 standarden. MB funksjonstemplaten skal overvåke alle binære innganger og utganger på et anlegg basert på definerte logiske parametere. SBE funksjonstemplaten beskriver kontroll av en enhet, som for eksempel motor, pumpe, varmeovn eller vifte. Det er bare en utgang, Y, som gir en åpen/lukket beskjed til enheten [9] (kilde lagt til som vedlegg i innleveringsmappen).

1.5 Ønsket resultat

Prosjektet vil sette søkelys på utviklingen av et Python-skript, som skal automatisere konverteringsprosessen fra Siemens-kode til Beckhoff kode. Skriptets hovedoppgave har som mål å ta inn en eksportert Siemens-fil, deretter produsere tilsvarende TwinCAT (Beckhoff)-filer som kan kjøres på en Beckhoff PLS. Skriptet skal videre kjøres som en GitHub Action-funksjon, noe som vil muliggjøre effektiv konvertering via nettleser. Vi ønsker samtidig å lage en applikasjon med et brukervennlig grafisk brukergrensesnitt, slik at brukeren enkelt kan gjennomføre en konvertering i applikasjonen på lokal datamaskin.



Figur 2. Siemens PLS, Python og Beckhoff PLS

Figur 2 viser en overordnet representasjon av hvordan konverteringsprosessen skal se ut. Her illustreres det hvordan Siemens-koden går gjennom et Python-skript, og returnerer en kjørbar fil som importeres i TwinCAT, deretter kobles opp mot en Beckhoff PLS.

Et vellykket resultat vil være en applikasjon som ved hjelp av noen tastetrykk kan konvertere en funksjonsblokk fra Siemens til en tilsvarende funksjonsblokk for Beckhoff. Denne applikasjonen skal også bestå av et brukervennlig grafisk brukergrensesnitt, utviklet i Python.

2 Kravspesifikasjon

Hovedkravet er utviklingen av Python-skript som automatiserer prosessen av manuell konvertering over ulike systemer, spesifikt Siemens til Beckhoff. Applikasjonen skal gjøre vedlikehold og oppdatering av PLS-bibliotek enkelt, uten noe omfattende arbeid.

2.1 Liste over krav

2.1.1 Generelt

1. Opprette et Python-skript som skal konvertere MB- og SBE-blokker fra Siemens TIA Portal til TwinCAT.
2. Python-skriptet skal kunne kjøres på en GitHub runner.

2.1.2 Brukervennlighet

1. Applikasjonen skal ha et intuitivt brukergrensesnitt.
2. Applikasjonen skal tilby forståelig tilbakemeldinger på brukerhandlinger.

2.1.3 Funksjonalitet

1. Applikasjonen må identifisere og konvertere nøkkelord som representerer forskjeller mellom leverandørene.
2. Om konverteringen ikke er mulig, må applikasjonen informere om at konvertering ikke kan gjennomføres.

2.1.4 Fleksibilitet

1. Applikasjonen må være fleksibel nok til å kunne konvertere oppdaterte blokker. Den må altså ta hensyn til funksjoner som ville vært naturlig å bruke i en MB- eller en SBE-blokk.

2.2 Tilleggskrav

2.2.1 Liste over tilleggskrav

1. Programmet skal gi tilbakemelding på feil, som hvor og hva som ikke fungerer.
2. Lage en tag-tabell til en global variabel liste (GVL), som i tillegg også skal konverteres.
3. Utvikle en tilsvarende konverteringsmetode hvor kunstig intelligens utfører konverteringen, som et alternativt forsøk på konvertering.
4. En tilsvarende konverteringsmetode som går fra Beckhoff tilbake til Siemens.

2.2.2 Beskrivelse av tilleggskrav

Tilleggskravene representerer videreutvikling som vil forbedre applikasjonen videre i større grad enn hovedkravene. Disse tilleggskravene blir prioritert om tidsplanen tillater det. En feilmelding med indikasjon på hvor det var feil, en funksjon for konvertering av global variabel liste, et alternativt konverteringsforsøk ved hjelp av KI, og reversering av konverteringsprosessen fra Beckhoff til Siemens, er alle tilleggsmål som vil forbedre programmets fleksibilitet, brukervennlighet, og funksjonalitet.

3 Analyse av problemet

En grundig gjennomgang av kravspesifikasjonen gjør det mulig å bryte problemstillingen i mindre seksjoner. Det overordnede problemet vi ønsker å løse er konversasjon av MB- og SBE-blokker fra Siemens til Beckhoff. Tiden det tar å konvertere manuelt er lang og arbeidet er svært repetitiv. Dette er fordi brukeren må gå gjennom linje for linje i koden og sjekke for forskjeller som må oversettes til TwinCAT standard. Dette må også gjøres for alle programfiler som eventuelt skal importeres til TwinCAT programmet.

3.1 Dekomponering av kravspesifikasjon

Kravspesifikasjonen deles inn i fire inndelingsoverskrifter. Det består av et generelt krav, brukervennlighet, funksjonalitet og fleksibilitet. Det generelle kravet omhandler å løse hovedproblemet, konverteringen. For at dette skal være en god løsning må applikasjonen være mer effektiv enn nåværende fremgangsmåte, som er å konvertere det manuelt.

Applikasjonen som skal benyttes må være **brukervennlig**. Et brukergrensesnitt som er vanskelig å bruke, vil ha motvirkende effekt. Derfor er det viktig at når bruker skal gjennomføre en konvertering, skal det ikke være spørsmål angående hvordan det gjøres. Derfor vil vi utvikle oversiktlig applikasjon, samt en brukermanual for å oppnå kravet om brukervennlighet.

Funksjonaliteten. skriptet må ta hensyn til visse nøkkelord. Nøkkelord vil være ord som vi har identifisert som ulike tvers av leverandørene. Disse nøkkelordene må derfor oversettes. Skriptet må fungere slik at det detekterer nøkkelord, deretter oversetter dem, slik at det nye TwinCat programmet vil ha tilsvarende funksjonalitet som det tidligere TIA Portal programmet.

Så skal skriptet være **fleksibelt**, som innebærer at den skal ta hensyn til funksjoner som kan dukke opp i en MB- eller SBE-blokk ved oppdateringer av funksjonsblokkene.

3.2 Utforming av mulige løsninger

Oppgaven vi valgte er veldig åpen utenom begrensningen av valg av programmeringsspråk, hvor valgene stod mellom C# eller Python. Det finnes mange måter å løse en slik oppgave på. Vi har utforsket ulike løsninger og vedlagt de mest sentrale alternativene under.

3.2.1 Løsningsalternativ 1: Kodeoversetting ved bruk av Kunstig Intelligens (KI)

Bruk av KI for å automatisk oversette programmeringskoden fra TIA Portal til TwinCAT sitt format. KI vil da ta imot en komplett kode fra Siemens TIA Portal, analysere og oversette forskjellene, ved å benytte en ChatGPT API-nøkkel som man skal koble opp mot Python skriptet.

3.2.2 Løsningsalternativ 2: Omgjøring av funksjonsblokker

Strukturere Siemens-kode på forhand, slik at koden på enklest mulig måte kan konverteres til Beckhoff kode. Dette innebærer å lage nye funksjonsblokker som tilsvarer innebygde funksjoner i TIA Portal. Skriptet vil da håndtere færre forskjeller, og hovedsakelig ta hånd om de fundamentale forskjellene i filoppbygningen.

3.2.3 Løsningsalternativ 3: En fullstendig konvertering ved hjelp av et Python-skript

Ved bruk av dette løsningsforslaget, skal skriptet behandle alle forskjellene på tvers av leverandørene. Applikasjonen skal lese en Siemens TIA Portal SCL-fil, konvertere og deretter legge de nye Beckhoff tilsvarende filene i ønsket mappe.

3.2.4 Vurderinger i forhold til verktøy og HW/SW komponenter

En fellesfaktor for alle løsningsforslagene er behandling av tekst, og utforming av filer. Derfor er det essensielt med et programmeringsspråk som er effektivt i form av strengbehandling. «FreeCodeCamp» har publisert en oversiktlig Python brukerhåndbok kalt «Python String Manipulation Handbook» [10]. Denne kilden ga oss oversikt over Python sine sterke og intuitive strengmanipulasjonsfunksjoner. Her får vi oversikt over hvordan Python sin tekststreng manipulasjon kan splitte, strippe og enkelt hente ut et ord fra en setning. Dermed konkluderte vi med å bruke programmeringsspråket Python for denne oppgaven.

For testing av applikasjonen vil vi benytte en Beckhoff PLS. Ved å bruke de utleverte blokkene som utgangspunkt på et fysisk anlegg og deretter teste de konverterte blokkene. vil vi kunne sammenligne funksjonaliteten, og sikre at konverteringen gikk som den skal.

Under kodingen av programmet har vi benyttet oss av mange ulike typer programvare for å øke effektiviteten vår. Disse programvarene har også bidratt til bedre samarbeid mellom oss som skriver bachelor, samt med arbeidsgiver. Under har vi en liste av ulike programmer og metoder som vi har benyttet.

3.2.4.1 GitHub CoPilot

GitHub CoPilot er en kunstig intelligent kodeassist som kommer med hjelpeforslag til kode basert på utformingen av prosjektet man jobber med [11]. Vi har benyttet oss av CoPilot sine kodeforslag da dette i stor grad effektiviserer kodingsprosessen. Når vi skal produsere metoder for å utføre handlinger som for eksempel å lese en tekstfil, begynner CoPilot tidlig å foreslå forskjellige alternativer.

3.2.4.2 Ruff

Da vi begynte med programmeringen, var vi bevisste på viktigheten av å produsere leselig og forståelig kode, som vi ble informert om i ELE205, videregående programmering. Dette gjør det lettere å vedlikeholde og oppdatere kode.

Tidlig i kodingsfasen introduserte oppdragsgiver oss for Ruff, et raskt og effektivt linter-verktøy for Python som har vært med å opprettholde en oversiktlig kode gjennom prosjektet. Linter er programmer man kan bruke i tråd med kodingen som gir beskjed, i form av feilmeldinger, om koden man har skrevet inneholder feil eller er utenfor PEP-8 standarden. Det kan for eksempel være om man mangler en parentes eller variabler som ikke blir brukt i noen sammenheng [12]. Vi lastet ned Ruff direkte på Visual Studio Code [13]. PEP-8 standarden er en måte å skrive kode på, som gjør at den skal være enklere å lese og utføre endringer på [14]. Dette er regler som for eksempel maksimal lengde på en setning [15], eller hvorvidt man skal bruke tabs- eller mellomrom for avstander. Dette fikser Ruff automatisk, derfor ønsker vi å benytte ruff, for å opprettholde en oversiktlig kode, gjennom hele prosjektet.

3.2.4.3 Samarbeid over GitHub

Discord ble tidligere benyttet som et bindeledd mellom programkode. Problemet med det er at det kan være vanskelig å ha oversikt over endringer i koden. Eksempelvis har vi ikke oversikt om den nye koden vi har produsert vil ha konflikter med eksisterende kode. Derfor velger vi å benytte GitHub som bindeledd mellom programkoden. Oppdragsgiver var behjelpelig med å opprette et GitHub-arkiv (repository) som vi kunne bruke for å videreutvikle programmet. Vi lastet ned GitHub Desktop lokalt på datamaskinen [16]. GitHub Desktop gjør det mulig å koble det som skjer på din lokale datamaskin opp til et GitHub arkiv [17].

GitHub Desktop gjør det mulig for oss å «pushe» (dytte) endring til arkivet og «pulle» (hente) nyere versjon av prosjektet til vår lokale datamaskin. Dette sikrer at alle medlemmene i prosjektet alltid har tilgang til den nyeste versjonen av koden, samt bidrar til en effektiv implementasjon av de nyeste endringene som er gjort.

Det som også har gjort samarbeidet over GitHub svært gunstig, er at Ruff blir automatisk kjørt dersom man lager en pull request (PR). En PR er et forslag om endring til hovedprosjektet (main-branch) fra en av grenene (branch) du arbeider på. For at denne endringen skal bli innvilget, har vi gjort det slik at en av samarbeidspartnerne i prosjektet må godkjenne endringene [18]. Når en PR blir godkjent blir det en automatisk test av programkoden i hele prosjektet ved hjelp av ruff, som forteller om vi har forbeholdt PEP-8 standarden.

3.3 Valg av løsningsalternativ

Løsningsalternativ 1, ''Kodeoversetting ved bruk av Kunstig Intelligens (KI)''. Dette løsningsforslaget har blitt testet ved å sende inn Siemens kode til ChatGPT og har i mange tilfeller konvertert koden med suksess. Ved noen tilfeller har deler av koden forsvunnet. Konkrete spørringer vil i de fleste tilfeller returnere fungerende kode, men brukeren vil likevel være nødt til å gå gjennom koden for å sjekke at resultatet samsvarer den opprinnelige koden. Derfor har vi valgt vekk denne fremgangsmåten, da sluttresultatets pålitelighet kan variere.

Løsningsalternativ 2, ''Omgjøring av funksjons-blokker''. Dette svarer godt til oppgaven da vi med stor trygghet kan konvertere de utdelte blokkene med et lite skript ved å kun endre de fundamentale forskjellene i filoppbygningen. Med denne fremgangsmåten blir programmet lite fleksibelt, og legger ansvar på dem som ønsker å oppdatere biblioteket videre. Dette løsningsforslaget bidrar også til en mer manuell konvertering, som da gjør prosessen mer tidkrevende.

Vi konkluderer med at løsningsalternativ 3, ''En fullstendig konvertering ved hjelp av et Python-skript'' vil ha den mest brukervennlige tilnærmingen. Til tross for at dette vil være en mer krevende oppgave, mener vi at dette alternativet vil gi det beste sluttproduktet. Med feil i konverteringen fra applikasjonen, mener vi at å benytte løsningsalternativ 1 som et ekstra forsøk for konvertering, med en liten advarsel om at det er en sjanse for upålitelighet, er et bra tilleggsmål om det skulle vært tid til det.

4 Strukturering og progresjon

4.1 Strukturering i gruppen

Vi bestemte oss raskt for å sette en standard på oppmøtetidspunkter for bachelorprosjektet, noe som har vært vesentlig for å oppnå en jevn og stabil fremgang i arbeidet med bacheloroppgaven. Da vi ikke har erfaring med programmeringsspråket Python, tenkte vi at det var hensiktsmessig å møte fysisk for å samarbeide.

4.1.1 Valg av arbeidssted og kommunikasjonsverktøy

Vi valgte elektro-laboratoriet (Hammer) på Høgskulen på Vestlandet i Førde som vårt primære arbeidssted. Arbeidstedet er utstyrt med arbeidsstasjoner med store skjermer, noe som bidrar til økt produktivitet. Selv om vi har møtt opp fysisk har behovet for kommunikasjon med oppdragsgiver vært nødvendig. Vi har derfor brukt Discord som kommunikasjonsverktøy [19]. Discord har også vært et godt verktøy på dager vi ikke har hatt mulighet for å møte opp fysisk på arbeidstedet.

4.1.2 Planlegging og fremskritt

Vedlagt er fremdriftsplanen vår. Under utviklingen og planlegging av fremdriftsplanen fikk vi en dypere forståelse for hvordan vi skulle sette i gang med oppgaven. Dette gjorde at vi enklere kunne estimere tidsbruk for hver arbeidsoppgave, og ble da lettere å forstå omfanget av prosjektet i en helhet. Fremdriftsplanlegging er et godt verktøy å bruke for å identifisere hvilke oppgaver som avhenger av hverandre. Eksempel på dette var når vi først forstod at før vi kunne begynne på konvertering av spesielle metoder, måtte vi ha et Python-metode som utfører nøkkelord deteksjon.

Fremdriftsplanen er også et godt hjelpemiddel for å se hvor mange timer vi måtte jobbe med for å oppnå nok timer på prosjektet. Vi brukte Microsoft Excel for å utarbeide framdriftsplanen. Excel får vi gratis gjennom Høgskulen på Vestlandet og er et godt verktøy for slike planer [20].

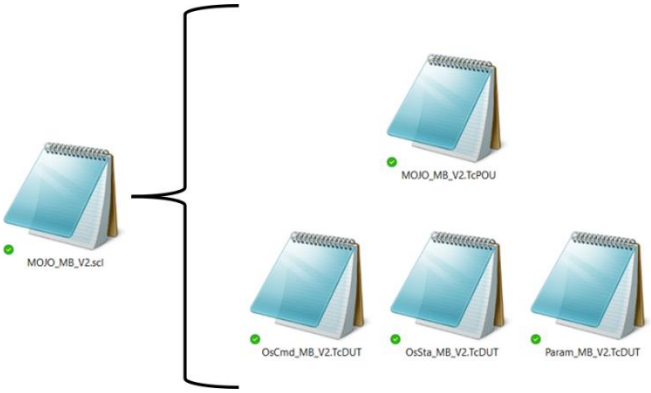
5 Forskjeller på tvers av leverandørene

5.1 Fundamentale forskjeller

For å kunne konvertere fra Siemens TIA Portal til TwinCAT, må man ta hensyn til de fundamentale forskjellene. De fundamentale forskjellene baserer seg på filstrukturen, hvordan innholdet til hver fil er bygget opp og forskjeller i selve koden. Disse forskjellene er funnet ved å sammenligne filene som representerer MB- og SBE-blokker for begge leverandørene.

5.1.1 Filstrukturen

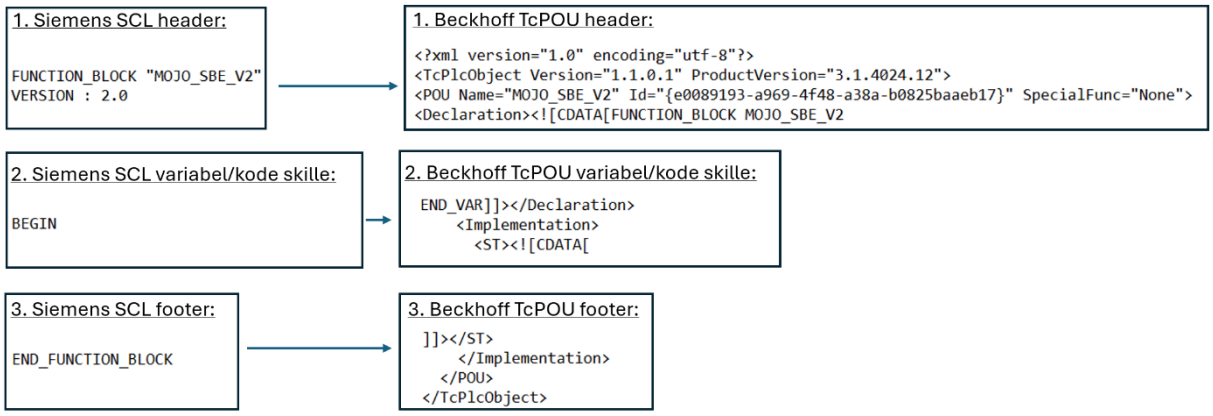
Når man eksporterer en fil fra Siemens TIA Portal, kan man velge å eksportere med avhengige blokker. Da får man en SCL-fil som inneholder alt av nødvendig informasjon. Beckhoff derimot ønsker å dele opp filstrukturen til TcPOU- og TcDUT-filer, hvor TcPOU-filen beskriver funksjonsblokken, og TcDUT-filene beskriver de egendefinerte datatypene. Figur 3 viser filstrukturen til TIA Portal for en funksjonsblokk sammenlignet med TwinCAT.



Figur 3. Siemens fil og tilsvarende Beckhoff filer

5.1.2 Filoppbyggingen

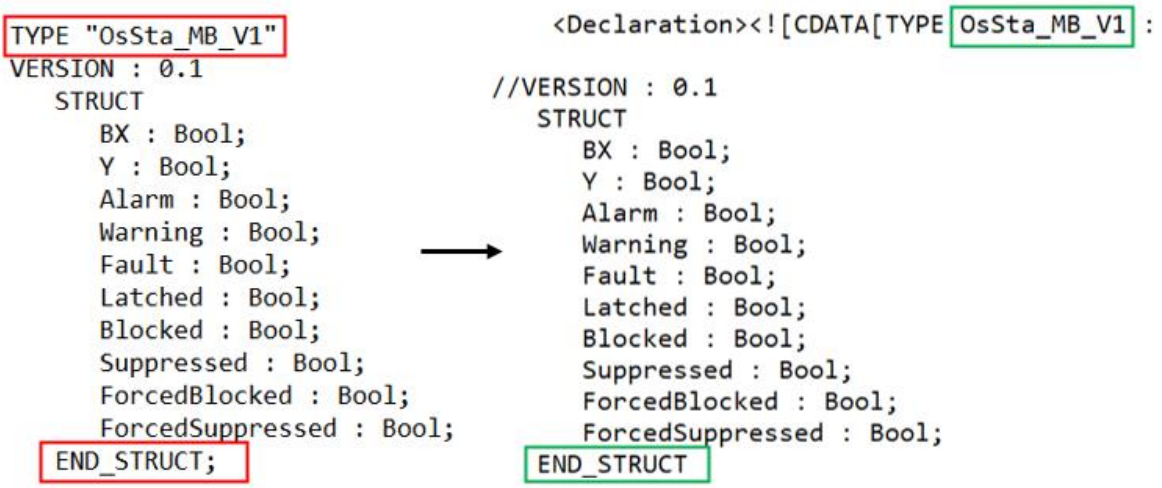
Figur 4 viser oppbygningen av filene for både Siemens TIA Portal og TwinCAT. Siemens-filen har en standard «header» med hvilken type blokk det er, prosjektnavn og versjonen. Deretter kommer variabelteksten, koden, og en standard «footer», som i Siemens vil være «END_FUNCTION_BLOCK». Beckhoff har også en standard «header», med prosjektnavn og versjonen, men den er ulik fra Siemens sin fil. Variabelteksten og koden i Beckhoff er adskilt av en standard tekst, før selve koden også avsluttes med en standard «footer».



Figur 4. Fundamentale forskjeller mellom Siemens og Beckhoff sin filoppbygging

5.1.3 Fundamentale forskjeller i koden

I tillegg til forskjellene i filstrukturen og filoppbygningen er det noe få fundamentale forskjeller i koden. Når en benytter variabler i Siemens TIA Portal, bruker man «hashtag» foran variabelen for eksempel: «#variabel := 2;». Tilsvarende i TwinCAT ville vært: «variabel := 2». Hashtagene blir fortsatt brukt når en skal sette variabler lik tidsintervall i TwinCAT for eksempel: «tids-variabel := t#2s». Siemens TIA Portal benytter anførselstegn rundt navnet til de egendefinerte datatypene, dette gjelder ikke TwinCAT, da de skriver navnet uten. Semikolon etter forekomsten av ordet «END_STRUCT» viser seg også å være en forskjell da dette forekommer i TIA Portal, men ikke i TwinCAT, vist i Figur 5.



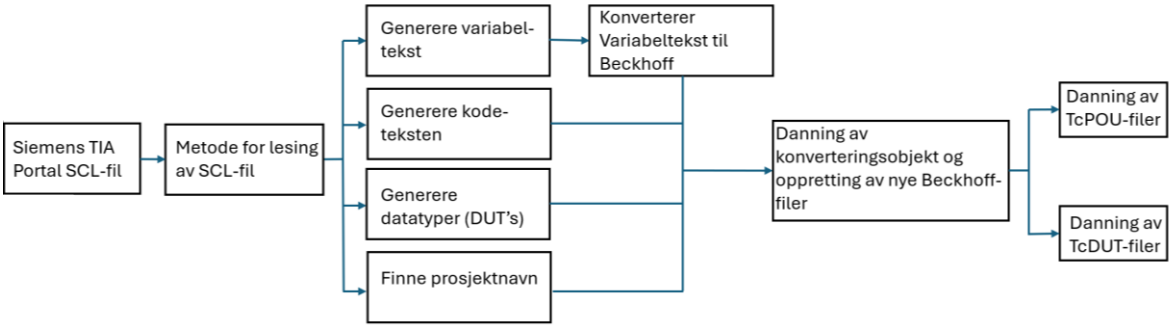
Figur 5. TIA Portal egendefinert datatype til venstre og TwinCAT egendefinert datatype til høyre

5.2 Andre forskjeller

Sammen med de fundamentale forskjellene, er det også forskjeller i hvordan man skriver koden. I de utleverte MB- og SBE-blokkene oppdaget vi forskjell i hvordan man deklarerer tidtakere, som for eksempel TON_TIME. Disse type tidtakere er en del av IEC 61131 standarden, [21] men blir deklarert forskjellig over de to leverandørene.

En annen forskjell, er hvordan tidsvariabler fungerer på tvers av leverandørene, vi ble oppmerksomme på denne forskjellen da vi leste oppgaveteksten gitt fra arbeidsgiver, da denne forskjellen ble brukt som et eksempel. I Siemens TIA Portal begynner tidsvariabelen på -24 dager, og stiger til 24 dager [22]. I TwinCAT går tidsvariabelen fra 0 til 49 dager [23]. Tidsvariabler er derimot ikke benyttet i de utleverte blokkene, og er derfor også en forskjell, vi ikke trenger å ta hensyn til.

6 Realisering av løsning



Figur 6. Oversikt over egendefinerte metoder

6.1 Kode

Applikasjonen benytter en rekke egendefinerte metoder for å fullføre konverteringen. I Figur 6 ser vi et diagram som beskriver prosessen av en fullverdig konvertering av en eksponert Siemens TIA Portal SCL-fil til tilsvarende TwinCAT-filer.

6.1.1 Lese fra fil

Applikasjonen starter med å lese innholdet i en valgt SCL-fil. Vi benytter «Path» som er en del av «Pathlib» biblioteket. **Dette** biblioteket gjør det mulig å åpne filen, og senere returnere alt av filens innhold i form av en lang tekst streng [24].

```
def read_scl_file(scl_file_path: str) -> str:
    """Read the SCL file from the given file path and store the content in SCLConvertion.SCL_Full_Text."""
    _LOGGER.debug(f"Reading SCL file from {scl_file_path}")
    try:
        with Path(scl_file_path).open(encoding="utf-8-sig") as fil:
            return fil.read()
    except FileNotFoundError:
        _LOGGER.critical(f"Filen {scl_file_path} ble ikke funnet.")
        sys.exit(1)
    except Exception as e:
        _LOGGER.critical(f"En uventet feil oppstod: {e}")
        return ""
```

Figur 7. Metode for lesing av fil

6.1.2 Skille og generering av variabeltekst

variabelteksten og koden i TwinCAT-filene er adskilt av noe standard tekst, vi var derfor nødt til å lagre variabel teksten til en egen variabel. Dette gjorde vi med å utnytte at variabel teksten i Siemens TIA Portal sin SCL-fil, alltid begynte etter følgende tekst «VAR_INPUT» og sluttet ved teksten «BEGIN». På denne måten lagres all tekst imellom disse to indeksene.

```
def generate_variable_text(full_text: str) -> str:
    """Generate the variable text from the SCL file."""
    start_index = full_text.find("VAR_INPUT")
    stop_index = full_text.find("BEGIN")
    converted_variable_text = full_text[start_index + len("VAR_INPUT") : stop_index].strip()
    return converted_variable_text
```

Figur 8. Metode for generering av variabeltekst

6.1.3 Skille og generere kodetekst

På samme måte som vi skilte og lagret variabel teksten, har vi lagret kodeteksten. Her utnyttet vi at kode segmentet alltid kommer etter forekomst av ordet «BEGIN», og slutter ved «END_FUNCTION_BLOCK» i SCL-filen. Tegnet hashtag (#) skal også vekk i alle tilfeller utenom når en benytter seg av tidsfunksjoner i TwinCAT. For å fjerne hashtagene brukte vi Python sin innebygde funksjon «replace». Replace-funksjonen lar oss erstatte tegn med andre tegn [25]. For å sikre at hashtagene ikke forsvinner i tilfeller hvor tidsintervall benyttes, fjernet vi kun hashtagene som ikke er pakket inn i tekst, som for eksempel «t#5s».

```python
def generate_code(full_text: str) -> str:
    """Generate the code from the SCL file."""
    try:
        start_index = full_text.find("BEGIN") + len("BEGIN")
        end_index = full_text.find("END_FUNCTION_BLOCK")
        code_section = full_text[start_index:end_index]
        code_section_done = code_section.replace(" #", " ")
        code_section_done = code_section_done.replace("\t#", "\t")
        code_section_done = code_section_done.replace("(#", "(")
        return code_section_done
    except Exception as err:
        raise ValueError("The code section could not be extracted from the SCL file.") from err
```

Figur 9. Metode for generering av kode

6.1.4 Skille og generere liste av egendefinerte datatyper

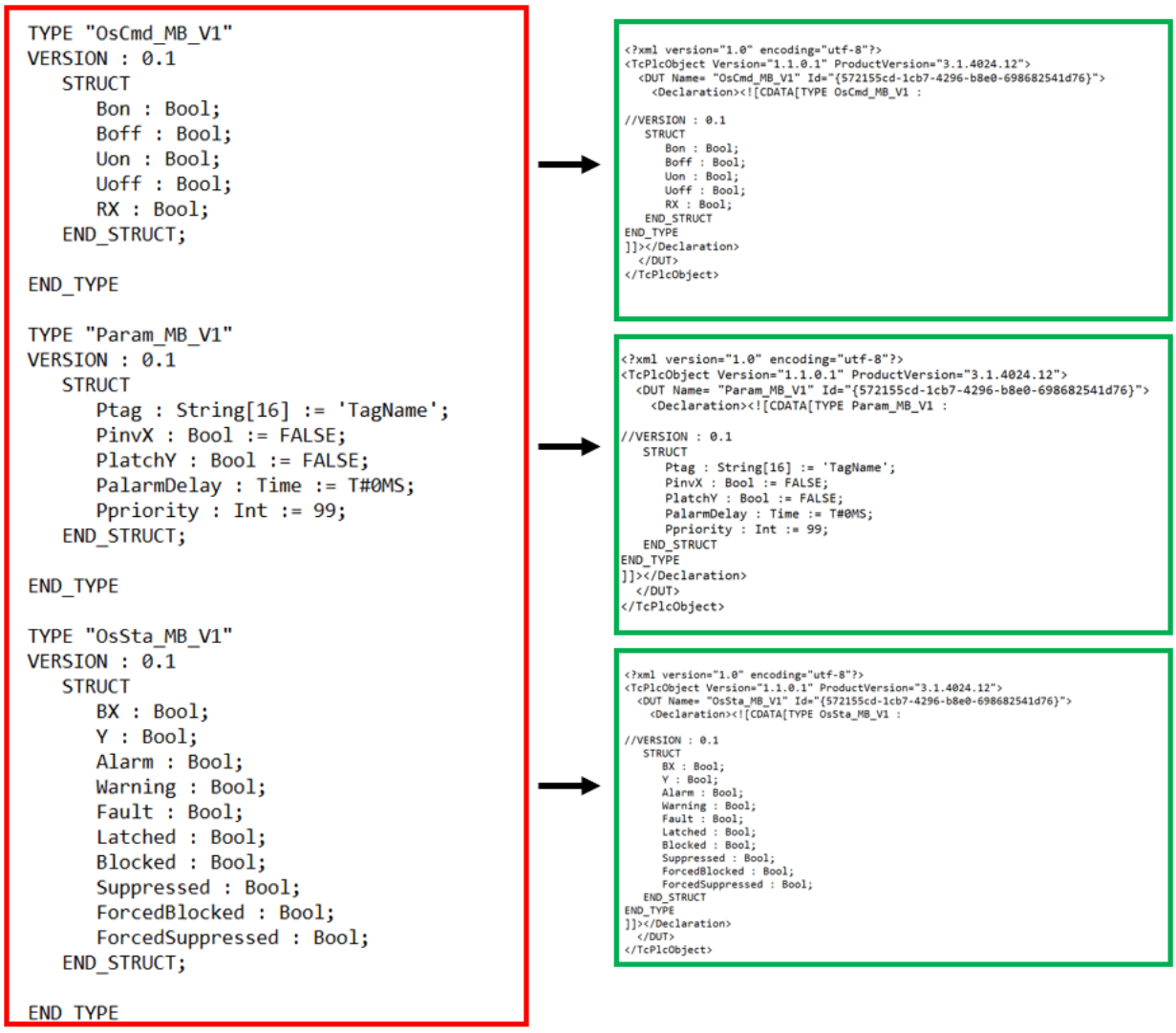
De egendefinerte datatypene til Siemens kommer i starten av SCL-filen når en eksporterer med avhengige blokker fra TIA Portal. Når disse datatypene skal konverteres til TwinCAT, må dem skilles hver for seg, da disse skal være i egne filer. Første steg er å lagre teksten mellom starten av SCL-filen og helt til forekomsten av teksten «FUNCTION_BLOCK». Dette lagrer hele seksjonen som inneholder egendefinerte datatyper. Videre splitter vi teksten ved forekomsten av «END_TYPE», ved hjelp av biblioteket re. [26] re biblioteket har funksjonen «re.split», som kan dele opp en større tekststreng i flere mindre tekststrenger. Vi deler opp tekstrengene på teksten «END_TYPE» da dette kommer i enden av tekst seksjonene som inneholder egendefinerte datatyper i fra TIA Portal sin SCL-fil.

```python
def generate_dut_list(full_text: str) -> list[Tcdut]:
    """Generate the dut list from the SCL file."""
    stop_index = full_text.find("FUNCTION_BLOCK")
    dut_text: str = full_text[:stop_index].strip()
    dut_lines = dut_text.split("\n")
    for i in range(len(dut_lines)):
        if "VERSION" in dut_lines[i]:
            dut_lines[i] = dut_lines[i].replace(dut_lines[i], "//" + dut_lines[i])
    dut_text = "\n".join(dut_lines)
    dut_list: list[str] = re.split(r"END_TYPE", dut_text)
    dut_list_done: list[Tcdut] = []

    for dut in dut_list[:-1]:
        dutcode = dut
        dutcode += "\nEND_TYPE"
        dutcode = dutcode.replace("\n\n", "")
        start_index = dutcode.find('"')
        stopp_index = dutcode.find('"', start_index + 1)
        dut_name = dutcode[6:stopp_index]
        dutcode = dutcode[: stopp_index + 1] + " :\n" + dutcode[stopp_index + 1 :]

        lines = dutcode.split("\n")
        lines[0] = lines[0].replace(";", "")
        lines[0] = lines[0].replace('"', "")

        for line in range(len(lines)):
            if "END_STRUCT" in lines[line]:
                lines[line] = lines[line].replace(";", "")

        dutcode = "\n".join(lines)
        dut_list_done.append(Tcdut(dut_name, dutcode))
    return dut_list_done
```

Figur 10. Metode for generering av liste av egendefinerte datatyper

Det er også en del fundamentale forskjeller i hvordan de egendefinerte datatypene er presentert i filen. Linjer som inneholder «VERSION» oppstår ikke i TwinCAT. Derfor har vi valgt å sette to skråstrek-tegn fremfor linjen som inneholder «VERSION», på denne måten, blir denne linjen kommentert ut, og har derfor ingen påvirkning på koden. forekomster av dobbel «Enter» (\n\n) må fjernes, anførselstegn må fjernes fra første linje, og semikolon må fjernes fra både første linje, og linjer som inneholder teksten «END_STRUCT». På denne måten oppnår filene en filoppbygning som er lik TwinCAT sine filer for egendefinerte datatyper, se Figur 11.

Vi ønsker også at egendefinerte datatypene skal beholde navnet sitt. Vi lagret dermed teksten som er mellom første de to første forekomstene av anførselstegn. Vi utnytter dette, og lagrer navnet til variabel. Til slutt returnerer metoden listen av egendefinerte datatype objekter bestående av koden og navnet, på TwinCAT sin standard.



Figur 11. Egendefinerte datatyper fra TIA Portal til TwinCAT

6.1.5 Finne prosjektnavnet

På samme måte som vi lagret navnet til de egendefinerte datatypene, har vi lagret navnet til selve funksjonsblokken. Vi utnyttet at navnet alltid oppstår i samme linje som inneholder forekomsten av teksten «FUNCTION_BLOCK» i SCL-filen vist i Figur 13. I den samme linjen er navnet alltid mellom to anførselstegn. Vi lagrer dermed navnet med å lagre teksten mellom første forekomst av anførselstegn og andre.

```python
def find_project_name(full_text: str) -> str:
    """Find and store the project name from the SCL file."""
    lines = full_text.split("\n")
    for i in range(len(lines)):
        if "FUNCTION_BLOCK " in lines[i]:
            start_index = lines[i].find('"')
            stop_index = lines[i].find('"', start_index + 1)
            project_name = lines[i][start_index + 1 : stop_index]
    return project_name
```

Figur 12. Metode for å finne prosjektnavn

```
FUNCTION_BLOCK "FB_TEST"
{ S7_Optimized_Access := 'TRUE' }
VERSION : 0.1
```

Figur 13. Siemens funksjonsblokk navn

6.1.6 Fikse forskjeller i variabelteksten

De utleverte blokkene benyttet funksjonen TON_TIME. Selv om denne funksjonen forholder seg til IEC 61131-3 standarden, blir funksjonene deklarert ulikt i variabel teksten for de to leverandørene. I Siemens TIA Portal blir funksjonen deklarert som «TON_TIME» [27], mens Beckhoff deklarerer den til «TON» [28]. I SCL-filen er det også en del beskrivende tekst i deklarasjonen av funksjonen, for eksempel biblioteks-versjonen som vist i Figur 15. Dette forekommer ikke i TwinCAT. Første ordet i linjen som inneholder «TON_TIME» vil alltid være variabel navnet. Metoden bytter derfor ut hele linjen som inneholder «TON_TIME», med variabelnavnet, og deretter «: TON». I Figur 15 ser vi resultatet av konverteringen med hensyn på tidtaker funksjonen. Metoden tar også hensyn til andre type tidtakere og tellere, om disse funksjonene noen gang skulle blitt brukt i MB- eller SBE-blokker i fremtiden.

```python
def convert_timers_and_counters_in_variabletext(variable_text: str) -> str:
    """Convert the timers and counters in the variable text."""
    variable_text_lines = variable_text.split("\n")
    for i in range(len(variable_text_lines)):
        if "TON_TIME" in variable_text_lines[i]:
            variable_text_lines[i] = variable_text_lines[i].strip()
            line_words = variable_text_lines[i].split(" ")
            variable_text_lines[i] = variable_text_lines[i].replace(variable_text_lines[i], "\t"
            + line_words[0] + ": TON;")

        if "TOF_TIME" in variable_text_lines[i]:
            variable_text_lines[i] = variable_text_lines[i].strip()
            line_words = variable_text_lines[i].split(" ")
            variable_text_lines[i] = variable_text_lines[i].replace(variable_text_lines[i], "\t"
            + line_words[0] + ": TOF;")

        if "TP_TIME" in variable_text_lines[i]:
            variable_text_lines[i] = variable_text_lines[i].strip()
            line_words = variable_text_lines[i].split(" ")
            variable_text_lines[i] = variable_text_lines[i].replace(variable_text_lines[i], "\t"
            + line_words[0] + ": TP;")

        if "CTU_INT" in variable_text_lines[i]:
            variable_text_lines[i] = variable_text_lines[i].strip()
            line_words = variable_text_lines[i].split(" ")
            variable_text_lines[i] = variable_text_lines[i].replace(variable_text_lines[i], "\t"
            + line_words[0] + ": CTU;")

    variable_text = "\n".join(variable_text_lines)
    return variable_text
```

Figur 14. Metode for å konvertere variabelteksten

```
sbBB : Bool;
sAlarmDelay {InstructionName := 'TON_TIME'; LibVersion := '1.0'} : TON_TIME;
sbAlarmDelayed : Bool;
sOutputDelay {InstructionName := 'TON_TIME'; LibVersion := '1.0'} : TON_TIME;
sbOutputDelayed : Bool;
```

→

```
sbBB : Bool;
sAlarmDelay: TON;
sbAlarmDelayed : Bool;
sOutputDelay: TON;
sbOutputDelayed : Bool;
sbLatched : Bool:
```

Figur 15. Til venstre er deklarasjon av timer i TIA Portal og til høyre er tilsvarende for TwinCAT

6.1.7 Danning av konverteringsobjekt og oppretting av filer

Metoden translate danner et konverteringsobjekt bestående av tekstinnholdet i SCL-filen, samt den nye koden, variabelteksten og prosjektnavnet. Videre produserer metoden de nye filene. Denne metoden benytter alle metodene nevnt tidligere for å utføre en konvertering, som vist i Figur 16.

```python
def translate(filepath: str, new_file_path_tc: str) -> None:
    """Translate the SCL file and generate the TCPou and DUT files."""
    full_text = read_scl_file(filepath)
    code = generate_code(full_text)
    dut_list = generate_dut_list(full_text)
    variable_text = convert_timers_and_counters_in_variabletext(generate_variable_text(full_text))
    project_name = find_project_name(full_text)
    scl_full_text = full_text
    convertion_object = SCLConvertion(scl_full_text, code, variable_text, project_name)
    convertion_object.dut_list = dut_list
    log_stream.truncate()
    log_stream.seek(0)

    _LOGGER.debug("Generating TcPOU files...")
    try:
        generate_tcpou_file(new_file_path_tc, convertion_object)
        _LOGGER.info(f"POU file generated successfully in \n{new_file_path_tc}")
    except Exception as err:
        _LOGGER.critical(f"Error in generating TCPou file. {err}")

    _LOGGER.debug("Generating DUT files...")
    try:
        generate_dut_files(new_file_path_tc, convertion_object)
        _LOGGER.info(f"DUT files generated successfully in \n{new_file_path_tc}")
        convertion_object.dut_list = []
    except Exception as err:
        _LOGGER.critical(f"Error in generating DUT files. {err}")
```

Figur 16. Metode for danning av konverteringsobjekt og oppretting av filer

6.1.8 Sjekkliste om konvertering er mulig

Metoden «check», sjekker konverteringsobjektet og gir en tilbakemelding på om konvertering er mulig eller ikke. metoden leter etter nøkkelord som er nødt være med i teksten til den nye TwinCAT funksjonsblokk-filen, som «END_FUNCTION_BLCOK» og «BEGIN». Metoden leter også etter nøkkelord som ikke må forekomme i den nye TwinCAT-filen som for eksempel «TON_TIME» eller «TOF_TIME», da disse deklarasjonene ikke fungerer i TwinCAT. Ved et av disse tilfellene vil metoden returnere false, noe som tilsier at konvertering ikke er mulig. Metoden vil også gi beskjed om hvilket ord som ble/ikke ble detektert, som vist i Figur 18.

```python
def check(full_text: str) -> bool:  # Check funksjonen tar inn en streng og returnerer en bool
    """Check the full text."""
    converting_object = create_object(full_text)  # Lager et converting_object med data fra full_text
    result = True
    potential_converted_full_info = find_full_info(converting_object)

    must_have_keywords = ["END_FUNCTION_BLOCK", "BEGIN"]
    error_list = ["TON_TIME", "TOF_TIME", "TP_TIME", "CTU_INT"]
    found_errors = [keyword for keyword in error_list if keyword in potential_converted_full_info]
    not_found_keywords = [keyword for keyword in must_have_keywords if keyword not in full_text]
```

Figur 17. Utklipp av kodesnutten til metoden "check"

```
2024-04-30T13:49:09+0200 - src.plctranslator.tia_translator - tia_translator.py:24 - INFO -
2024-04-30T13:49:09+0200 - src.plctranslator.tia_translator - tia_translator.py:29 - DEBUG -
2024-04-30T13:49:09+0200 - src.plctranslator.tia_translator - tia_translator.py:32 - DEBUG -
dut's found: 3...
2024-04-30T13:49:09+0200 - src.plctranslator.tia_translator - tia_translator.py:35 - DEBUG -
xt...
2024-04-30T13:49:09+0200 - src.plctranslator.tia_translator - tia_translator.py:40 - DEBUG -
 Finding Project Name...
2024-04-30T13:49:09+0200 - src.plctranslator.tia_translator - tia_translator.py:83 - ERROR -
 Check Complete: Error found - TON_TIME
```

Figur 18. Feilmelding ved deteksjon av nøkkelord

6.1.9 Generering av nye TwinCAT filene

Til slutt skal vi generere de nye TcPOU- og TcDUT-filene. Metodene i Figur 19 og Figur 20 tar inn de allerede genererte konverteringsobjektene. Deretter danner metodene de nye TwinCAT-filene. Vi sjekket de utleverte blokkene og deres respektive filer. Begge disse filene benytter UTF-8 formatet. UTF-8 formatet sikrer at spesialtegn lastes inn riktig. [29] Noe som gjorde det vesentlig å produsere disse filene kodet i UTF-8. Dette ble gjort ved hjelp av parameteren: encoding = «UTF-8».

```python
def generate_tcpou_file(folder_path: str, converting_object: SCLConvertion) -> None:
    """Generate the TcPOU file based on the provided folder path."""
    filsti = rf"{folder_path}/{converting_object.project_name}.TcPOU"
    with Path(filsti).open("w", encoding="UTF-8") as file:
        file.write(converting_object.header())
        file.write(converting_object.variable_text())
        file.write(converting_object.code())
```

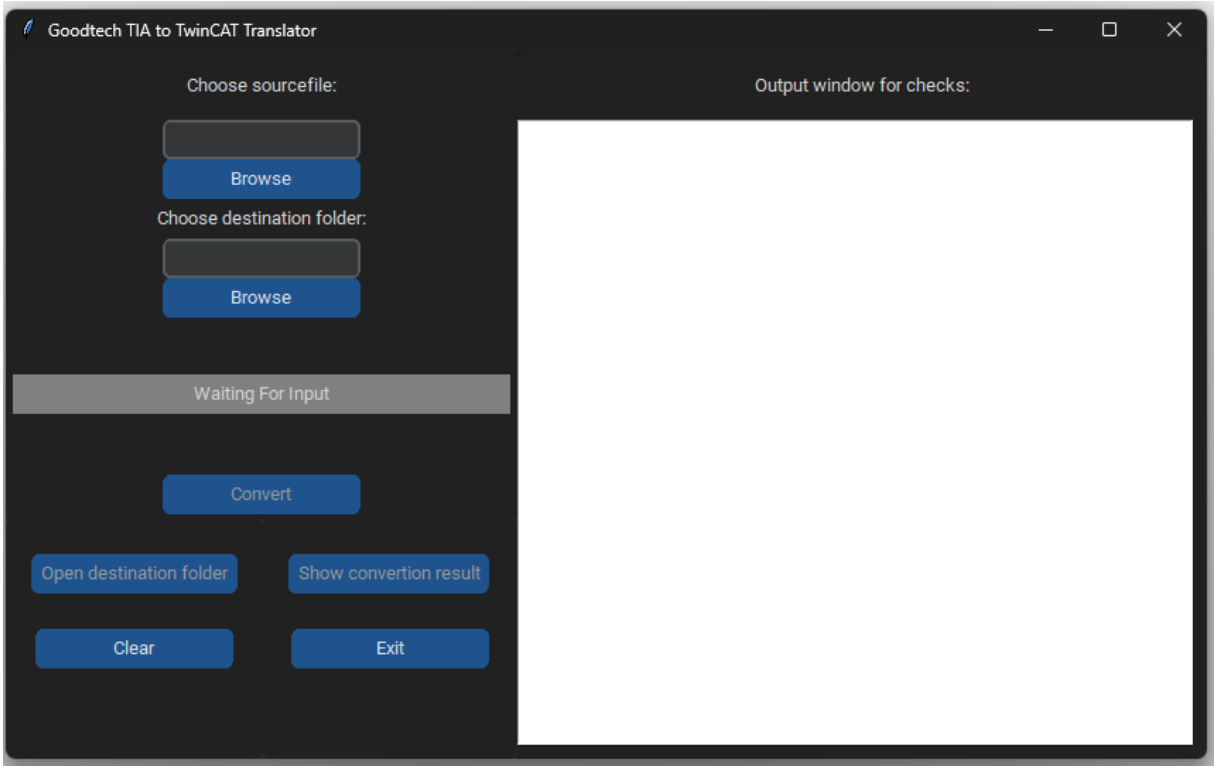
Figur 19. Metode for generering av funksjonsblokk-fil

```python
def generate_dut_files(folder_path: str, converting_object: SCLConvertion) -> None:
    """Generate the dut files based on the provided file path."""
    for dut in converting_object.dut_list:
        filsti = rf"{folder_path}/{dut.name}.TcDUT"
        with Path(filsti).open("w", encoding="utf-8-sig") as file:
            file.write(dut.header())
            file.write(dut.code)
            file.write(dut.footer)
```

Figur 20. Metode for generering av egendefinerte datatype-filer

6.2 Utvikling av brukergrensesnitt

For å opprette et brukergrensesnitt i Python, valgte vi å importere bibliotekene Tkinter og Customtkinter [30] [31]. Disse bibliotekene er gode verktøy for å utvikle grafiske brukergrensesnitt. Tkinter har standard grafiske elementer, mens Customtkinter har grafiske elementer av mer moderne stil. Vi benyttet Tkinter for den store tekstboksen til høyre vist i Figur 21. På venstre side benyttet vi Customtkinter da vi ønsket at knappene og etikettene skal ha en moderne stil.



Figur 21. Grafisk brukergrensesnitt

6.2.1 Valg av kildefil

Tkinter biblioteket inneholder klassen «filedialog». Denne klassen gjør det mulig å hente filer som ligger lokalt på datamaskinen, funksjonen åpner filutforsker og lar deg velge hvilken fil du ønsker [32]. I valg av kildefil sjekker metoden om konvertering er mulig ved hjelp av den egendefinerte metoden «check». Ved mulig konvertering, vil en etikett bli grønn og fortelle at konvertering er mulig.

```
def choose_sourcefile():
    """Open a file dialog to select a source file."""
    filepath = filedialog.askopenfilename(filetypes=[("SCL files", "*.scl")])
    if filepath:
        sourcefile_var.set(filepath)
        if plctranslator.check(plctranslator.read_scl_file(filepath)):
            print(plctranslator.log_stream.getvalue())
            status_label.configure(text="Convertion is possible", text_color='black', fg_color="orange", )
            # Tømmer textboxen før ny tekst legges til
            textbox.delete("1.0", "end")
```

Figur 22. Metode for å velge kildefil

6.2.2 Valg av destinasjonsmappe

På samme måte som vi valgte kildefil, velger vi destinasjonsmappe, ved hjelp av tkinter sin filedialog klasse. Dette er mappen hvor de ferdig konverterte filene skal opprettes.

```
def choose_destinationfolder():
    """Open a file dialog to select a target folder."""
    mappesti = filedialog.askdirectory()
    if mappesti:
        destinationfolder_var.set(mappesti)
        check_convertion()
```

Figur 23. Metode for å velge destinasjonsmappe

6.2.3 Konvertering

Metoden «converter()», bruker i all hovedsak den egendefinerte metoden translate. Dette samler alle de konverteringsrelaterte metodene og utvikler de nye TwinCAT-filene i valgt destinasjons-mappe. Metoden kjøres når en trykker på convert-knappen i det grafiske brukergrensesnittet.

```
def converter():
    """Converts the source file to the target folder."""
    file = sourcefile_var.get()
    plctranslator.translate(file, destinationfolder_var.get())
    end_index = textbox.index(ctk.END)
    lines = textbox.get("1.0", end_index).count("\n")
    textbox.insert(float(lines), plctranslator.log_stream.getvalue())
    plctranslator.log_stream.truncate()
    plctranslator.log_stream.seek(0)
    status_label.configure(text="Convertion successful",text_color='white', fg_color="green")
    converting_btn.configure(state="disabled")
```

Figur 24. Metode for å fullføre konverteringen

6.2.4 Ekstra funksjoner

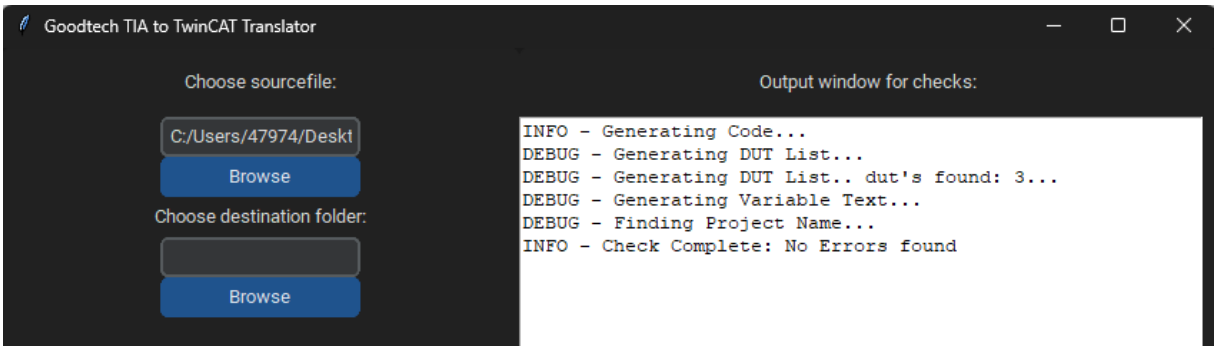
I utformingen av brukergrensesnittet, valgte vi å legge til ekstra funksjoner som «open destination folder»- og «show convertion result»-knappene som en kan se i Figur 21. «Open destination folder»-knappen åpner destinasjonsmappen, som er gunstig etter en har konvertert, slik at en enkelt kan finne de nye filene i destinasjonsmappen uten behov for å lete gjennom filutforsker selv. «Show convertion result»-knappen, viser den konverterte koden i tekstboksen til høyre i det grafiske brukergrensesnittet. Dette kan bidra til mer effektiv manuell verifikasjon av koden, om det skulle vært nødvendig.

6.3 Bruk av logging

I flere av figurene som beskriver Python-kode, kan en se bruken av funksjonen logging for eksempel i Figur 7. Vi benytter logging biblioteket som et alternativ til Python sin innebygde metode «print()». Dette tillater farget, og oversiktlig tekst i terminal vinduet [33], under utføring av metoder vist i Figur 25. Brukergrensesnittet tillatte ikke farget tekst etter vår forståelse, men hadde fortsatt den oversiktlige strukturen vist i Figur 26.

```
2024-04-23T13:26:33+0200 - src.plctranslator.tia_helpers - tia_helpers.py:61 - DEBUG - Reading SCL file from C:/Users/jomar/OneDrive/
or/twincat_folder_for_all/MOJO_MB_V2.scl
2024-04-23T13:26:33+0200 - src.plctranslator.tia_translator - tia_translator.py:24 - INFO - Generating Code...
2024-04-23T13:26:33+0200 - src.plctranslator.tia_translator - tia_translator.py:29 - DEBUG - Generating DUT List...
2024-04-23T13:26:33+0200 - src.plctranslator.tia_translator - tia_translator.py:32 - DEBUG - Generating DUT List.. dut's found: 3...
2024-04-23T13:26:33+0200 - src.plctranslator.tia_translator - tia_translator.py:35 - DEBUG - Generating Variable Text...
2024-04-23T13:26:33+0200 - src.plctranslator.tia_translator - tia_translator.py:40 - DEBUG - Finding Project Name...
2024-04-23T13:26:33+0200 - src.plctranslator.tia_translator - tia_translator.py:94 - INFO - Check Complete: No Errors found
```

Figur 25. Logging i praksis, terminal vinduet i Visual Studio Code



Figur 26. Logging-output i tekstvinduet, grafisk brukergrensesnitt

6.4 Enhetstesting

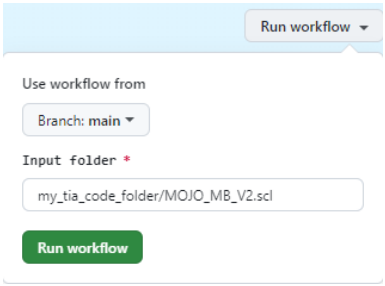
For å sikre effektiv fremgang og raskt bli oppmerksom på feilkode ved endring av metoder, har vi laget enhetstester for hver metode. I Figur 27 ser vi et eksempel på en enhets test. Vi har utviklet en SCL-fil som inneholder en del av forskjellene typisk sett i MB- og SBE-blokker. Deretter har vi manuelt gjort metodene sine oppgaver for så å sammenligne metodens svar på oppgaven, med svaret vi fikk av manuell håndtering. På denne måten kan vi gjøre endringer i metodene, deretter sjekke at metoden fortsatt produserer riktige resultater.

```
def test_generate_variable_text(self):
    expected_output = """X : Bool;
  Safetysensor : Bool;
  MyInput : Bool;
  MyReset : Bool;
  MyPV : Int;
END_VAR

VAR_OUTPUT
  Y : Bool;
  Alarm : Bool;
  EmergencyStop : Bool;
  Qatt : Bool;
  MyCounter : Int;
END_VAR

VAR
  TimerTON {InstructionName := 'TON_TIME'; LibVersion := '1.0'; S7_SetPoint := 'False'} : TON_TIME;
  TimerTOF {InstructionName := 'TOF_TIME'; LibVersion := '1.0'} : TOF_TIME;
  TimerTP {InstructionName := 'TP_TIME'; LibVersion := '1.0'} : TP_TIME;
  InvertedX { S7_SetPoint := 'True'} : Bool;
  AlarmTimer {InstructionName := 'TON_TIME'; LibVersion := '1.0'; S7_SetPoint := 'False'} : TON_TIME;
  Param : "Param_MB_V1";
  OsSta : "OsSta_MB_V1";
  CTU {InstructionName := 'CTU_INT'; LibVersion := '1.0'} : CTU_INT;
END_VAR"""
    result = generate_variable_text(TestTiaTranslator.full_text)
    self.assertEqual(result, expected_output)
```

Figur 27. Eksempel på en enhetstest for metode

6.5 GitHub Runner

Et av kravene er å kunne kjøre skriptet på en GitHub Runner. Med hjelp fra oppdragsgiver har vi utviklet et ekstra skript i tillegg til det grafiske brukergrensesnittet, som fungerer på en GitHub Runner. GitHub Runneren kan kjøre skriptet som en workflow. Det vil si å kunne kjøre Python kode direkte i GitHub [34]. Dette skriptet benytter metoden Translate, og bruker kun kildefil som inndata. «Run workflow» knappen i Figur 28, utfører konvertering av gitt fil. Dette gjør det mulig å kunne utføre konverteringer direkte i GitHub, noe som kan effektivisere prosessen av automatisk konvertering [35].



Figur 28. Start av en GitHub workflow

```python
if __name__ == "__main__":
    """App will read arguments from the command line and translate the files in the input folder.
    The translated files will be saved in the output folder.
    """
    _LOGGER.debug("Starting")
    expected_number_of_arguments = 2 + 1  # 2 arguments + 1 for the script name
    if len(sys.argv) != expected_number_of_arguments:
        _LOGGER.error("Usage: python app4realz.py <input_folder> <output_folder>")
        sys.exit(1)
    input_folder = sys.argv[1]
    output_folder = sys.argv[2]

    _LOGGER.debug(f"Translating files in {input_folder} to {output_folder}")
    translate(input_folder, output_folder)
```

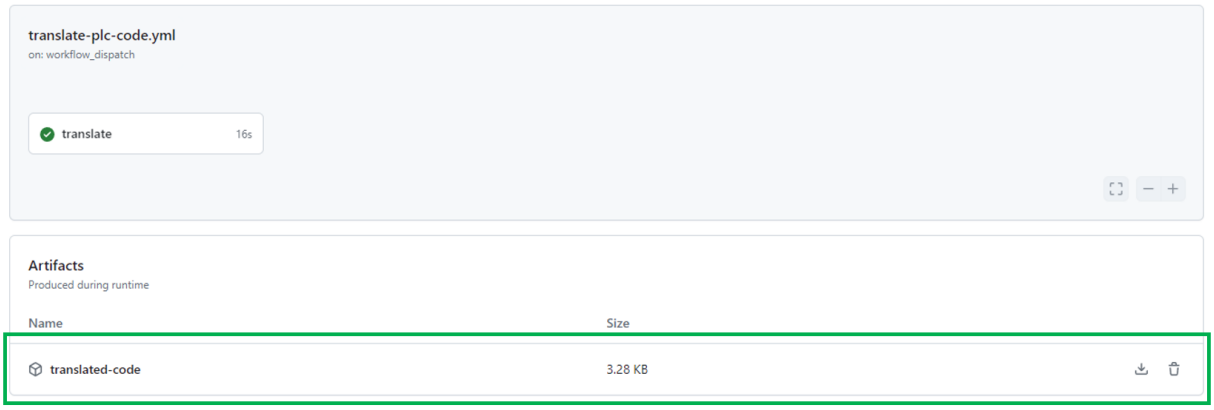
Figur 29. Kodeutklipp av GitHub Runner-skript

```yaml
name: Translate PLC code
on:
  workflow_dispatch:
    inputs:
      input_folder:
        description: 'Input folder'
        required: true
        default: 'my_tia_code_folder'
        type: string
jobs:
  translate:
    runs-on: ubuntu-latest

    steps:
      - name: Checkout repository
        uses: actions/checkout@v4

      - name: Set up Python 3.12
        uses: actions/setup-python@v5
        with:
          python-version: 3.12

      - name: Install dependencies
        run: |
          python -m pip install --upgrade pip
          pip install .

      - name: Translate PLC code
        run: |
          python app4realz.py ${{ github.workspace }}/${{ github.event.inputs.input_folder }}
          ${{ github.workspace }}/my_translated_code_folder

      - name: Upload translated code
        uses: actions/upload-artifact@v4
        with:
          name: translated-code
          path: ${{ github.workspace }}/my_translated_code_folder
```

Figur 30. Konfigurasjonsfil for GitHub Actions

Figur 30, viser konfigurasjonsfilen for GitHub Actions, sendt fra arbeidsgiver. «On: workflow_dispatch», tillater manuell kjøring av Python-skriptet med «input folder» som inndata i GitHub [36]. Konfigurasjonsfilen kjører også skriptet «app4realz.py» som er Python-filen vist i Figur 29. «app4realz.py» benytter hovedsakelig metoden «translate», som konverterer de ønskede SCL-filene.

Inndata som representerer destinasjonsmappe, er satt til «my_translated_code_folder» som standard. Det vil si at det ikke er nødvendig å gi en destinasjonsmappe som inndata. Skriptet lager en mappe med navnet «my_translated_code_folder» som inneholder de nye konverterte filene. Videre kan du laste ned mappen, og importere disse i TwinCAT prosjektet, se Figur 31 .



Figur 31. Resultat av GitHub Runner, hvor konverterte filer kan lastes ned

7 Testing

7.1 Testing av brukervennlighet

I kravspesifikasjonen står det et punkt om brukervennlighet, som innebærer at programmet skal være intuitivt å bruke. En måte å verifisere dette på er ved å gjennomføre en rekke tester. Disse testene tar tiden på hvor lang tid hver student bruker på å gjennomføre en konvertering. Da fikk vi hjelp av syv studenter ved Høgskulen på Vestlandet i Førde, og de ble tildelt en beskrivelse på hvordan testen skal gjennomføres. Denne beskrivelsen omfatter bruk av konverteringsfunksjonen og andre hjelpsomme innebygde metoder i applikasjonen. Når studenten har lest gjennom hva som skal gjøres, åpner vi applikasjonen til dem og starter tiden. Figur 32 viser instruksjon som vi ga til studentene.

Start:
Alt av funksjoner skal gjennomføres med applikasjonens innebygde funksjoner.
1. Bruk appen til for å navigere deg frem til blokken/filen MOJO_MB_V2.scl.
2. Når denne filen er valgt skal du velge destinasjonsmappen du vil filen skal bli konvertert i .
3. Konverter filen du har valgt ved å trykke på "Convert".
4. Bruk appens innebygde funksjon for å åpne destinasjonsmappen med de konverterte filene.
Ferdig!

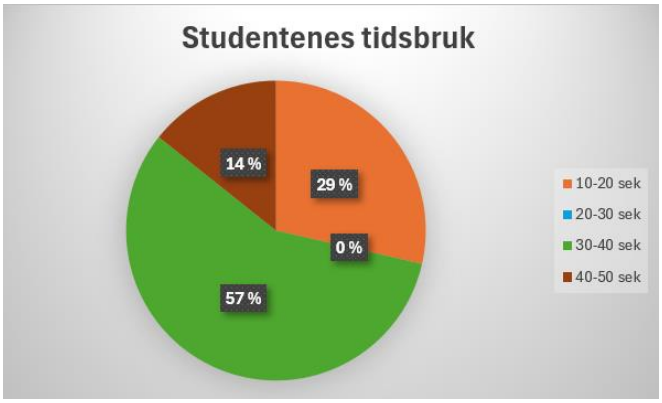
Figur 32. Beskrivelse av oppgave for testsubjektene

Figur 33 viser en tabell over studentenes tidsbruk i stigende rekkefølge. Alle testsubjektene lå innenfor tidsintervallet 12 til 46 sekund.

Student 1	12.58	sekunder
Student 6	18.49	sekunder
Student 2	32.38	sekunder
Student 4	32.44	sekunder
Student 5	37.37	sekunder
Student 7	38.47	sekunder
Student 3	45.92	sekunder

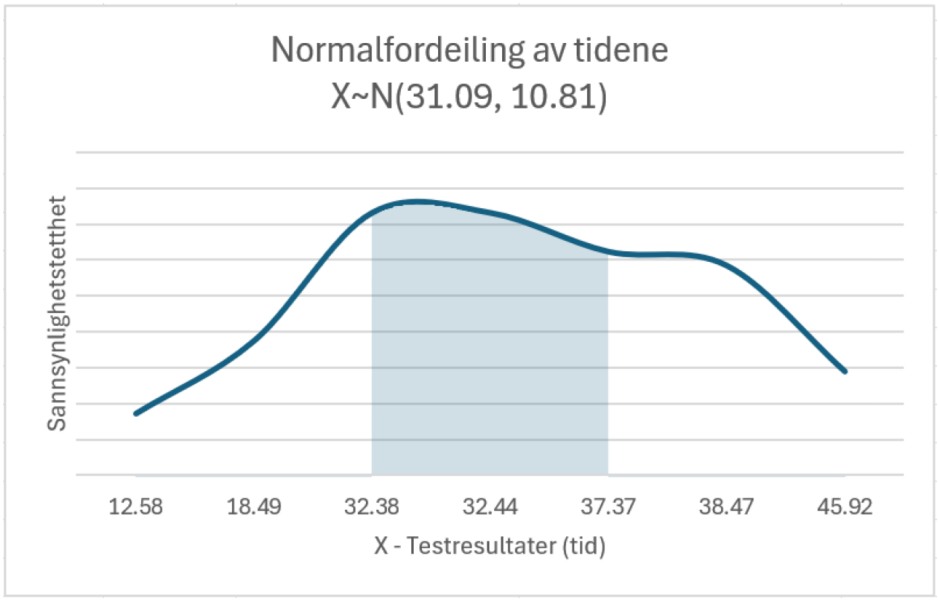
Figur 33. Tabell over tidsbruk på test

Figur 34 viser et diagram av testtidene i intervaller basert på deres varighet. Vi har valgt å fordele testresultatene i båser hvor hvert intervall er på 10 sekunder. Dette er for å gjøre det lettere se sammenheng mellom tidsbruken. Majoriteten av testresultater havner imellom 10-40 sekunder.



Figur 34. Kakediagram over tidsbruk

Når testsubjektene gjennomfører slike tester på applikasjonen vår, kan vi avdekke hvorvidt applikasjonen er enkel å bruke. Et eksempel på dette er for testsubjektet som brukte mellom 40-50 sekunder, han var ikke oppmerksom på «Show destination folder»-knappen er en innebygd funksjon i applikasjonen, han så derfor etter de konvertere filene i filutforsker. Dette kan enten tyde på en dårlige instrukser, eller at grensesnittet ikke var tydelig nok.



Figur 35. Normalfordeling av tidsbruk

Tid	Sannsynlighetstetthet	Kumuliativ sannsynlighet
12.58	0.008522705	-100
18.49	0.018706941	-100
32.38	0.036628671	0.036628671
32.44	0.036603929	0.036603929
37.37	0.03117028	-100
38.47	0.029231726	-100
45.92	0.014412024	-100

Figur 36. Tabell av sannsynlighet for tidsbruk

Gjennomsnitt	Standardavvik
31.09285714	10.81466163

Figur 37. Gjennomsnitt og standardavvik

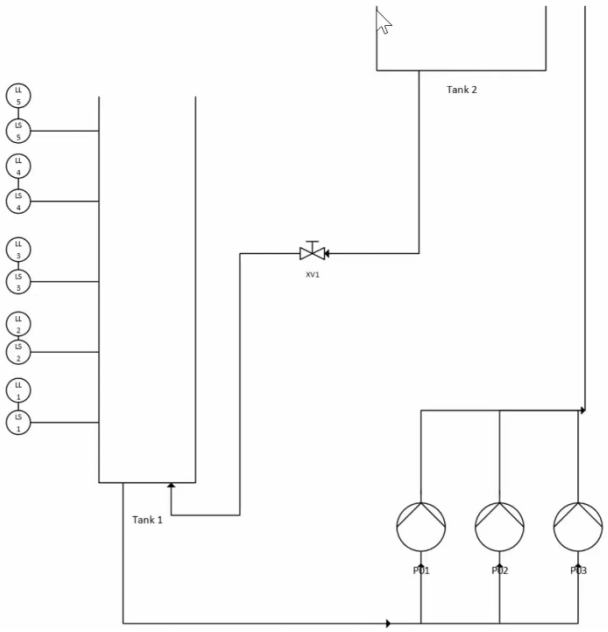
7.2 Normalfordeling

For å forstå spredningen av testtidene, brukte vi normalfordeling. Dette ble analysert ved hjelp av Excel-funksjonene «GJENNOMSNITT» og «STDAV.P», for sannsynlighetstetthet brukte vi «NORM.FORDELING(verdi; gjennomsnitt; standardavvik; USANN)». Dette ga oss en visuell representasjon av test dataen, vist i Figur 35, som viser at de fleste testresultatene befinner seg i tidsperspektivet mellom 32 til 37 sekunder. Normalfordelingen er ikke en ren bjelleformasjon, fordi det bare var syv testsubjekter som kunne gjennomføre testene. Figur 37 viser et gjennomsnitt på 31,09 sekunder og standardavvik på 10,81. Standardavviket er relativt høyt, noe som indikerer en bred spredning i tidsbruken blant testsubjektene som betyr at enkelte tester ble brukt betydelig mer tid enn gjennomsnittet.

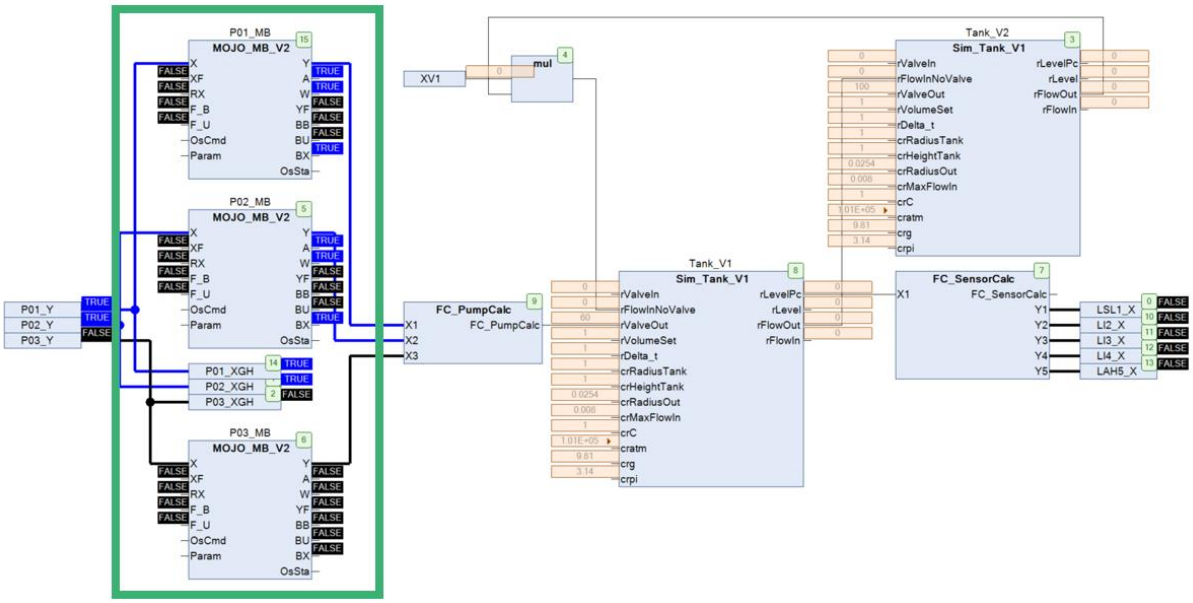
7.3 Testing av programkode

7.3.1 Simulering i TwinCAT

For å teste funksjonalitet har vi utført simulasjoner i TwinCAT, med de konverterte blokkene. Vi benyttet en veilednings video, tilsendt fra arbeidsgiver for å konstruere et simulert pumpeanlegg. [37] Det simulerte pumpeanlegget representerer ett modell pumpeanlegg som vi har tilgjengelig, Figur 38 viser et rør- og instrumentdiagram ofte kalt P&ID, av modell pumpeanlegget. Videre ble de konverterte MB-blokkene implementert i simulasjonen, se Figur 39. De konverterte blokkene fungerte som forventet, og var kompatible med TwinCAT.



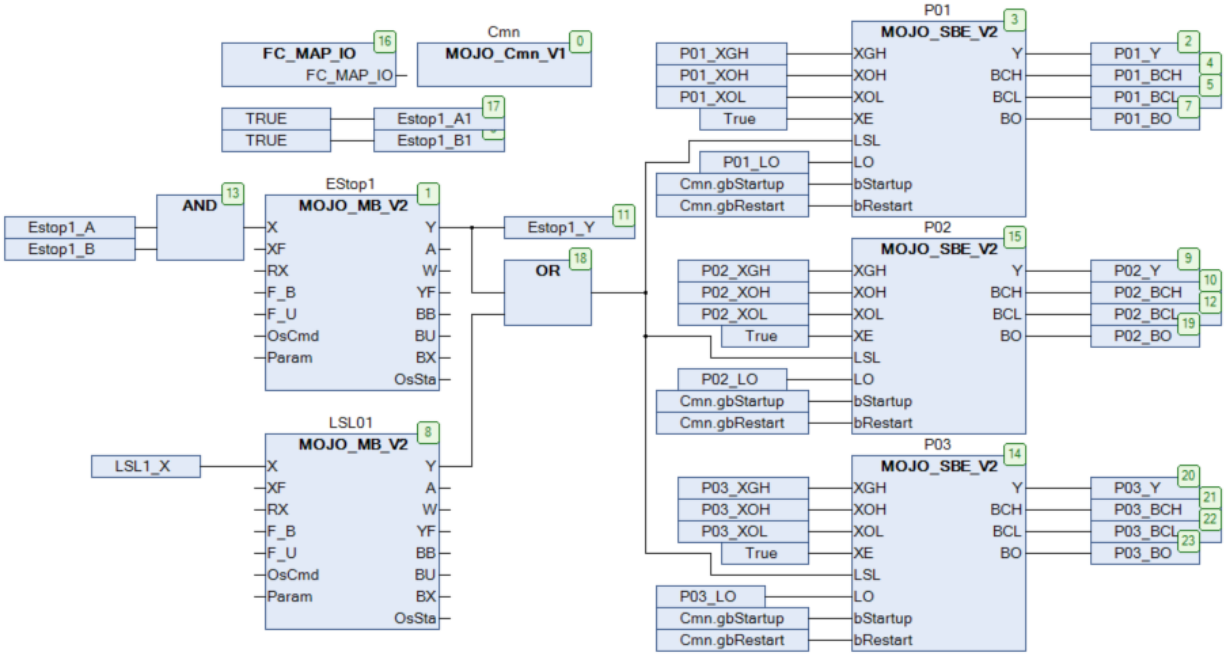
Figur 38. P&ID av modell pumpeanlegg



Figur 39. Simulasjon av MPA i TwinCAT (skjematisk diagram)

7.3.2 Testing på fysisk MPA

Vi begynte med å utvikle programmet som skulle kjøre på anlegget vist i Figur 40. Her tok vi utgangspunkt i en veiledningsvideo tilsendt fra arbeidsgiver. Videoen viser hvordan en kan utvikle et TwinCAT program som passer til styring av pumpeanlegg med en Beckhoff-PLS, med funksjoner som for eksempel å styre de forskjellige pumpene og alarmsystem. I veiledningsvideoen blir ferdigkonstruerte MB- og SBE blokker benyttet. Vi byttet disse blokkene ut med Siemens kompatible blokker, som ble konvertert ved hjelp av Python skriptet. Vi organiserte et møte med arbeidsgiver, for å utføre testene på modell pumpeanlegget som vist i Figur 41. Funksjonaliteten til Beckhoff programmet var den samme både med funksjonsblokker utviklet i TwinCAT, og blokker utviklet i TIA portal, som ble konvertert ved hjelp av Python skriptet.



Figur 40. Beckhoff program for modell pumpeanlegg

Figur 41. Bilde av modell pumpeanlegg

8 Diskusjon

Gjennom dette prosjektet har vi utforsket muligheten for å automatisere konverteringsprosessen mellom PLS-systemer, spesifikt Siemens og Beckhoff. Prosjektet har gitt oss en verdifull innsikt i automatisering av prosesser, men også noen utfordringer knyttet til metodene vi brukte.

8.1 Metodevalg og implementasjon

Valget av programmeringsspråk er sentralt for å løse denne oppgaven på mest mulig effektiv måte. Vi valgte Python, fordi det tilbyr kraftige biblioteker som er veldig sentralt i denne typen oppgaveløsning, nemlig filmanipulasjon og tekstbehandling. Videre refleksjon antyder at utforsking av andre mulige brukergrensesnitt ikke hadde forbedret det aktuelle grensesnittet vi har nå. Måten brukergrensesnittet er oppbygget nå er godt og valget av Tkinter, og Customtkinter var det beste brukergrensesnittet vi kom over. Andre valg kunne vært «PyQt5» [38], som også er et populært brukergrensesnitt for Python. Vi valgte denne bort, fordi tkinter var mye enklere å bruke, og veldig er fleksibelt, samt lett og utføre endringer. Det eneste vi måtte gjøre for å bruke tkinter var å installere biblioteket og importere det i Python programmet, i motsetning til «PyQt5», hvor man må laste ned et eksternt program.

8.2 Bruk av teknologiske verktøy

Bruk av kunstig intelligens gjennom GitHub CoPilot for å assistere i kodingsprosessen har vist seg å være til stor hjelp. Den har effektivisert prosessen betraktelig, ved å foreslå kodealternativer. Vi syns CoPilot er svært god på å tilføye vanlige programmeringsproblemer, noe som ellers hadde tatt tid å skrive selv. På en annen side fører dens begrensende forståelse for mer komplekse kontekster til feil. Dette kan medføre skjulte feil som kan være vanskelig å oppdage, og som kan skape problemer med programmeringsutskriften. CoPilot har derimot møtt våre forventninger i stor grad, og har vist seg å være et veldig godt verktøy.

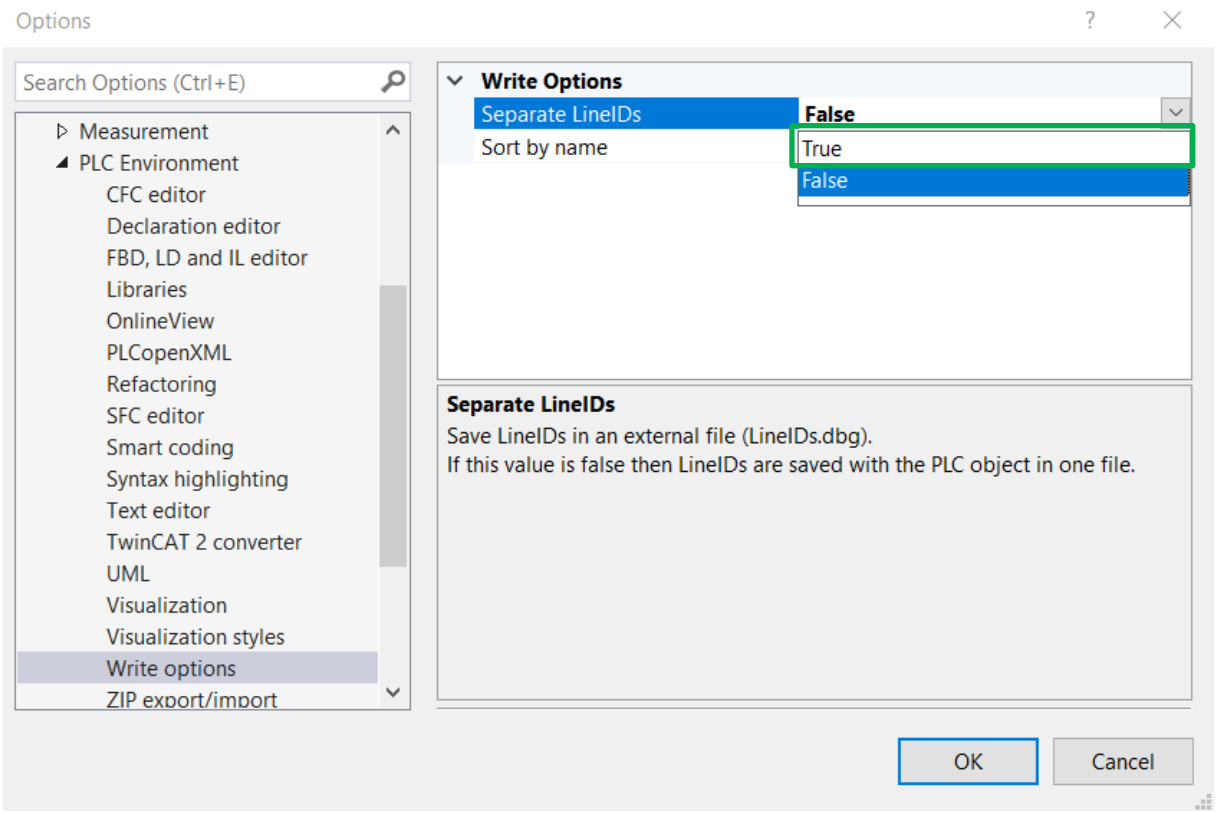
8.3 Progresjon og utfordringer

I starten av prosjektet fokuserte vi på å få en god oversikt over forskjellene mellom Siemens- og Beckhoff-kode. Ved hjelp av utleverte blokker for Siemens og tilsvarende blokker for Beckhoff, kunne vi sammenligne og adressere forskjellene på tvers av leverandørene. Etter vi hadde opparbeidet oss en god oversikt over forskjellene, startet arbeidet med å utvikle et Python-skript. Som et resultat av god oversikt og forståelse av forskjellene, har vi hatt en stabil progresjon gjennom utviklingen av Python-skriptet. Når vi skulle importere de nye filene inn i TwinCAT sine systemer, møtte vi på en utfordring. Filene lot seg ikke importere, etter mye utarbeiding av kode, og nøye vurdering av forskjellene fra vår konverterte fil sammenlignet med den utleverte fungerende filen, innså vi at problemet aldri lå i TwinCAT koden. Problemet var at de konverterte filene ikke var produsert med UTF-8 formatet, noe som førte til feiltolkning av TwinCAT, på grunn av spesialtegn. Vi løste dette problemet med å gi UTF-8 som en parameter i metoden som produserte filene. Det var altså en veldig enkel løsning, men tidkrevende da vi feilsøkte i feil område.

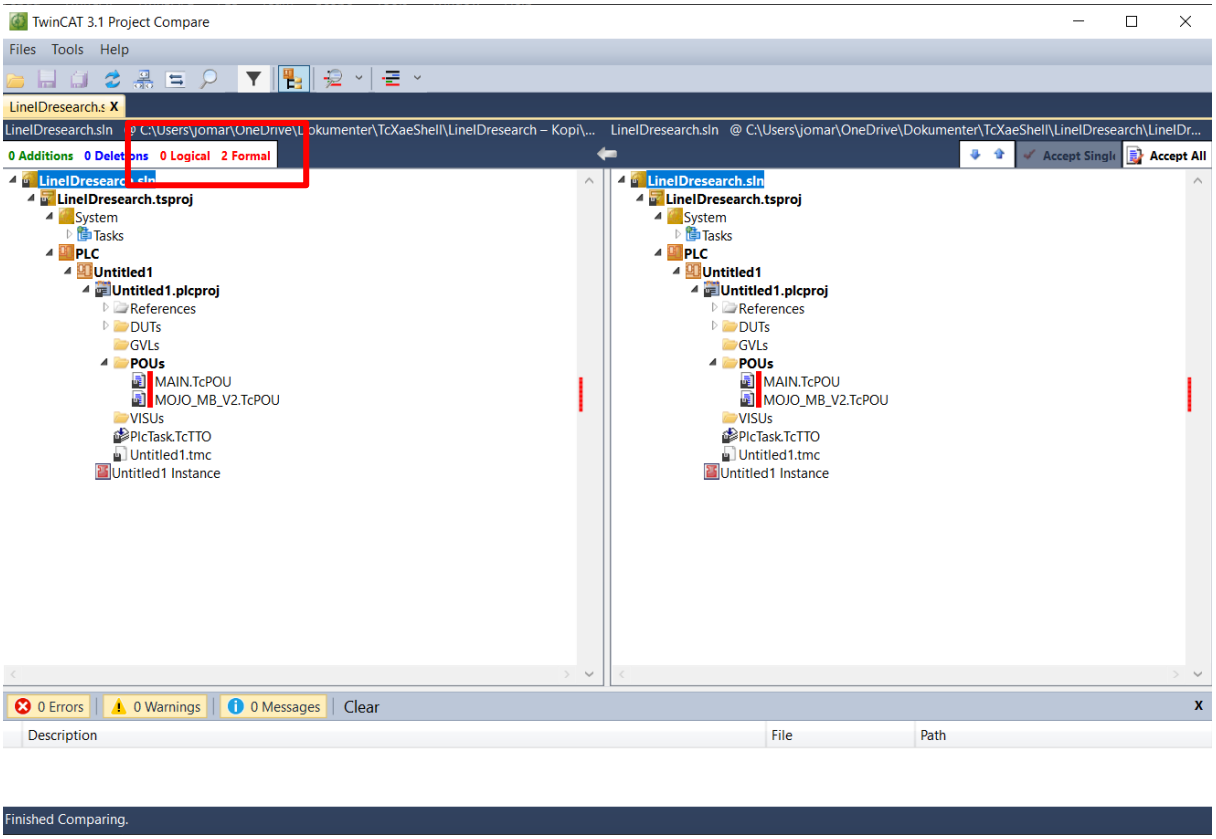
Når vi undersøkte de representative MB-blokk filene til både Siemens og Beckhoff, oppdaget vi et lite segment på slutten av TwinCAT filen som inneholdt LineID, vist til høyre i Figur 42. Dette forekommer ikke i Siemens sine filer. Dette segmentet er vanskelig å forstå, da det viste svært liten sammenheng med resten av filens innhold. Vi prøvde å fjerne segmentet, og sjekket om filen fortsatt er kompatibel med TwinCAT, det er den. Vi merket også at når vi lagret TwinCAT programmet, så oppdaterte TwinCAT sin egen fil med LineID infoen. Dette viste seg da å være en fundamental forskjell vi ikke trengte å ta hensyn til. En standard innstilling for TwinCAT er også at «sort by name» innstillingen er satt til sant. Dette betyr at TwinCAT forholder seg til navn, og ikke id'en som er representert i filen. [39]

Figur 42. Beckhoffs oppdatering av LineID

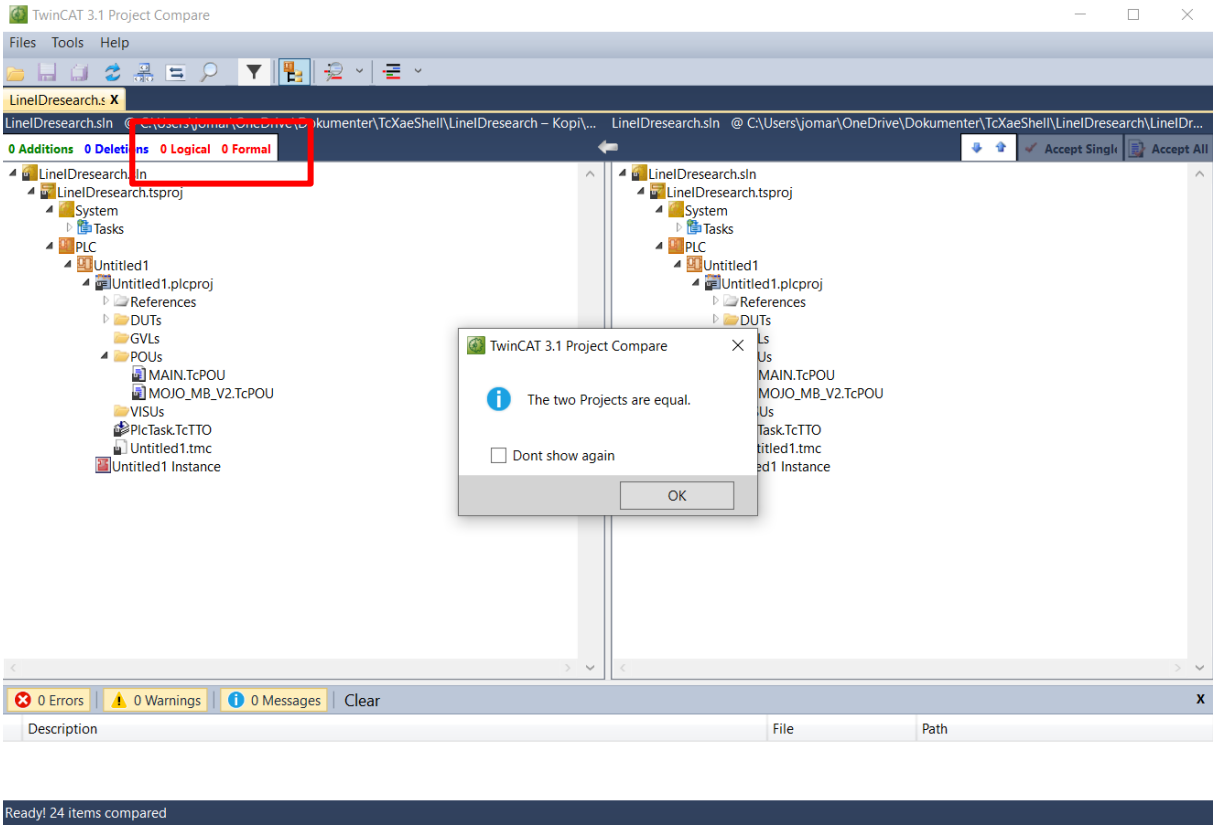
TwinCAT har en sammenligningsapplikasjon for å kunne sammenligne prosjekt. Applikasjonen funker slik at en velger to prosjekt, og får tilbakemelding om eventuelle forskjeller mellom prosjektene [40]. Når vi testet å sammenligne prosjekt med ferdig konstruerte blokker sammenlignet med våre konverterte blokker, oppdaget vi noen formelle forskjeller, se Figur 44. Selv om prosjektet så helt likt ut i grensesnittet, var det noen underliggende forskjeller i filene. Disse forskjellene hadde en sammenheng med «LineID». Vi fant en diskusjon på GitHub hvor andre med samme problem hadde funnet en løsning. Løsningen var å endre en innstilling i TwinCAT, ved å endre «Seperate LineIDs» til «True», se Figur 43. Denne innstillingen sørger for at «LineID» segmentet havner i en egen fil, noe som resulterer i null formelle forskjeller, se Figur 45, når en sammenligner prosjektene i sammenligningsapplikasjonen [41].



Figur 43. «Seperate LineIDs », true/false



Figur 44. «TwinCAT compare»-vindu med to formelle varslinger



Figur 45. «TwinCAT compare»-vindu med null formelle varslinger

8.4 Videre arbeid

Skriptet slik det er nå tar kun for seg forskjeller som typisk forekommer i MB- og SBE-blokker. En naturlig videreutvikling av skriptet ville vært å også konvertert forskjeller som kan oppstå i andre blokker. Om vi skulle skalert opp skriptet for flere forskjeller, ville vi brukt samme fremgangsmåte som tidligere. Dette vil si, å spørre om flere blokker som forholder seg til IEC 61131-3 standarden fra arbeidsgiver, sammenligne dem, og deretter lage metoder som utfører konvertering.

Et av tilleggskravene er å kunne benytte KI som et alternativt forsøk på konvertering dersom skriptet ikke skulle klare å håndtere konverteringen. Tidsplanen tillot ikke dette tilleggskravet, men vi mener det kan være en naturlig videreutvikling. Ettersom at vi har testet å sende inn Siemens-kode til ChatGPT, med relativ høy suksessrate, mener vi at dette ville vært en god ekstra egenskap, som kunne gjort programmet mer funksjonelt.

Vi mener også at konvertering tilbake fra Beckhoff til Siemens kunne vært en grei egenskap, dersom man hypotetisk sett skulle gjort endringer i noen av blokkene for Beckhoff. Oppdragsgiver har nevnt at dem oftest bruker Siemens TIA Portal når dem vedlikeholder blokkene, som betyr at denne funksjonaliteten ikke nødvendigvis er så viktig. Dette er også et tilleggskrav, vi ikke hadde tid til å utføre.

For å gjøre prosessen enda mer automatisk, kunne man undersøkt TIA Portal sin «Version Control Interface». Dette brukergrensesnittet lar deg gjøre endringer i filer i GitHub, direkte i TIA Portal. [42] En kunne videre konfigurert GitHub runneren til å kontinuerlig oppdatere eventuelle TwinCAT-filer, ved endringer i de tilsvarende Siemens-filene.

9 Konklusjon

Vi har utviklet et Python-skript for automatisert konvertering av filer fra Siemens TIA Portal til filer kompatible med Beckhoff. Formålet med skriptet er å kunne effektivisere prosessen av vedlikehold av forskjellige IEC 61131-3 standardiserte blokker, spesifikt MB- og SBE-blokker. Vi hadde også et krav om brukervennlighet, at applikasjonen skulle være enkel å bruke. Alle konverteringene fra testsubjektene var vellykkede ved bruk av applikasjonen, derfor konkluderer vi med at grensesnittet er oversiktlig og brukervennlig nok til å godkjenne kravet.

Skriptet fungerer også på en GitHub Runner, noe som også er et krav. Denne funksjonen bidrar til enda høyere effektivitet enn det grafiske brukergrensesnittet, ettersom konvertering foregår direkte i GitHub nettleseren.

Python-skriptet vi har utviklet klarer å konvertere funksjonsblokker fra Siemens til funksjonsblokker kompatible med Beckhoff. Simuleringen og de fysiske testene på modell pumpeanlegget viser at funksjonsblokkene opprettholder all funksjonalitet etter konvertering. Vi konkluderer derfor med at Python skriptet vil kunne automatisere det som tidligere var en manuell konverteringsjobb.

Python skriptet oppfyller kravene til oppgaven, men vi mener at det er naturlige videreutviklinger som kunne ha økt funksjonaliteten i stor grad, som å skalere skriptet opp, for håndtering av flere forskjeller på tvers av leverandørene. Metoden vi har brukt, med å sammenligne fungerende filer for begge leverandørene, deretter å lage Python metoder som konverterer fra Siemens til Beckhoff, har vist seg å være effektivt. Derfor ville det vært lurt å bruke denne metoden, og sammenlignet andre blokker som forholder seg til IEC 61131-3 standarden.

Selv om vi ikke hadde tid til å utføre flere av tilleggskravene, er vi svært fornøyde med resultatet. De fysiske testene og simuleringene viser at applikasjonen kan brukes i praksis. Vi vil takke Goodtech for samarbeidet, og håper at applikasjonen vil bli brukt i fremtiden.

10 Referanser

[1] «Goodtech,» [Internett]. Available: https://www.goodtech.no/industri/.

[2] Goodtech, «Goodtech,» Wikipedia, 18 Aug 2023 . [Internett]. Available: https://no.wikipedia.org/wiki/Goodtech#cite_note-wikidata-a14b092a4e15dbc17fdfbe5b7b876b2adaf05ce3-v2-3. [Funnet 20 Mars 20].

[3] «Goodtech, hvem er vi,» Goodtech, [Internett]. Available: https://www.goodtech.no/hvem-er-vi/. [Funnet 20 Mars 2024].

[4] «Beckhoff Information System,» Beckhoff, [Internett]. Available: https://infosys.beckhoff.com/english.php?content=../content/1033/tcquickstart/5281654411.html&id=. [Funnet 25 April 2024].

[5] «Guide to Standardization,» Siemens TIA Portal, September 2018. [Internett]. Available: https://cache.industry.siemens.com/dl/files/737/109756737/att_961818/v2/109756737_Standardization_Guideline_DOC_V10_en.pdf. [Funnet 25 April 2024].

[6] I. 61131-3, «IEC 61131-3,» RTA (Real Time Automation), 2024. [Internett]. Available: https://www.rtautomation.com/technologies/control-iec-61131-3/. [Funnet 8 Mai 2024].

[7] S. Dietrich, «PLC Programming With Function Block Diagrams,» Controll automation, 29 Aug 2023. [Internett]. Available: https://control.com/technical-articles/plc-programming-with-function-block-diagrams/. [Funnet 8 Mai 2024].

[8] D. Peterson, «Alternative PLC Programming Languages,» Control automation, 27 Aug 2020. [Internett]. Available: https://control.com/technical-articles/alternative-plc-programming-languages/. [Funnet 8 Mai 2024].

[9] n. standard, «System control diagram,» NORSOK STANDARD I-005, Lysaker, 2013.

[10] R. M. Ferreira, «freeCodeCamp,» 22 Mars 2021. [Internett]. Available: https://www.freecodecamp.org/news/python-string-manipulation-handbook/. [Funnet 5 April 2024].

[11] GitHub, «About GitHub CoPilot and Visual Studio Code,» GitHub Docs, 2024. [Internett]. Available: https://docs.github.com/en/copilot/using-github-copilot/getting-started-with-github-copilot#about-github-copilot-and-visual-studio-code. [Funnet 7 Mai 2024].

[12] «Linting Python in Visual Studio Code,» Visual Studio Code, 15 Juni 2023. [Internett]. Available: https://code.visualstudio.com/docs/python/linting. [Funnet 8 Mai 2024].

[13] Ruff, «Ruff extension for Visual Studio Code,» marketplace.visualstudio, [Internett]. Available: https://marketplace.visualstudio.com/items?itemName=charliermarsh.ruff. [Funnet 8 Mai 2024].

[14] «Improve your python code quality – a python linters overview,» DS STREAM, 2023. [Internett]. Available: https://dsstream.com/improve-your-python-code-quality/. [Funnet 17 April 2024].

[15] G. v. Rossum, «PEP 8 – Style Guide for Python Code,» Python Enhancement Proposals, 9 Des 2023. [Internett]. Available: https://peps.python.org/pep-0008/#maximum-line-length. [Funnet 8 Mai 2024].

[16] I. GitHub, «GitHub Desktop,» desktop.github, 2024. [Internett]. Available: https://desktop.github.com/. [Funnet 1 Mai 2024].

[17] GitHub, «Getting started with GitHub Desktop,» GitHub Docs, 2024. [Internett]. Available: https://docs.github.com/en/desktop/overview/getting-started-with-github-desktop#creating-adding-and-cloning-repositories. [Funnet 19 Mai 2024].

[18] GitHub, «GitHub Docs,» GitHub, 2024. [Internett]. Available: https://docs.github.com/en/pull-requests/collaborating-with-pull-requests/proposing-changes-to-your-work-with-pull-requests/about-pull-requests. [Funnet 7 Mai 2024].

[19] «Discord,» Discord, [Internett]. Available: https://discord.com/. [Funnet 10 April 2024].

[20] H. p. Vestlandet, «Høgskulen på Vestlandet,» hvl, 14 Mars 2024. [Internett]. Available: https://www.hvl.no/student/it-hjelp/Programvareoversikt/microsoft-office-365/. [Funnet 25 Mars 2024].

[21] fernhillsoftware, «Common Elements,» Fernhill SCADA, 2024. [Internett]. Available: https://www.fernhillsoftware.com/help/iec-61131/common-elements/index.html. [Funnet 22 April 2024].

[22] «Step 7 Elementary Data Types,» PLC dev, [Internett]. Available: https://www.plcdev.com/step_7_elementary_data_types. [Funnet 8 Mai 2024].

[23] «Date and time data types,» Beckhoff Information System, [Internett]. Available: https://infosys.beckhoff.com/english.php?content=../content/1033/tc3_plc_intro/2529415819.html&id=. [Funnet 8 Mai 2024].

[24] «pathlib - Object-oriented filesystem paths,» Python.org, 22 April 2024. [Internett]. Available: https://docs.python.org/3/library/pathlib.html. [Funnet 16 April 2024].

[25] «w3schools,» Python, 2024. [Internett]. Available: https://www.w3schools.com/python/ref_string_replace.asp . [Funnet 16 April 2024].

[26] «Python Regex Split String Using re.split(),» PYnative Python Programming, 2024. [Internett]. Available: https://pynative.com/python-regex-split/. [Funnet 16 April 2024].

[27] «PLC & Automation With Liam Bee,» liambee, 30 Juli 2023. [Internett]. Available: https://liambee.me/siemens/ton-tof-tp-timers-and-different-use-cases/. [Funnet 16 April 2024].

[28] Beckhoff, «Beckhoff Information System,» Beckhoff, [Internett]. Available: https://infosys.beckhoff.com/english.php?content=../content/1033/tcplclib_tc2_standard/74406539.html&id=. [Funnet 15 April 2024].

[29] V. A. Merge, «UTF-8 format,» Addo Sign, [Internett]. Available: https://support.vismaaddo.net/hc/no/articles/360017259559-UTF-8-format#h_01GDZTX4ZMTTWZW1YYZB7769R5. [Funnet 30 April 2024].

[30] «tkinter - Python interface to Tcl/Tk,» Python.org, 22 April 2024. [Internett]. Available: https://docs.python.org/3/library/tkinter.html. [Funnet 15 April 2024].

[31] «CustomTkinter,» CustomTkinter, 2024. [Internett]. Available: https://customtkinter.tomschimansky.com/. [Funnet 16 April 2024].

[32] «Tkinter Dialogs,» Python.org, 22 April 2024. [Internett]. Available: https://docs.python.org/3/library/dialog.html#module-tkinter.filedialog. [Funnet 16 April 2024].

[33] A. Zaharia, «Make your own custom color formattingwith Python logging,» github.io, 2023. [Internett]. Available: https://alexandra-zaharia.github.io/posts/make-your-own-custom-color-formatter-with-python-logging/ . [Funnet 16 April 2024].

[34] «Manually running a workflow,» GitHub Docs, 2024. [Internett]. Available: https://docs.github.com/en/actions/using-workflows/manually-running-a-workflow. [Funnet 7 Mai 2024].

[35] B. Whittle, «github,» github, 2024. [Internett]. Available: https://resources.github.com/learn/pathways/automation/intermediate/create-reusable-workflows-in-github-actions/. [Funnet 25 April 2024].

[36] GitHub, «GitHub Docs,» GitHub, 2024. [Internett]. Available: https://docs.github.com/en/actions/using-workflows/triggering-a-workflow#defining-inputs-for-manually-triggered-workflows. [Funnet 7 Mai 2024].

[37] J. Møgster, «TC 03 Creating MPA Simulator,» YouTube, April 3 2017. [Internett]. Available: https://www.youtube.com/watch?v=jVB1LxsxUM4&t=110s. [Funnet 29 April 2024].

[38] «The complete PyQt5 tutorial — Create GUI applications with Python,» pythonGUIs, 4 Feb 2024. [Internett]. Available: https://www.pythonguis.com/pyqt5-tutorial/. [Funnet 7 Mai 2024].

[39] Beckhoff, «Beckhoff New Automation Technology,» Beckhoff, [Internett]. Available: https://infosys.beckhoff.com/english.php?content=../content/1033/tc3_userinterface/18014403202147467.html&id=. [Funnet 7 Mai 2024].

[40] Beckhoff, «TwinCAT3 project comparison window,» Beckhoff, [Internett]. Available: https://infosys.beckhoff.com/english.php?content=../content/1033/project_compare_tool/7609457291.html&id=. [Funnet 10 Mai 2024].

[41] J. Sagatowski, «LineIDs necessary? #85,» GitHub, 3 Jan 2013. [Internett]. Available: https://github.com/tcunit/TcUnit/issues/85. [Funnet 10 Mai 2024].

[42] R. A. A. Yahia, «SolisPLC,» [Internett]. Available: https://www.solisplc.com/tutorials/an-introduction-to-tia-portals-version-control-interface-using-git-and-github#version-control-interface-in-tia-portal. [Funnet 7 Mai 2024].

[43] H. Dvergsdal, «Unicode,» Store Norske Leksikon, 30 Nov 2021. [Internett]. Available: https://snl.no/Unicode. [Funnet 30 April 2024].

Appendiks A Forkortelser og ordforklaringer

API	Application Programming Interface
ChatGPT	Gratis kunstig intelligens system
CI/CD	Continous Integration/ Continous Deployment
C#	Programmeringsspråk utviklet av Microsoft, (C-sharp)
Discord	Kommunikasjonsverktøy
Footer	Seksjon som befinner seg nederst i dokumenter
Funksjonsblokk	Representerer funksjoner eller operasjoner i PLS
GitHub	Applikasjon for versjonskontroll og samarbeid
GitHub Actions	En CI/CD-plattform som tillater automatisering av testing, kodeutførelser og kjøring av GitHub repositorier
Goodtech	Oppdragsgiver
Header	Seksjon som befinner seg øverst i dokumenter
IEC 61131-3	Internasjonal standard for PLS-programmeringsspråk
KI	Kunstig Intelligens
P&ID	Piping instrumentation diagram (rør- og instrumentdiagram)
PEP-8	Python Enhancement Proposal 8
PLS	Programmerbar logisk styring
Python	Programmeringsspråk utviklet av Microsoft
SCL	Structured Control Language
TcDUT	TwinCAT Data Unit Types
TcPOU	TwinCAT Program Organization Unit
TwinCAT	Automatiseringsverktøy for programmering av Beckhoff PLS-er
UTF-8	Kodingssystem for Unicode
Unicode	“Unicode er et utvidbart tegnesett for representasjon av tekstlig informasjon digitalt» [43]

Appendiks B Prosjektledelse og styring

Prosjektet begynte i begynnelsen av januar, 9.jan.2024, hvor vi kontaktet arbeidsgiver på e-post og fikk tilbakemeldinger hva vi skulle gjøre for å sette i gang med prosjektet. Dette innebar hovedsaklig videoer som forklarte systemene vi skulle fordype oss i, i tillegg til at oppdragsgiver kom opp for å undervise oss. På den tiden var oppdragsgiver vår prosjektleder, helt til vi fikk god kontroll på prosjektet i slutten av februar, og begynte å ta styring selv.

B.1 Prosjektorganisasjon

Gjennom prosjektet har vi samarbeidet veldig tett, som er enklere å gjennomføre når man er to på gruppen. Det gjør at man er avhengig av å møte opp til avtalte tidspunkter for å igangsette arbeidsdagen. Det har ikke vært noe prosjektleder i våres tilfelle, da begge stemmene har i stor grad veid like mye. Organiseringen av prosjektet har gått bra, da vi er to stykker, blir det ikke store variasjoner i hvordan vi ønsker at fremdriftsplaner og oppgaver skal se ut eller gjennomføres.

B.2 Fremdriftsplan

fremdriftsplanen vår er vedlagt. Den utviklet vi raskt for å få et overblikk på arbeidsmengden. Det har blitt gjort endringer underveis, som for eksempel «Arbeidspakke 1», hvor vi hadde lagt for mange timer til. Vi måtte redusere dette underveis da vi så at denne delen ikke var så tidkrevende.

B.3 Arbeidslogg

Arbeidsloggen ligger også vedlagt. Vi har benyttet arbeidslogg gjennom hele prosjektet. Dette har vært en fordel til skriving av rapport, men også å sette en finger på hva vi faktisk har gjort den dagen og ulike nye oppdagelser. Vi har inndelt de forskjellige kategoriene med dato, tid, hva som er gjort, hvem som gjorde det, om det var utfordringer og hva som kan/må gjøres til neste arbeidsøkt. Arbeidsloggen har også bidratt til å hva vi burde fokusere mer på, hva som må gjøres til neste økt og alvorlighetsgraden av å ikke få det gjort. Arbeidsloggen ligger som vedlegg i innleveringsmappen.

B.4 Risikoliste

Risikovurdering ovenfor ulike scenarioer, rangert på en skala fra 1-5.

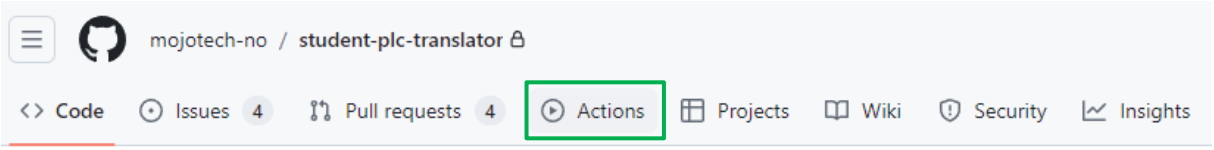
Nr.	Risiko	Påvirkning (skala 1-5)	Sannsynlighet	Risikoverdi (Påvirkning x Sannsynlighet)	Mulige tiltak
1	Uforutsette endringer i funksjonsblokken	5	2	10	Bruker må forholde seg til standarden. Programmet tar høyde for flere forskjeller.
2	Fundementale forskjeller som ikke er tatt høyde for	5	1	5	Gjøre gode undersøkelser og ta høyde for alle fundamentale forskjeller.
3	Oppdatering i programvaren (TIA eller TwinCAT) som resulterer i endringer i oppbyggingen av programfiler	3	3	9	Gjøre skriptet lett leselig og lett å vedlikeholde.

Figur 46. Tabell over risikovurderinger

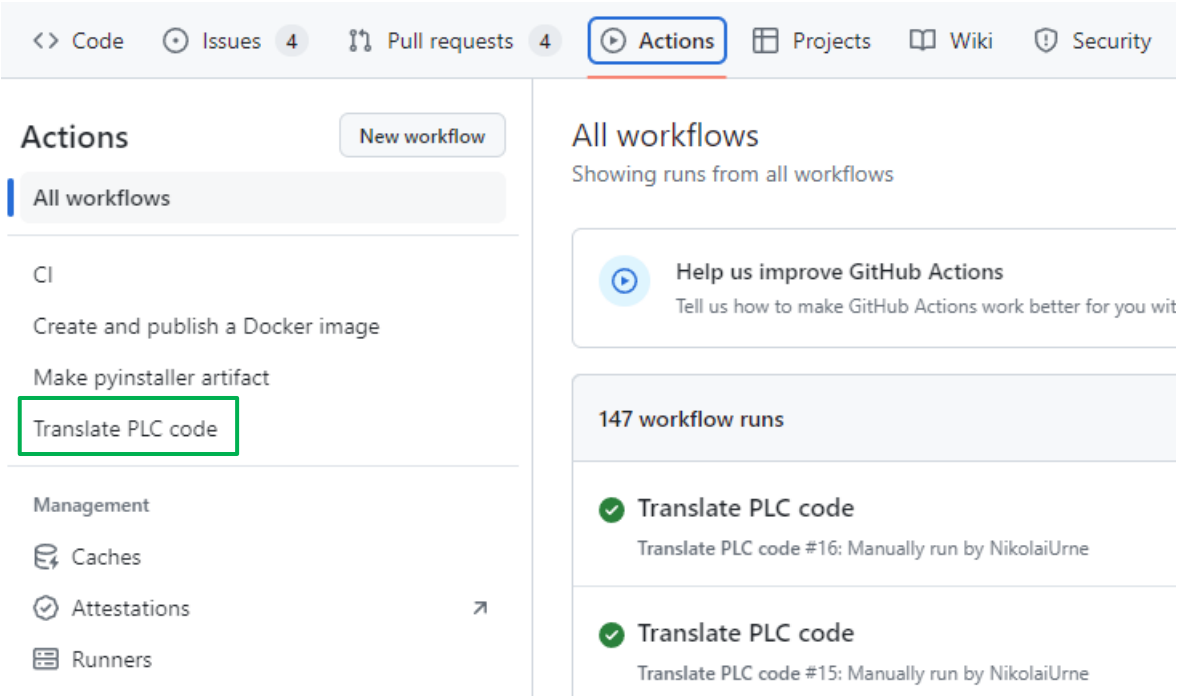
Appendiks C Brukermanualer

GitHub Runner brukermanual er beskrevet i avsnitt C.1, og for konvertering fra Siemens TIA Portal til Beckhoff i C.2. Disse gir en stegvis beskrivelse som leder brukeren gjennom prosessen for konvertering av fil/filer.

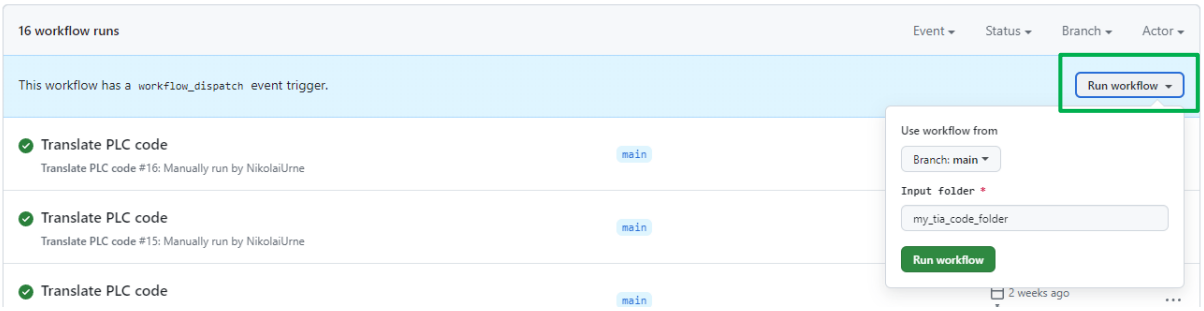
C.1 GitHub Runner



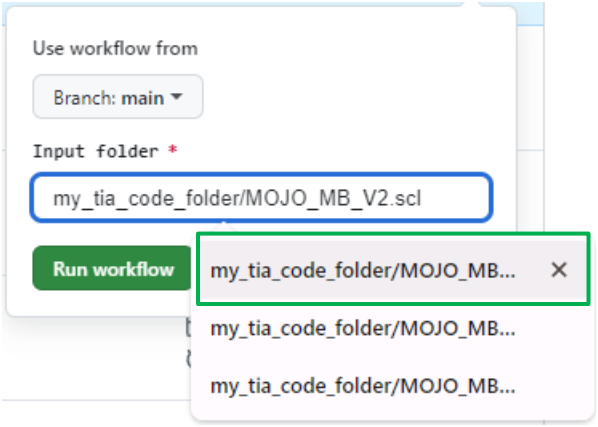
Figur 1. Klikk på "Actions"-knapp



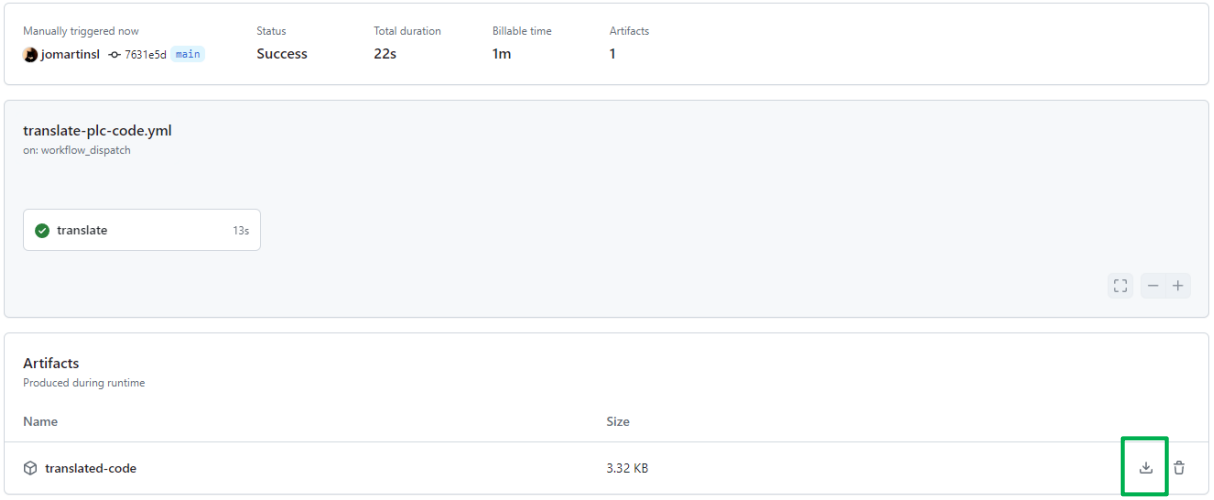
Figur 2. Velg "Translate PLC code"-workflow



Figur 3. Klikk "Run workflow"

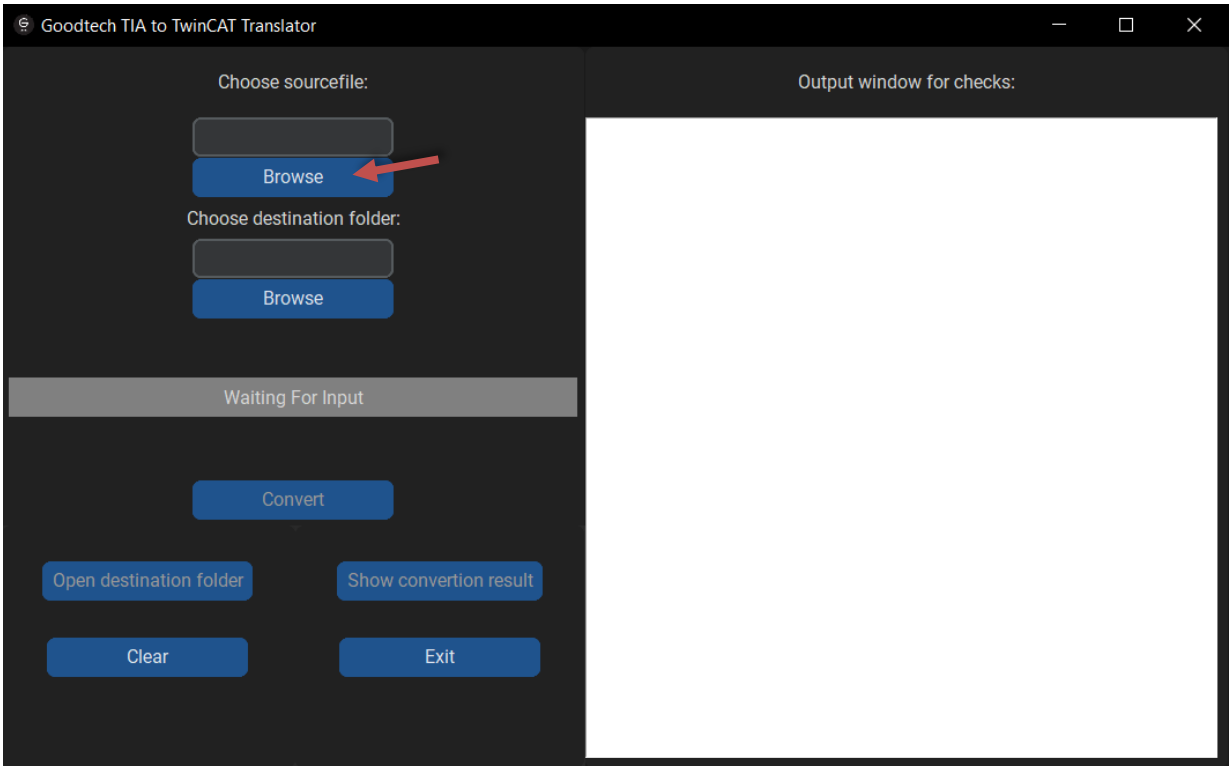


Figur 4. Velg ønsket SCL-fil og "Run workflow"

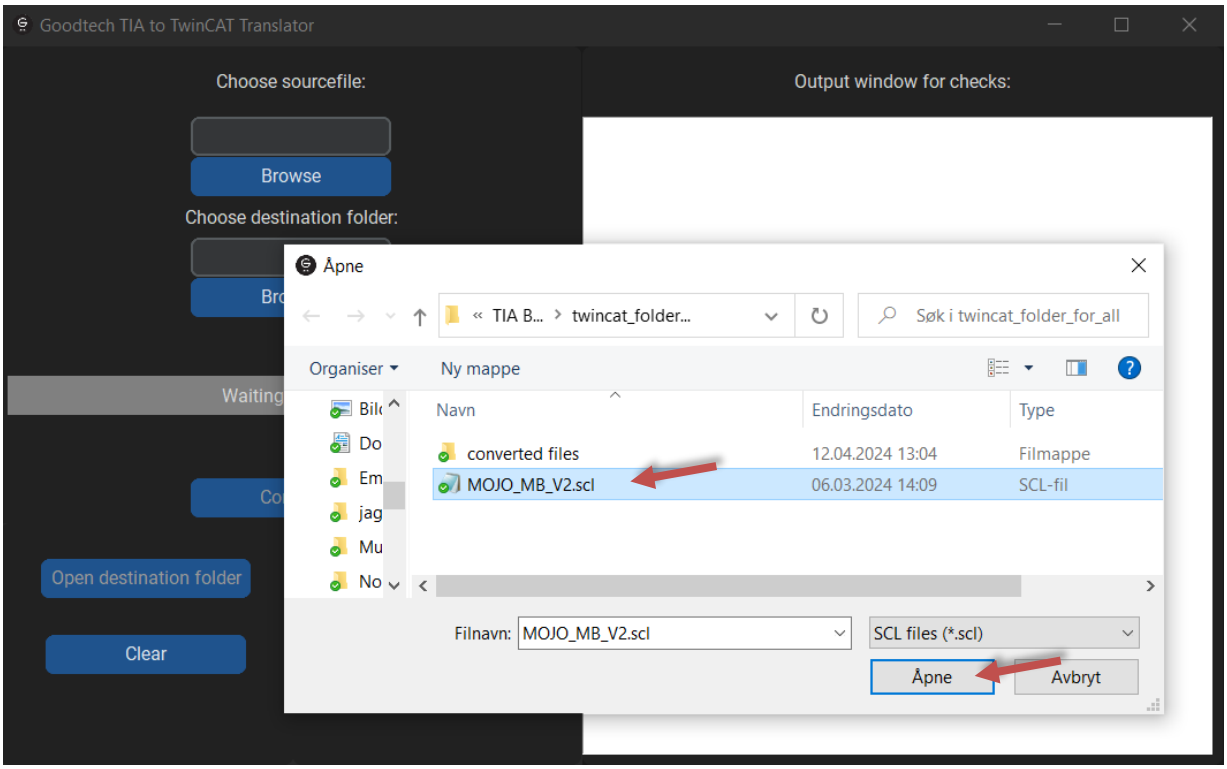


Figur 5. Last ned mappen "translated-code", denne inneholder konverterte filer

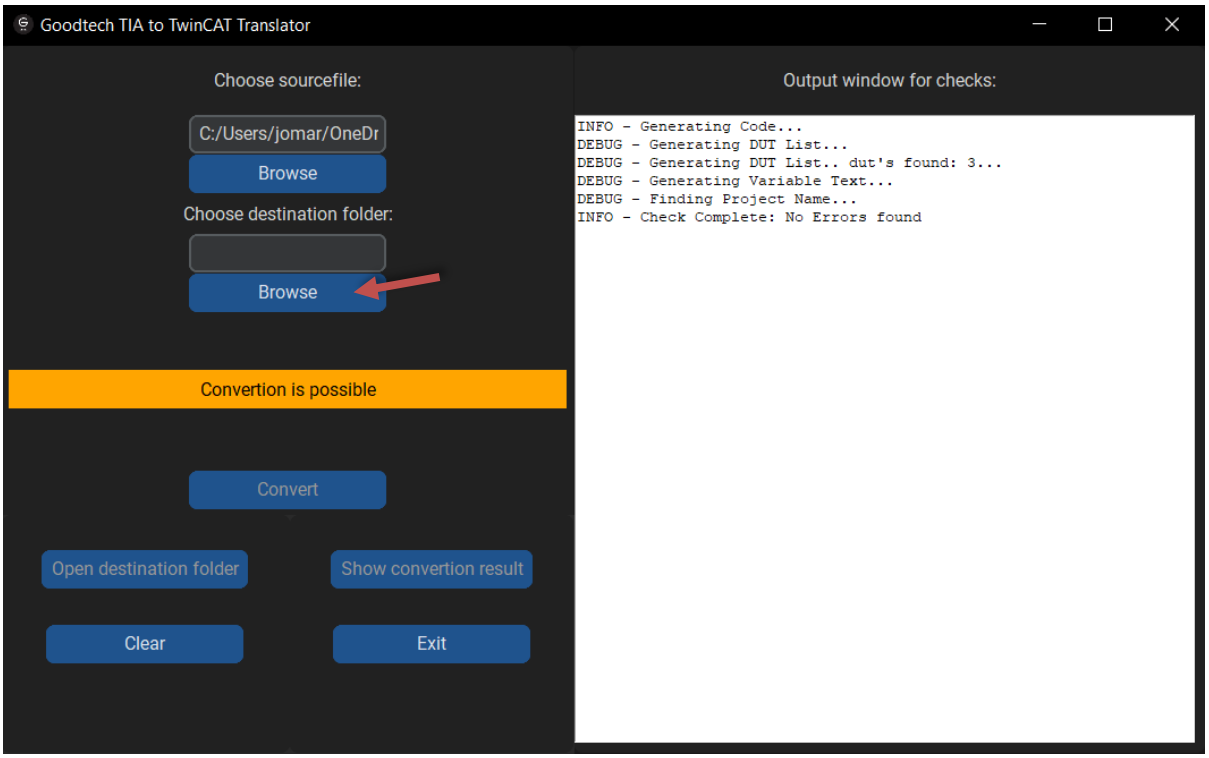
C.2 Brukergrensesnitt



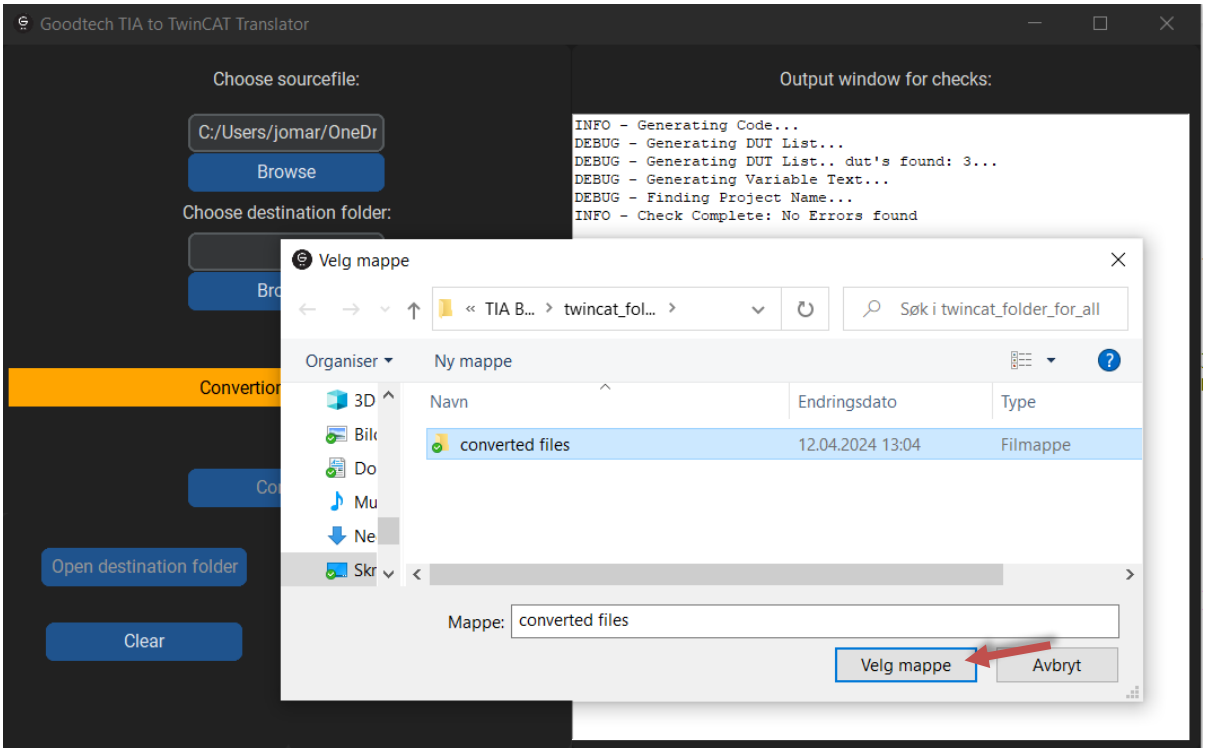
Figur 1. Finn SCL-filen som ønskes konvertert ved å «Browse»



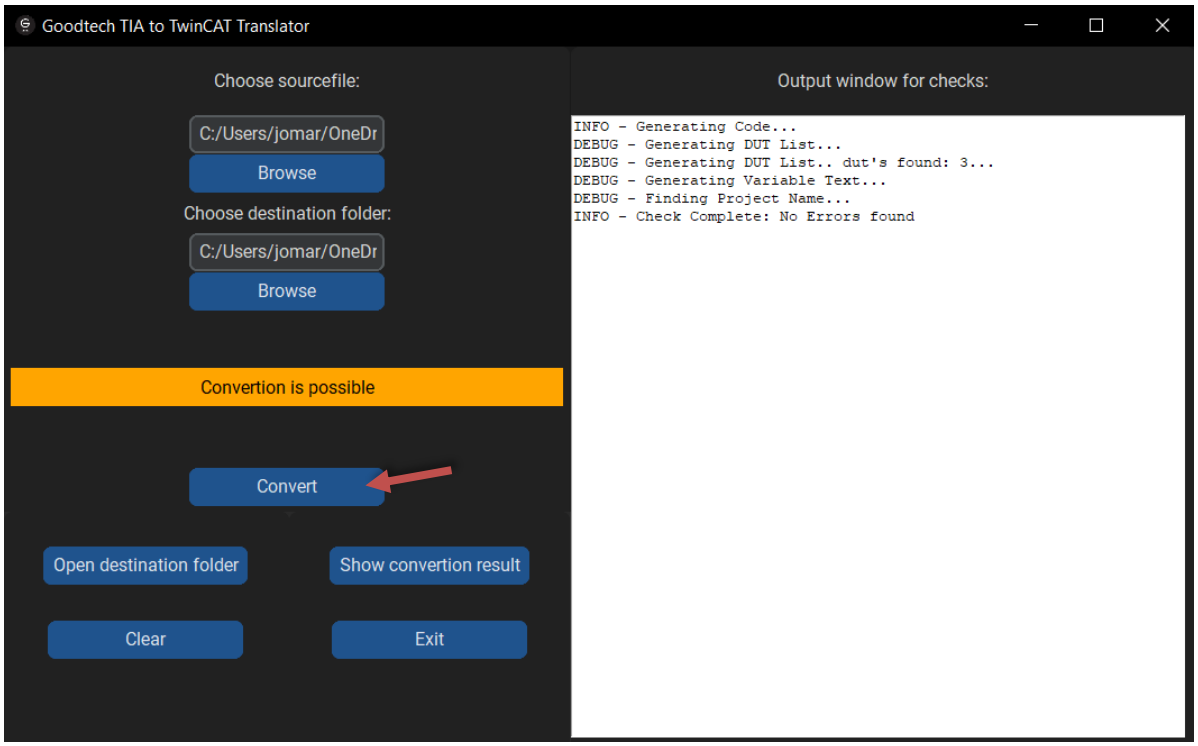
Figur 2. Trykk åpne



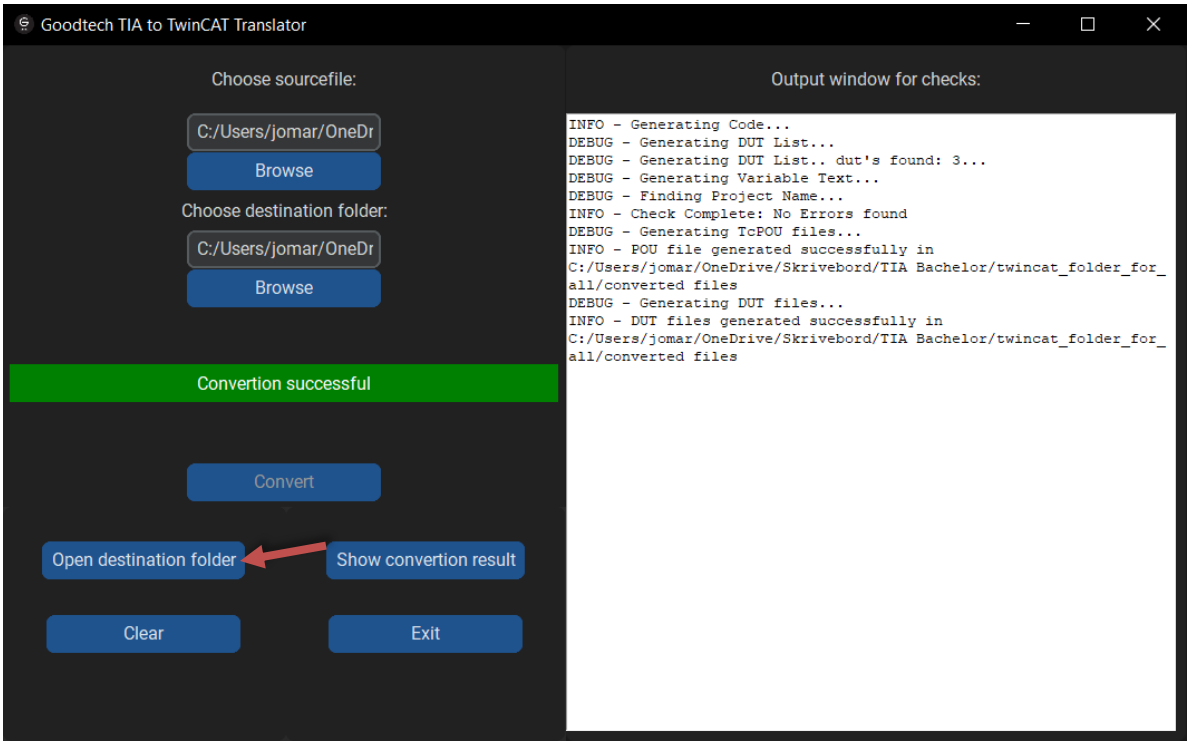
Figur 3. Velg ønsket mappe de konverterte filene skal legges i



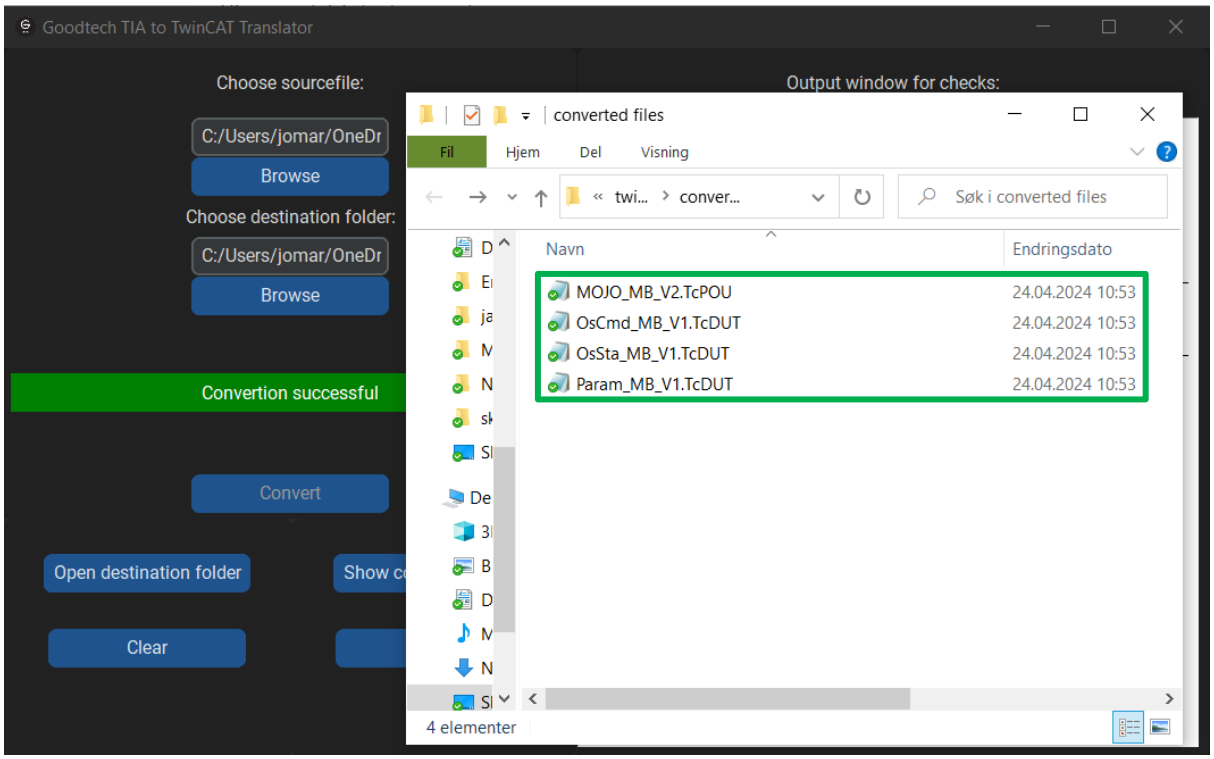
Figur 4. Velg ønsket mappe



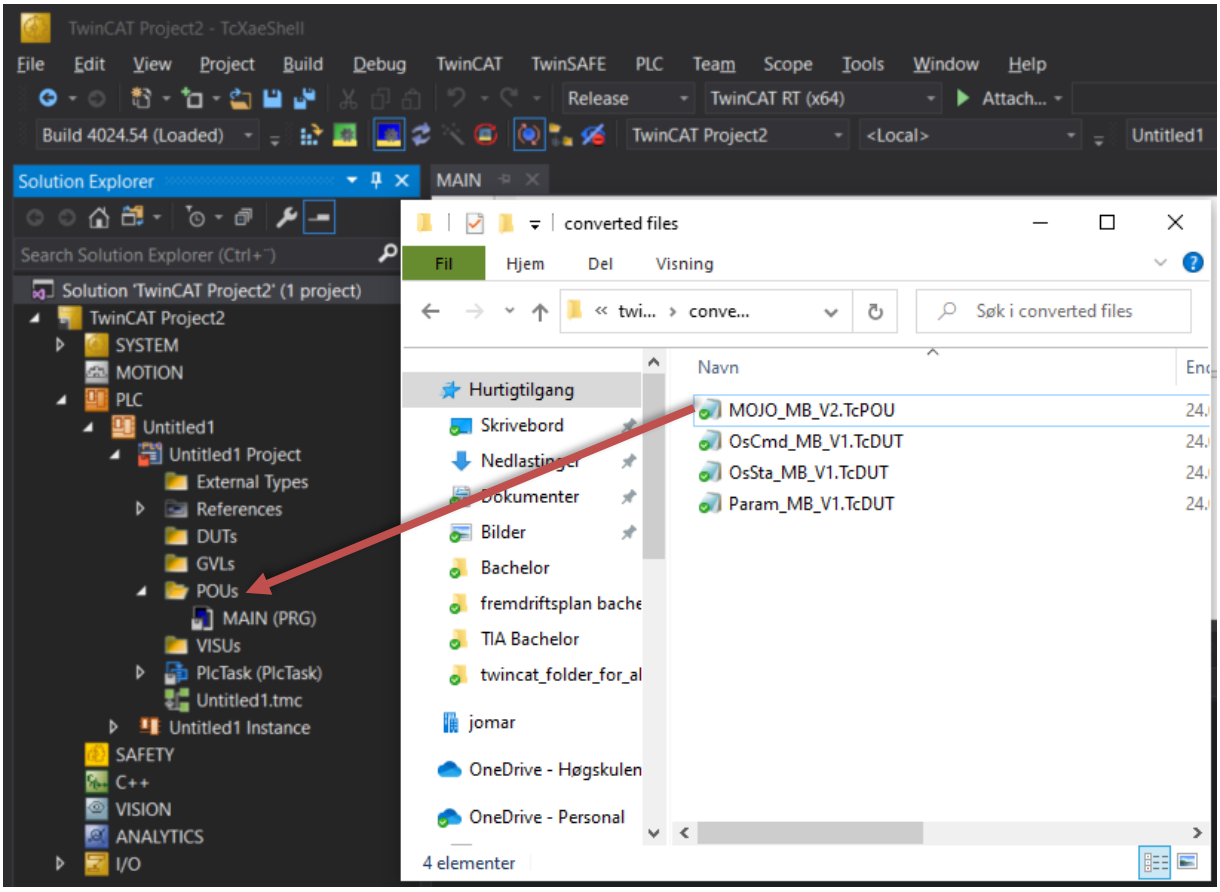
Figur 5. Klikk på «Convert»-knapp



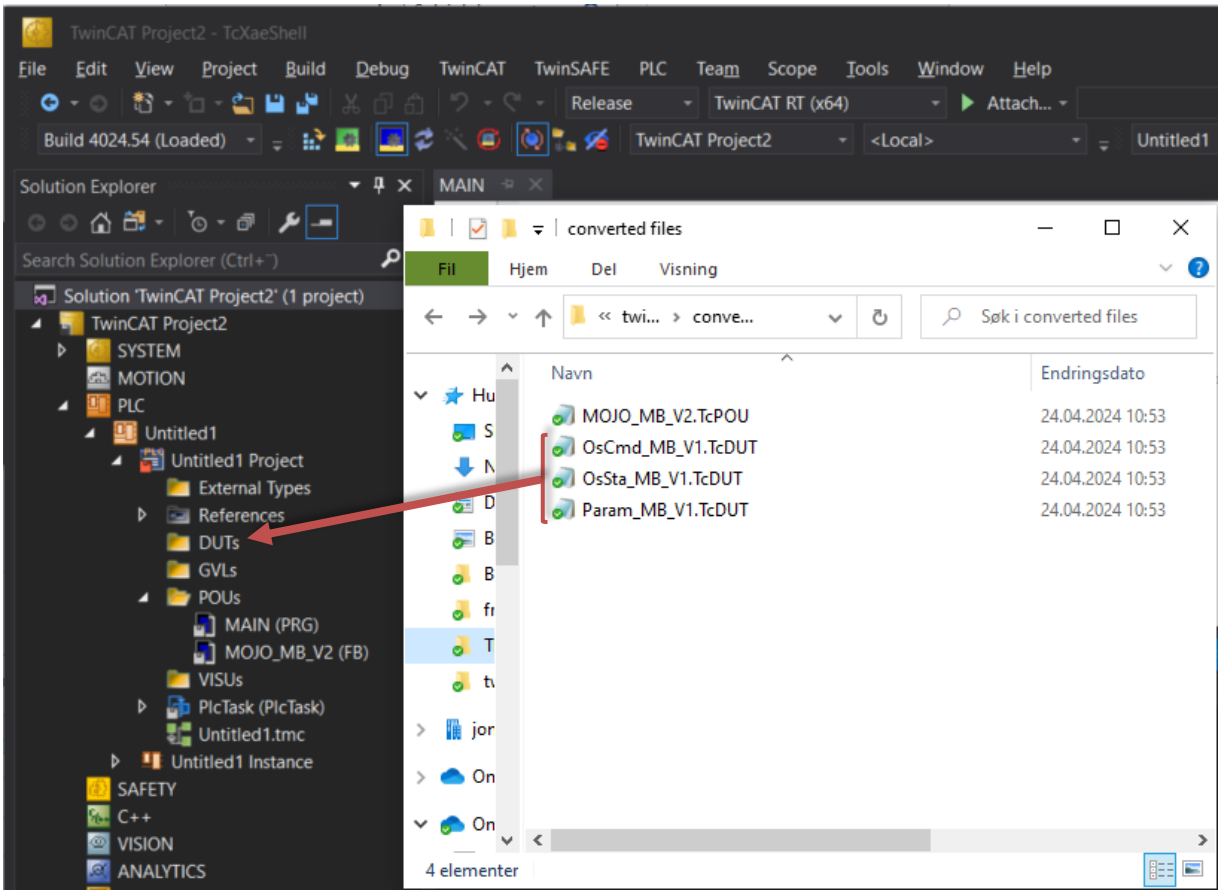
Figur 6. Åpne målmappen for å finne de konverterte filene



Figur 7. Viser destinasjonsmappen med konverterte filer



Figur 8. Dra TcPOU-filen inn i TwinCAT prosjektet



Figur 9. Dra TcDUT-filene inn i TwinCAT prosjektet