UNIVERSITETET I AGDER

IT infrastruktur for tilstandsbasert vedlikehold

Joachim Gahrsen Skogstad
Øystein Jamtveit

Veileder
Geir Hovland (UiA)
Eivind Mogensen (Aker Solutions)

Dette er en konfidensiell rapport.

Masteroppgaven er gjennomført som ledd i utdanningen ved Universitetet i Agder og er godkjent som del av denne utdanningen. Denne godkjenningen innebærer ikke at universitetet innestår for de metoder som er anvendt og de konklusjoner som er trukket.

Universitetet i Agder, 2012
Fakultet for teknologi og realfag
Institutt for ingeniørvitenskap

Sammendrag

Aker Solutions er et internasjonalt selskap med hovedkontor i Lysaker, med en underavdeling i Kristiansand som designer og produserer boreutstyr og borepakker til offshoreindustrien. Utstyret til Aker Solutions er hele tiden under utvikling for å imøtekomme konkurransen fra markedet.

Hardware-in-the-loop (HIL) simulering og systemer for logging og sanntids overvåking av prosesser er en økende trend i markedet, og gir nye muligheter for utvikling av produkter. HIL simulering er relativt nytt innenfor offshore industrien og interessen er økende da dette er tids og kostnadsparende. Dette gjør det mulig å teste og utvikle kontrollsystem for nye produkter i en tidlig fase av produktutviklingen.

Det ble i 2011 utviklet et HIL system i forbindelse med en masteroppgave ved Universitetet i Agder med oppdrag fra Aker Solutions. Dette er en velfungerende løsning som gir mulighet for videre arbeid i form av utvidelse av systemet.

Kommunikasjonen i HIL simuleringen er testet, vurdert og forbedret ved implementering av en ny kommunikasjonsprotokoll. Det har blitt sammenlignet to forskjellige kommunikasjonsprotokoller med hensyn på ytelse og pålitelighet. Systemet er også utvidet med et historisk arkiv for sanntidslogging, som logger og systematiserer data. Dette gir muligheter for å gå tilbake i tid for analysering av data. Utvidelsen av systemet gir en god IT infrastruktur som gir muligheter for tilstandsbasert vedlikehold. En algoritme for syklustelling er konfigurert og koblet til systemet. Data fra HIL simulering av en toppmontert hivkompensator er brukt for konfigurering av algoritmen. Det er også utviklet et konsept for deteksjon av friksjonsendring i hydrauliske sylindere som kan implementeres i det totale systemet.

Forord

Denne rapporten utgjør den avsluttende masteroppgaven, kurskode MAS500, som er en del av master programmet i Mekatronikk ved Universitetet i Agder.

Takk til veilederne våre Geir Hovland (UiA) og Eivind Mogensen (Aker Solutions) for råd og hjelp gjennom prosjektet, samt Marcel Groothuis hos 20Sim.

Grimstad
Mai 2012

Joachim Gahrsen Skogstad og Øystein Jamtveit

Innhold

Figurliste

Tabelliste

Forkortelser

ACE	Advanced Computing Engine
AF	Asset Framework
AHC	Active Heave Compensator
API	Application Programming Interface
COM	Component Object Model
DA	Data Access
DB	Data Block
DCOM	Distributed Component Object Model
DL	Data Link
DVR	Design Verification Report
FB	Function Block
FBD	Function Block Diagram
FC	Function
HIL	Hardware-in-the-loop
HMI	Human Machine Interface
HTTPS	Hypertext Transfer Protocol Secure
ICU	Interface Configuration Utility
IE	Industrial Ethernet
IP	Internet Protocol
OB	Organization Block
OLE	Object linking and embedding
OPC	OLE for process control
OSI	Open System Interconnection
PI	Process information
PLS	Programmerbar logisk styring
S7	Step7
SQL	Structured Query Language
TCP	Transmission Control Protocol
TP	Twisted Pair
UA	Unified Architecture
UDP	User Datagram Protocol

1 Introduksjon

1.1 Bakgrunn

Offshoreindustrien er hele tiden i utvikling. Det er hele tiden økene krav til kvalitet, automatisering og effektivisering. Kundene ønsker å lete etter hydrokarboner nye steder, noe som gir utfordringer i form av dypere hav og kaldere klima. Utstyret skal være så kostnadseffektivt som mulig. Dette gjelder både under utvikling, produksjon og under drift.

Hardware-in-the-loop (HIL) og tilstandsbasert vedlikehold er metoder for å effektivisere utviklingsprosessen og levetid av nye produkter. HIL gir mulighet for utvikling av styringssystemer og testing av feilscenarioer ved et tidlig stadie i utviklingsprosessen. Tilstandsbasert vedlikehold øker driftssikkerheten og levetiden på utstyret fordi det hele tiden er under observasjon. Dette gjør at tid blir spart under utvikling, og nødvendigheten av service på utstyr kan bestemmes ut fra tilstand og ikke faste tidsintervaller.

1.2 Oppgavebeskrivelse

Oppgaven er delt opp i tre deler. Fokuset på den første delen handler om testing av en ny kommunikasjonsprotokoll mellom Programmerbar logisk styring (PLS) og simuleringsenheten i et allerede eksisterende HIL system.

Den andre delen omhandler utvikling av en IT infrastruktur. Dette er et historisk arkiv for sanntids logging av data fra HIL systemet. Det er også sett på hvilke muligheter et slikt system gir.

Siste del omhandler utvikling av et system for tilstandsbasert vedlikehold ved bruk av IT infrastruktur fra del to. To eksempler er på syklustelling og deteksjon av friksjonsendring i hydrauliske sylindere.

1.3 Problem løsning

Den eksisterende Transmission Control Protocol (TCP) protokollen på HIL systemet har blitt erstattet med en User Datagram Protocol (UDP) protokoll. Denne protokollen har deretter blitt testet og sammenlignet mot TCP protokollen.

Det har blitt satt opp et databasesystem fra OSIsoft sitt PI system. Systemet er designet for å samle inn og logge data. Systemet er satt opp til å lese av verdier fra PLSen i HIL systemet og lagrer dataen i en database.

For tilstandsbasert vedlikehold er det satt opp et system som tillater Matlab å kommunisere med databasen. Deretter er det implementert en lastsyklusteller ved bruk av Rainflow-algoritme. Det er også satt opp en friksjonsmodell som simulerer friksjons endringer i hydrauliske sylindere, og utviklet et konsept for implementering av deteksjon av friksjonsendring.

1.4 Rapportstruktur

Kapittel to gjennom går HIL systemet, hvordan det er satt opp og fungerer. Her det det lagt vekt på kommunikasjonen mellom simuleringsenheten og PLSen. Det er gjennomgang og testing av to ulike kommunikasjonsprotokoller, TCP og UDP.

Det tredje kapittelet gir en innføring i OSIsoft PI system, og en presentasjon av OSIsoft sine produkter. Det er så satt opp en IT infrastruktur med et databasesystem som logger data fra HIL simuleringen i sanntid.

Fjerde kapittel omhandler tilstandsbasert vedlikehold. Det er vurdert, testet og konfigurert kommunikasjon mellom PI systemet og Matlab for en kompleks kalkulering av sykluser ved bruk av Rainflow algoritmen. En innføring og konsept av deteksjon av friksjonsendring er også presentert.

Rapporten er oppsummert og konkludert i kapittel 5.

Vedleggene til rapporten inneholder tre veiledninger, en for installasjon av PI Server, en for konfigurasjon av UDP kommunikasjon på PLS i Step7 og en for konfigurering av OPC server. Det er også lagt ved Matlab-script for Rainflow algoritmen, kommunikasjon mellom Matlab og det historiske arkivet for sanntidslogging.

2 Hardware-in-the-loop system

HIL simulering er en metode for å teste og utvikle komplekse sanntidssystemer. HIL gir muligheten til å teste nye og eksisterende systemer, og simulere ulike problemstillinger. Her er det mulig å teste scenarioer som kan være vanskelige å teste i virkeligheten. Hovedbruken av HIL testing er å teste hvordan kontrollsystemet på den fysiske kontrolleren[1] oppfører seg ved forskjellige feilscenarioer. Dette kan være sensorfeil, ventilfeil eller mer ekstreme scenarioer som ikke er mulig å utføre i praksis [12].

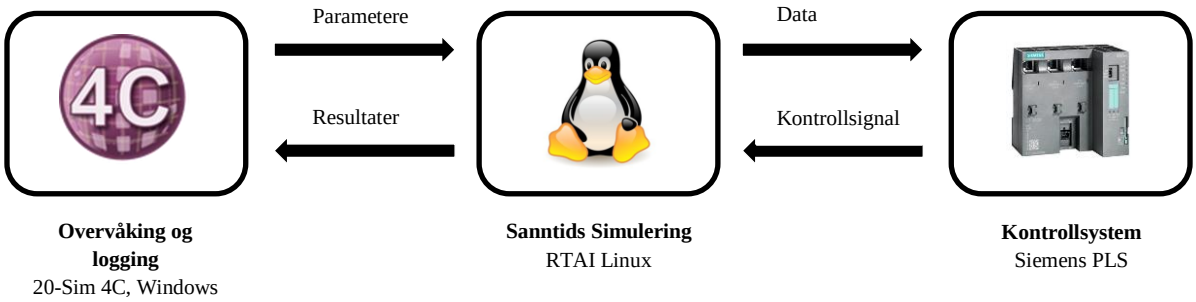
HIL går ut på at det blir laget en matematisk dynamisk modell av en applikasjon, som kan inneholde elektroniske, mekaniske og hydrauliske komponenter for så å simulere oppførselen til applikasjonen under forskjellige omstendigheter. Modellen blir laget så nær virkeligheten som mulig og med like fysiske begrensninger. Når en simuleringsmodell av det fysiske systemet er ferdig utviklet kan den bli kjørt i et HIL system der den fysiske kontrolleren blir koblet til, noe som gir mulighet av utvikling av kontrollsystemet allerede før maskinen er satt i produksjon. Dette gjør at testingen av nye applikasjoner kan begynne i et tidligere stadie av utviklingsprosessen og viktige feil kan oppdages tidligere Ved å utvikle kontrollsystemet med hensyn på simuleringsmodellen er det enklere å gå tilbake å endre på designet. HIL testingen gjør at kontrollsystemet kan implementeres direkte på maskinen når den er ferdig produsert og det trengs kun fin-tuning av systemet.

HIL er en testplattform som kan gjøre utvikling av komplekse applikasjoner enklere, og gir mulighet for testing av ideer og utforinger ved et tidligere stadie. Ved å luke ut feil og utvikle ting som kontrollsystem i en tidligere fase, gir det også muligheter for å spare penger på utviklingen. Samtidig kan utviklingen av et nytt produkt gå raskere ved bruk av HIL testing.

Det finnes i dag simulatorer for opplæring av personell av ulike maskiner. De fleste simulatorene representerer kun maskinene visuelt og har i liten grad den fysiske responsen maskinene har i virkeligheten. Ved å implementere HIL systemet i en simulator vil derfor gjøre systemet mer virkelighetsnært da de fysiske egenskapene også blir implementert.

[1] En fysisk kontroller kan være PLS, mikrokontroller, FPGA etc.

2.1 HIL konfigurasjon



Figur 1 - Oversikt over HIL system [2]

HIL systemet som er brukt består av to datamaskiner og en PLS. Der den ene maskinen kompilerer applikasjonen som er modellert, og overvåker og logger simuleringen under simulering. Den andre maskinen er satt opp med sanntids RTAI Linux Target. På PLSen er kontrollsystemet.

2.1.1 HIL Komponenter

2.1.1.1 Modelleringsverktøy - 20sim

Modelleringsverktøyet som er brukt i HIL systemet er 20sim. Det er et modellering og simuleringsprogram som kjører på Microsoft Windows. Det gir mulighet for dynamisk simulering av elektriske, mekaniske og hydrauliske modeller eller en kombinasjon av disse. Det bruker et grafisk brukergrensesnitt som gir et intuitivt bilde av modelleringen. 20Sim har et stort bibliotek med forskjellige ferdige komponenter i elektroniske-, mekaniske-, og hydrauliske komponenter som gir et bredt utvalg av muligheter. Ønskede komponenter hentes ut fra biblioteket og kobles sammen i modellen. Det er ikke mulig å simulere i sanntid direkte i 20sim, og det er heller ikke mulig å kommunisere med eksterne enheter. [3]

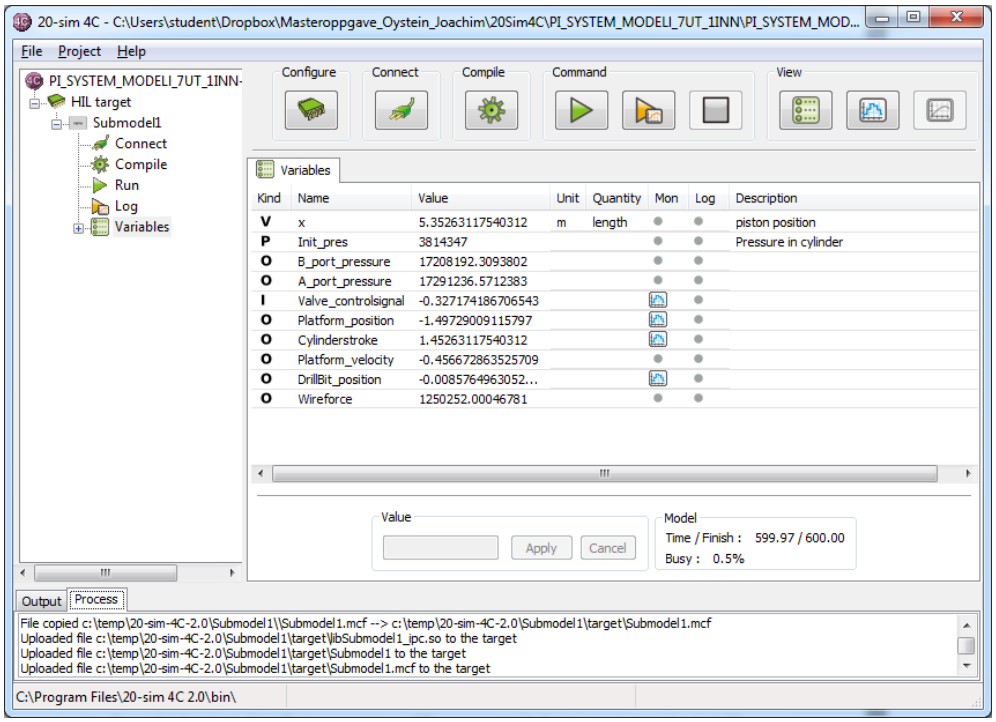
For å kjøre modellen i HIL systemet må den kompileres til 20Sim 4C, C-kode. Det blir først laget en modell av applikasjonen i 20Sim, så valgt ønsket solver. En solver er en bestemt metode for å løse et matematisk problem. Beregningen kommer som et klokke-avbrekk (interrupt) med gitte tidsintervaller. Det er derfor viktig at solveren er ferdig med å utføre beregningene før neste klokke-avbrekk kommer. I dette tilfellet er det brukt solveren Runge-Kutta 4, med et tidsintervall på 0,001 sekunder, eller 1000Hz. Modellen blir så eksportert som C-kode tilpasset 20Sim4C, ved hjelp av sanntids verktøykassen i 20sim.

2.1.1.2 RTAI sanntids Target

RTAI star for Real Time Application Interface. RTAI er en ekstrapakke til Linux kernel som gir et sanntids operativsystem. Det er som andre applikasjoner i Linux-familien, gratis. Dette gir muligheten til å simulere dynamiske applikasjoner med virkelighetsnær oppførsel i ekte sanntid (Hard Real-Time (HRT)) ved hjelp av 20Sim 4C [5].

2.1.1.3 20sim 4C

20sim 4C er en tilleggspakke til modelleringsverktøyet 20Sim. Det er flere forskjellige kommunikasjonsmål, kalt targets som 4C kan kobles opp mot, og det er i denne oppgaven brukt sanntid RTAI Linux. 4C gir også muligheter for kommunikasjon med eksterne enheter, som for eksempel PLS. Modellen blir så kompilert og sendt til RTAI sanntids target. Det er også mulighet til å velge stimuleringstid og tidsintervaller. Det gir også mulighet til å overvåke utvalgte variabler under simuleringen samt logging av resultater fra simuleringen [4]. I 20Sim 4C blir utgangene og inngangen til modellen koblet til de bestemte adressene for sending og mottaging fra og til PLS under simuleringen.



Figur 2 - 20Sim 4C skjermbilde

Figur 2 viser et skjermbilde fra en HIL Simulering av en toppmontert hivkompensator. Ønskede variabler er valgt for logging, og kan overvåkes under simuleringen.

2.1.1.4 PLS

PLSen som brukes er fra Siemens, og er en ET200S IM151-8. Den har tre ethernet porter, der to av portene brukes i HIL systemet. PLSen støtter Industrial Ethernet (IE) kommunikasjon via UDP, TCP og PROFINET. Programmeringsspråket som er brukt er Function Block Diagram (FBD), og gjort med programvaren Step7.

PLSen fungerer ved å kjøre hovedfunksjonen så fort og ofte som mulig. Denne hovedfunksjonen (OB1) inneholder funksjonsblokker (FB) og funksjoner (FC) som blir kjørt etter tur. Etter initialiseringen ved oppstart av PLSen kjører den gjennom rutinene i OB1 så fort som mulig. Det er også brukt et tidskontrollert avbrekk ved hjelp av en organisasjons blokk (OB35). Denne settes til å avbryte programmet med et fast tidsintervall, for så å fortsette hovedprogrammet der den avbrøt.

2.2 Kommunikasjon

En av hovedutfordringene i denne oppgaven er å se på muligheter til å forbedre kommunikasjonen mellom 20Sim 4C og PLSen. Systemet er satt opp med en TCP kommunikasjon. Utfordringen ligger i å få kommunikasjonen til å bli så pålitelig som mulig, og med så stor hastighet som mulig. En deterministisk kommunikasjon er å foretrekke.

Det er i hovedsak tre alternativer for kommunikasjons protokoller:

- TCP
- UDP
- PROFINET

2.2.1 OSI-modell

Open Systems Interconnection (OSI) modellen gir en grafisk fremstilling av hvordan datakommunikasjon er bygd opp med de forskjellige lagene. Modellen viser syv steg data må gjennom for å flytte informasjon mellom to enheter. Hvert lag har mange ulike muligheter og standarder avhengig av applikasjon og bruk. Det er bare valgt å ta med de vanligste, og det som er relevant for kommunikasjonen i denne oppgaven. Kommunikasjonen i HIL simuleringen mellom PLS og sanntids target handler om å flytte data raskest mulig med størst mulig pålitelighet. Som modellen viser avhenger både TCP og UDP av IP-adresser, som igjen bruker Ethernet standarden. Dataen går gjennom disse stegene fra 1 til 4 og andre veien 4 til 1, avhengig om enheten sender eller mottar data.

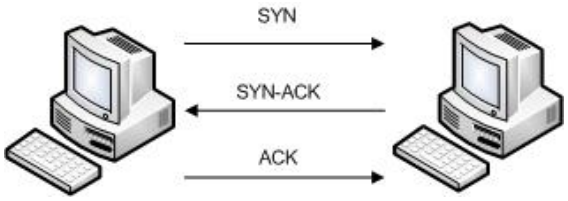
Tabell 1 - OSI Modell

Lag:	Standard:	Datatype:
1. Fysisk	Ethernet	Bits
2. Datalink	Ethernet	Frames
3. Nettverk	IP-Adresse	Pakker
4. Transport	TCP, UDP	Segmenter
5. Sesjon	Sockets, HTTP, Bilder	Data
6. Presentasjon		Data
7. Applikasjon		Data

2.2.2 TCP

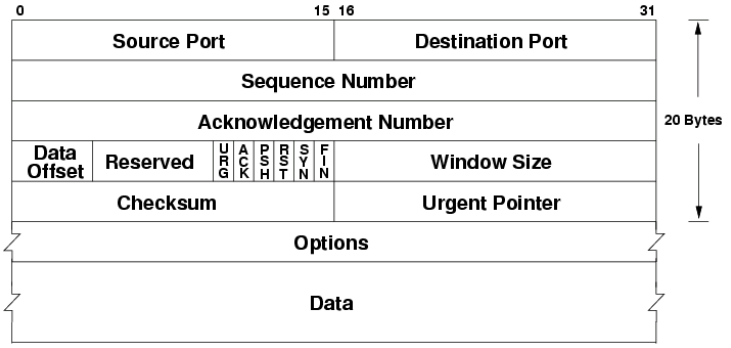
TCP står for Transmission Control Protocol og er forbindelsesorientert, noe som vil si at senderen må ha en mottaker før det kan bli sendt data. Det er ikke mulighet å sende ut data med TCP protokollen uten å ha en definert mottaker. Mottaker og sender har begge en unik IP-adresse på nettverket, og en valgt port det skal kommuniseres gjennom. Dette sikrer at pakkene går til og fra riktig mottaker og sender. En TCP forbindelse har tre faser, opprettelse av forbindelse, overføring av data og avslutning av forbindelse. TCP garanterer levering av data, og også at pakkene som blir mottatt kommer i samme rekkefølge som de ble sendt. TCP er ikke deterministisk, og data blir sendt så fort som mulig til mottaker. Det opprettes først forbindelse mellom klient og server ved hjelp av et tre stegs håndtrykk (Figur 3).

Treveis håndtrykk:



Figur 3 - TCP håndtrykk

Kommunikasjonen opprettes først ved at den ene maskinen sender et SYN-pakke til den andre maskinen. Er maskinen klar til å motta data sender den tilbake en SYN-ACK-pakke. Den første maskinen sender så tilbake en ACK-pakke til den andre maskinen, og tilkoblingen er opprettet. Dette sikrer at enhetene som kommuniserer med hverandre har en fast tilkobling hele tiden mens det overføres data som er viktig for påliteligheten i overføringen.



Figur 4 - TCP Header

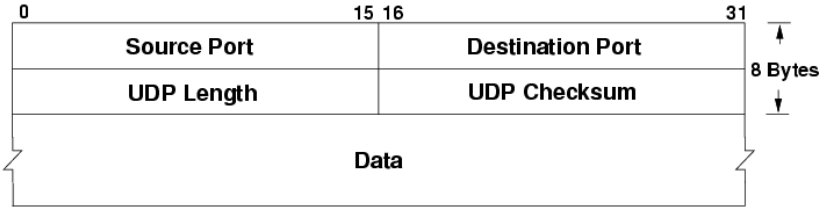
Figur 4 viser hvordan en TCP pakke er bygget opp, og det er flere bit som er reservert til spesifikke egenskaper for hver enkelt TCP pakke. Disse forteller blant annet om hvor pakken skal, hvor stor den er, og hvilket nummer den er i rekken. Dette gjør at mottakeren kan sjekke hver enkelt pakke, for å se om den inneholder det den skal. Mangler pakken data sender mottakeren en forespørsel om å få sendt pakken på nytt.

2.2.3 UDP

UDP står for User Datagram Protocol og er i motsetning til TCP ikke forbindelsesorientert. Det betyr at for å sende data over UDP protokollen trengs det ingen bestemt mottaker. Fordi UDP ikke bygger opp en bestemt tilkobling med en partner, får heller ikke senderen noe tilbakemelding på pakkene som er sendt ut om den er mottatt. Det er ingen garanti for at pakkene kommer frem, og data kan gå tapt på veien til mottaker. UDP sender ut data gjennom en valgt port, og enheter tilkoblet samme nettverk kan lytte til denne porten, og motta dataene som blir sendt ut, dataen blir streamet ut gjennom denne porten. Fordelen med denne måten å overføre data på er at det kreves mindre tilbakemelding under sending, og hver pakke inneholder mindre ekstra informasjon i tillegg til selve dataen som skal bli sendt. Dette gjør at hver pakke kan bli sendt med en høyere frekvens, men med mindre pålitelighet. Akkurat som TCP er UDP heller ikke deterministisk, og det er på ingen måte noe garanti at pakkene kommer frem på ønsket tidspunkt.

Ulempen med UDP er som nevnt over at det er ingen garanti for at data kommer frem til ønsket destinasjon, og data kan gå tapt. HIL systemet er et lukket nettverk, med bare tre enheter koblet til. Disse er koblet sammen fysisk med TP-kabler. Ved korte avstander, fysisk tilkobling og små nettverk vil sannsynligheten av tap av data være ekstremt liten. Det er ingen trådløse tilkoblinger som kan forstyrres av radiosignaler, mobiltelefoner eller andre elektroniske enheter. Den korte avstanden på TP-kablene gjør også at dataen må fysisk reise kortere fra enhet til enhet, noe som også øker påliteligheten fremfor lengere avstander. Det er heller ingen huber eller svitsjer i systemet som kunne påvirket påliteligheten.

Tap av data ved bruk av UDP blir derfor ikke noe problem i dette HIL systemet.



Figur 5 - UDP Header

Figur 5 viser hvordan en UDP pakke er bygget opp. Sammenlignet med en TCP pakke er denne mye enklere og inneholder mindre informasjon. Pakken inneholder kun adresser fra sender, mottaker og data.

Både UDP og TCP ligger i transportlaget i OSI modellen, og bruker IP adresser for kommunikasjonen, og blir ofte referert til TCP/IP og UDP/IP.

2.2.4 PROFINET

PROFINET er en åpen standard for IE (Industrial Ethernet). PROFINET er utviklet av Siemens og PROFINET User Organization. Det er basert på Ethernet grensesnittet og baserer seg på TCP/IP, men har integrert flere protokoller for å sikre sanntids ytelse. Det brukes to typer PROFINET klasser, den ene heter PROFINET CBA og bruker TCP/IP protokollen for kommunikasjon og passer best til komponent-basert kommunikasjon. Den andre heter PROFINET IO og blir brukt for sanntids kommunikasjon, typisk for bevegelses kontroll, og tidssensitive applikasjoner [9].

Det finnes tre typer PROFINET standarder:

- **TCP/IP** - PROFINET CBA, med kommunikasjonshastighet ned til 100ms
- **RT (Real-Time)** - PROFINET CBA og PROFINET IO, med kommunikasjonshastighet ned til 10ms.
- **IRT (Isochronous Real-Time)** – for PROFINET IO, med kommunikasjonshastighet ned til 1ms

2.2.5 Oppsummering

Det eksisterende HIL systemet fungerer allerede med TCP kommunikasjon. Dette er en løsning som fungerer bra, men det er ønskelig å se på alternativene, og om disse bedrer kommunikasjonen ved å gjøre den mer pålitelig, og raskere.

PROFINET ser ut til å være den beste løsningen ved bruk av enten RT eller IRT standardene. PLSen som er brukt har støtte for alle PROFINET standardene mens andre deler av utstyret som blir brukt i prosjektet har ikke støtte for PROFINET RT eller IRT, og dette blir derfor utelatt. Det er derfor foretatt testing med UDP kommunikasjon, og sammenlignet med den eksisterende TCP kommunikasjonen.

2.3 Testing av kommunikasjon

Det er satt opp et testprogram i 20Sim 4C som kommuniserer med PLS for å måle hvor deterministisk kommunikasjonen mellom disse enhetene er. Det er testet med sendehastighet på 6ms, 8ms, 10ms, og 12ms. Testen er utført med ved å sende 10.000 pakker med data til 20Sim 4C, og tiden mellom hver mottatte pakke er målt i millisekunder.

Det er i forbindelse med denne oppgaven laget UDP støtte for 20Sim 4C. Dette skal bli implementert i kommende versjoner av 20Sim 4C. Både TCP og UDP kommunikasjonen er integrert i samme kildekode, og bruker flere av de samme konfigurasjonsparameterne.

Felles for begge kommunikasjonsprotokollene er i 20Sim 4C er at det blir brukt en postkasse-løsning. Dette fordi kommunikasjonen og simuleringen kan kjøres på hver sin tråd av prosessoren, uavhengig av hverandre, og da unngå å stjele ressurser fra hverandre. For en mest mulig pålitelig kommunikasjon er det viktig at kommunikasjonsprosessen kjører så stabilt og uavbrutt som mulig, for å sende og motta data på de bestemte tidspunktene. postkasse-løsningen gir også en god løsning ved å lagre sendt og mottatt data i en buffer, og gir simuleringen- og kommunikasjons-prosessen tilgang til denne dataen uten å påvirke hverandre.

Kommunikasjonen er satt opp til å sjekke for data hvert 1ms, mens sending av data tilbake til PLS blir styrt av en ratio bestemt i kildekoden for kommunikasjonen (haptic_socket.h) (Se Figur 6). Denne ratioen blir multiplisert med en vente-funksjon i koden som også styrer hvor ofte 20Sim 4C skal se etter ny data. Denne vente-funksjonen er satt til 1ms.

```
12  /* Socket settings */
13  #define SOCKET_SERVER_HOSTNAME "192.168.1.41"
14  #define SOCKET_SERVER_PORT     12400      // used for TCP and UDP
15  #define SOCKET_REPLY_PORT      12401      // used for UDP
16  #define SOCKET_SERVER_CONNECT_TIMEOUT 30  // {s}
17
18  #define NUMBER_OF_VARIABLES_IN  1         // In defines from PLC to RTAI
19  #define NUMBER_OF_VARIABLES_OUT 1         // Out defines from RTAI sim to PLC
20  #define RATIO 1                    // Ratio between receiving and sending. Sending
```

Figur 6 - Utdrag av kommunikasjonskildekode

Før simuleringen må de forskjellige parameterne til 20Sim 4C settes opp i henhold til systemet, med IP-adresser, porter, hvor mange variabler som skal sendes/mottas og med hvilket tidsintervall dette skal gjøres med.

2.3.1 Testing av TCP

2.3.1.1 Oppsett av TCP på PLS

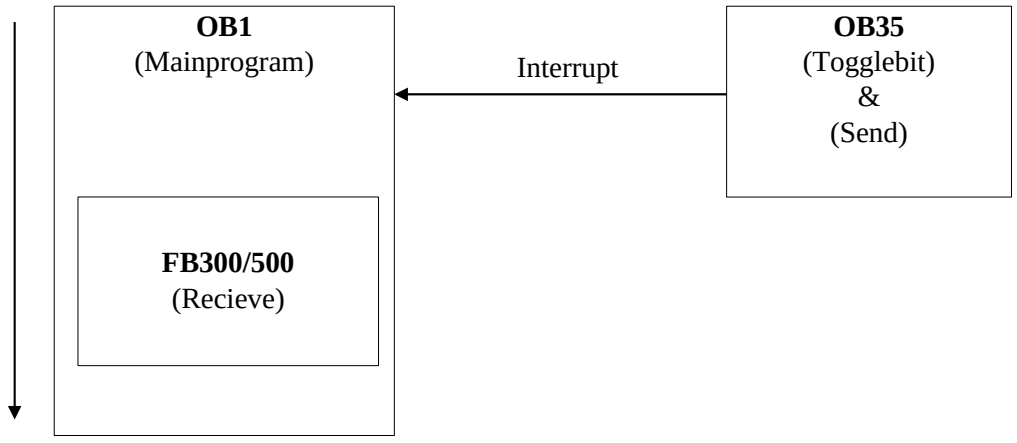
For oppsett av TCP kommunikasjonen på PLSen er det brukt et åpent eksempelprogram hentet fra Siemens Automation sine sider. Dette inneholder en ferdig satt opp kommunikasjonsrutine med sende og mottak funksjoner. Programmet bruker de allerede eksiterende funksjonsblokkene i Step7, FB63 (sending) og FB64 (mottak).

Dette er i utgangspunktet et program satt opp for kommunikasjon ned mot 100ms, og bruker PLSen et eget reservert byte (M10) som clock memory. M10 sine bits representerer en gitt frekvens. Bit nummer 0 eller M10.0 er det med høyeste frekvens, Mens M10.6 har en periode på over et sekund. Dette gjør at bittet oscillerer og går høyt hvert 100ms med M10.0, noe som trigger sendefunksjonen i programmet. 100ms er for treg kommunikasjon i et HIL system, og det er gjort modifikasjoner i programmet for å bedre ytelsen.

Det er laget et eget bit som brukes til å trigge sendefunksjonen, som settes og resettes ved hjelp av OB35. OB35 er en organisasjonsblokk i step7 som kan gir faste avbrekk i programmet med et fast tidsintervall. Ved å sette OB35 med en syklustid på avbrekket på 3-6ms, vil trigger-bittet gå høyt annenhver gang OB35 kjøres, og aktivere sendefunksjonen. Dette gjør at det kan bli sendt ut data enda raskere enn ved bruk av standard oppsettet, og fortsatt å ha kontroll over tidspunktet når data blir sendt ut.

Standard oppsettet for kommunikasjonen fungerer ikke optimalt, og sendefunksjonen FB63 ble flyttet direkte inn i OB35 for å få enda bedre kontroll og pålitelighet for sendefunksjonen (Figur 7). Dette forbedret resultatene betraktelig (

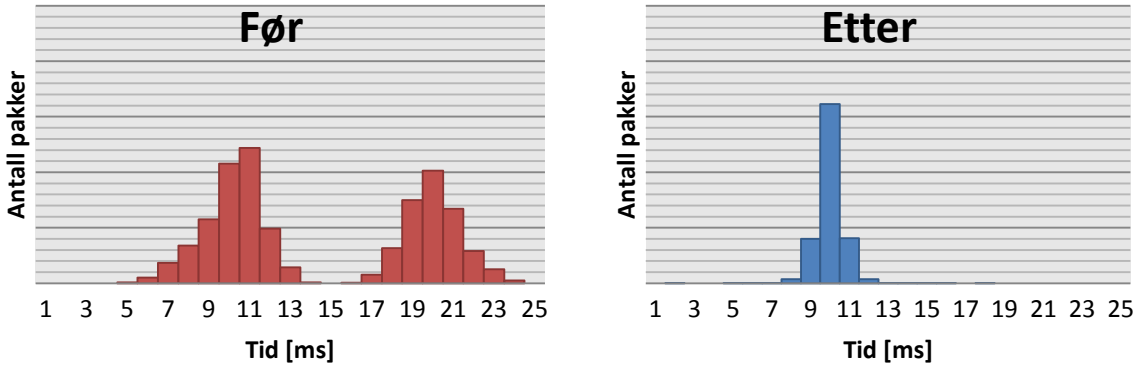
Figur 8).



Figur 7 - Kommunikasjons oppsett PLS

Ved å sette hyppige avbrekk på OB35 vil større programmer på PLSen bli avbrutt under kjøring, og kan også bli avbrutt flere ganger i samme sekvens. Det er derfor relativt hvor raskt kommunikasjonen kan gå i forhold til størrelsen på programmet PLSen skal kjøre.

PLSen er satt opp til å bruke 50% av ressursene tilgjengelig til kommunikasjonen, dette for å få kommunikasjonen til yte optimalt.

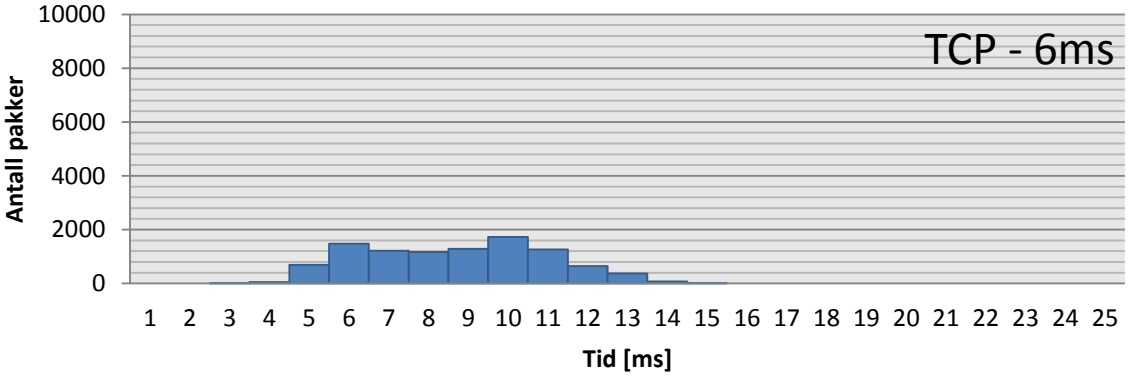


Figur 8 – Sammenligning av resultater før og etter endring i kommunikasjon på PLS

Histogrammene over viser forbedringen i påliteligheten til overføringen før og etter flytting av sendefunksjonen (FB63) til OB35 ved 10ms kommunikasjon.

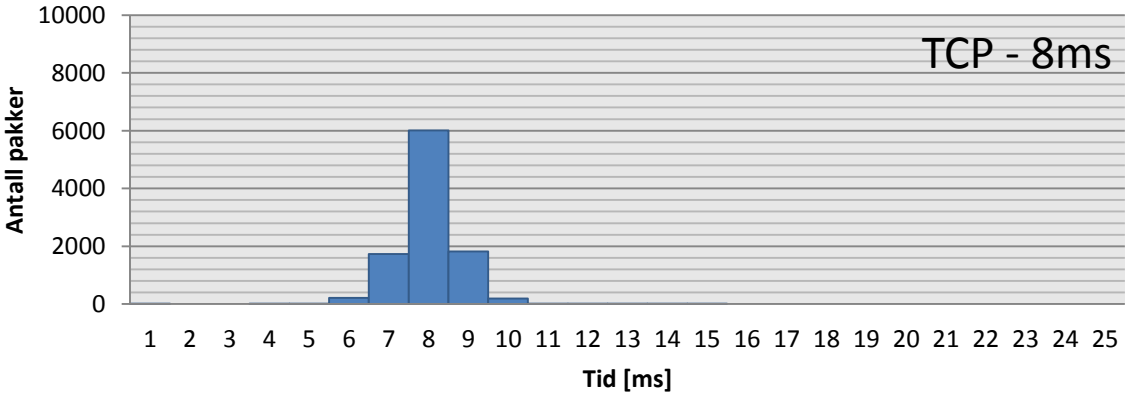
2.3.1.2 TCP resultater

Det er blitt sendt 10.000 pakker fra PLS til 20Sim 4C, og målt hvor ofte de treffer det ønskede tidspunktet. Det er tatt målinger på 6ms, 8ms, 10ms og 12ms for å måle determinismen til kommunikasjonen. Oppløsningen på målingene er i hele millisekunder.



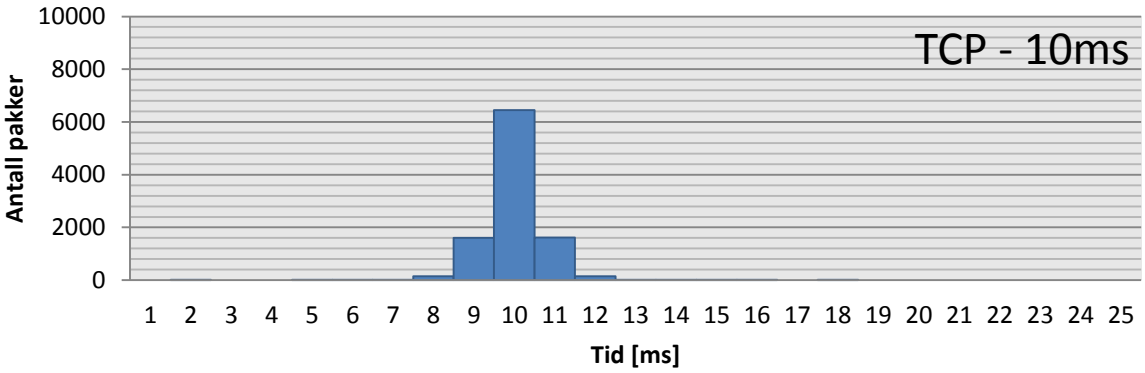
Figur 9 - TCP resultat for 6ms

Figur 9 viser resultatene ved 6ms sending. Oppsettet som ble laget for testing hadde store problemer med å kjøre 6ms, da dette blir for hypping avbrudd i programmet til å kjøres uten problemer. Testoppsettet ble derfor forenklet, noe som tydelig påvirker resultatene. Her er det kun 14.8% av pakkene som treffer ønsket tidspunkt på 6ms.



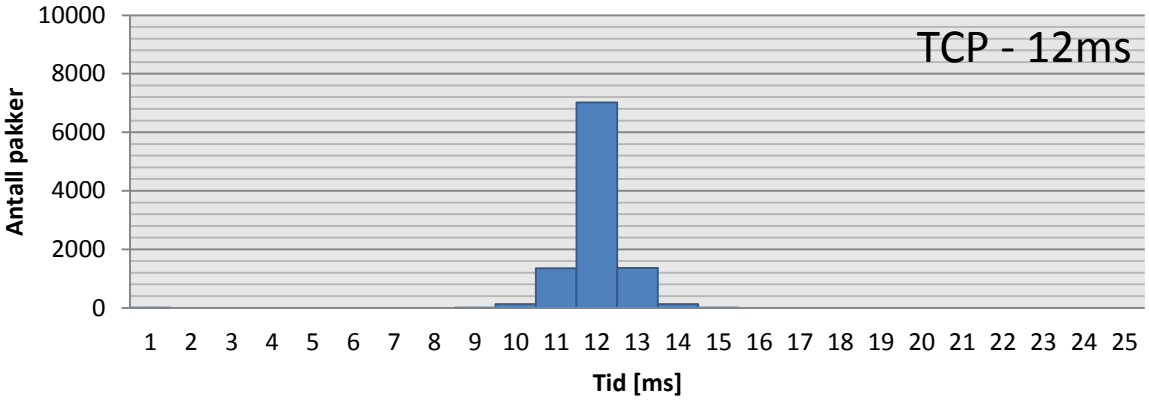
Figur 10 - TCP resultat for 8ms

Oppsettet fungerte bra på 8ms, og resultatene er tilnærmet normalfordelt, og 60% pakkene treffer på det ønskede tidspunktet, 8ms. 17,3% treffer ved 7ms og 18,2% treffer ved 9ms.



Figur 11 - TCP resultat for 10ms

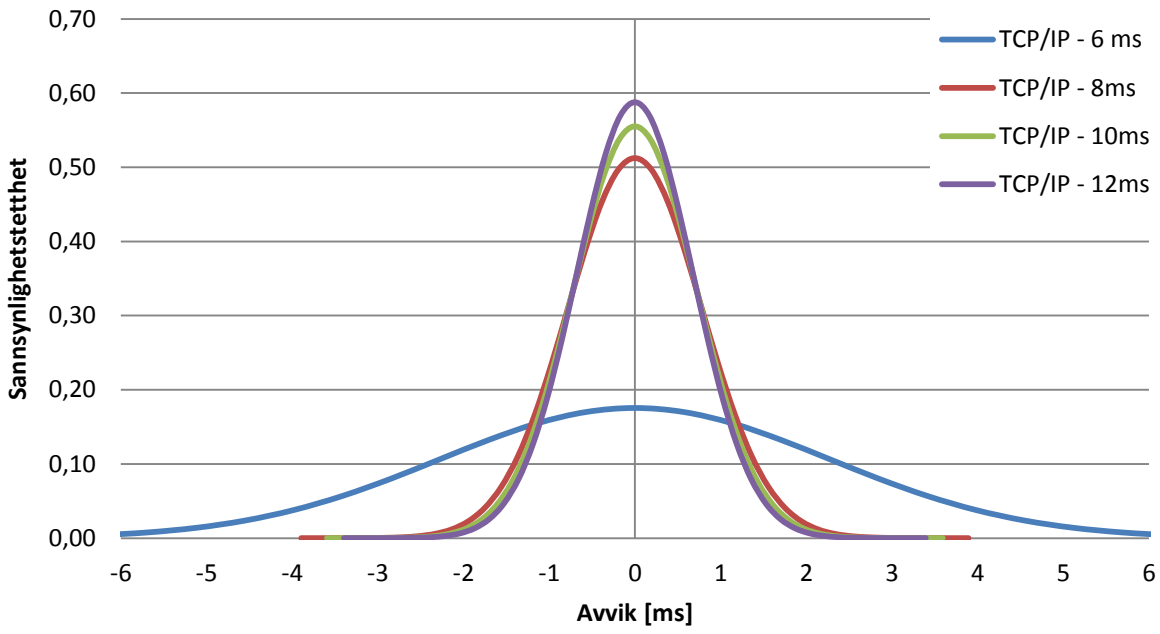
Resultatene for 10ms er bedre, der 64,6% av pakkene treffer det ønskede tidspunktet, og er bedre enn 8ms. 16% av pakkene treffer ved 9ms og 11ms.



Figur 12 - TCP resultater for 12ms

12ms viser enda bedre determinisme enn ved raskere hastighet, og 70,2% av pakkene kommer frem ved ønsket tidspunkt. 13% treffer ved 11ms og 14ms.

2.3.1.3 Sammenligning av de forskjellige senderatioene for TCP kommunikasjon



Figur 13 - Normalfordeling av resultater for TCP

Det er laget en normalfordeling av resultatene for TCP kommunikasjonen. Fordelingen viser en trend der jo høyere sendehastighet som brukes, jo oftere mottas pakkene på ønsket tidspunkt. Dette er naturlig, da det kreves mindre ressurser ved å gjøre samme oppgaver med en lavere frekvens. Kommunikasjonen ved 12ms gir som vist på figuren over den mest deterministiske dataoverføringen. Tabell 2 viser standardavvikene til de forskjellige senderatene. Dette viser også at 6ms er for raskt for PLSen med dette oppsettet, mens 8ms fungerer OK.

Tabell 2 - Standardavvik for TCP

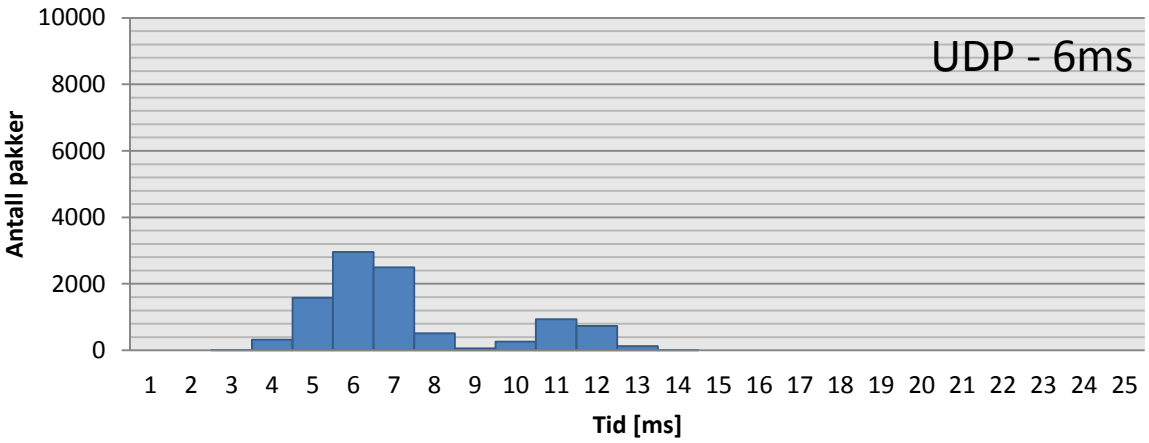
Tid [ms]	Standardavvik
6	2,2274
8	0,7787
10	0,7187
12	0,6789

2.3.2 Testing av UDP kommunikasjon

2.3.2.1.1 Oppsett av UDP kommunikasjon på PLS

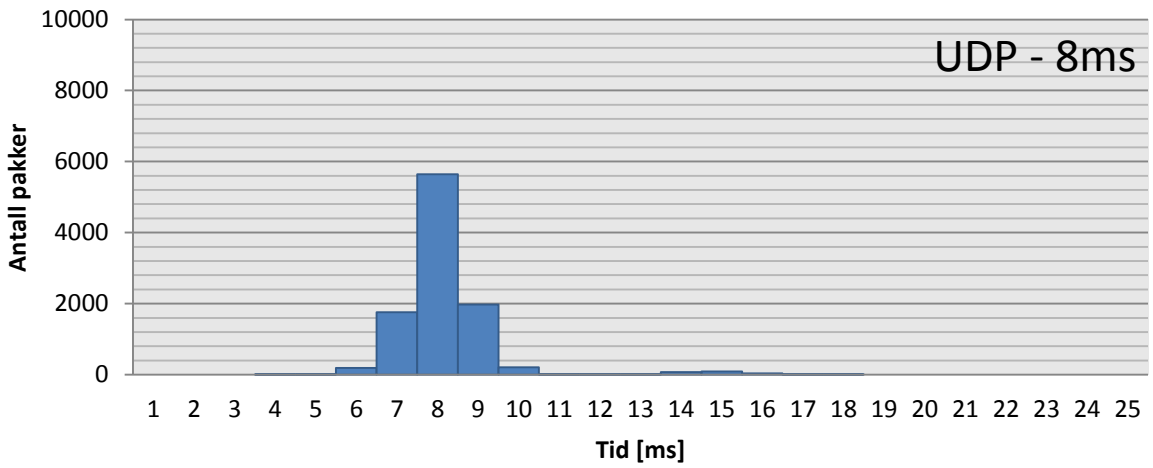
Det er også her brukt et åpent eksempel hentet fra Siemens Automation sine hjemmesider. Dette programmet bruker standard sende- (FB67) og motta-funksjoner (FB68) som er integrert i Step7. Også her ble programmet modifisert for å få bedre resultater, og sendefunksjonen ble på samme måte som i ved TCP testen flyttet til OB35 for bedre kontrollere tidspunktet på sendingen. Det ble også her testet med sendetider på 6ms, 8ms, 10ms og 12ms. PLSen er også satt opp til å bruke 50% av ressursene på kommunikasjonen. Triggerbitet til sendefunksjonen ligger her også på OB35.

2.3.2.1.2 UDP resultater



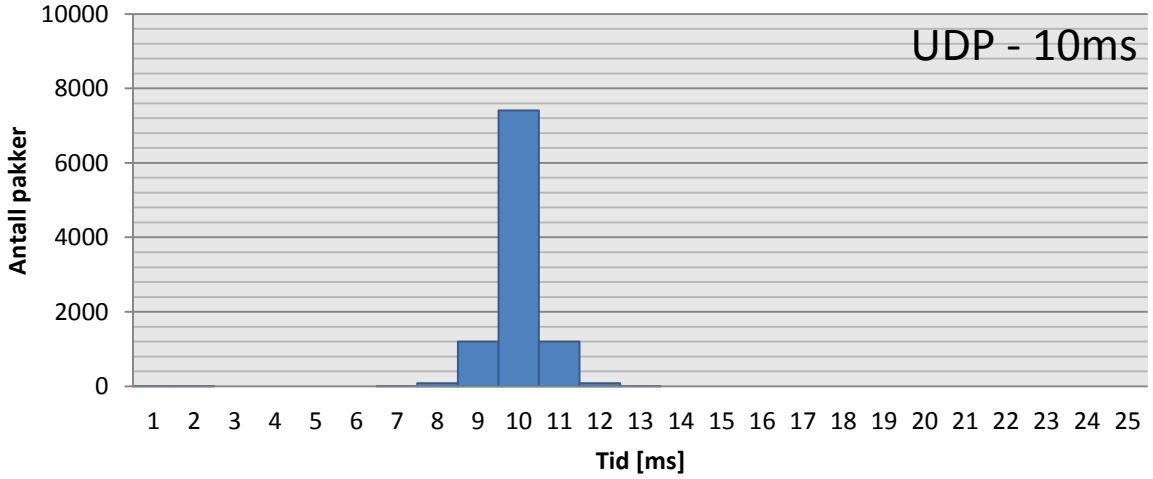
Figur 14 - UDP resultat for 6ms

Resultatene ved senderatio på 6ms gir unøyaktige resultater. Histogrammet viser at 25% av pakkene treffer på det ønskede tidspunktet, og de andre tidspunktene for mottatte pakker fordeler seg nærmest tilfeldig rundt.



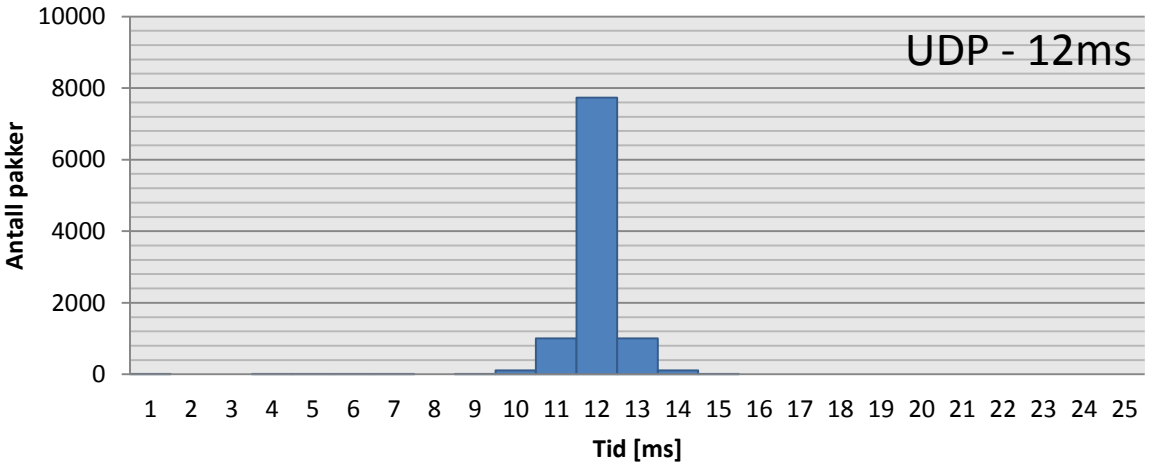
Figur 15 - UDP resultater for 8ms

Ved 8ms senderatio treffer 56,5% av pakkene, noe som er betraktelig bedre enn ved 6ms. De andre pakkene fordeler seg jevnt rundt 6ms, og 17,6% treffer ved 7ms, og 19,8% treffer ved 9ms.



Figur 16 - UDP resultater for 10ms

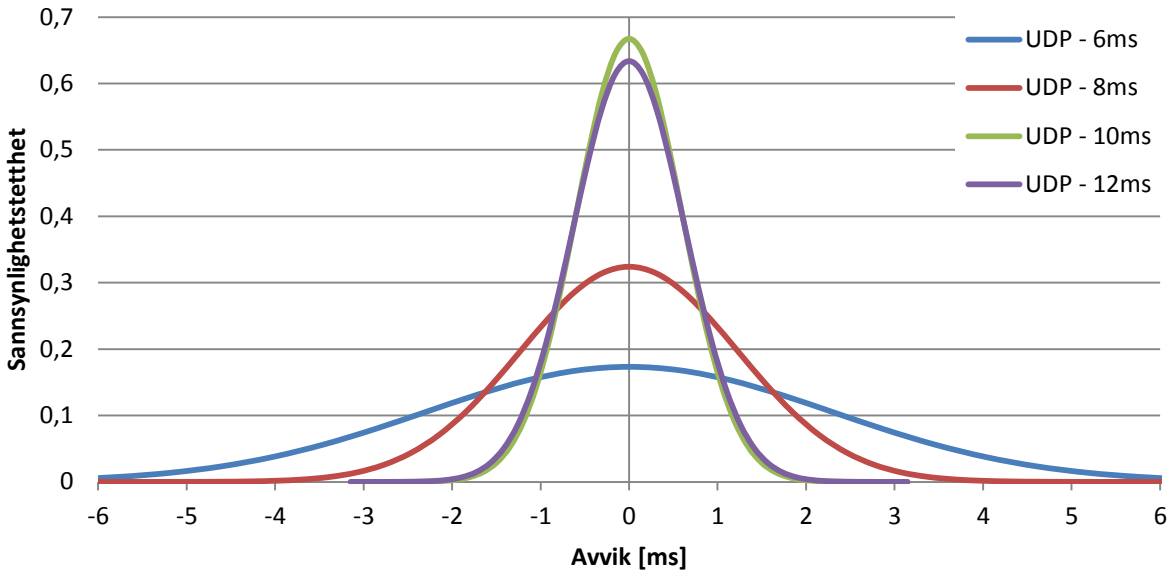
Figur 16 viser UDP testene er ved 10ms. Her treffer 74,1% av pakkene ønsket tidspunkt, og 12% ved 9ms og 11ms.



Figur 17 - UDP resultater for 12ms

Resultatene ved 12ms er omtrent det samme som ved 10ms. 77% av pakkene treffer på ønsket tidspunkt, og 10% treffer ved 11ms og 12ms.

2.3.2.2 Sammenligning av de forskjellige senderatioene for UDP kommunikasjon



Figur 18 - Normalfordeling av resultater for UDP

Normalfordelingen over viser sannsynligheten for at pakkene kommer frem på ønsket tidspunkt. Ved 12ms treffer flest pakker på ønsket tidspunkt, men standardavviket i tabellen under viser at 10ms gir det minste avviket, og sannsynligheten for å treffe eksakt ønsket tidspunkt ved 10ms større enn ved 12ms.

Normalfordelingen viser her en et lite knekk ved overgangen fra 8ms til 10ms, der påliteligheten i overføringen ble forbedret med rundt 50%. Dette skyldes trolig at halvparten av ressursene til PLSen blir brukt til kommunikasjon, noe som gjør at selve programmet ikke får nok ressurser til å kjøres så ofte det skal.

Tabell 3 - Standardavvik for UDP

Tid [ms]	Standardavvik
6	2,3020
8	1,2312
10	0,5978
12	0,6296

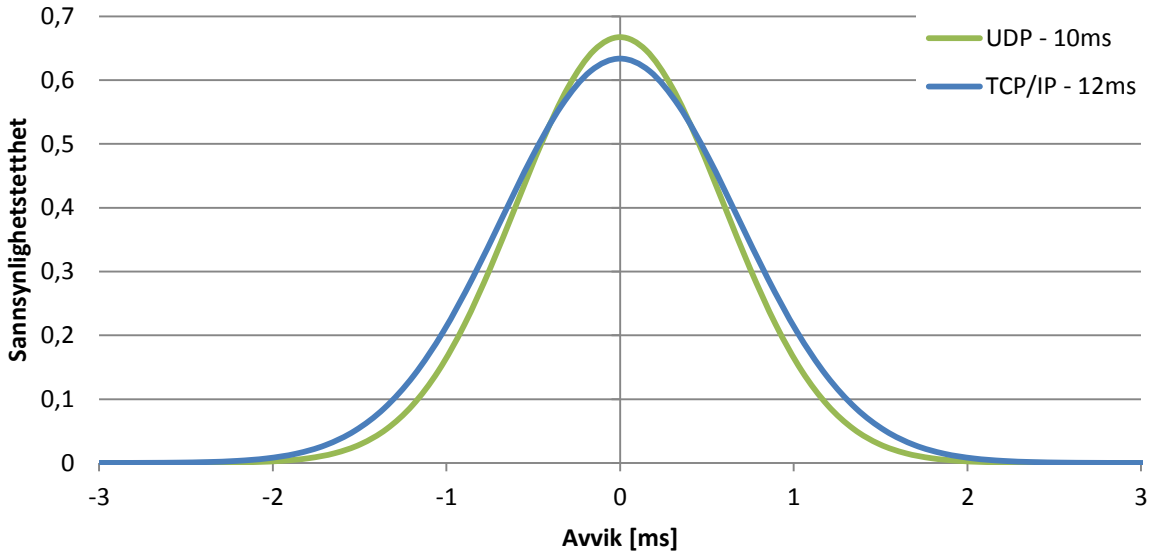
Tabell 3 viser standardavvikene til de forskjellige målingene gjort med UDP kommunikasjonen. 10ms senderatio gir det mest pålitelige resultatet.

2.4 Sammenligning av TCP og UDP

Tid [ms]	Standardavvik for TCP	Standardavvik for UDP	Forskjell
6	2,2274	2,3020	-0,0746
8	0,7787	1,2312	-0,4525
10	0,7187	0,5978	0,1209
12	0,6789	0,6296	0,0493

Tabell 4 - Sammenligning av TCP og UDP

Tabellen over viser sammenligning av standardavvikene til de forskjellige kommunikasjonsprotokollene. Ved raskere senderatio, gir TCP med henholdsvis 6ms og 8ms best resultater i pålitelighet. Mens ved lavere senderatio, 10ms og 12ms gir UDP det beste resultatet. Det oppsettet som gir mest deterministisk resultat er UDP ved 10ms senderatio.



Figur 19 - Sammenligning av TCP og UDP

Normalfordelingen på Figur 19 viser de to beste resultatene fra testene med TCP og UDP kommunikasjon for å vise forbedringene det er ved å implementere UDP protokoll i HIL systemet.

3 Database og sanntids logging

Logging av data med historie er noe som er nytt og i vinden innenfor drilling. Det har ikke vært vanlig før å lagre data over tid, for så å ha muligheten til å gå tilbake å analysere data. Dette gir også muligheter for overvåking og analysering av applikasjoner uavhengig hvor applikasjonen befinner seg. Dette er interessant med hensyn på flere ting.

Det jobbes hele tiden etter å finne metoder for å drive mest mulig kostnadseffektivt, og det å kunne redusere bemanningen på rigg, eller slippe å sende noen på rigg for å gjøre målinger kan gi innsparinger, dersom man kan overvåke og analysere data fra land kan dette oppnås.

Data kan gjøres tilgjengelig for flere, alt fra riggeier, riggsjef, leverandører av utstyr, vedlikeholds personell er eksempler. Dette kan være aktuelt både for utvikling av nye produkter og oppfølging av utstyr som er allerede utplassert. Database og sanntids logging gir også mulighet til å overvåke hvordan utstyret blir brukt, med hensyn på garantier og i henhold til spesifikasjoner, vedlikeholdsrutiner, overvåke produktivitet og aktivitet på rigg er eksempler på nyttige bruksområder.

Sanntidsovervåking og logging av utstyr er også gunstig med hensyn på dokumentasjon, fordi det hele tiden er data å kunne vise tilbake til. Dersom utstyr skal godkjennes for utvidet levetid er det gunstig å kunne henvise til driftsdata på utstyret. Dersom det for eksempel er snakk om løfteutstyr som er designet for en gitt levetid så vil et kontrollorgan godkjenne dette utstyret. Det kan tenkes at dersom det finnes data som beviser at den faktiske belastningen for utstyret over tid er mye mindre enn de antatte beregningene kan utstyr godkjennes for utvidet levetid.

Det eksisterende systemet ønskes derfor utvidet med database og logging av data over tid. Muligheter for å gå tilbake til et bestemt tidsintervall og se på data er et kriterie. Til dette vil det ikke fungere med en klassisk Structured Querty language (SQL) database og Historian en bedre løsning. Historian er et databasesystem som logger data med unike tidsnøkler, og systematiserer data ved hjelp av ulike tags. Denne måten å lagre data på er mest brukt for sanntids løsninger og lagring og arkivering av store mengder data. Dette gjør det enklere å gå tilbake i tid og analysere data fra et bestemt tidspunkt [6]. Kjente systemer som gjør dette er Proficy Historian og OSIsoft PI system.

Aker Solutions bruker allerede OSISoft PI system, og ønsket å vite mere om mulighetene dette gir. I denne oppgaven er det blitt lagt hovedvekt på å finne en eller flere løsninger til en infrastruktur for tilstandsbasert vedlikehold. Men det er i tillegg gjort undersøkelser for hvilke andre muligheter man har i anvendelsen av PI system.

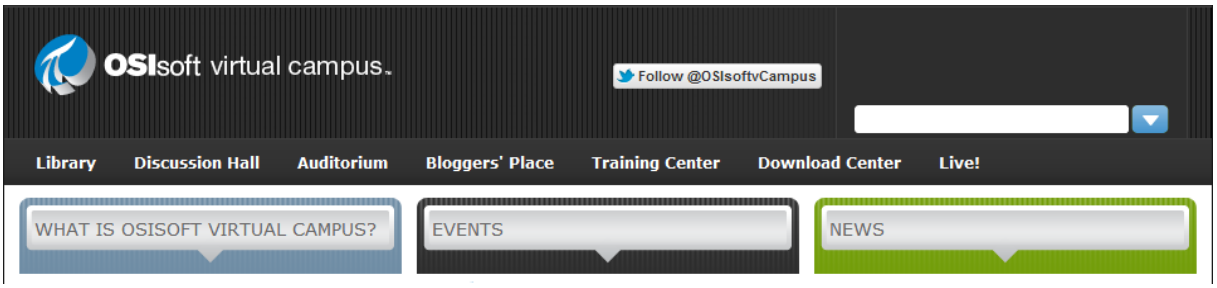
3.1.1.1 OSIsoft

OSIsoft er et anerkjent firma som har eksistert siden 1980 med godt og stødig renome. De tilbyr en omfattende pakke for innsamling og lagring av data. Eksempler på bruk kan være prosessdata fra fabrikker og økonomisk data for børs.

Dette består da av løsninger som omfatter innsamling av data, lagre data, organisering av data, lete opp data, analysere, levere og visualisere.

OSIsoft har et omfattende samarbeid med blant annet Microsoft, Cisco, SAP, og IBM, og har løsninger som fungerer sømløst sammen med produkter levert av disse aktørene.

OSIsoft har også gode hjemmesider og egen Youtube kanal som de kaller for OSIsoft Learning Channel for opplæring av brukere av systemet. Her finnes det gode og informative instruksjonsvideoer for installering og konfigurering av de forskjellige produktene. De tilbyr også en løsning de kaller Virtual Campus (Figur 20). Som navnet sier er det et virtuelt Campus som tilbyr online forelesninger samt et bibliotek med artikler rundt problemer, utfordringer, utvikling og testing av applikasjonene deres. Det er også et forum der folk stiller spørsmål og fagfolk hjelper til.



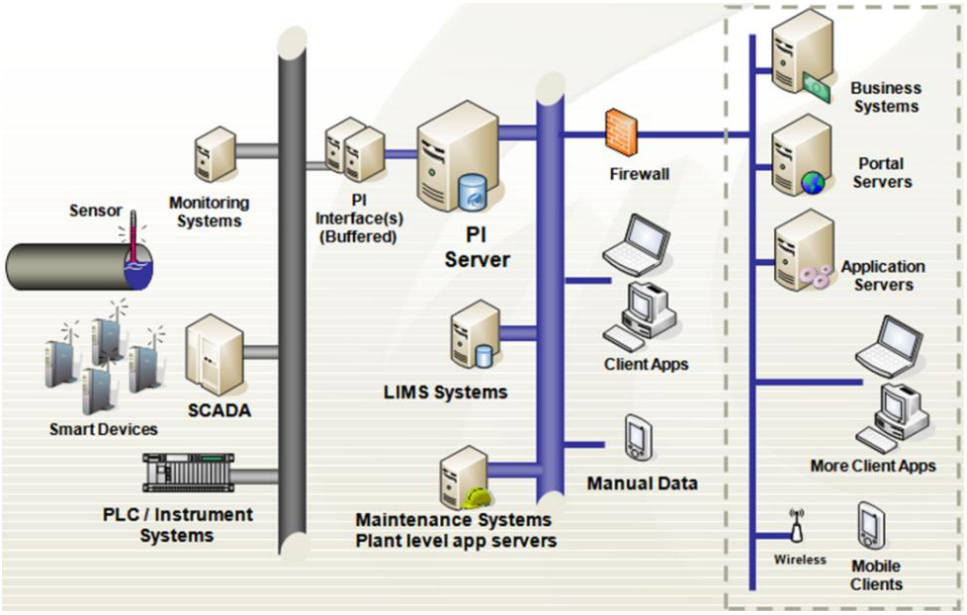
Figur 20 - Virtual campus

3.2 OSIsoft PI System

Process Information (PI) er et begrep laget for å beskrive all informasjon laget av alle typer elektriske, hydrauliske eller mekaniske prosesser. Prosessene ute på en boreplattform genererer enorme mengder data, og til det trengs spesialprogrammer og systemer til å systematisere dataen som generes. OSIsoft sitt PI system er et system laget spesielt til å håndtere rådata av denne typen og systematiserer dataen. Rådata blir systematisert ved bruk av tags og tidsnøkler. Den prosesserte dataen kan så bli brukt for videre avlesning eller analyse.

Ved å overvåke prosessene i sanntid kan man til en hver tid se hvordan utstyret brukes, hvordan det oppfører seg og å følge prosessene. PI systemet logger også all dataen, så det er mulig å gå tilbake i tid for å se hvordan prosessene har oppført seg over en gitt periode. Dette gjør det mulig å se forandringer i systemet over tid, og bruke dette til å beregne vedlikeholds og service rutiner ut i fra bruk i stedet for faste tidsintervaller.

PI systemet er PC basert, og er for eksempel koblet til en PLS ved hjelp av en OPC server. OPC serveren gjør at man kan lese av de rådataene direkte fra PLS, for så å sende det inn i PI Systemet. PI systemet er så satt opp til å tolke disse dataene og vise de i en mere forståelig form. Det finnes mange tilleggspakker til OSIsoft sitt PI system, og systemet kan skreddersys etter hvilken og hvordan en prosess ønskes å bli overvåket. Figur 21 viser strukturen til et typisk PI system der det er mulig å få tilgang til sanntids data fra applikasjonen som overvåkes.



Figur 21 - Eksempel på PI system

3.3 OSIsoft produkter

OSIsoft tilbyr flere produkter avhengig av hvilke oppgaver som skal løses. Produktene er laget for innsamling av data, lagre data, organisering av data, lete opp data, analysere, levere og visualisere. Det er valgt ut noen av de tilgjengelige produktene for en kort gjennomgang .

3.3.1 PI ProcessBook

PI ProcessBook (PB) er et verktøy for grafisk fremstilling av data. PB gir mulighet til å lage en visuell tilnærming av applikasjoner og prosesser ved hjelp av figurer og script designet med Visual Basic. PB gir muligheten for enkel fremstilling av grafer og trender av data fra PI databasen. Grafikk og figurer kan tilegnes valgte funksjoner ved bruk av script. Grensesnittet kan skreddersys og lages så informativt som ønskelig i forhold til oppgaven det skal brukes til [7].

3.3.2 PI DataLink

PI DataLink (DL) er en ekstrapakke til Microsoft Office sitt regneark Excel. Denne gir muligheten for å enkelt hente data fra PI databasen og implementere i Excel for videre analysering. Trender kan hentes direkte inn og vises. Lister med data fra et bestemt tidsintervall for så å bli analysert med Excel sine funksjoner. Dette gjør at data fra prosesser enkelt kan bli analysert i for eksempel et allerede eksisterende regneark [7].

3.3.3 PI Coresight

PI Coresight er et web-basert verktøy som gir tilgang til PI systemet fra en nettleser. Dette er også det nyeste produktet fra PI og ble lansert i desember 2011. Coresight er på mange måter en mer moderne versjon av ProcessBook, og har et enklere brukergrensesnitt. Brukergrensesnittet er laget så enkelt som mulig og designet for å raskt presentere rådata på en intuitiv måte. Brukergrensesnittet kan bli laget separat, og legges så til en webserver. Noe som gjør PI systemet tilgjengelig direkte i nettleseren og krever ingen andre programmer installert enn Windows Silverlight. Dette sikrer rask og effektiv tilgang til data fra PI systemet fra hvilken som helst datamaskin med nettleser så lenge det er internettilgang [7].

3.3.4 PI ActiveView

PI Active View er et verktøy til ProcessBook. Active View gir muligheten for deling av brukergrensesnitt i ProcessBook via nettleseren. Og gir derfor samme funksjon som Coresight, men er avhengig av ProcessBook [7].

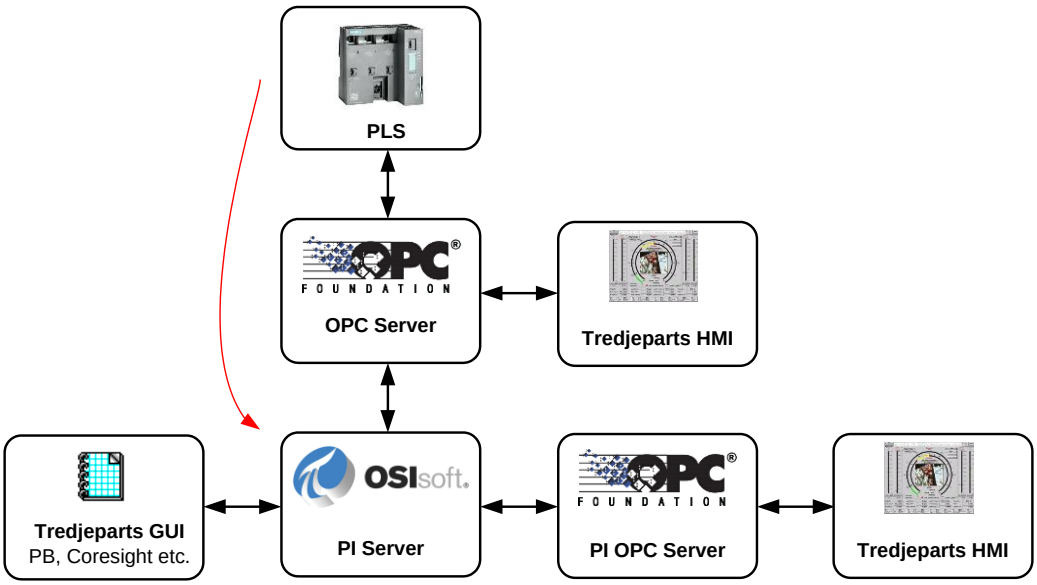
3.3.5 PI Advanced Computing Engine

PI Advance Computing Engine (ACE) gir mulighetene for analysering av data ved bruk av enkle eller komplekse ligninger. Dette sys inn i ProcessBook og utvider mulighetene for behandling av data som strømmer inn. Funksjonene og ligningene blir programmert med Visual Basic, og gir avanserte muligheter [7].

3.4 Konsept for dataflyt i PI System

PI systemet har mulighet til å bruke til et verktøy som heter PI OPC Server. Dette gjør at PI Serveren opptrer som en OPC Server. Dermed er PI Serveren tilgjengelig for programvare som er laget for å kommunisere med OPC. Dette gir muligheter for å koble flere programer som kommuniserer via OPC til å bruke data fra PI serveren. DrillView fra Aker Solutions og WinCC er eksempler på brukergrensesnitt, Human Machine Interface (HMI) som kommuniserer med OPC server.

Operatørene på en borerigg styrer boreoperasjonen via joysticks og skjermer på operatørstolene. Disse skjermene viser HMI for den aktuelle maskinen med data som er relevant for operasjonen som gjennomføres. Ved løsningen på Figur 22 gir det mulighet for direkte overvåking av denne prosessen og det kan også brukes som en co-pilot. Det samme brukergrensesnittet kan kobles til PI OPC serveren og linkes opp til de samme tagsene som blir brukt ute på riggen. Dette fører til et identisk bilde av skjermene operatøren har foran seg under operasjonen. Noe som forutsetter at alle de samme verdiene som blir brukt av operatøren blir logget av PI systemet.

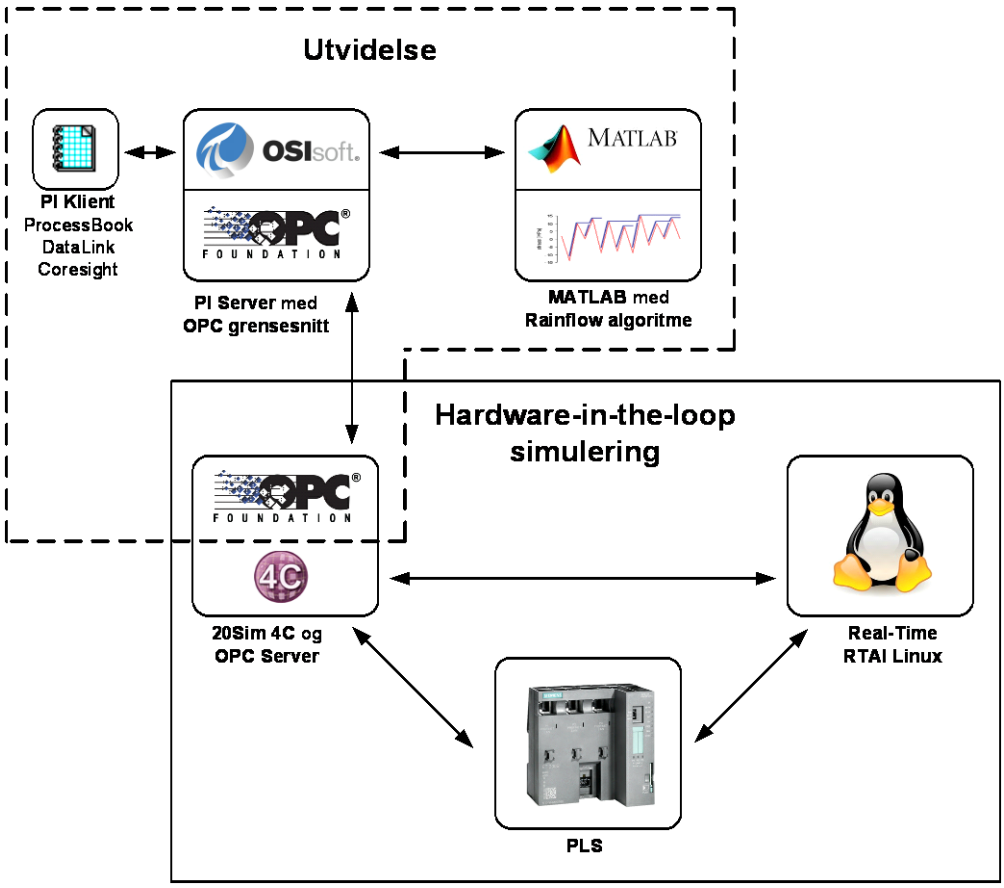


Figur 22 - Dataflyt i PI system

Den røde pilen på Figur 22 indikerer en alternativ måte å overføre data fra PLS til PI server, dette kan for eksempel gjøres via UDP. Den tradisjonelle måten å kommunisere mellom PLS og PI server er gjenom en OPC server.

3.5 Utvidelse av HIL system

Det eksisterende systemet har blitt utvidet en PI System som logger sanntids data fra HIL systemet.



Figur 23 - IT Infrastruktur

Figur 23viser hvordan det eksisterende HIL systemet er utvidet med en OPC server og et PI system som igjen er linket opp med Matlab for for videre analyse av simuleringsresultater.

Det er brukt en modell av en toppmotnert hivkompensator for testing av PI Systemet, og analyseverktøy er utviklet i Matlab med disse testresultatene fra HIL simulering av denne.

For å få testet systemet ordentlig er det ønskelig å overvåke flere variabler, 20 Sim modellen ble derfor utvidet med flere sensorer for lagring av flere typer data. For å få lagret data i databsen til PI systemet er det satt opp en OPC Server som leser av minnet til PLSen. PI systemet har et eget grensesnitt for å kommunisere med OPC servere. Dette gjør det mulig for PI systemet å få data fra PLSen og lagre det i en egen database basert på unike tidsnøkler og forskjellige tags. Dette databasesystemet for sanntids logging av data lagrer dataen systematisk, og gjør det mulig å enkelt hente frem data for de overvåkede sensorene i et gitt tidsintervall. Ved å koble PI systemet opp mot Matlab blir det mulig å gjøre avanserte kalkulasjoner og analyse med resultater fra HIL simuleringen lagret i databasen.

3.6 PI Server

Installasjon av PI server er forklart i detalj i vedlegg 7.1.

PI serveren består av bygger på to hovedelementer:

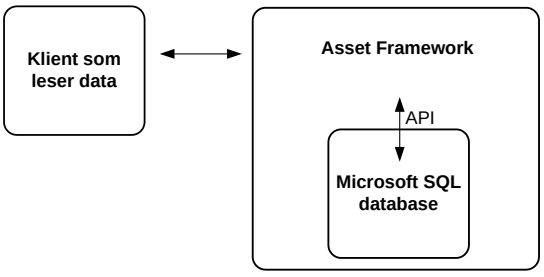
- Asset Framework (AF) Server
- SQL Server Express 2008R2

Det er brukt èn servermaskin for håndtering av AF database, SQL database, OPC Grensesnitt og PI Server. Dette gjør tilkoblingen mellom de forskjellige enhetene enklere med tanke på brukerrettigheter og sikkerhet. Det er også hensiktsmessig fordi systemet som blir brukt i denne oppgaven er relativt lite i denne type sammenheng og ikke krever enormt med datakraft for å fungere tilfredsstillende.

PI serveren bruker Microsoft sin SQL Server for lagring av data. Dette er et relasjonsdatabaseadministrasjonssystem utviklet av Microsoft. Versjonen som er brukt i denne oppgaven er SQL Server Express 2008 R2. Dette er den nyeste versjonen av SQL databasen og inneholder kun hovedelementene til Microsofts SQL Server. PI systemet trenger i hovedsak kun databasen som ligger i denne installasjonen og er derfor tilstrekkelig.

SQL databasen ligger i bunnen av systemet, og brukes til å arkivere data (Figur 24). OSIsoft har laget sitt eget databasesystem som heter Asset Framework (AF) som bygger på SQL databasen, og SQL kreves derfor for at AF skal fungere. Når det kalles etter data fra databasen er det AF som håndterer sikkerheten i forhold til brukere og kommunikasjonen til SQL serveren. AF bruker Application Programming interface (API) for kommunikasjon mellom de to programmene. Dette er et verktøy som gir AF mulighet til å kommunisere ved å skrive eller lese data fra SQL databasen.

I stedet for å ha flere tusen verdier liggende i en database uten noe form for systematisering i SQL databasen, gir AF muligheten til å sette dataen i system for enklere å lese av og håndtere dataen. AF gir mulighet til å kategorisere og sette opp de forskjellige dataene i hierarkier dette for eksempel basert på enkelte applikasjoner eller prosesser. Det kan for eksempel finnes flere av samme type sensor som genererer data med et tidsintervall, men er montert på forskjellige prosesser. Hver sensor får da sin unike tag det blir lest data til, og denne tagen knyttes opp til den bestemte prosessen.



Figur 24 - Databaseoppbygging

3.7 OPC

OPC er basert på Microsoft Windows teknologi og var i utgangspunktet en forkortelse for Object linking and embedding (OLE) for Process Control men står nå for Open Prosess Control ettersom OLE teknologien har blitt erstattet med Active X. OPC server er den eneste kommersielle måten å få tilgang til minnet på PLSen. Det finnes flere forskjellige leverandører av OPC servere, som for eksempel Matrikon, Novotek og Siemens. Fordi det blir brukt Siemens ET200S PLS i dette prosjektet er det brukt OPC server fra Simatic NET fra Siemens Automation. Denne baserer seg på Industrial Ethernet (IE). IE er Ethernet protokollen brukt i et industrielt miljø. Den er mere allsidig enn for eksempel Profibus og gir muligheten til å bruke utstyr som routere og switcher i allerede eksiterende nettverk for kommunikasjon mellom PLC og for eksempel PC [1].

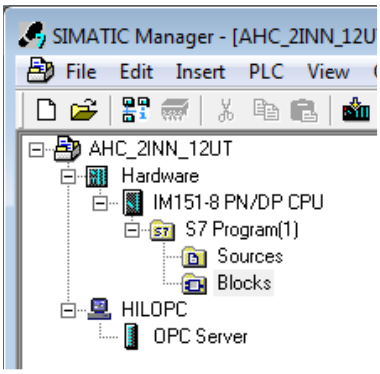
Det finnes flere ulike typer OPC standarder med forskjellig formål, under et ut utdrag av de mest brukte [10]:

- **DA (Data Access) –** DA er den første standarden utviklet av OPC Foundation, denne standarden kan kun skrive og lese fra og til PLS og PI systemet. DA gir muligheten til avlesning av minnet på PLS i sanntid og blir brukt til for eksempel HMI for operatører av maskiner.
- **HDA (Historical Data Access)** – HDA fungerer på samme måte som DA, men leverer data som allerede er sortert og arkivert. Passer bedre til mindre systemer, og vil gjøre det mere tungvint enn DA for lagring i PI database.
- **UA (Unified Architecture) -** UA er den nyeste og mest avanserte standarden, den har implementert flere OPC standarder i en standard. Noen av disse er DA, DA og Alarms and Conditions.
- **Alarms & Events –** Denne standarden brukes til å lese eller skrive data ved bestemte hendelser.

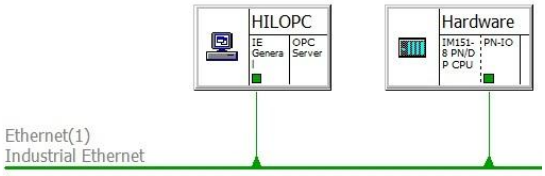
De mest aktuelle for systemet i denne oppgaven er UA og DA. OSIsoft sitt OPC grensesnitt støtter ikke UA standarden og har kun støtte opp til DA 2.05a, og det er derfor valgt å bruke dette DA som OPC standard videre.

3.7.1 OPC Server

OPC serveren er satt opp og konfigurert i Step7, dette er forklart i detalj i vedlegg 7.2.



Figur 25 - Step7 skjermbilde 1



Figur 26 - Step7 skjermbilde 2

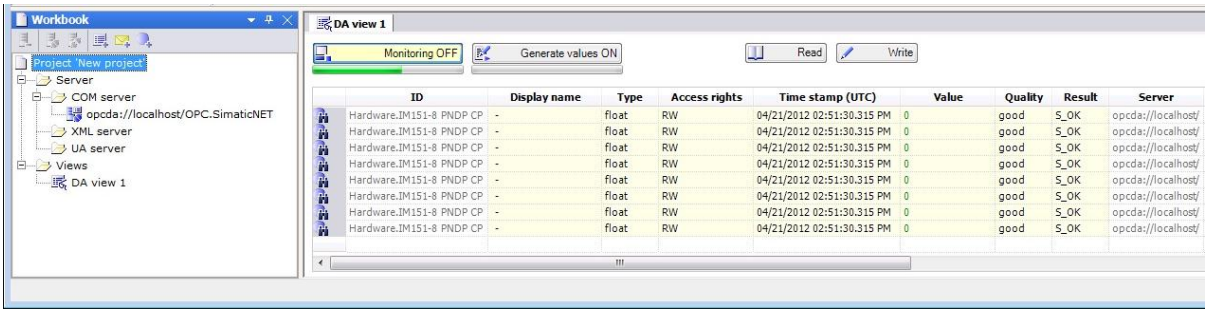
Som vist på Figur 25 er det satt inn en Simatic PC Station med navn "HILOPC". Denne representerer datamaskinen OPC Serveren står på. Her blir konfigurasjonen til OPC serveren satt opp. IP-adresser, oppdateringstid, kommunikasjonsprotokoll og utvalgte variabler blir så konfigurert.

Figur 26 viser hvordan kommunikasjonen mellom OPC serveren og PLSen er satt opp. Den grønne streken illustrerer en Step7 (S7) tilkobling som baserer seg på IE. PLSen og OPC serveren er koblet opp mot hverandre via en vanlig TP kabel.

Når konfigureringen er gjort lastes den opp til Station Manager som er et program fra Simatic Net programpakken til Siemens, og OPC serveren kan startes.

3.7.2 Test av OPC lokalt

For å teste OPC serveren og se at den leser av de riktige verdiene samt oppdateringshastigheten er tilfredsstillende er det brukt to forskjellige programmer. Det ble først testet lokalt med OPC Scout som er med i Simatic NET pakken. OPC Scout leter etter OPC servere som kjøres lokalt på systemet, kobler seg opp og gir mulighet for å lese av verdiene. Den har også mulighet til å skrive til PLS for å endre eventuelle verdier.



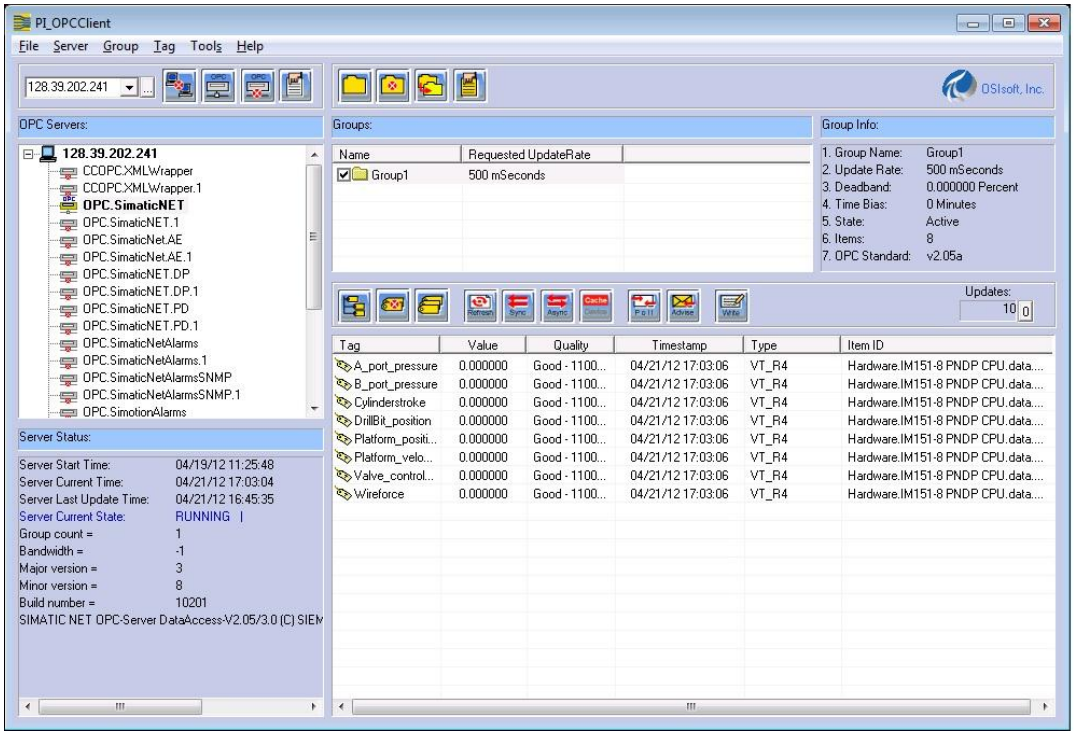
Figur 27 - Skjermbilde fra OPC Scout

Figur 27 viser et utdrag av et skjermbilde fra OPC Scout som leser av verdier fra OPC serveren som er satt opp i kapittel 0. Klienten gir informasjon om de forskjellige variablene som er tilgjengelig gjennom OPC serveren. Datatype, rettigheter, siste oppdatering, verdi og adresse blir vist i listen.

3.7.3 Test av OPC eksternt

For å få tilgang til OPC serveren med PI Serveren ble OPC serveren sin tilgjengelighet testet med en klient på PI Server-maskinen (Figur 28). Denne kommunikasjonen foregår via Distributed Component Object Model (DCOM), som er en Microsoft-teknologi for kommunikasjon mellom programvarekomponenter fordelt på datamaskiner i nettverk. Dette er en eldre standard som er delvis utgått, og tatt over av Microsoft.NET. Som nevnt tidligere er støtten for ulike standarder til OPC grensesnittet til OSIsoft begrenset (3.7), og det derfor valgt å bruke DA 2.05a, som er den eldste standarden. DCOM ga utfordringer i forhold til rettigheter på de forskjellige maskinene som skulle snakke sammen, og det ble opprette like brukerkontoer på PI servermaskinen og OPC servermaskinen for å få dette til å fungere [15].

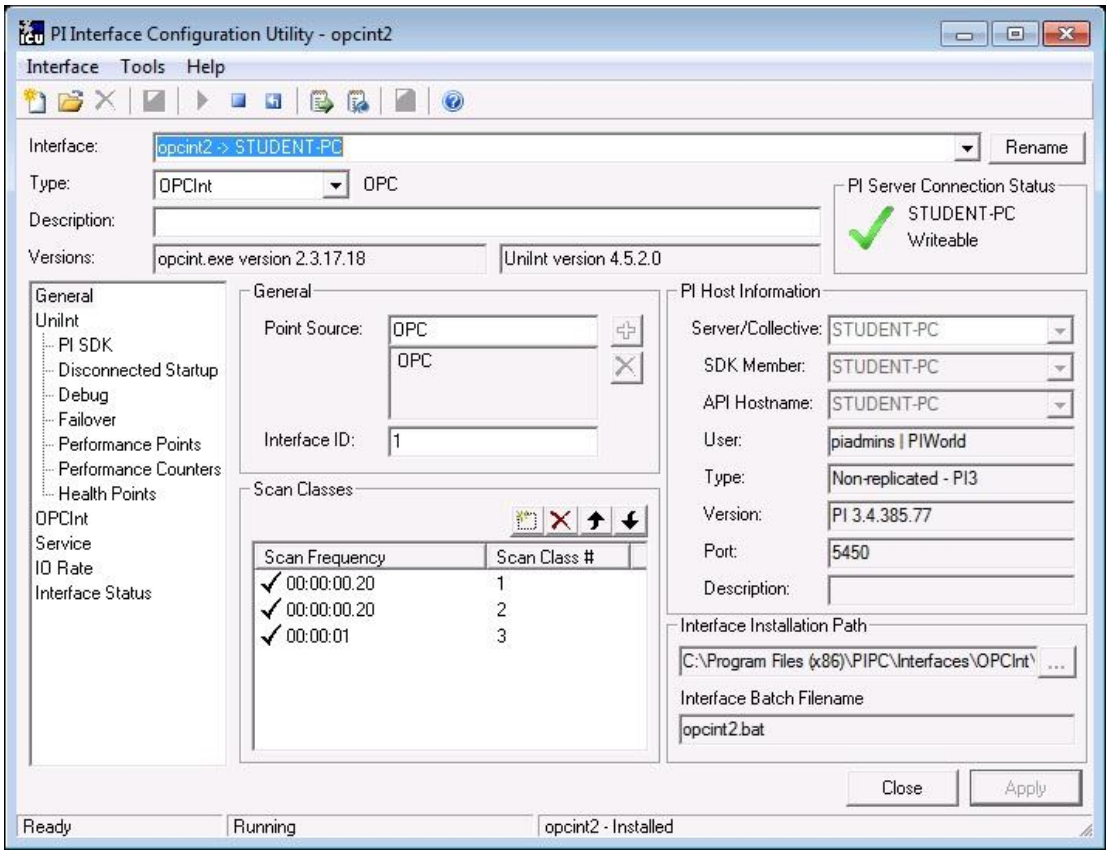
OSIsoft har laget en egen OPC klient som gir mulighet for å teste OPC servere. Denne fungerer ved at man kobler seg til ønsket maskin som kjørere OPC server, og tilgjengelige OPC servere kommer opp listen til venstre. Man må så lage en gruppe der en oppdateringsfrekvens blir bestemt, i dette eksempelet er den 500ms. Det må så velges hvilke verdier av serveren som skal leses. Denne klienten gir også mulighet til å skrive til PLS.



Figur 28 - Skjermbilde fra OSIsoft OPC klient

3.7.4 OSIsoft OPC Grensesnitt

For å logge verdiene fra HIL systemet er det som nevnt tidligere (0) satt opp en OPC server som leser av verdiene til PLSen til en hver tid. OSIsoft har utviklet et eget grensesnitt for å kommunisere med OPC servere for å kunne lese og logge dataen til PI Serveren. Dette OPC grensesnittet har mulighet for Historical Data Access (HDA), DA og "Alarms and Events" standardene (3.7). HDA begrenser seg med å kun ha mulighet til å sende arkivert data, og "Alarms and Events" er begrenset til å kun fungere ved spesielle hendelser. Derfor er det kun DA som tilfredsstiller kravet om å lagre data fra OPC serveren i PI Serveren.

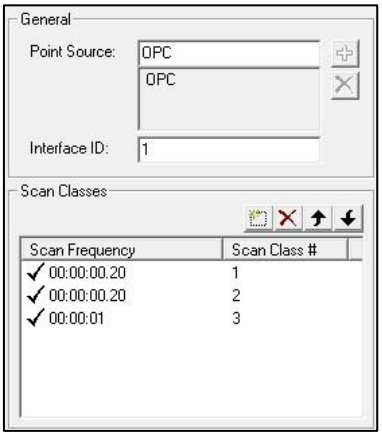


Figur 29 - Skjermbilde fra PI Interface Configuration Utility med OPC Grensesnitt

Ved hjelp av PI Interface Configuration Utility (ICU) (Figur 29) konfigureres OPC grensesnittet. Her bestemmes det blant annet hvilken OPC server grensesnittet skal kobles opp mot og hvilken PI Server det skal skrives til.

All data som leses av OPC serveren og blir lagret i PI Serveren blir linket opp til bestemte variabelnavn kalt tags som PI Systemet bruker for å systematisere dataen i databasen. Det gjør det lettere å holde orden på tags som blir lest inn til PI Systemet, og man kan enklere filtrere ut verdiene som kommer fra en bestemt applikasjon ved avlesning.

En viktig ting som blir satt opp her er frekvensen på avlesningene. Avhengig av hva slags data som skal lagres kan denne settes til ønsket tidsintervall. Tester viste at frekvenser over 1sekund ga svært upresise resultater derfor er frekvensen på avlesningen i denne oppgaven satt til å være 200ms. Dette gir 5 avlesninger hvert sekund, og gir korte nok tidsintervaller for å vise trender med jevne kurver med god oppløsning.



Figur 30 - ICU - Scan class

Den første avlesningsklassen (Scan Class) er reservert til Advise Tags, mens de resterende klassene er til Polled Tags. Advise tags oppdaterer kun verdien sin når det blir lest inn en ny verdi av OPC serveren, mens Polled tags leser av verdier i en bestemt frekvens uavhengig om verdien forandrer seg. I denne oppgaven er det mest hensiktsmessig å bruke Advise tags, fordi PI systemet kjører hele døgnet mens HIL simuleringen bare blir brukt i perioder. Så for ikke å fylle systemet med unødvendig mye informasjon og identiske verdier er Advise tags valgt. Advise tags har begrensninger på 800 tags i forhold til hvor mange tags denne tag-gruppen kan inneholde for et system, noe som ikke er noe har noe betydning i denne oppgaven.

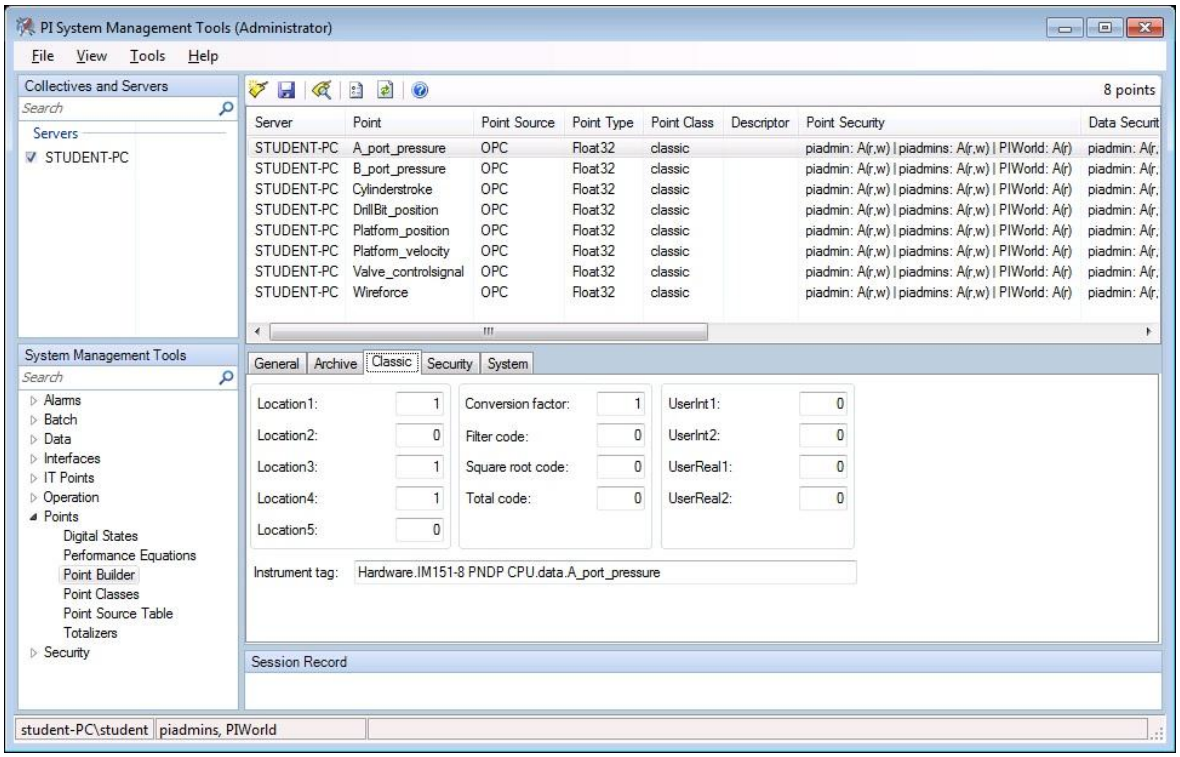
Det er i tillegg er aktivert en mulighet som kalles “bufserv”. Dette er en buffer som er laget for å hindre tap av data hvis tilkoblingen til PI Serveren skulle gå ned. Verdiene blir da lagret i bufferen og lastet opp automatisk med riktige timestamps når tilkoblingen gjenopprettes. Denne buffermekanismen bygger på API grensesnittet. Det finnes også en nyere type buffer for denne typen oppgaver som blir kalt PI Buffer Subsystem, eller ”PIbufss”. Dette er et undersystem av PI Serveren som kjører på OPC grensesnittet. Dette gir mulighet for lagring av mer data og dataen blir komprimert i bufferen i stedet for på PI Serveren [8].

3.7.5 OPC til PI system

Data i PI databasen skilles ved å ha unike navn på tags og forskjellige grupperinger. Som nevnt tidligere er gruppen fra OPC serveren satt opp som «OPC» i Point Source når OPC grensesnittet ble satt opp (3.7.4). Dette gjør filtreringen av verdiene til OPC serveren enklere. Hver enkelt verdi blir linket opp til en bestemt tag i PI databasen.

Konfigurering av tags kan gjøres på flere måter, det er her valgt å bruke PI System Management Tools for konfigurering av tags (Figur 31). Øverst til venstre er tilgjengelige PI Servere som kan kobles til, og under dette er det et utvalg av muligheter for konfigurering. For å lage nye tags brukes Point Builder. Disse blir satt opp med ulike parametere for å få de rette spesifikasjonene. Format, oppdateringsklasse, tag-klasse og adresse er egenskaper som blir konfigurert her.

Figur 30 viser åtte forskjellige tags som er laget for å lagre verdier i PI databasen fra PLSen. Figur 30 viser hvordan tagen «A_port_pressure» er satt opp. «Instrument Tag» representerer adressen til hvor verdien til tagen hentes og linkes direkte opp til OPC serveren. Adressen til OPC serveren her er ”Hardware.IM151-8 PNDP CPU.data.<variabel navn>”. Det er ikke noe bruk av IP adresser i denne adressen selv om den befinner seg på en annen enhet på nettverket. Dette er fordi denne rutingen allerede er konfigurert i OPC grensesnittet (3.7.4).



Figur 31 - Point Builder i PI SMT

- **Location1** er satt til 1. Det representerer ID nummeret til det grensesnittet det skal få data fra. I dette tilfellet har OPC grensesnittet ID = 1.
- **Location2** er brukt til spesifiesere hvilken datatype dataen skal lagres, det er brukt flyttall i PLS og det ønskes også å lagres som flyttall. Og verdien her er derfor satt til 0. For eksempel ved å sette denne til 1 blir verdiene som blir lest inn lagret som en string, ved å sette denne til 2 blir verdiene lagret som boolean.
- **Location3** bestemmer hvilken type tag det skal være. Som nevnt tidligere er det brukt Advise tag for å kun oppdatere verdiene til PI systemet når verdiene forandres. Dette blir derfor satt til verdien 1.
- **Location4** sier noe om hvilken Scan Class som brukes til verdien. Dette ble satt opp i OPC grensesnittet, og er satt til 200ms for Advise tags som har Scan Class ID 1.
- **Location5** er brukt for å sette et dødbånd for tagen. Dette kan brukes til å filtrere ut ukorrekte verdier som for eksempel kan komme av støy.

3.8 Applikasjon

Maskinen som er brukt i oppgaven er en toppmontert hivkompensator, og er modellert i 20Sim. Oppgaven til maskinen er å kompensere for bølgebevegelsene riggen blir utsatt for på sjøen. Dette for å få jevn vekt på drillbitet ved boring av brønner for søk etter hydrokarboner. Uten heave kompensering ville borestrengen fulgt riggens bevegelse påført av bølgene, noe som hindrer jevn vekt på drillbitet. Konsekvensene av dette kan føre til ødelagt utstyr, og ineffektiv boring. Heave kompensering gjør det også mulig å opprettholde driften i et større værbilde.

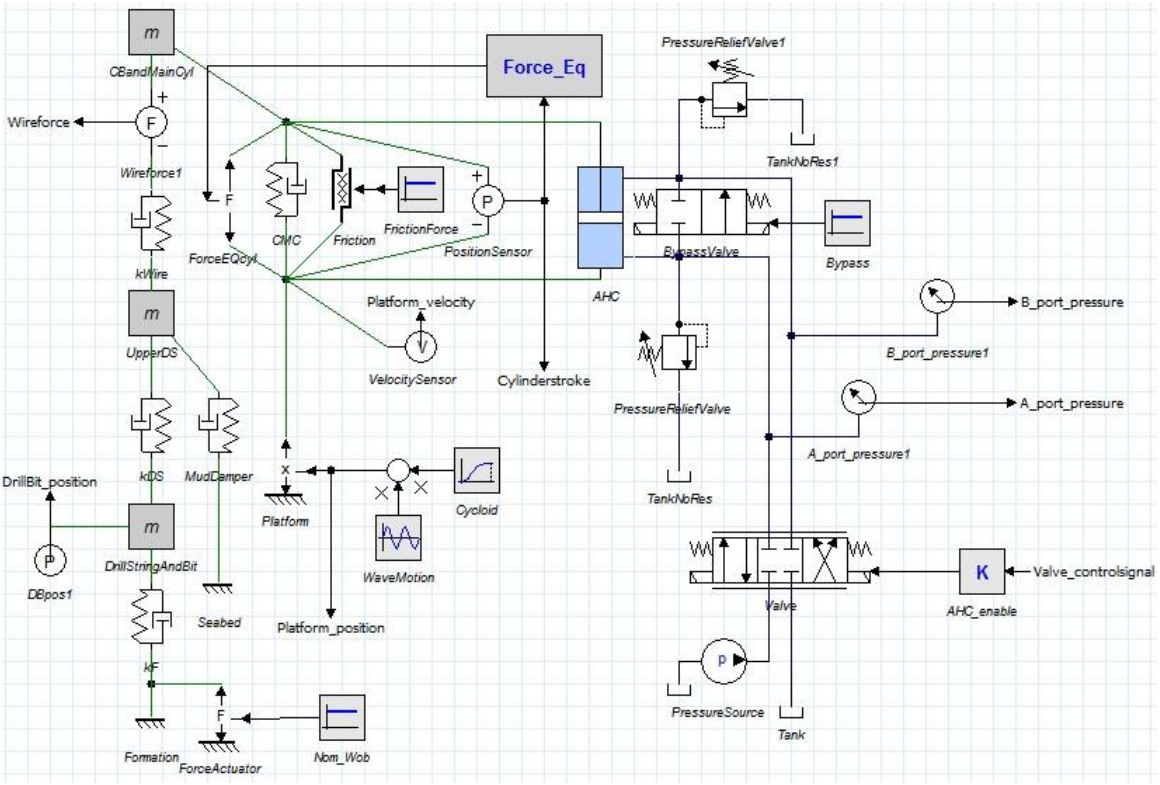
Kompensatoren har en aktiv og en passiv kompensering funksjon. Den passive kompenseringsfunksjonen består av en hoved sylinder og to kraft-likevekts sylindere, og fungerer som en hydro-pneumatisk fjær ved hjelp av akkumulatorer [2].

Den aktive kompenseringsdelen (AHC) består av to større sylindere og gir mulighet for kompensering av bølger opptil 5meter. Disse større sylinderne blir kontrollert med hydraulisk system styres ved hjelp av en kontroller.

Figur 32 - Toppmotnert hivkompensator

3.8.1 20sim modell

Kompensatoren er modellert i 20Sim, med tilnærmet like fysiske egenskaper som virkeligheten. Det er gjort noen forenklinger som å kombinere to ventiler for den aktive kompenserings delen for å gjøre kalkulasjonene under simluering lettere. De to hovedsylinderene for den aktive delen er slått sammen til èn større sylinder. Den passive kompenseringsdelen er i stedet for hydro-pneumatisk, modellert som en trandisjonelt fjær-demper. Modellen er modifisert i forhold til originalen, og det er lagt til flere sensorer for logging til databasen til PI systemet. Kontrollsystemet er også flyttet fra 20Sim modellen til PLSen.



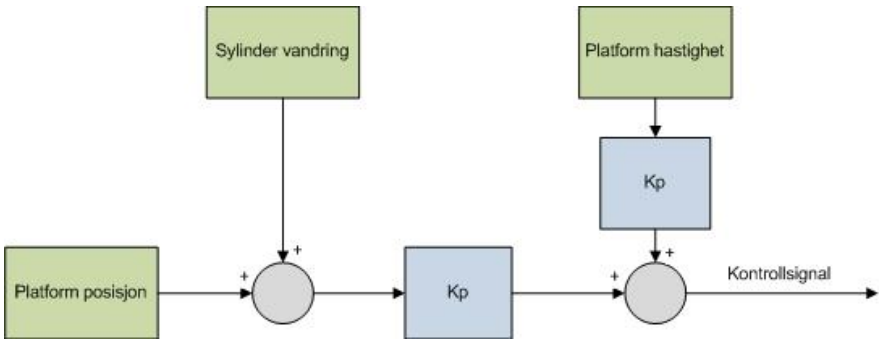
Figur 33 - 20Sim modell

3.9 PLS Oppsett

Det er valgt ut syv variabler som skal overvåkes i modellen til en hver tid. Samt kontrollsignalet som blir beregnet ved hjelp av kontrollsløyfa på bildet under.

- Platform posisjon / Bølge
- Platform hastighet / Bølgehastighet
- Sylindervandring
- Drill Bit posisjon
- Vaierkraft
- Trykk i A port
- Trykk i B port
- Kontrollsignal til ventil

Variablene blir sendt fra modellen på sanntids targetet og lagret i minnet på PLSen. PLSen bruker så noen av disse til å beregnet kontrollsignalet som sendes tilbake til modellen i HIL simuleringen for å heavekompensere bølgebevegelsen. Som vist på bildet under legges Platform posisjon og Sylindervandring sammen, før den multipliseres med en forsterkning. Denne verdien blir så lagt sammen med Plattformhastighet og en ny forsterkning. Den samlede verdien av disse variablene er da kontrollsignalet som sendes til ventilen i det hydrauliske systemet til modellen. Parameterne som er brukt som forsterkning (Kp, Figur 34) er hentet fra [2].



Figur 34 - Reguleringssløyfe for modellen

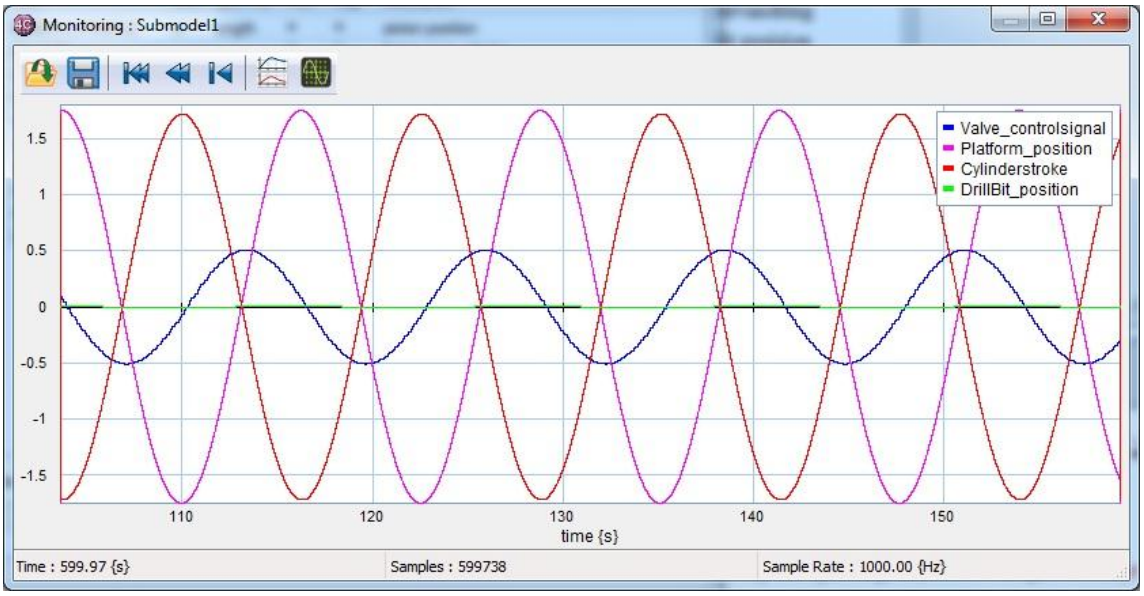
3.10 Resultater

3.10.1 20Sim 4C

Figur 34 viser et utdrag av resultatene til en 10 minutters HIL simulering fra 20Sim 4C.

- Rosa – Platform posisjon
- Rød – Sylindervandring
- Blå – Kontrollsignal til ventil
- Grønn – Drill Bit posisjon

Som vist på Figur 34 er bølgebevegelsen kompensert meget bra ved hjelp av kontrollsystemet på PLSen, og Drillbit posisjonen varierer kun med ±5 centimeter.

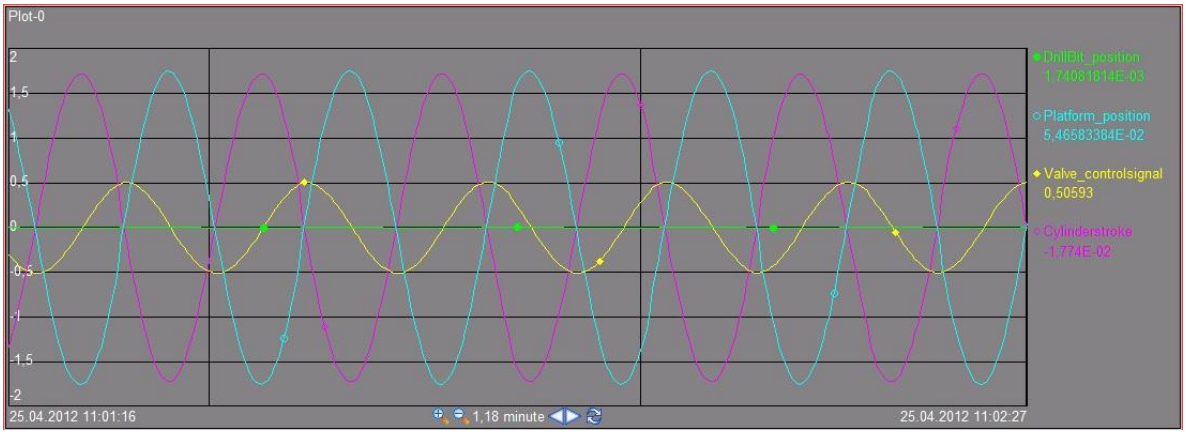


Figur 35 - 20Sim 4C Simulering

3.10.2 PI System og ProcessBook

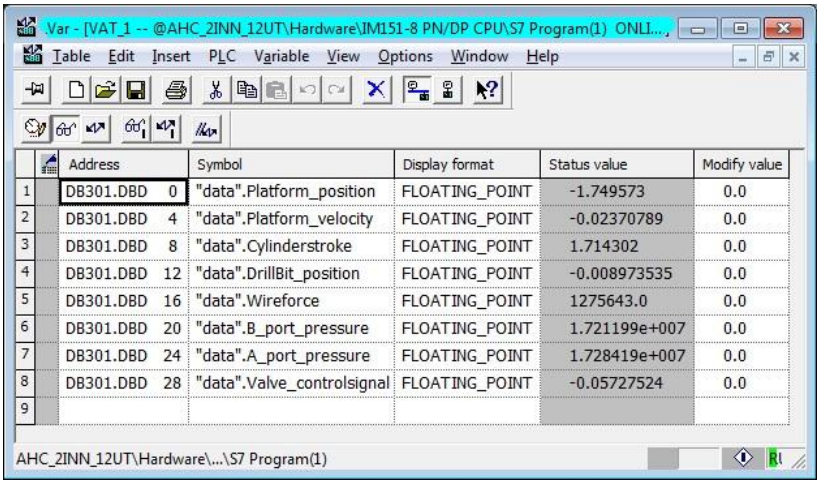
PI systemet er satt opp til å logge de utvalgte verdiene fra OPC serveren. Som nevnt tidligere er oppdateringsfrekvensen på avlesinger av verdier fra OPC server til PI system satt til 200ms (3.7.4). Det ble først prøvd ut med lavere oppdateringshastighet, men dette gjør at kurvene blir unøyaktige og videre beregninger blir derfor vanskelig. Under er et bilde fra ProcessBook som viser nøyaktig samme data som dataen lest ut direkte fra 20 Sim 4C under HIL simuleringen.

- Blå – Platform posisjon
- Rosa – Sylindervandring
- Gul – Kontrollsignal til ventil
- Grønn – Drill Bit posisjon



Figur 36 - ProcessBook skjermbilde

Data som blir logget blir også overvåket og kryssjekket ved hjelp av en varibeltabell i Step7 (Figur 37). Denne viser nå-verdi for utvalgte minnelokasjoner.



	Address	Symbol	Display format	Status value	Modify value
1	DB301.DBD 0	"data".Platform_position	FLOATING_POINT	-1.749573	0.0
2	DB301.DBD 4	"data".Platform_velocity	FLOATING_POINT	-0.02370789	0.0
3	DB301.DBD 8	"data".Cylinderstroke	FLOATING_POINT	1.714302	0.0
4	DB301.DBD 12	"data".DrillBit_position	FLOATING_POINT	-0.008973535	0.0
5	DB301.DBD 16	"data".Wireforce	FLOATING_POINT	1275643.0	0.0
6	DB301.DBD 20	"data".B_port_pressure	FLOATING_POINT	1.721199e+007	0.0
7	DB301.DBD 24	"data".A_port_pressure	FLOATING_POINT	1.728419e+007	0.0
8	DB301.DBD 28	"data".Valve_controlsignal	FLOATING_POINT	-0.05727524	0.0
9					

Figur 37 - Variabeltabell i step7

4 Tilstandsbasert vedlikehold

Formålet med tilstandsbasert vedlikehold er å redusere det totale vedlikeholdsbehovet. Tilstandsbasert vedlikehold overvåker tilstanden til komponenter til en hver tid, eller ved faste tidsintervaller. Måledata fra overvåkingen blir så analysert, og brukes til å anslå tilstanden til utstyret. Ved bruk av disse resultatene fattes beslutninger om vedlikehold eller service er nødvendig. Det gjør at vedlikehold på utstyr kan bli gjort med hensyn på tilstanden til utstyret fremfor kalenderbasert vedlikehold.

4.1.1.1 System for tilstandsbasert vedlikehold

For å kunne gjennomføre tilstandsbasert vedlikehold er det nødvendig å samle data fra utstyret det dreier seg om, og deretter bruke verktøy for å analysere disse dataene. Det er i denne rapporten sett på mulighetene til å implementere en lastsyklusteller (Kapittel 4.4)og en funksjonalitet for å detektere friksjonsendringer i hydrauliske sylinder (Kapittel 4.6). Begge deler ved hjelp av matematiske og logiske funksjoner/algoritmer.

Mulighetene for å drive behandling og analyse ved hjelp av matematiske og logiske algoritmer er veldig begrensede i PI serveren fra OSIsoft, da dette primært er en løsning for å logge og lagre store mengder data. Det er derfor bestemt å bruke Matlab til å gjennomføre dataanalyser og for å implementere algoritmer.

Matlab er et høynivå programmeringsspråk og en interaktiv plattform som gjør det mulig å utvikle programmer for komplekse matematiske oppgaver raskere enn med tradisjonelle programmeringsspråk som C, C++ og Fortran. Matlab kan brukes i et bredt spekter av applikasjoner, inkludert signal og bildebehandling, kommunikasjon, kontrolldesign, testing og måling, økonomisk modellering og analyse og beregningsorientert biologi. Veg hjelp av verktøykasser såkalte “Toolboxes” som selges separat kan man utvide og tilpasse Matlab etter det behovet man som er i bedriften/organisajonen. Matlab koden kan også integreres med andre språk og programmer.

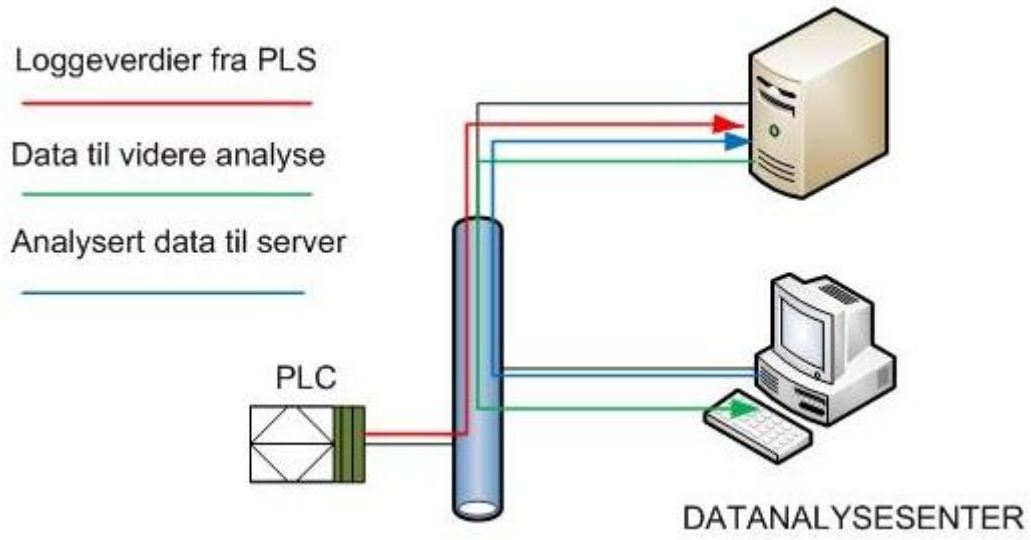
Det er tatt utgangspunkt i at brukeren av databasen ikke nødvendigvis har kjennskap til Matlab, men allikevel har behov for de opplysningene Matlab kan bidra med, og det er derfor ønskelig å komme frem til en løsning der brukeren bruker ProcessBook som eneste applikasjon for å gjøre de operasjonene som er nødvendig for å kjøre beregninger og analyser. Hvilket betyr at alle funksjoner i Matlab må aktiveres av PI server eller ProcessBook.

For å dra nytte av kapasiteten til PI serveren er det ønskelig å kunne behandle lange tidsserier med data.

Systemet skal være så universelt som mulig. Det vil at løsningen også kan brukes på andre PI servere uten store endringer.

Løsningen skal ikke påvirke PI serverens evne til å samle data.

Utkastet på løsningen ser slik ut:



Figur 38 - Utkast til infrastruktur

Det har blitt valgt å gå frem på følgende måte:

1. Sette opp kommunikasjon
2. Teste kommunikasjon
3. Implementer lastsyklusteller
4. Teste lastsyklusteller
5. Utvikle modell for deteksjon av friksjonsendring
6. Teste deteksjon av friksjonsendring
7. Ferdigstille systemet til en helhetlig løsning

4.2 Matlab til PI kommunikasjon

4.2.1 Mulige løsninger

Det første som ble sett på var hvordan data skulle sendes og mottas mellom PI serveren og Matlab. Det er ønskelig å komme fram til en løsning som både kan skrive og lese mellom Matlab og PI server.

Det er blitt gjort undersøkelser for å finne ut hvilke muligheter man har for å skrive og hente data fra PI serveren og til Matlab. OSIsoft har gjort arbeid på dette området før [13]. Her er det testet et utvalg av forskjellige fremgangsmåter og arbeidet er presentert på supportsidene til OSIsoft sitt Virtual Campus. Det er da tatt utgangspunkt i dette for å sette opp kommunikasjonen mellom Matlab og PI.

Følgende metoder er vurdert:

1. ADO og PI OLEDB provider
2. Matlab sin Database Toolbox og PI JDBC Driver
3. Matlab sin OPC Toolbox og PI OPC DA Server
4. Integration Objects OPC HDA toolbox og PI OPC HDA Server
5. Programmering med en blanding PI og Matlab funksjoner
6. Ved hjelp av kommaseparerte filer.

4.2.2 Vurdering av alternativer

1. ADO - ActiveX Data Objects

Dette er ett sett Component Object Model (COM) objekter som gjør at man kan lese og manipulere data fra et utvalg av kilder gjennom en OLE DB provider. I dette tilfellet PI Object linking and embedding database (OLEDB) provider som gir tilgang via COM funksjonalitet til PI databasen. Data blir da lest og skrevet ved hjelp av pekere.

- **Fordeler:** Lett å bruke, høy hastighet, og et lite diskavtrykk. ADO støtter nøkkelfunksjoner for å bygge klient/server funksjoner. Og krever generelt lite forkunnskaper for å sette opp.
- **Ulemper**: Kommuniserer kun direkte med serveren, støtter ikke kommunikasjon med ProcessBook. Matlab må kjøre hele tiden.

2. Matlab sin database Toolbox og PI JDBC Driver

JDBC er kort for Java Database Connectivity, og PI JDBC Driver gir tilgang til å kunne bruke Java programmeringsspråk for å få tilgang til data. Også her blir data lest og skrevet ved hjelp av pekere.

- **Fordeler**: Gir sikker nettverkskommunikasjon Hypertext Transfer Protocol Secure (HTTPS), og støtter kommunikasjon både på Windows og Linux plattformer.
- **Ulemper:** Snakker kun med serveren, ikke med ProcessBook. Matlab må kjøre hele tiden.

3. Matlab sin egen OPC Toolbox og OPC DA/HDA server

Med denne løsningen blir det satt opp en OPC grensesnitt på PI serveren på en slik måte at den er representert som en OPC server for tredjeparts programvare. Med Matlab OPC Toolbox kan man lese og skrive til OPC servere og på den måten behandle data. OPC DA gir tilgang til serverens nå verdier, men gir ikke tilgang til historisk data (tidligere målte verdier).

- **Fordeler**: Gir sanntidsdata fra serveren.
- **Ulemper:** Gir ikke historisk data, man kan derimot sette Matlab til å logge verdier som gir historisk data til en viss grad, men man kan ikke gå tilbake i tid. Matlab må kjøre hele tiden.

4. Integration Objects OPC HDA Toolbox og PI OPC HDA Server

OPC HDA Toolbox er et verktøy levert av Integration Objects, som integreres i Matlab på samme måte som Matlab sine egne verktøy. Man kan da sette opp PI OPC HDA server og få tilgang til historiske data.

- **Fordeler**: Kan koble til mange servere samtidig og liste opp alle tilgjengelige servere.
- **Ulemper**: ikke et grensesnitt mot ProcessBook, HDA Toolbox til Matlab er tredjeparts programvare. Matlab må kjøre hele tiden. Er ikke kompatibel med Windows 7.

5. Programmering med en blanding PI og Matlab funksjoner

Med språk som VB6, VB.NET eller C#, kan man utnytte funksjonalitet som ligger i PI og Matlab og lage rutiner for å hente/sende data, samt kunne vekke funksjoner i både Matlab og PI.

- **Fordeler**: ProcessBook har egen VB script editor, som gjør at man kan lage skriptet direkte i ProcessBook. Med VB6/VB.NET og kan man starte Matlab og Matlab-funksjoner eksternt fra for eksempel ProcessBook.
- **Ulemper**: Kan ta lang tid å sette seg inn i med lite kunnskap fra VB6/NET.

6. Kommaseparerte filer (.csv) er en mulighet for å overføre data mellom Matlab og PI.*

- **Fordeler:** Kan brukes til å overføre mye data fra PI til Matlab, Excel og andre applikasjoner som leser csv-filer.
- **Ulemper:** dette er ikke en automatisert kommunikasjonsform. Man må selv generere et sett csv-filer for de dataene man ønsker å analysere og deretter laste disse opp i den applikasjonen som skal analysere dataene. Det er også ansett som en form en veis dataoverføring ettersom det ikke uten videre kan ta en csv fil og importere til PI server.

4.2.3 Oppsummering

Det er valgt å ikke bruke OPC server, dette fordi det trengs en kombinasjon av to løsninger for å kunne utnytte både DA og HDA funksjonaliteten til OPC. Det er også et underordnet mål å holde seg til OSIsoft og Mathworks som leverandør, dette for å slippe ekstra utgifter dersom det ikke er en åpenbart bedre løsning. ActiveX og JDBC henter data på tilnærmet lik måte, men det er valgt å gå videre med ActiveX fordi dette er en mer kjent metode enn Java. Det er derfor gjort tester med ActiveX for å vurdere denne kommunikasjonen. Fordelen med ActiveX og JDBC er at man får tilgang rett inn i SQL databasen, og på den måten slipper å gå via andre applikasjoner og grensesnitt.

Det har blitt satt opp kommunikasjon for å hente data fra PI serveren. Denne koden er delt opp i flere subrutiner. Men prinsippet går ut på at man genererer en tilkoblings-string som beskriver type oppkobling, hva den skal kobles opp mot og hvilke data som er ønskelig. Deretter brukes det ulike pekere (4.3.1.1) til å hente de dataene som ønskes.

4.3 Testing av kommunikasjon

Programmet som er laget for å teste kommunikasjonen mellom Matlab og PI serveren kan bli funnet i Vedlegg 7.5. Under følger en forklaring av rutinene programmet inneholder.

1. cn = adodbcn(cnstr). Denne funksjonen åpner kommunikasjon mellom Matlab og SQL, ved hjelp av en connection string som er samsavarer med databasen . Denne stringen definerer tilkoblingstype, datakilde, leverandør, brukerID og Passord. Adodbcn funksjonen kobler deretter opp og definerer pekeren.Pekeren kan da enten bli settes til å være server peker eller klient peker.

2. PI_SampledData Lager forespørselen om hvilke data som skal hentes/kopieres, og setter deretter igang adodbquery,

3. adodbquery Denne funksjonen får pekeren til hente ut de dataene som er blitt forespurt

4. invoke(cn,'close') Lukker kommunikasjonen.

For å ha data til å teste med ble det kjørt en HIL simulering slik at det ble generert måledata til databasen. Måledataene ble deretter hentet via ActiveX protokollen i Matlab. For å verifisere dataene er tallene i Matlab sammenlignet med PI databasen. Måleverdiene og tidsserier stemmer overens og verdiene er korrekte.

Derimot var formatet feil, og dataene kom i celleformat, dette kan ikke brukes til å gjøre kalkuleringer med, derfor er måledataene formatert til dataformatene double-word og tidsseriene til char/string.

4.3.1.1 Test av kapasitet for henting av data fra PI til Matlab

Det har blitt testet kapasitet på kommunikasjonen. Det ble da testet hvor mange måleverdier det var mulig å hente på en forespørsel, og resultatet endte da opp med å bli 1.000.000 verdier som maks antall verdier. Dette ble testet med serverpeker. I en sql database ligger verdiene i rader, det blir så sendt en forespørsel om å ta ut ett sett med data. Denne dataen blir lagt i pekerne, som fungerer som et mellomlager der dataen blir behandlet. Hvor dette mellomlageret ligger bestemmes i protokollen. Og kan settes til server eller klient. Det har også blitt gjort tester med pekeren liggende på klientside. Og det ble da mulig å hente ut 1.500.000 verdier. Begrensningen ligger nå i rettighetene til serveren. Og er det behov for å hente flere verdier så kan rettighetene på serveren endres. 1.500.000 verdier er tilstrekkelig for denne applikasjonen.

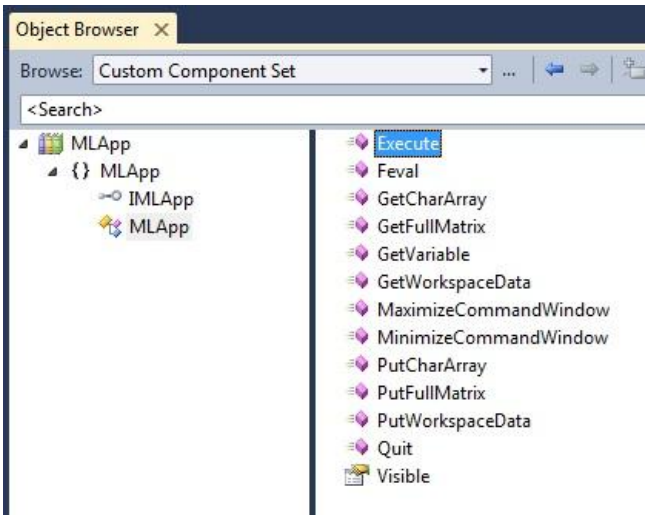
4.3.1.2 Testing av start av programsekvenser

Testing viste at det fungerte dårlig å bruke ActiveX til å starte programsekvenser, fordi målet er å ha en løsning der syklustelleren blir kjørt direkte fra ProcessBook.

ProcessBook har i utgangspunktet ikke støtte for ActiveX (dette kan programmeres med VB), og det blir en omvei dersom alle kommandoer sendes via serveren. Et annet problem er at Matlab hele tiden må kjøre og lese verdier fra serveren da denne som er satt opp som klient.

For å kunne starte opp Matlab, kjøre kommandoer/funksjoner, og sende startbetingelser er det derfor laget et VBA script i ProcessBook. For å kunne bruke Matlab kommandoer i VBA scriptet må man legge til Matlab som referanse og deretter er det ett sett med kommandoer tilgjengelig

.



Figur 39 – Matlab-kommandoer i VBA

Figur 39 viser kommandoene som da er tilgjengelig i VBA. Det ble deretter bekreftet ved testing at det er mulig å starte Matlab, kjøre funksjoner og skrive kommandoer på en ganske enkel måte, og få data tilbake fra PI og inn i ProcessBook. Dette er da tall som kun ligger i minne til ProcessBook, og dersom man da ønsker og skrive diss verdiene vil/fra serveren må man legge inn i scriptet funksjoner for dette også (Vedlegg 7.6).

4.4 Rainflow syklusteller algoritme

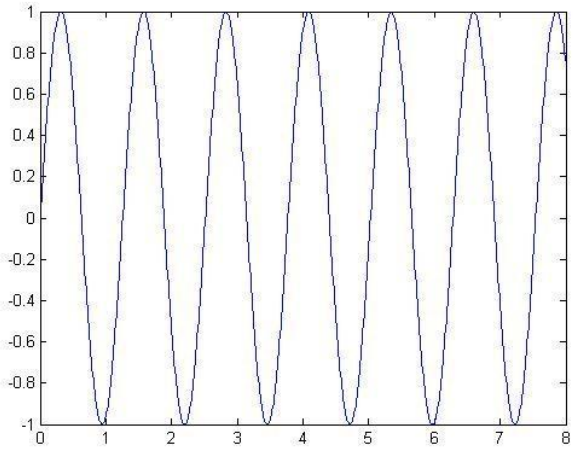
Det er ønskelig å implementere en syklusteller med hensyn på lastvariasjonene i konstruksjonen. Mulighetene her er mange dersom det er mulig å se på lange tidsserier med data og analyserer dette, dataene kan være nyttig informasjon med hensyn på levetidsberegninger, utmatting og fastsettelse av serviceintervaller. Det er stort sett i dag designet utstyr på en slik måte at man tar utgangspunkt i en antatt belastning og antatt operativ drift, det samme er ofte utgangspunktet hva serviceintervaller gjelder. Dette har resultert i at det ofte har blitt byttet og overhalt komponenter og konstruksjoner som fortsatt kunne hatt mange driftstimer igjen før det har vært nødvendig å gjøre noe. Dette er blitt gjort ettersom det ikke har vært muligheter til å overvåke utstyret, og belastningene dette blir utsatt for, eller logge det slik at data som kan analyseres.

Rainflow er en metode/algoritme for å beregne lastsykluser med hensyn på utmatting i tilfeller der det er komplekse lastsituasjoner. Det være seg når det er utstyr der lasten varierer både med størrelse og frekvens. Da kan man ikke lenger bare telle perioder på samme måte som om lasten var representert ved en sinus med fast amplitude.

Rainflow syklusteller algoritmen ble utviklet av de japanske ingeniørene Tatsuo Endo og M. Matsuisk, og selv om det er mange måter å beregne syklusantall på, er Rainflow den mest populære. Rainflow har også blitt kalt pagodatak metoden. Metoden kan forklares ved at det blir vurdert hvordan en strøm av vann renner ned et pagodatak[2] [14].

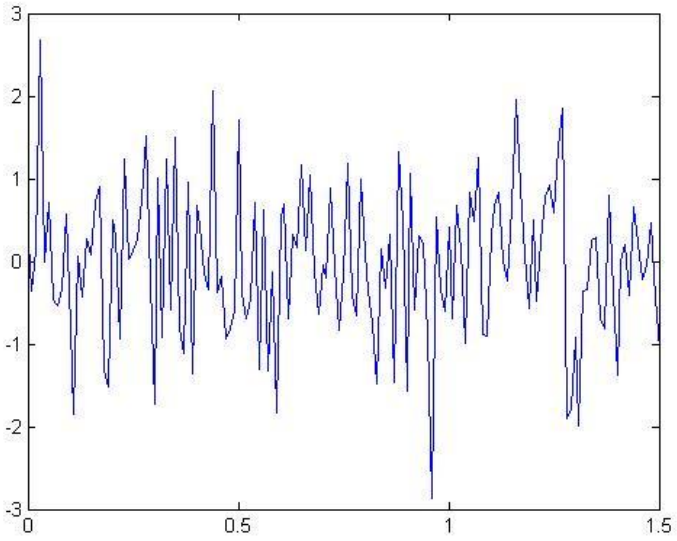
[2] Pagodatak er en typisk takform i mange asiatiske land der taket er delt opp i flere nivåer.

Eksempel:



Figur 40 - Lasttilfelle med fast amplitude og frekvens

Figur 40, her kan man telle antall lastsykluser lett med å kalkulere frekvensen eller bare telle toppunkter. Da er det ganske enkelt å gå rett videre på beregne utmatting. Dessverre er lasttilfellene på store maskiner like enkle, noe som gjør analyse av dette mere kompleks.

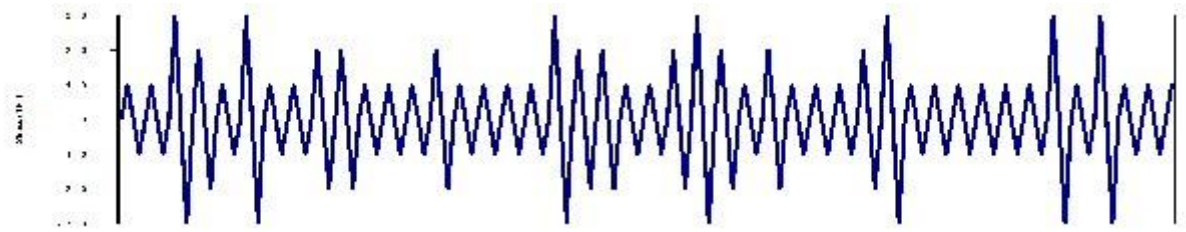


Figur 41 - Kompleks lastsituasjon

Figur 41 viser ett mer typisk lasttilfelle. For å telle antall lastsykluser ved dette tilfellet må man gå frem på en litt annen måte for å få ett resultat som vil representere lastens betydning på konstruksjonen med hensyn på utmatting. Dette er jobben til Rainflow syklus teller.

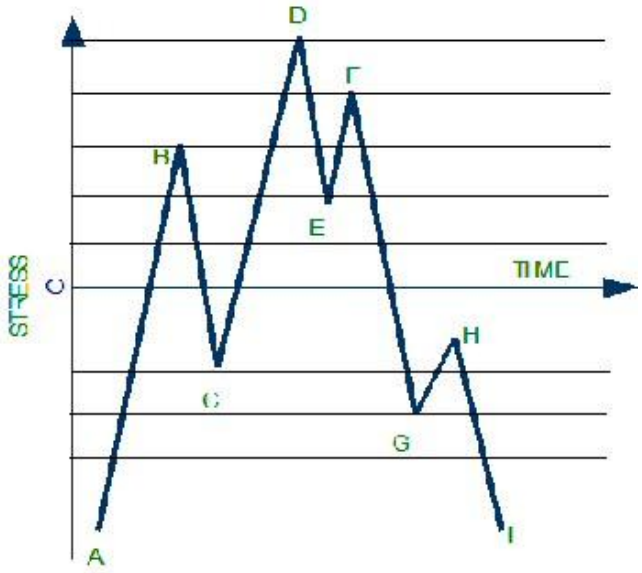
4.4.1.1 Fremgangsmåten ved bruk av Rainflow:

1. Ta utgangspunkt i ett sett med måledata som representerer belastningen i konstruksjonen som skal vurderes.



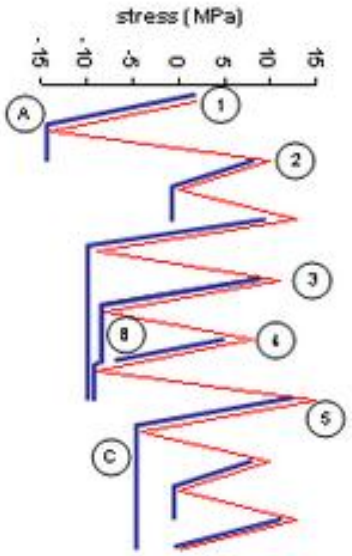
Figur 42 - eksempel på måledata

2. Reduser målingene til å kun innebære topp og bunnpunkter.

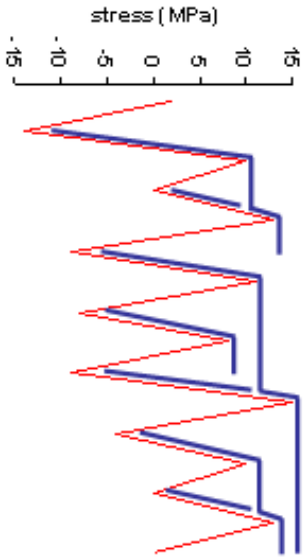


Figur 43 - Topp og bunnpunkter

3. Snu grafen 90° i klokkens retning, toppene og bunnene kan nå ansees som taktopper.



Figur 44 - Rainflow illustrasjon 1



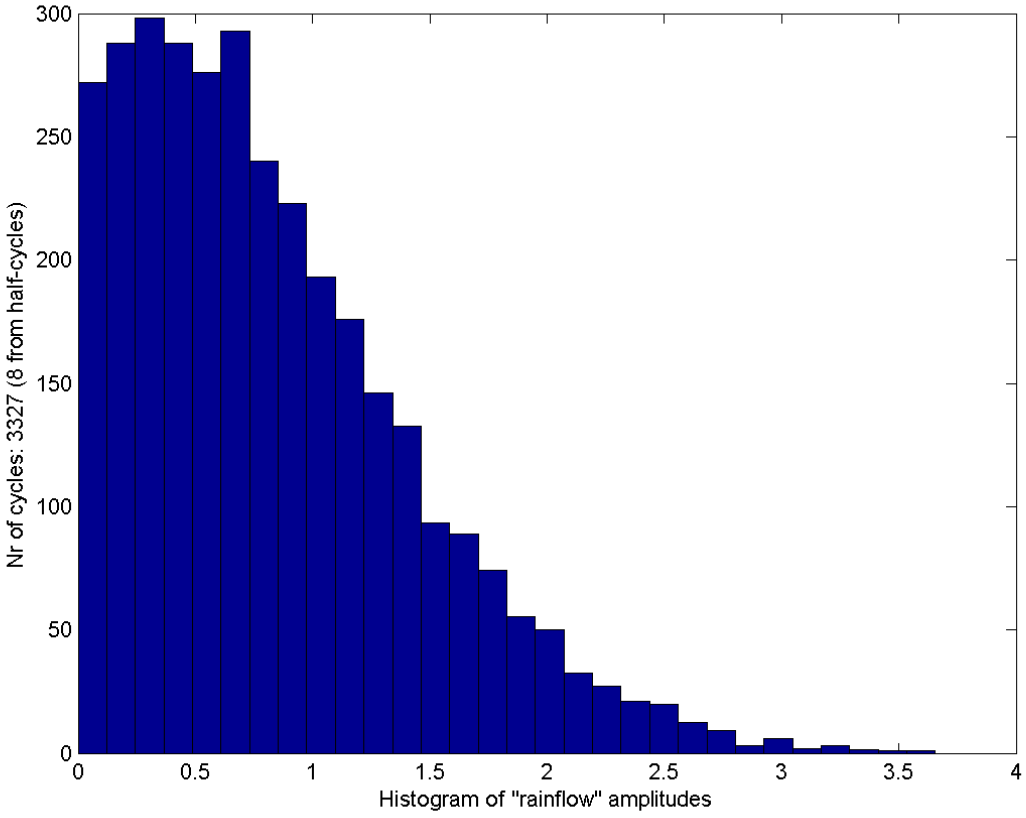
Figur 45 - Rainflow illustrasjon 2

4. Anse hver topp som en kilde til vann som renner nedover taket.
5. Tell antall halv-sykluser ved å se etter avslutninger i grafen som oppstår når enten:
    1. Enden på målingene er nådd.
    2. Strømmen treffer en strømning som startet tidligere
    3. Strømmen renner motsatt av en spenningstopp med større amplitude.
6. Gjenta deretter trinn 5 men denne gangen anser man hver bunn som kilden til vann som renner nedover taket.
7. Størrelsen til hver halvsyklus vil da være lik lengden på en slik strøm
8. Deretter parrer man de halvsyklusene med identisk størrelse men motsatt retning for å få en komplett syklus, det vil da typisk også være halve sykluser igjen som ikke har identiske motparter[14].

4.4.2 Løsning i Matlab

Den syklustelleren som er satt opp til å gjøre beregninger på data fra PI serveren er delt opp i 3 separate funksjoner.

1. sign2ext – fjerner alle data som ikke er vendepunkter(topp og bunnpunkter). Slik at man får en graf bestående av rette linjer. Denne funksjonen fjerner også de tilhørende tidsnøklene slik at man også får tatt hensyn til tiden på måledataene. Se vedlegg 7.4.
2. Rainflow – er programmert i C og kompilert som en mex funksjon. Dette gjør at den gjør kalkulasjonene kjappere, men det er også en annen syntax. Det er her selve Rainflow algoritmen ligger. Den bruker da resultatet fra sign2ext til å beregne antall sykluser og halve sykluser, amplitudene til syklusene og tid, Resultatet blir da gjengitt som en 4*n matrise. Se vedlegg 7.4
3. Denne matrisen kan deretter behandles på ønskelig måte. Under testing er det blitt vurdert at den mest hensiktsmessige måten å presentere resultatet er ved hjelp av histogram. Det er derfor i dette prosjektet valgt å lage et histogram, med hensyn på antall sykluser og amplitude.

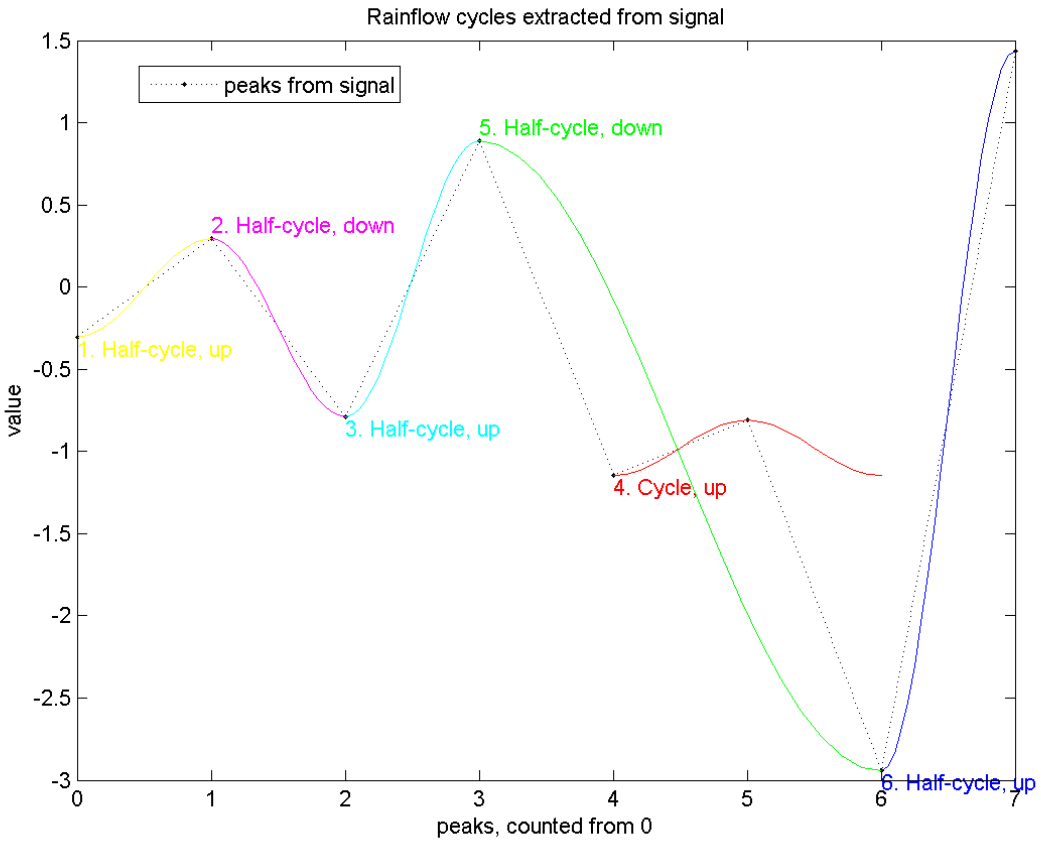


Figur 46 - Histogram av Rainflow amplituder

Figur 46, viser hvordan et typisk resultat av 10.000 tilfeldige målinger ser ut.

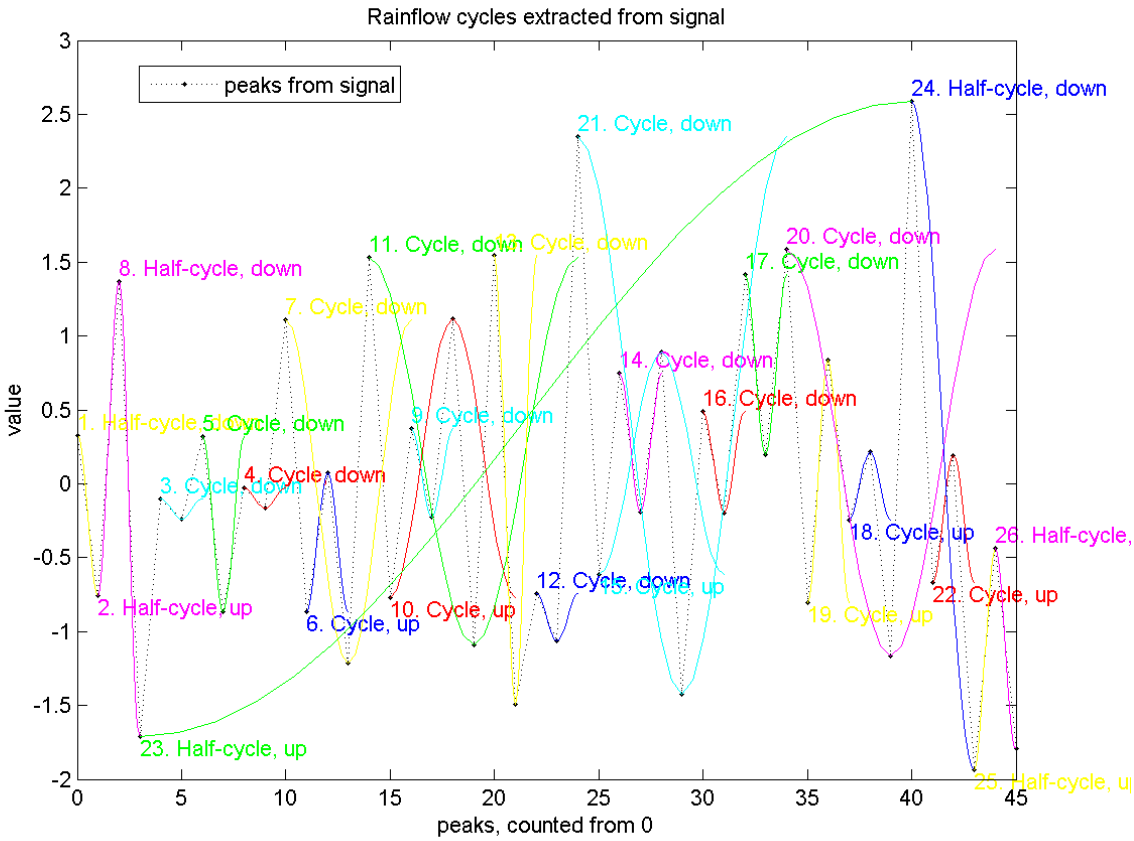
4.4.3 Testing i Matlab

Det har blitt ytret ønske fra oppdragsgiver at man med syklustelleren skulle kunne analysere lengre tidsserier. Da blir det mye data som skal gjengis visuelt, og det er da gjort noe testing for å se hvordan man presenterer resultatet.



Figur 47 – Rainflow-graf

Figur 47 var første graf. Det gir et veldig godt bilde over hvordan syklustelleren kalkulerer sykluser, og ser visuelt veldig bra ut i dette eksempelet.

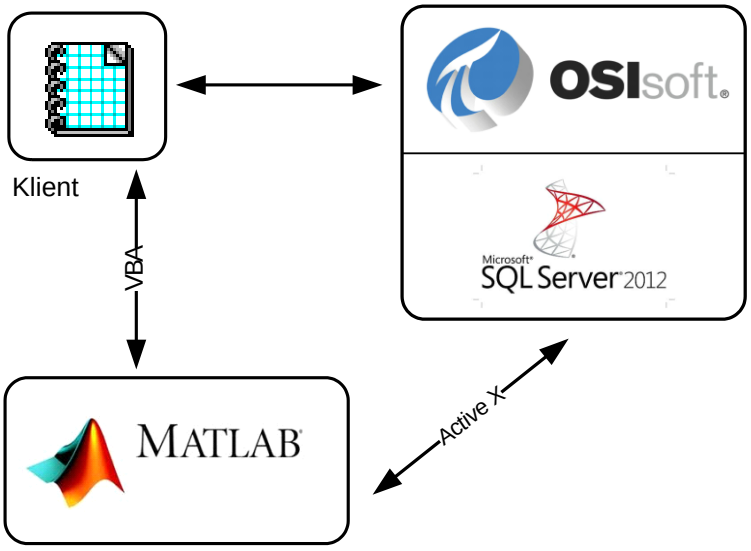


Figur 48 - Resultat av lang tidsserie

Figur 48 er samme representasjon som Figur 47. Men denne gang med en langt større datamengde. Man kan da tydelig se at dette blir en dårlig visualisering, da det blir veldig komprimert. Figur 48 representerer et datasett med 100 vendepunkter(topp og bunnpunkt), er det da et ønske om å analysere data som kan inneholde flere tusen punkter vil bli kaos og det kan derfor ikke representeres på denne måten. Det har derfor blitt valgt å gå bort i fra denne løsningen ettersom man har tatt utgangspunkt i at det er ønskelig å bruke dataene til videre kalkulasjoner. Det er derfor som nevnt tidligere at det mest illustrer bare metoden å presentere resultatet ville være ved hjelp av histogram, se Figur 46.

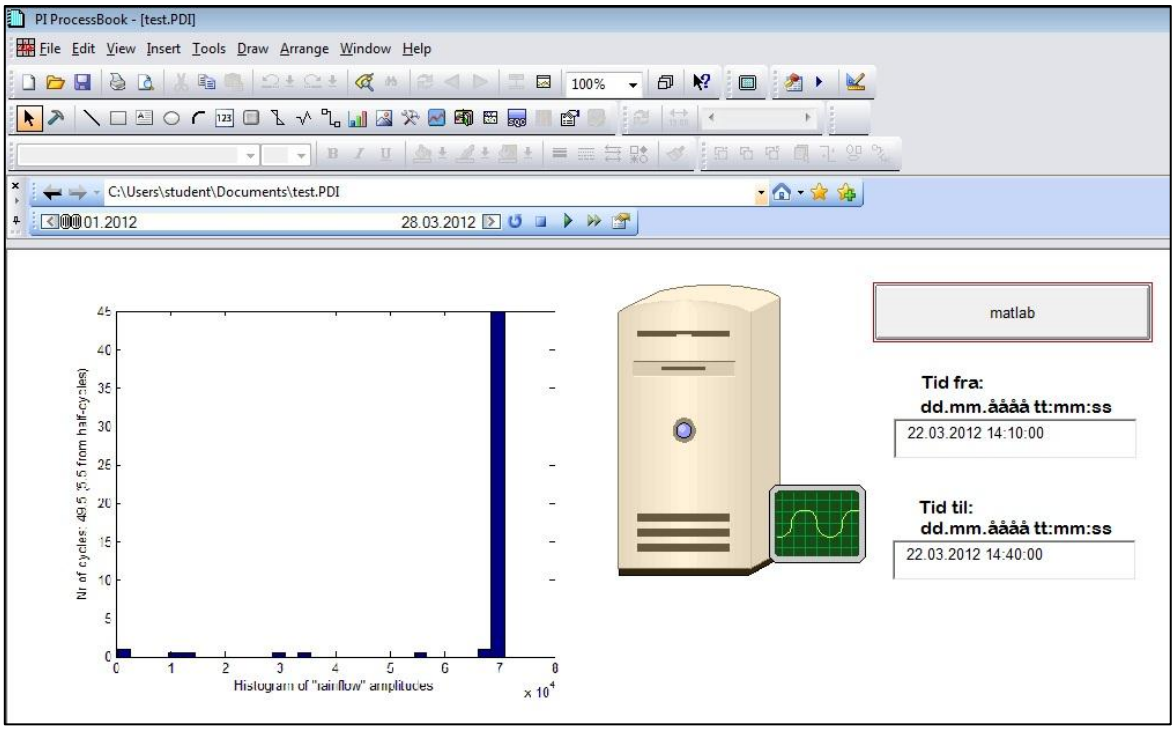
4.5 Endelig løsning på Rainflow rutine

Den endelige løsningen av Rainflow rutinen består av ProcessBook, PI server og Matlab. Det blir sendt data for tidsintervallet man ønsker å kjøre syklustelleren i og kommando for hvilken funksjon man ønsker å kjøre i Matlab fra ProcessBook. ActiveX protokollen for å hente de dataene som trengs er implementert i Matlab koden som kjører syklustelleren (Vedlegg 7.7). Resultatet blir deretter representer som et histogram, dette histogrammet lagres på datamaskinen og lastes deretter inn i ProcessBook via det samme VBA scriptet som starter og kjører matlab koden (Vedlegg 7.6).



Figur 49 - Oversikt over endelig løsning

Figur 49 viser hvordan systemet henger sammen, der data fra serveren går direkte til Matlab, og data fra ProcessBook går direkte til Matlab, og tilbake.



Figur 50 - Rainflow i ProcessBook

Figur 50 viser hvordan løsningen ser ut i ProcessBook. I to tekstbokser til høyre skrives ønsket dato fra og til for perioden som skal analyseres. Over disse tekstboksene er det en knapp, som starter analysen når man trykker på den. Til venstre vises resultatet i form av et histogram som vist til venstre.

Histogrammet til venstre viser en Rainflow analyse for måledata generert ved HIL simulering av kompensatoren (3.8). Bølgene som blir simulert er reine sinuskurver, noe som gjør at belastningen blir jevn og lik over tid. Dette vises tydelig i histogrammet der den ene stolpen representerer den hyppigste belastningen.

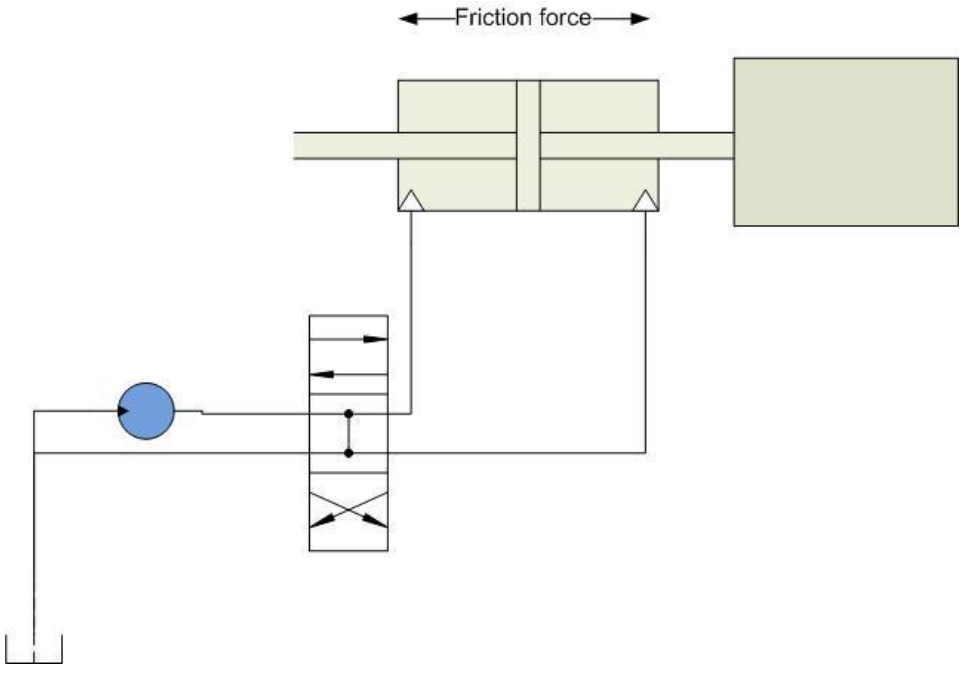
4.6 Deteksjon av friksjonsendring

Økt friksjon i maskinkomponenter er ofte en indikasjon på slitasje. Denne friksjonen oppstår ofte i opplager, og tetninger på deler i bevegelse. Det er vanlig å sette faste serviceintervaller på enkelte av disse komponentene, mens det i andre tilfeller ikke byttes før komponenten er ødelagt. På maskinen som er modellert i oppgaven er det derfor ønskelig å finne en metode for å detektere friksjonsendringer i de hydrauliske sylinderne for AHC. Endring av friksjon i disse sylinderne kommer ofte av slitasje på tetninger og foringer i sylinderen.

4.6.1 Valgt metode

Det er valgt å bruke en metode der det blir generert en modell av systemet, og representert som en matematisk funksjon. Deretter blir det brukt målte verdier fra den faktiske prosessen som deretter blir brukt som input i den matematiske modellen. Denne kan da anses som en observatør.

Det er laget et forenklet system som skal brukes i utviklingen av løsning for deteksjon av friksjonsendring. Dette systemet består av en hydraulisk sylinder med masse, friksjon og eksterne krefter.



Figur 51 - Forenklet hydraulisk system

Det er tatt utgangspunkt i Newtons 1. lov som sier at summen av krefter er lik null.

$$\sum F = O$$

Systemet kan da representeres matematisk, ved hjelp av en differensialligning.

$$\sum F = (m \cdot \ddot{x}) = A(P_1 \cdot P_2) - C \cdot sign(\dot{x})$$

$$(m \cdot \ddot{x}) - A(P_1 \cdot P_2) + C \cdot sign(\dot{x}) = 0$$

Her er:

$\sum F =$ Summen av krefter

m = Masse (kg)

$\ddot{x}$ = Akselerasjon

A = Arealet oljen har å trykke på i sylinderen

P_1 og P_2 = Trykket i hvert kammer på cylinderen

C = Friksjon (konstant)

$sign(\dot{x})$ = Bevegelsesretning (-1, 0 eller1)

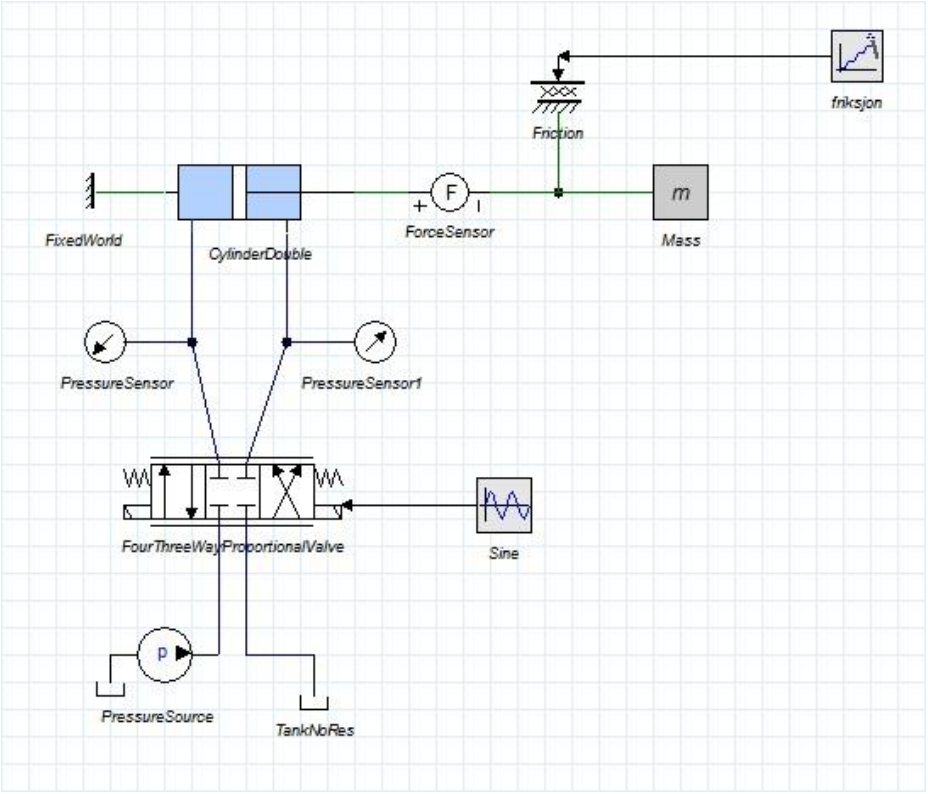
Prinsippet er at dersom man bruker måleverdiene fra den aktuelle maskinen, i form av hastighet på sylinder trykk i kammer 1 g 2 og den samlede massen som blir satt i bevegelse, og setter dette inn i ligningen, så vil resultatet være null ved normale omstendigheter. Da kan det antas at friksjonen er uendret. Dersom man derimot sitter igjen med verdier (rester), kan det antas at noe har endret seg. Og dersom den eneste variabelen som ikke er målbar vil være friksjonen, så vil det være her endringen er. Verdiene fra denne matematiske fremstillingen kan anses som restkrefter (residual forces), derav navnet residual testing [11].

4.6.1.1 Utfordringer

Konseptet er en forenklet fremstilling, men det forklarer godt prinsippet. Dessverre vil ikke implementering av et slikt system være like enkel. I denne oppgaven er det gått ut ifra at systemet er så enkelt som beskrevet over. Virkeligheten er en annen, og en maskin med hydrauliske, maskintekniske og kontrollmessige aspekter vil ha mange parametere som vil spille inn på resultatet. Det kan være store utfordringer i det å lage en modell som tar høyde for alt dette. Ting som da må tas hensyn til kan være, hydrauliske lekkasjer, friksjon i andre deler av konstruksjonen som også påvirker resultatet i modellen. En mye mer kompleks friksjonssituasjon, som kan innebære både stiksjon og friksjon som kan gi hysterese og så videre. Komplekse lastsituasjoner. Målestøy som ofte krever at dataene filtreres, og problemer med å ha sensorer til å måle akkurat de verdiene som er ønskelige. Dette er ikke tatt hensyn til ettersom hovedvekten i denne oppgaven går ut på å presentere og sette opp infrastrukturen i systemet. Løsningen skal derimot bli produsert i så måte at det enkelt skal kunne implementeres en mer kompleks modell. Se [11] for ytterligere informasjon om residualtesting.

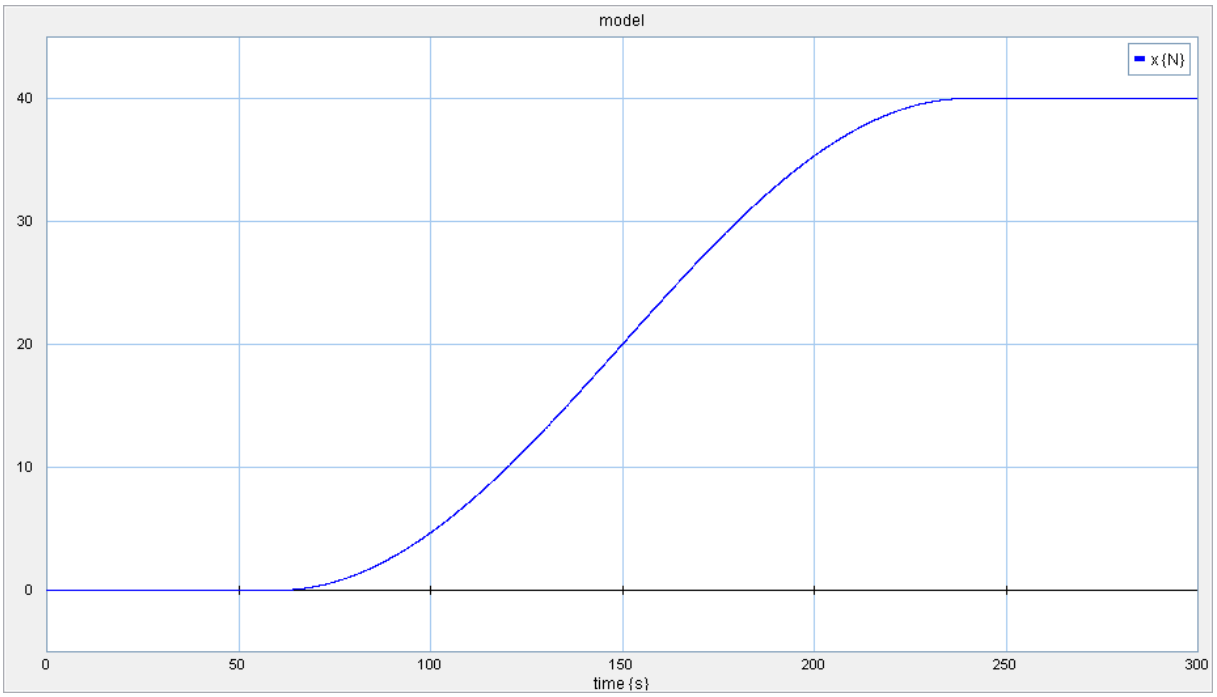
4.6.1.2 Tenkt løsning

Det er laget en egen modell i 20sim for å utvikle dette systemet. Denne modellen er bygget på samme måte som vist i Figur 51. Denne skal så kjøres i HIL systemet og logges av PI serveren.



Figur 52 - Friksjonsmodell i 20Sim

Figur 52 viser hvordan friksjonsmodellen ser ut i 20sim. Denne modellen er satt opp med en friksjon som øker gradvis etter en viss tid. Og deretter flater ut igjen ved en senere anledning.



Figur 53 - Friksjonskraft

Figur 53 viser hvordan friksjonskraften øker med tiden. Dette kan justeres slik at det passer applikasjonen.

Det er tenkt at systemet skal fungere på samme måte som Rainflow syklusteller løsningen. Det vil si at det blir sendt en kommando fra ProcessBook sammen med et tidsintervall om å kjøre friksjonsanalyse. Matlab bruker da ActiveX til å hente ønsket data, og legger dataen i lister. Det kan da kalkuleres restverdier/residualverdier for hvert tidsintervall. Det må tas hensyn til at støy på måleverdiene kan forekomme. Resultatet trenger da å filtreres. Etter at filtrering er gjennomført kan dataene behandles. Verdiene kan da visualiseres i en graf som lagres og lastes opp i ProcessBook. Det er også mulig å implementeres en algoritme som kontrollerer at restverdiene holder seg innenfor en gitt toleranse, og dersom det går utenfor denne toleransen kan det visualiseres ved hjelp av grafer eller lignende.

Det er også interessant å se om det var mulig å sette opp et system der det ble kjørt en kontinuerlig deteksjon av friksjonsendring i sanntid Da må det gjøres nye vurderinger og utvidet testing på kommunikasjon, for å se hvilke protokoller som er best egnet til dette formålet.

Ettersom systemet ikke er ferdigstilt og testet må de to siste avsnittene sees på som et konsept. Testing av dette systemet kunne ført til endringer av konseptet.

5 Konklusjon

Denne rapporten tar for seg utvidelse og testing av et eksisterende HIL system. Testing av ny kommunikasjonsprotokoll i HIL systemet ga gode resultater. Ved å bytte protokoll fra TCP til UDP ga det forbedringer i simuleringen i form av pålitelighet. Dette gjør HIL simuleringen mer nøyaktig.

HIL systemet er utvidet med OSIsoft sitt PI system og det er designet en god IT infrastruktur for arbeid med tilstandsbasert vedlikehold. Databasesystemet funger tilfredsstillende og har veldig mange muligheter til utvidelser og kommunikasjon med andre enheter. Systemet gir også enkelt tilgang til sanntidsdata samt historisk data, som fungerer veldig bra.

Det har blitt utviklet et system for tilstandsbasert vedlikehold. Der har det blitt implementert en Rainflow syklusteller algoritme for telling av lastsykluser ved gitte tidsintervaller. Resultater fra HIL simulering av en toppmontert hivkompensator ble brukt til å teste og konfigurere Rainflow algoritmen.

Trolig vil PROFINET RT eller IRT gi bedre resultater enn UDP og TCP kommunikasjon i HIL systemet. Som nevnt i kapittel 2.2.5 gir PROFINET deterministisk kommunikasjon. Testing av PROFINET RT og IRT kan gjøres, og ved bedre resultater enn UDP og TCP kan dette implementeres i HIL systemet og forbedre kommunikasjonen ytterligere.

For å utnytte HIL simulering bedre i bedrifter kan det utvikles et modellbibliotek med ferdige modeller tilpasset utstyret som produseres i bedriften. Dette gir mulighet for raskt og effektivt å kunne simulere og teste endringer av kontrollsystemet til maskiner.

Løsningen for deteksjon av friksjonsendring må testes og verifiseres da dette bare er et konsept. Etter ferdigstilt løsning kan det bli implementert i IT infrastrukturen.

6 Referanser

[1] OPC Foundation, 10 Mai 2012, http://www.opcfoundation.org/Default.aspx/01_about/01_whatis.asp?MID=AboutOPC.

[2] Jørgen Haaø and Steffen Vangen. *Impelentation of a Hard Real-Time system for Hardware-in-the-loop testing*, UiA Master's Thesis 2011.

[3] *20Sim by Controllab Products B.V.*, 10 Mars 2012, http://www.20sim.com.

[4] *20Sim 4C by Controllab Products B.V.*, 10Mars 2012, http://www.20sim.com/product/20sim4C/20sim4C.html.

[5] *RTAI - Real Time Application Interface Official Website*, 2 Mai 2012, http://www.rtai.org.

[6] *OSI PI Blog*, 15 Mai 2012, http://osipi.wordpress.com/2010/07/05/relational-database-vs-process-historian-for-process-data-use-both/.

[7] *OSIsoft Products*, 20 Februar 2012, http://www.OSIsoft.com/software-support/products.

[8] *OSIsoft techsupport API Buffer*, 20 Februar 2012, http://www.techsupport.OSIsoft.com/Products/Interfaces/Buffering/API+Buffer+Server/API+Buffer+Server+Overview.htm

[9] *PROFIBUS Standard*, 21 Februar 2012, www.profibus.com.

[10] *OPC Foundation*, 21 Februar 2012, http://www.opcfoundation.org.

[11] Martin Choux and Mogens Blanke, *Fault Diagnosis for Nonlinear Hydraulic-Mechanical Drilling Pipe Handling System*. European Control Conference, 12-15 Desember 2011.

[12] *World Oil - Hardware-in-the-loop*, 20 Mai 2012, http://www.worldoil.com/Hardware-in-loop_simulator_for_offshore_rigs.html

[13] *OSIsoft White Paper – Using PI Data with Matlab*, 2 April 2011, http://vcampus.osisoft.com/library/library.aspx

[14] Seçil Ariduru, *Fatigue life calculation by Rainflow cycle counting method*. A thesis submitted to the graduate school of natural and applied sciences of Middle East technical university. Desember 2004.

[15] OSIsoft DCOM Configuration Guide, 21 Februar 2012, http://interfaces.osisoft.com/osiif/docs/OPCInt/DCOM%20Configuration%20Guide.pdf

7 Vedlegg

7.1 Installasjon av PI Server

7.2 Konfigurering av OPC server i Step7

7.3 Konfigurasjon av UDP kommunikasjon på PLS i step7

7.4 Matlab Rainflow algoritme

7.5 Matlab activeX protokoll

7.6 Visual Basic Application script

7.7 Hovedprogram for Rainflow algoritme

Vedlegg 7.1 - Installasjon av PI Server

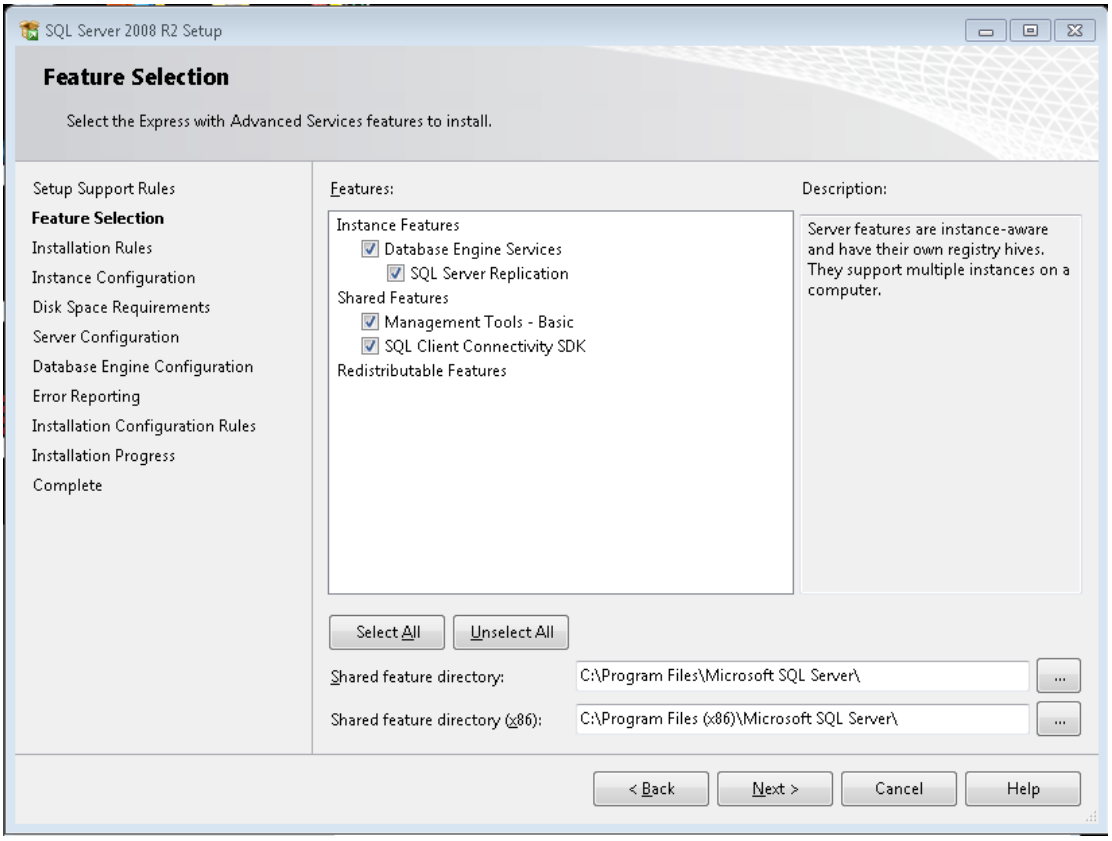
Før installasjonen av selve PI serveren må det først installeres noen programmer for at PI serveren skal fungere. Disse må installeres i riktig rekkefølge. Alle programmene lastes ned fra OSIsoft.com.

1. SQL Server Express 2008 R2 with Tools (64-bit)
2. PI Asset Framework (PI AF) Server with Event Frames 2010 R3 Install Kit
3. OSIsoft Prerequisites Kit - Standalone version
4. PI Asset Framework (PI AF) Client 2010 R3 Install Kit
5. Generere lisensfil
6. PI Server 2010 R3 Install Kit (64-bit)

1 SQL Server Express 2008 R2

1 Velg *New installation of add features to an existing installation*

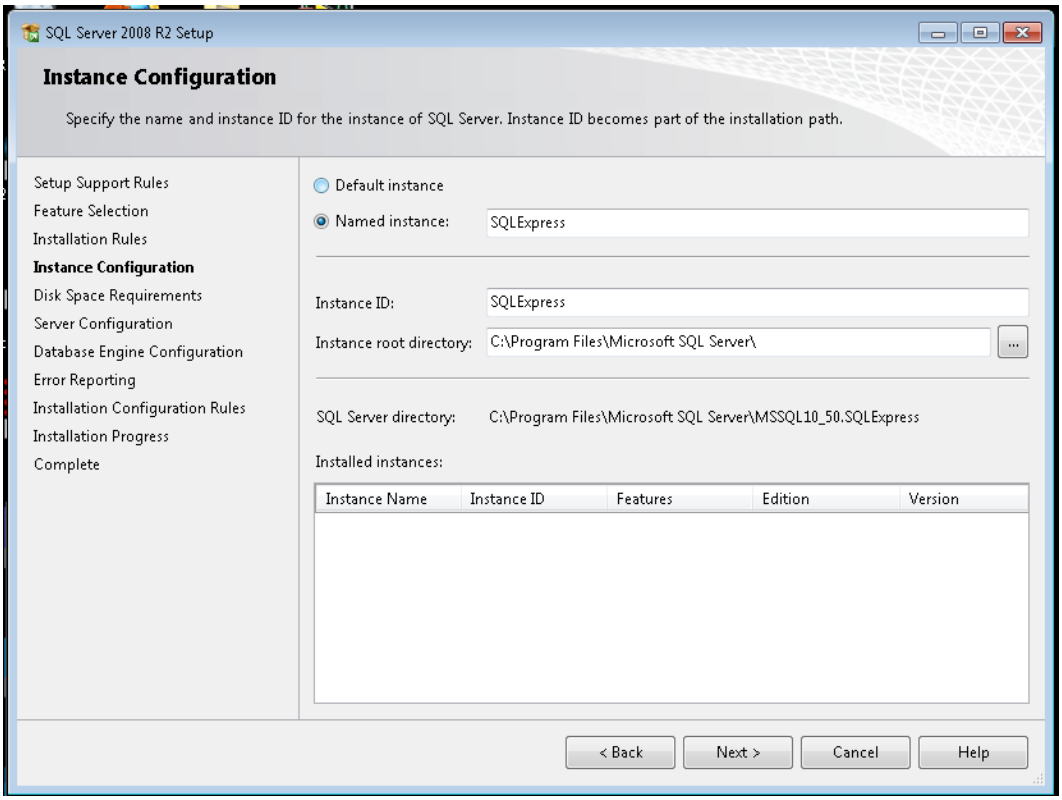
2 Trykk *Next* frem til bildet under kommer opp:



Her velges alle alternativene. Ved å velge *Management Tools* slipper man å installere *Management Studio* separat etter SQL serveren er installert. Dette er et

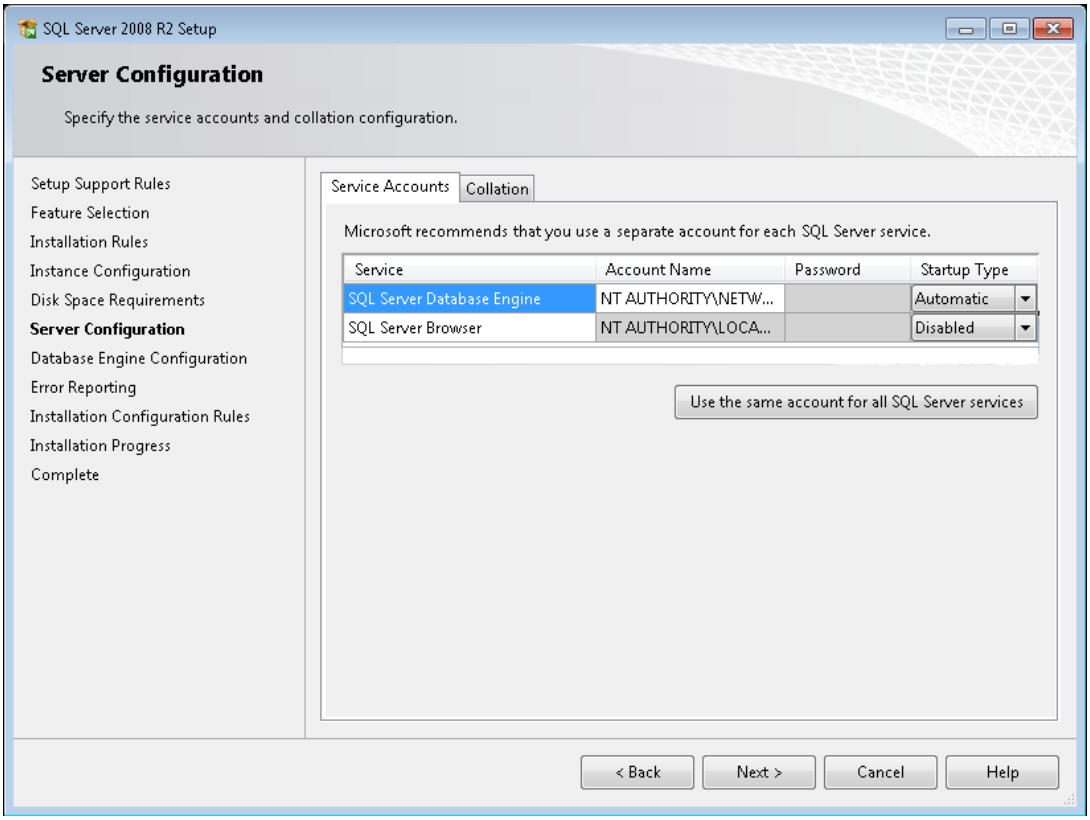
verktøy til å overvåke SQL databasen, og til å sjekke at databasen fungerer som den skal.

3 Det neste vinduet bestemmer hva SQL serveren skal hete, det kommer an på hvilken versjon av SQL server som installeres. Men det vanligste er å gi serveren navnet SQLExpress.

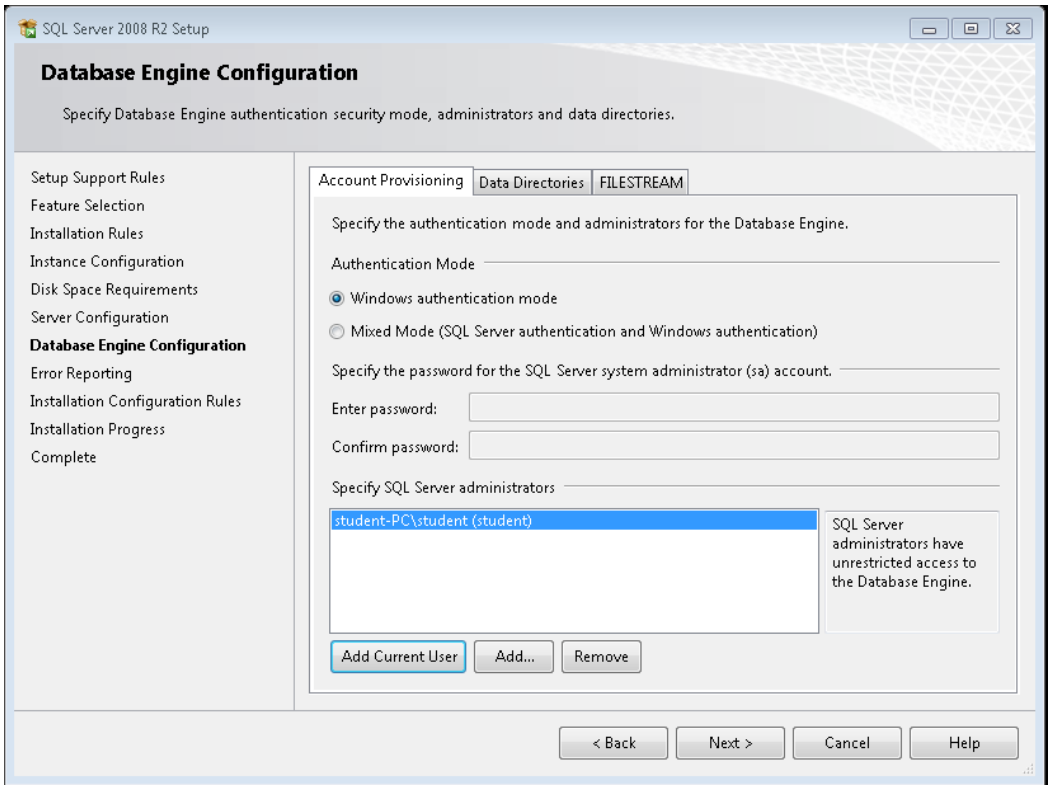


4 Neste valg omhandler serverkonfigurasjon. Her settes begrensninger for tilgangen til SQL serveren med brukernavn og passord. Dette brukes for å logge inn på serveren for for eksempel konfigurering eller avlesning av data. Serveren som brukes i denne oppgaven skal brukes lokalt, og installeres på samme PC uten andre brukere, derfor installeres den uten noen spesielle bruker og passord.

Service:	Account Name	Password	Startup Type
SQL Server Database Engine	NT AUTHORITY\NETWORK SERVICE		Automatic
SQL Server Browser	NT AUTHORITY\LOCAL SERVICE		Disabled



5 Neste valg bestemmer hvilke brukerkontoer maskinen SQL serveren installeres på som skal ha tilgang til serveren. I denne oppgaven er det samme brukerkonto som installerer og skal bruke serveren. Denne brukerkontoen kan enkelt leges til ved å trykke på *Add Current User*.



6 SQL serveren skal nå starte filkopieringen og installasjonen fullføres.

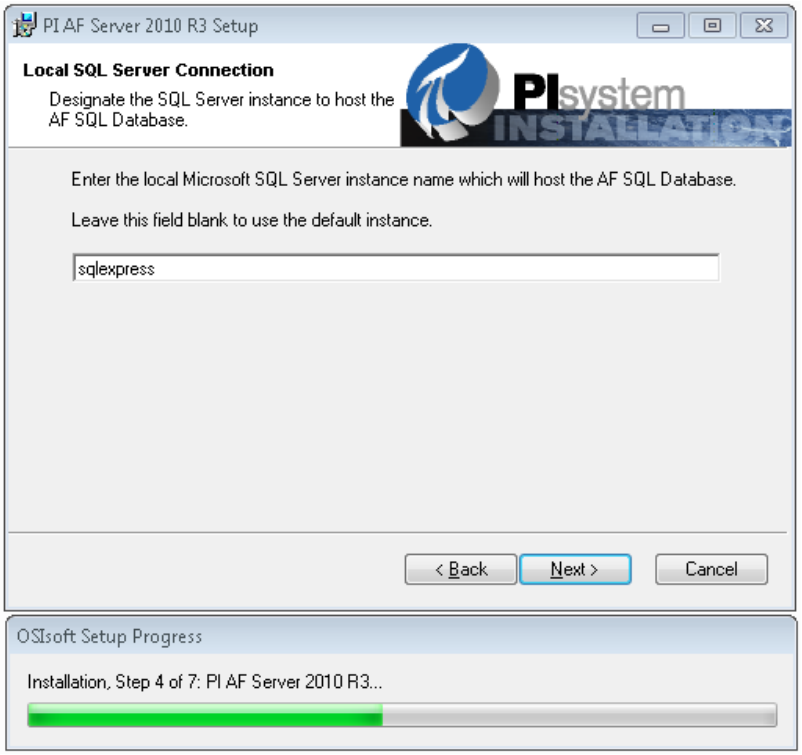
7 Gå så inn via startmenyen til *Microsoft SQL Server Management Studio*. Her kan man sjekke SQL serveren at den fungerer som den skal, og se hvilke verdier som er lagret i databasen.

For å koble til SQL serveren skrives servernavn inn: **.\SQLEXPRESS**
Det ble under installasjonen ikke valgt noe brukernavn eller passord for å koble til SQL serveren, derfor blir *Authentitaction* stående. Trykk så *connect*.

2 PI Asset Framework (PI AF) with Event Frames 2010 R3

Det neste som skal installers er PF AF server. Denne installasjonen er rett frem, og default valgene brukes hele veien.

På steg 4 av 7 under installasjonen kommer spørsmålet om hva SQL serveren PI systemet skal bruke heter. I dette tilfellet er SQL servernavnet: **SQLEXPRESS**.

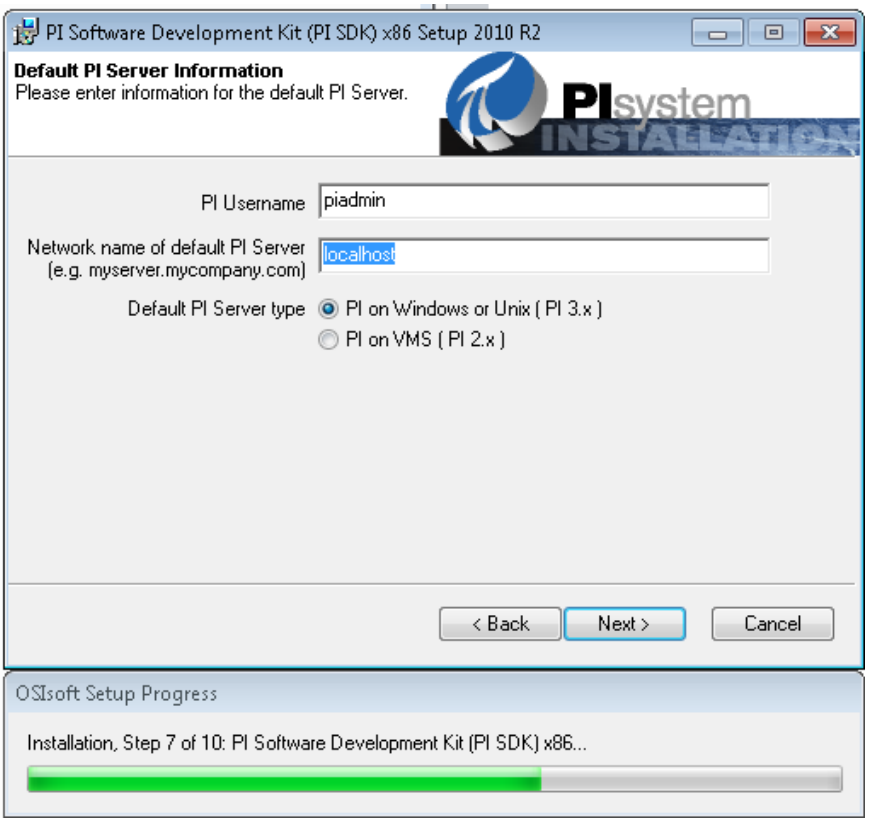


3 OSIsoft Prerequisites Kit - Standalone version

Installasjonen av OSIsoft prerequisites kit er rett frem, ingen valg som må tas.

4 PI Asset Framework (PI AF) Client 2010 R3 Install Kit

På steg 7 av 10 under installasjonen av AF klienten kommer valget om hvilket PI brukernavn som brukes. Dette er valgfritt, men i denne oppaven brukes **piadmin**. PI serveren skal kjøre på samme maskin som AF klienten installeres på, derfor brukes **localhost**. Dette gjør at AF klienten søker etter PI serveren på samme maskin som den blir installert, og finner den uavhengig av servernavnet til PI serveren.



På steg 9 av 10 kommer installasjonen av PI SDK (Software Development Kit). Her kommer det få valg, men før startknappen for installasjonen kommer en oppsummering av konfigurasjonene til installasjonen. Det er ikke mulig å velge *Default PI Servers* under denne installasjonen. Eneste valget er å trykke Start, for å starte installasjonen.

På steg 10 av 10 kommer selve PI Klient installasjonen, her der det bare å følge installasjonen.

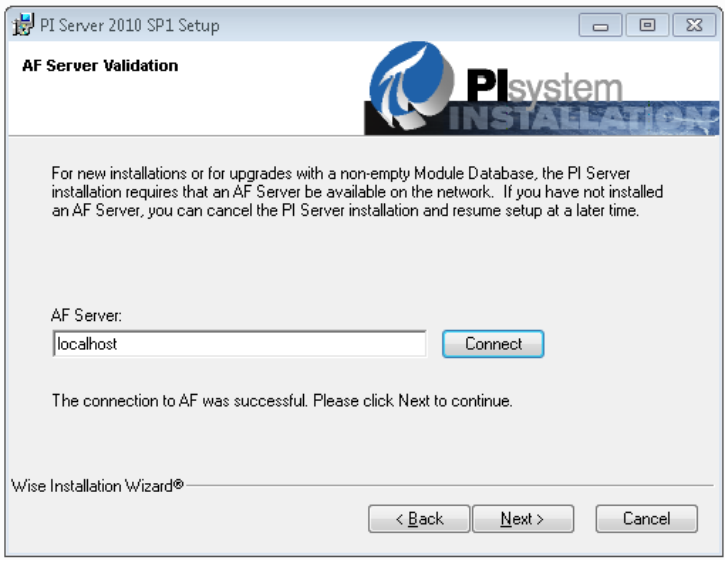
5 Generere lisensfil

Det første som må gjøres før PI serveren installeres er å generere en lisens fil på maskinen PI serveren skal installeres på. Det er viktig at lisensen blir generert på samme maskin, med eksakt samme hardware og innstillinger som de servermaskinen skal ha mens den kjører PI serveren, ellers vil den ikke fungere. For eksempel vil et bytte av IP-adresse, eller harddisk kunne gjøre lisensfilen ugyldig, og det vil ikke være mulig å gjennomføre installasjonen av PI serveren.

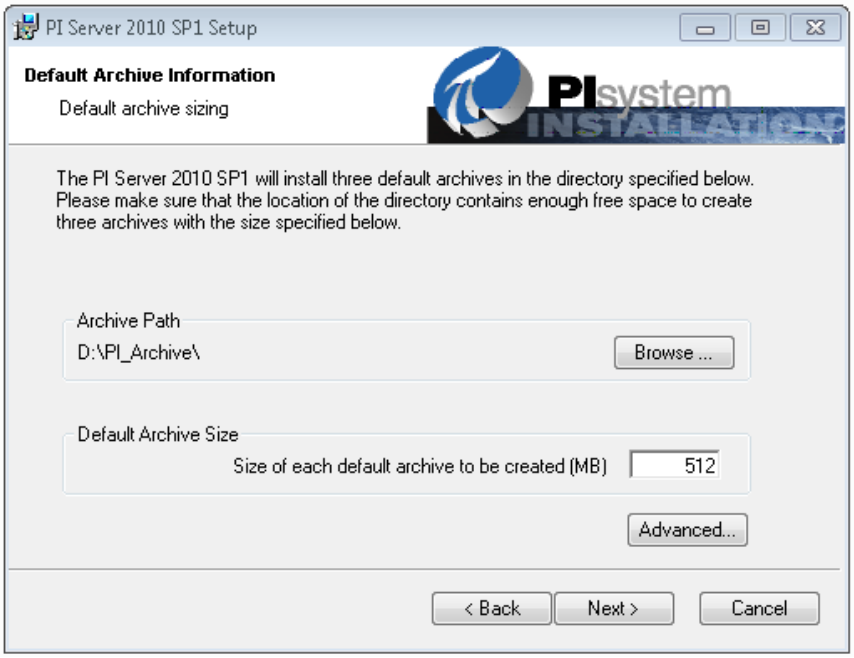
1. For å generere lisensen går man inn på OSIsofts hjemmesider, gå så til *Technical Support*. Oppe til venstre er *My Support*, etter å ha trykket på denne, kommer det opp en meny på venstre side, her er det en mulighet som heter *My License Activations*. Her ligger linken til å generere lisensfilen på servermaskinen. Dette tar deg videre til siden som genererer filen.
2. Det trengs en "maskin signatur" fil for å generere lisensen, denne er det link til på siden under punkt 2 på siden.
3. Når dette er gjort kan lisensfilen genereres, og installasjonen av PI serveren kan starte.

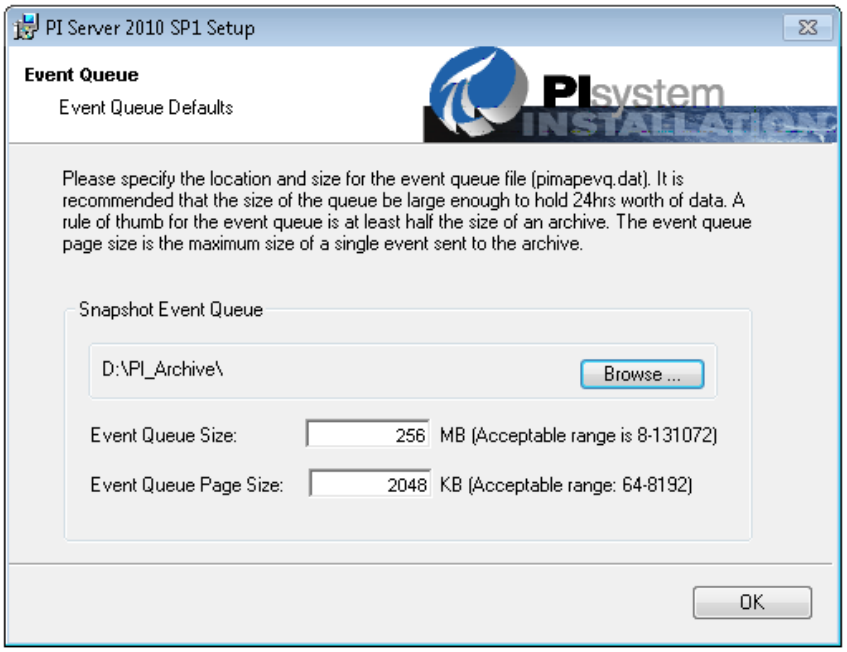
6 PI Server 2010 R3 Install Kit (64-bit)

1. Starte installasjonen
2. Du blir så bedt om å legge inn lisensfilen, bla så frem til lisensfilen som er generert og velg den.
3. PI installasjonen vil nå spørre hva AF serveren heter. AF serveren er på samme maskin som PI serveren installeres, og i stedet for å bruke datamaskinnavnet brukes **localhost**. For å teste at dette fungerer kan *Connect* knappen trykkes, og se om AF serveren finnes på nettverket/maskinen.

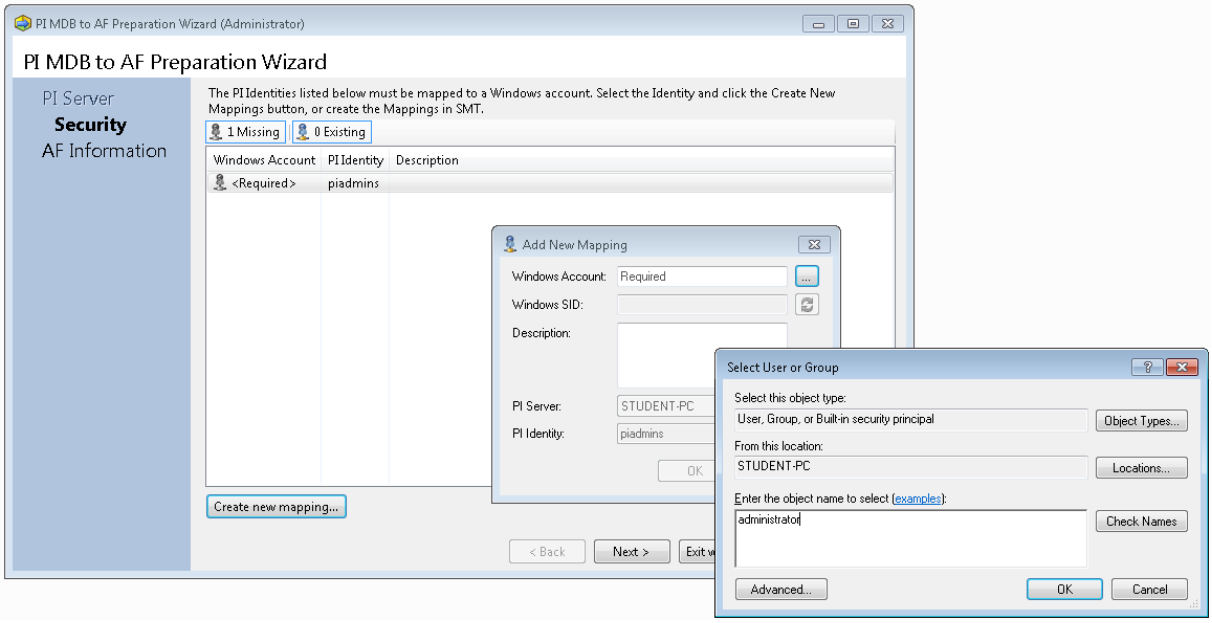


4. PI Module Database (MDB), velg *Yes*.
5. Disable password authentication for all PI Users, velg *Yes.*
6. Hukk av for *Default points?* Og *Auto start service?*
7. Angi hvor data skal lagres. Her er det lurt å legge arkivene på en annen partisjon eller disk enn hvor PI serveren er installert. Størrelsen på arkivene og queue er her satt til dobbelt av default. "Verdiene inne på Advanced er også satt til det dobbelt av default"

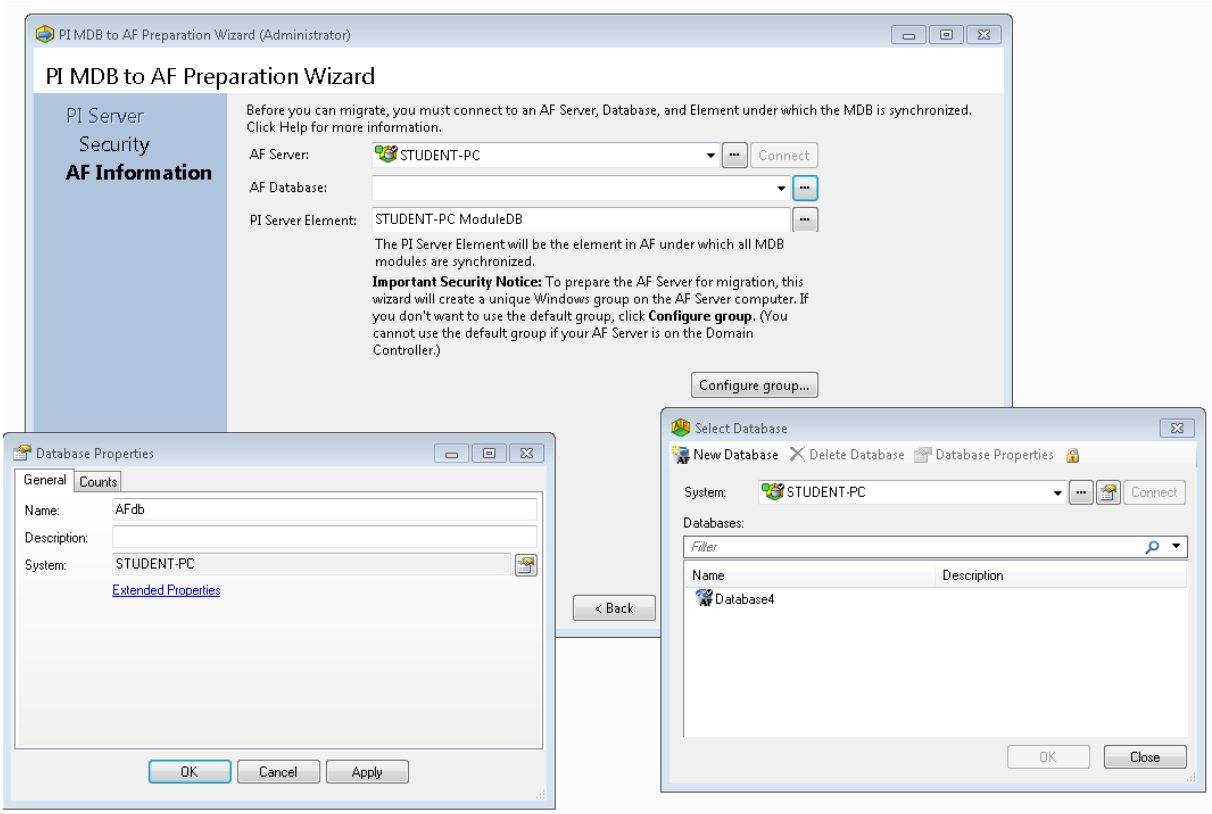




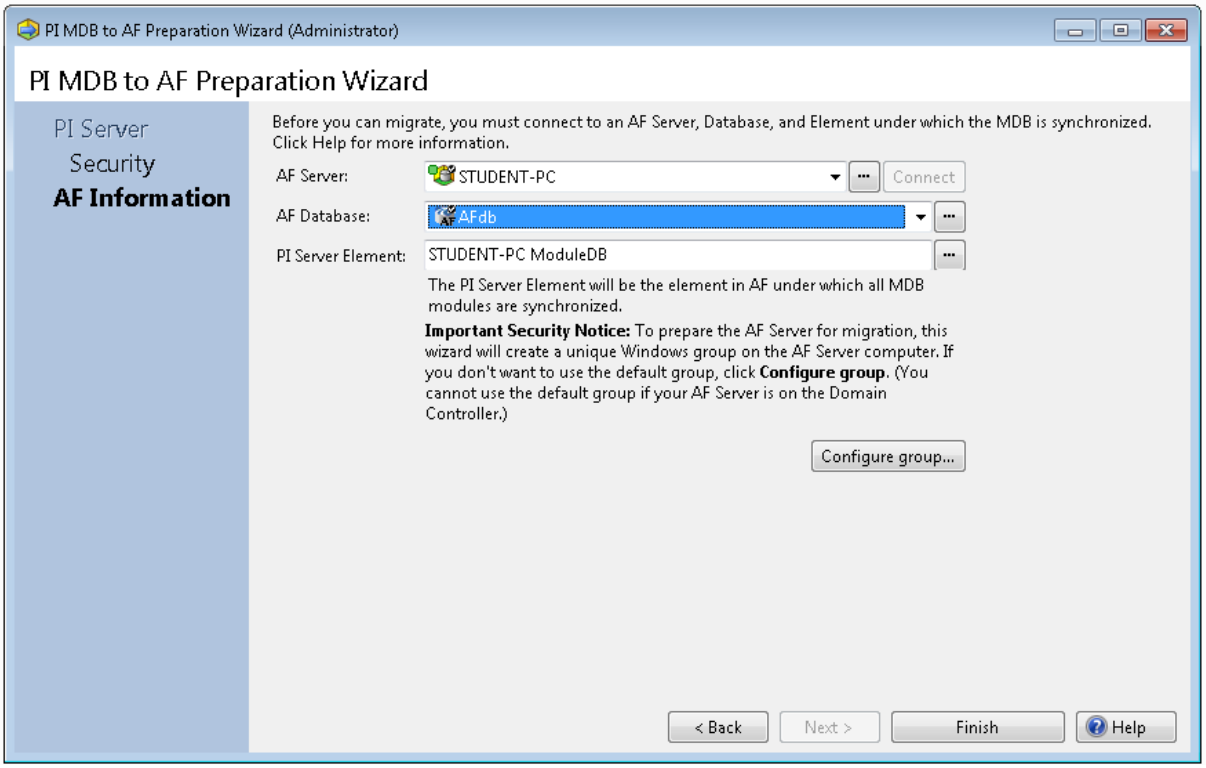
8. Legge til bruker til PI MDB databasen.
   Velg først *1 Missing*, velg så *Create new mapping*. Her må det velges den windows brukerkontoen som skal brukes, i dette tilfelle *Administrator/Student.* Trykker så OK på de to vinduene, og kontoen er lagt til.



9. Neste vindu ber om hvilken AF server som skal brukes, og hva databasen skal hete. Her må det lages en ny AF database. Dette gjøres ved å trykke på …, så velge *New Database*. Skriver så inn navnet på den ny databasen, i denne oppgaven er den kalt *AFdb.*



Det siste vinduet i installasjonen skal da se slik ut:

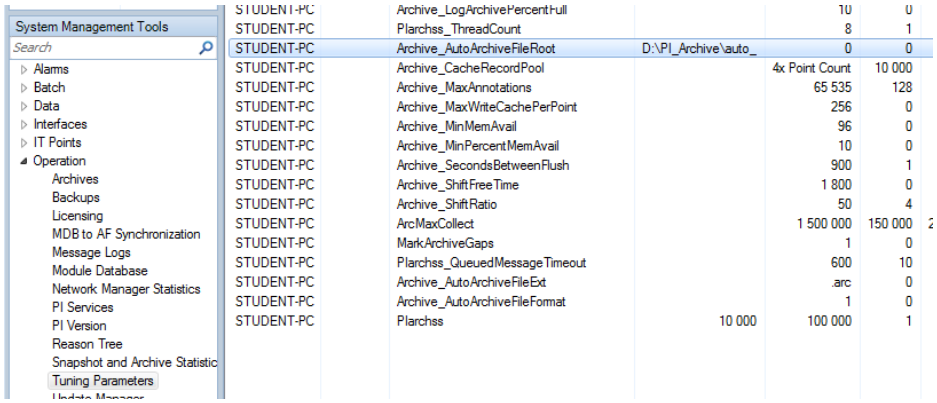


Trykk så *Finish.* PI Serveren er nå installert.

Auto arkivering

For å få PI systemet til å lage nye arkiver etter at et arkiv har blitt fullt må det legges til en ny verdi i *PI System Management Tools.*

1. Gå først inn i **PI System Management Tools**
2. Velg **Operation** i listen til høyre
3. Velg **Tuning Parameters**
4. Velg så **Archive** på toppen av siden som kommer frem
5. Finn filen **Archive_AutoArchiveFileRoot** og dobbeltklikk
6. Legg så til **D:\PI_Archive\auto_** i **Value.** De nye automatisk-genererte arkivene vil da bli lagret samme sted som de gamle, og filnavnet vil starte med auto_.



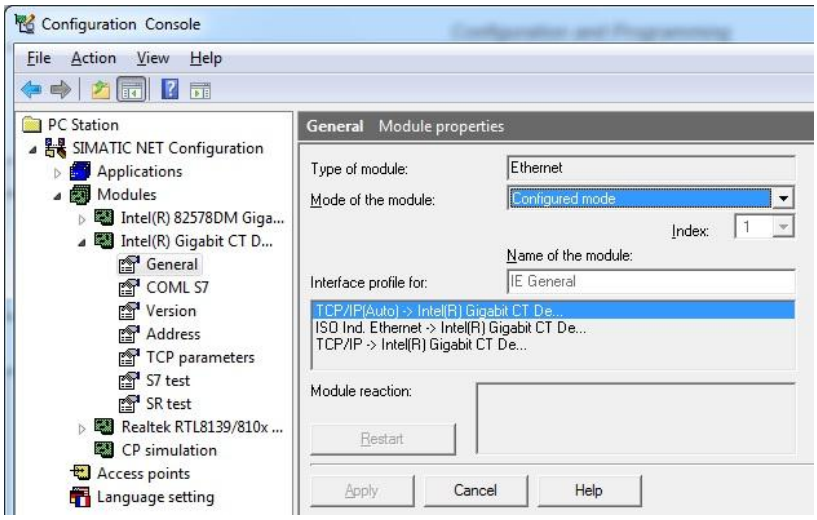
Vedlegg 7.2 – Konfigurering av OPC server i Step7

Programmer som er brukt:

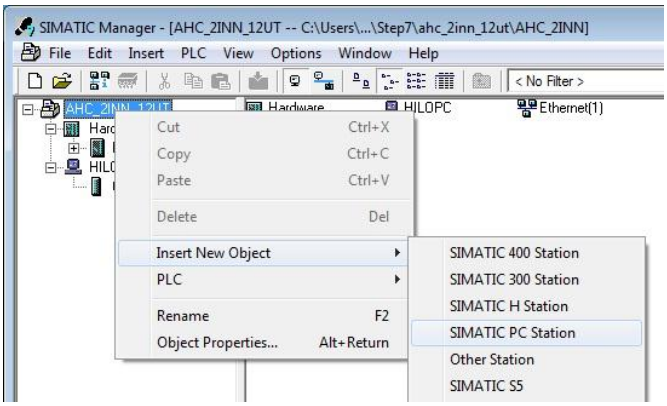
- Step 7 V5.5
- Simatic NET 2010

I eksempelet under er PLS programmet satt opp og laget ferdig.

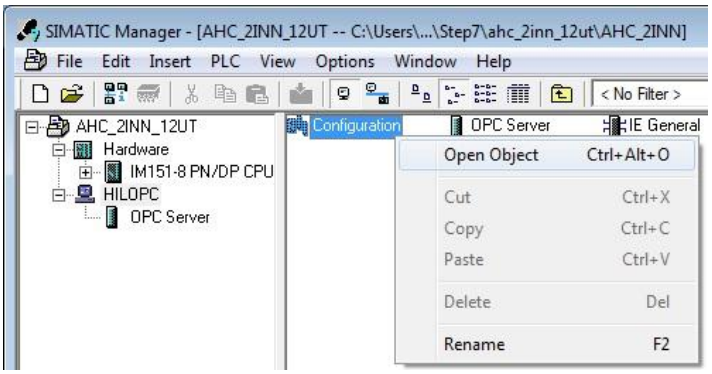
1. Nettverkskortet som er koblet til PLSen må først settes i *Configuration Mode*. Dette gjøres via Configuration Console. **Start -> Programmer -> Siemens Automation -> SIMATIC -> SIMATIC NET -> Configuration Console**



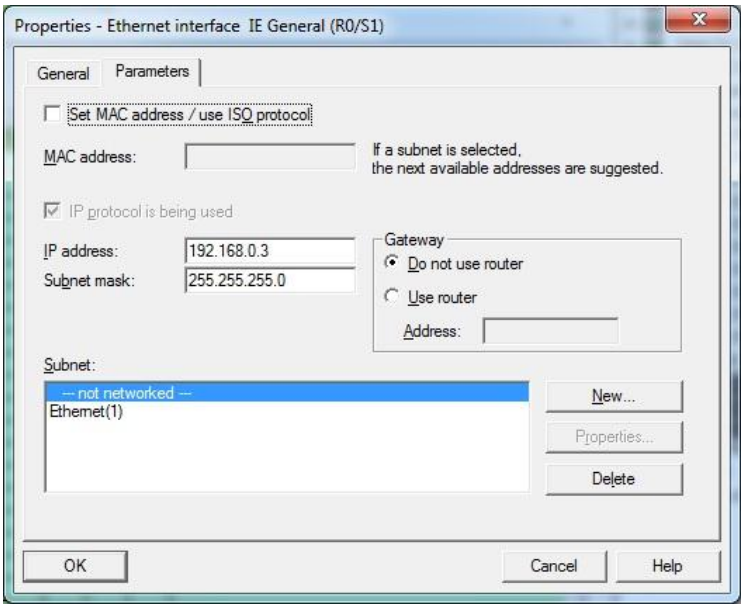
2. Sett inn PC STATION som skal konfigureres som en OPC server. Høyreklikk på prosjektnavnet i treet til høyre i Step7; **Insert New Object -> SIMATIC PC Station**



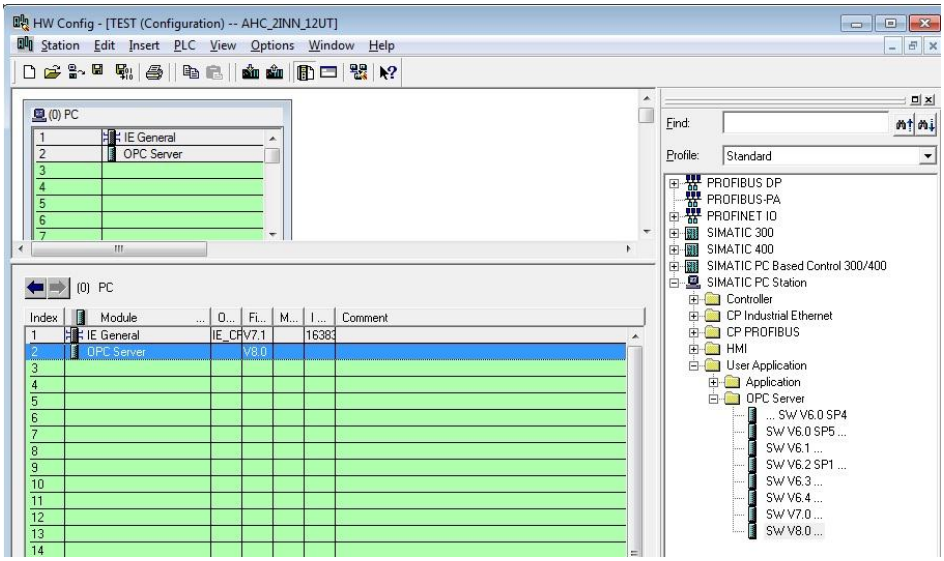
3. Konfigurering av PC STATION. Merk PC STATION i treet til venstre i Step7, og høyreklikk så på **Configuration -> Open Object.** Dette åpner hardware konfigurasjonen til PC STATION som skal bli OPC server.



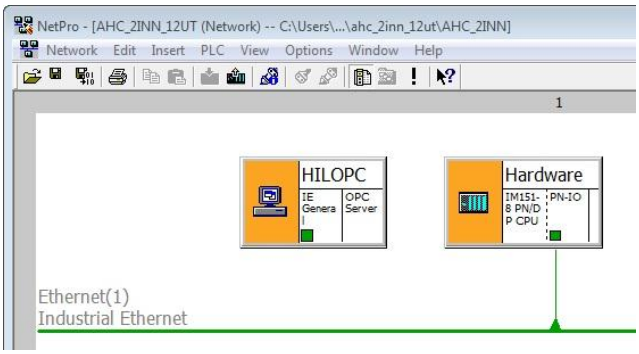
4. I hardware konfigurasjonen til PC STATION skal det legges til nettverksadapter og OPC Server. Dette gjøres ved å finne ønsket nettverksadapter i listen til høyre og dra over i konfigurasjonslisten til venstre. I dette eksempelet er det brukt *SW V7.1 fra IE General* som nettverksadapter. Egenskaper for nettverksadapteren kommer da opp som vist på bildet under, og IP-Adressen til maskinen OPC serveren skal stå på skrives inn under **Parameters.** Kommer det opp en beskjed om at den ikke er tildelt et nettverk, trykk OK.



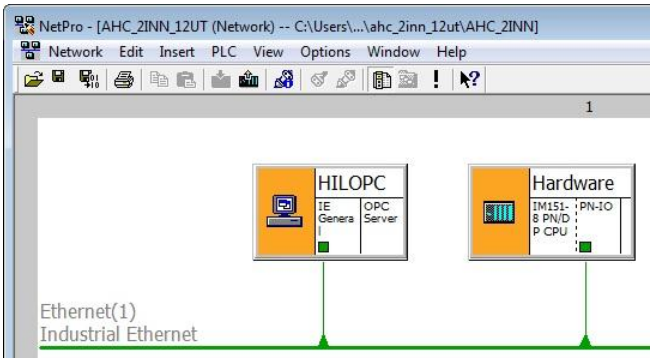
Legge til OPC serveren til PC Station konfigurasjonen. Samme som i punkt 4, finne OPC serveren og dra over til hardwarelisten på venstre side. I dette eksempelet er *OPC Server SW V8.0* brukt.



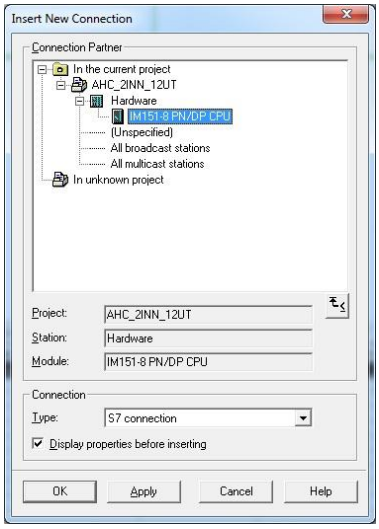
5. Gå så tilbake til hovedvinduet i Step7 velg **Options -> Configure Network.** Her konfigureres tilkoblingen mellom OPC serveren og PLSen.



Marker så den grønne firkanten på OPC server blokka, og dra den ned til *Industrial Ethernet* linja under.



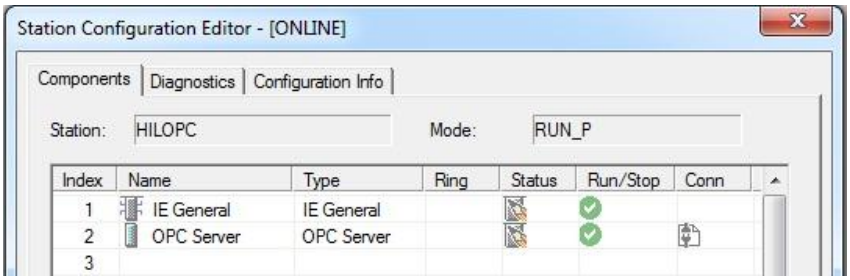
6. Høyreklikk så på *OPC Server* og velg **Insert New Connection.** Her velges hvilket target OPC serveren skal ha, i dette eksempelet er det PLSen; *IM151-8 PN/DP CPU.* Type tilkobling er *S7 Connection.* Trykk *OK*.



Egenskaper for tilkoblingen kommer så opp i et vindu. Her konfigureres IP-Adressen til PLSen og OPC serveren. I eksempelet har OPC Serveren 192.168.0.3, og PLSen 192.168.0.1.

7. Lagre og last ned konfigurasjonen til PLS.
8. Konfigurasjonen må så lastes opp til PCen som skal være OPC server. Dette gjøres via Station Configurator. Hvert Step7 prosjekt lager en konfigurasjons fil for PC STATION i .XDB format, denne importeres i Station Configurator og OPC serveren er oppe og kjører. .XBD konfigurasjons fila ligger i en undermappe der Step7 Prosjektet er lagret. (eksempel: .\XDBs\pcst_1.xdb)

   Start -> Programmet -> Station Configurator. Velg så **Import Station** og bla frem til konfigurasjonsfilen og velg OK. Når konfigurasjonsfilen er importert og OPC serveren er oppe og kjører skal ser det slik ut i Station Configuration Editor.



9. Konfigurasjon av DCOM er forklart i OSIsoft sin DCOM konfigurasjons guide på fra side 3 til 18. (Referanse [15])

Vedlegg 7.3 - Konfigurering av UDP kommunikasjon med step7

Stengene som følger forutsetter at PLSen er satt opp med nødvendig hardware konfigurasjon

1. Åpne UDP eksempelet ved å trykke på File → Retrieve og finn *Samlpe_open_upd.zip*

2. Kopier alle eksempel-blokkene til ditt PLS prosjekt

3. Funksjonsblokken FB500 inneholder alle funksjonene for kommunikasjonsrutinene i programmet. I nettverk 3 blir den lokale porten definert (LOC_PORT). I nettverk 7 blir enheten PLSen skal snakke med definert med IP-adresse og port. Portene i nettverk 3 og 7 må være forskjellige, da den ene blir brukt av klient og den andre av server. Disse må passe til enheten det skal kommuniseres med.

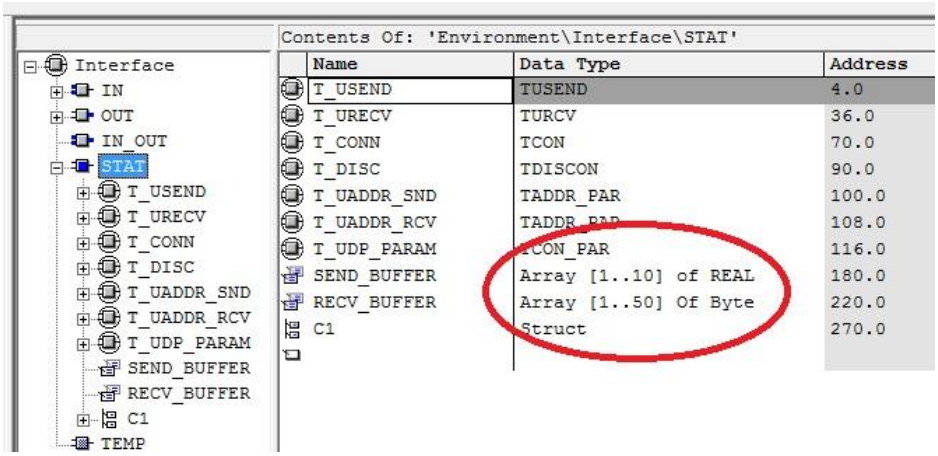
4. Nettverk 9 inneholder funksjonen som aktiverer sendingen av data. Dette blir som standard styrt av et reservert minne (som oftest M10.x, dette er konfigurert i hardwarekonfigurasjon). Hvert bit i bytet M10 representerer en gitt frekvens. M10.0 er det raskeste og gir en puls hvert 100ms. Her er det mulig å koble til et eget trigger bit for eksempel ved bruk av OB35.

5. Nettverk 8 er sendefunksjonen til UDP kommunikasjonen. Inngangene DATA og LEN definerer hvilket minneområde som skal sendes ut. (Syntax for DATA for mer enn 4bytes: *P#DBxxx.x BYTE <antall bytes>*)

   Det kan være lurt å lage en ny datablokk (eks. DB501) som inneholder en liste med reservert minne til de aktuelle variablene som skal sendes og motta for bedre oversikt. Uten en egen datablokk vil verdiene legge seg rett i mottaker eller sendebuffer som gjør dataen uoversiktlig.

6. Nettverk 11 inneholder mottakerfunksjonen. Inngangene DATA og LEN definerer hvilket minneområde dataen som mottas skal legges på. Her gjelder samme syntax for verdier over 4 bytes.

A. Ved endringer i FB500 kan det oppstå feilmeldinger til tider, som bygger på DB500. Det må da lages en ny Instance Data Block av DB500, og generere DB500 på nytt. Dette gjøres ved å høyreklikke ved siden av blokkene → Insert New Object → Data block. Velg så navn og type (Instance DB) og velg DB500 i listen øverst til høyre.



Figur 1 - Endre datatype for UDP kommunikasjon

B. Standard oppsettet for datatypen som sendes er laget i byte. Og sender ut et tog med bytes som legges i bufferen. Dette kan endres på ved å gå inn i FB500 blokken og endre øverst i vinduet. Sende og motta bufferen kan da endres til ønsket format. Under er et eksempel på sendebufferen som er endret fra 1-50 bytes, til 1-10 REAL verdier. Etter denne konfigurasjonen er det nødvendig å generere DB500 på nytt. (Se punkt A)

Vedlegg 7.4 – Matlab Rainflow algoritme

```
function [ext, exttime] = sign2ext(sig, time)
% SIGN2EXT - finner lokale extremer i tidsignalet
%
% function [ext, exttime] = sig2ext(sig, dt, clsn)
%
% SYNTAX
%   sign2ext(sig)
%   [ext]=sign2ext(sig)
%   [ext,exttime]=sig2ext(sig)
%   [ext,exttime]=sig2ext(sig, dt)
%   [ext,exttime]=sig2ext(sig, dt, clsn)
%
% OUTPUT
%   EXT     - signal
%   EXTTIME - timestamps
%
% INPUT
%   SIG     - signal
%   time    - timestamps


error(nargchk(1,3,nargin))

time=time(:);


sig=sig(:);



w1=diff(sig);
w=logical([1;(w1(1:end-1).*w1(2:end))<=0;1]);
ext=sig(w);
exttime=time(w);


w1=diff(ext);
w=~logical([0; w1(1:end-1)==0 & w1(2:end)==0; 0]);
ext=ext(w);

exttime=exttime(w);


w=~logical([0; ext(1:end-1)==ext(2:end)]);
ext=ext(w);

exttime=exttime(w);
```

```
if length(ext)>2,
    w1=diff(ext);
    w=logical([1; w1(1:end-1).*w1(2:end)<0; 1]);
    ext=ext(w);

    exttime=exttime(w);

end

if nargout==0,

    plot(dt,sig,'b-',exttime,ext,'ro')
    legend('signal','extrema')

    xlabel('time')
    ylabel('signal & extrema')
    clear ext exttime
end
```

```
/*  RAINFLOW  */


#include <math.h>
#include "mex.h"

/* ++++++++++ BEGIN RF3 [ampl ampl_mean nr_of_cycle] */
/* ++++++++++ Rain flow without time analysis */
void
rf3(mxArray *array_ext, mxArray *hs[]) {
    double *pr, *po, a[16384], ampl, mean;
    int tot_num, index, j, cNr;
    mxArray *array_out;

    tot_num = mxGetM(array_ext) * mxGetN(array_ext);
    pr = (double *)mxGetPr(array_ext);

    array_out = mxCreateDoubleMatrix(3, tot_num-1, mxREAL);
    po = (double *)mxGetPr(array_out);

    j = -1;
    cNr = 1;
    for (index=0; index<tot_num; index++) {
        a[++j]=*pr++;
        while ( (j >= 2) && (fabs(a[j-1]-a[j-2]) <= fabs(a[j]-a[j-1])) ) {
            ampl=fabs( (a[j-1]-a[j-2])/2 );
            switch(j)
{
                case 0: { break; }
                case 1: { break; }
                case 2: {
                    mean=(a[0]+a[1])/2;
                    a[0]=a[1];
                    a[1]=a[2];
                    j=1;
                    if (ampl > 0) {
                        *po++=ampl;
                        *po++=mean;
                        *po++=0.50;
                    }
                    break;
                }
                default: {
                    mean=(a[j-1]+a[j-2])/2;
                    a[j-2]=a[j];
                    j=j-2;
                    if (ampl > 0) {
                        *po++=ampl;
                        *po++=mean;
                        *po++=1.00;
                        cNr++;
                    }
                    break;
                }
            }
        }
    }
    for (index=0; index<j; index++) {
        ampl=fabs(a[index]-a[index+1])/2;
        mean=(a[index]+a[index+1])/2;
```

```
        if (ampl > 0){
            *po++=ampl;
            *po++=mean;
            *po++=0.50;
        }
    }
  /* you can free the allocated memeory */
  /* for array_out data                 */
    mxSetN(array_out, tot_num - cNr);
    hs[0]=array_out;
}
/* ++++++++++ END RF3 */

/* ++++++++++ BEGIN RF5 [ampl ampl_mean nr_of_cycle cycle_begin_time
cycle_period_time]*/
/* ++++++++++ Rain flow with time analysis */
void
rf5(mxArray *array_ext, mxArray *array_t, mxArray *hs[]) {
    double *pr, *pt, *po, a[16384], t[16384], ampl, mean, period, atime;
    int tot_num, index, j, cNr;
    mxArray *array_out;

    tot_num = mxGetM(array_ext) * mxGetN(array_ext);
    pr = (double *)mxGetPr(array_ext);
    pt = (double *)mxGetPr(array_t);

    array_out = mxCreateDoubleMatrix(5, tot_num-1, mxREAL);
    po = (double *)mxGetPr(array_out);

    j = -1;
    cNr = 1;
    for (index=0; index<tot_num; index++) {
        a[++j]=*pr++;
        t[j]=*pt++;
        while ( (j >= 2) && (fabs(a[j-1]-a[j-2]) <= fabs(a[j]-a[j-1])) ) {
            ampl=fabs( (a[j-1]-a[j-2])/2 );
            switch(j)
{
                case 0: { break; }
                case 1: { break; }
                case 2: {
                    mean=(a[0]+a[1])/2;
                    period=(t[1]-t[0])*2;
                    atime=t[0];
                    a[0]=a[1];
                    a[1]=a[2];
                    t[0]=t[1];
                    t[1]=t[2];
                    j=1;
                    if (ampl > 0) {
                        *po++=ampl;
                        *po++=mean;
                        *po++=0.50;
                        *po++=atime;
                        *po++=period;
                    }
                    break;
                }
                default: {
                    mean=(a[j-1]+a[j-2])/2;
                    period=(t[j-1]-t[j-2])*2;
```

```
                    atime=t[j-2];
                    a[j-2]=a[j];
                    t[j-2]=t[j];
                    j=j-2;
                    if (ampl > 0) {
                        *po++=ampl;
                        *po++=mean;
                        *po++=1.00;
                        *po++=atime;
                        *po++=period;
                        cNr++;
                    }
                    break;
                }
            }
        }
    }
    for (index=0; index<j; index++) {
        ampl=fabs(a[index]-a[index+1])/2;
        mean=(a[index]+a[index+1])/2;
        period=(t[index+1]-t[index])*2;
        atime=t[index];
        if (ampl > 0){
            *po++=ampl;
            *po++=mean;
            *po++=0.50;
            *po++=atime;
            *po++=period;
        }
    }
  /* free the memeory !!!*/
    mxSetN(array_out, tot_num - cNr);
    hs[0]=array_out;
}
/* ++++++++++ END RF5 */


/* mexFunction - main function called from MATLAB. */
void
mexFunction( int nlhs,       mxArray *plhs[],
int nrhs, const mxArray *prhs[] )
{
    mxArray *array_in0;
    mxArray *array_in1;
    double *pr, s0, s1, dt;
    int ind;

    if (nrhs < 1) {
        mexErrMsgTxt("RAINFLOW requires at least one input argument.");
    } else if (nlhs > 1) {
        mexErrMsgTxt("RAINFLOW requires only one output argument.");
    }

    if (mxIsComplex(prhs[0]) || !mxIsDouble(prhs[0])) {
        mexErrMsgTxt("RAINFLOW requires DOUBLE ARRAY as first input
argument.");
    } else { array_in0 = (mxArray *)prhs[0]; }

    switch(nrhs) {
        case 1: {
```

```
            rf3(array_in0, plhs);
            break;
        }
        case 2: {
            if (mxIsComplex(prhs[1]) || !mxIsDouble(prhs[1])) {
                mexErrMsgTxt("RAINFLOW requires two DOUBLE ARRAY input
arguments.");
            }
            s0 = mxGetM(prhs[0]) * mxGetN(prhs[0]);
            s1 = mxGetM(prhs[1]) * mxGetN(prhs[1]);
            if (s0 == s1) {
                array_in1 = (mxArray *)prhs[1];
                rf5(array_in0, array_in1, plhs);
            } else if (s1 == 1) {
                pr = (double *)mxGetPr(prhs[1]);
                dt = *pr;
                array_in1 = mxCreateDoubleMatrix(1, s0, mxREAL);
                pr = (double *)mxGetPr(array_in1);
                for (ind=0; ind<s0; ind++) {
                    pr[ind]=ind*dt;
                }
                rf5(array_in0, array_in1, plhs);
                mxDestroyArray(array_in1);
            } else {
                mexErrMsgTxt("RAINFLOW: Time Array size error.");
            }
            break;
        }
        default: {
            mexErrMsgTxt("RAINFLOW: To many input arguments.");
            break;
        }
    }
}
```

Vedlegg 7.5 – Matlab Active X kommunikasjon

Test.m

```
% Sett inn PI Server navn her:)
server = 'STUDENT-PC';

% genererer tilkoblingstring
cnstr = strcat('Provider=PIOLEDB.1;Data Source=',server,';Persist Security
Info=False');

% Åpner tilkobling
cn = adodbcn(cnstr);

% henter verdiene for tag Wireforce
% med 1 sekunds intervaller
SampledData = PI_SampledData(server,'Wireforce','22.03.2012
00:00:00','07.04.2012 09:00:00','1s',cn);

%x = SampledData(:,3);




% Skriv verdi;
%WriteValue = PI_WriteValue(server,'Run_rainflow','*','256',cn)

% henter en liste over tags som ender med 'force'
%TagSearch = PI_TagSearch(server,'*force',cn)


invoke(cn,'close');



clear server cnstr cn;
```

```
function cn=adodbcn(cnstr,cto)
% cn = adodbcn(cnstr,[cto])
%
% kobler til ADO OLEDB ved hjelp av Microsoft ActiveX Data Source Control
%
% Inputs:
%   cnstr,  str som inneholder informasjon som tilkoblingen trenger
%   cto,    CommandTimeout i sekunder (default=60 sekunder dersom den er
uspecifisert)
%
% Output:
%   cn,     connection



%genererer activeX control
cn = actxserver('ADODB.Connection');
%setter peker til server eller client
set(cn,'CursorLocation',3);
%set(cn,'Cursorlocation',2);

%åpner tilkobling
invoke(cn,'Open', cnstr);

%Spesifiserer timeout, 60 sekunder dersom det ikke annet er spesifisert.
if nargin>1
    set(cn,'CommandTimeout',cto);
else
    set(cn,'CommandTimeout',60);    %default
end
```

```
function Output = PI_SampledData(server,tag,starttime,endtime,interval,cn)
% function Output =
PI_SampledData(server,tag,starttime,endtime,interval,[cn])

% © Copyright OSIsoft, Inc.
% Returns a set of sampled values for a tag
% [tag time1 value1]
% ...
% [tag timeN valueN]
%
% Sample usage of the function.
% This example retrieves the values for cdt158 on
% server FLPVM2003 during the last hour every 5 minutes :
% PI_SampledData('FLPVM2003','cdt158','*-1h','*','5m',[cn])
%
% For better performance, one can pass an existing connection
% as the last parameter.

% Detect if an existing connection was passed
if nargin<6
    % Build connection string
    cnstr = strcat('Provider=PIOLEDB.1;Data Source=',server,';Persist
Security Info=False');

    % Open connection
    cn=adodbcn(cnstr);
end

% Query to execute
sql= strcat('SELECT tag, time, value FROM piarchive..piinterp2 WHERE tag =
''', ...
    tag,''' AND time BETWEEN ''',starttime,''' AND ''', endtime,''' AND
timestep=''',interval,'''');

% Run query and return results
Output=adodbquery(cn,sql);

% If no existing connection was passed
% (i.e. connection is not to be reused)
if nargin<6
    % Close connection
    invoke(cn,'close');
end

% Clear variables
clear sql cn cnstr server tag starttime endtime interval;
```

```
function x=adodbquery(cn,sql)
% [x]=adodbquery(cn,sql)
%
% adoledbquery    Executes the sql statement against the connection cn
%
% Inputs:
%   cn,     open connection to ADO OLEDB ActiveX Data Source Control
%   sql,    SQL statement to be executed
%
% Output
%   x,      cell array of query results
%
%
% Martin Furlan
% martin.furlan@iskra-ae.com
% January 2007
%
% Changed "invoke(r,'release');" for "invoke(r,'close');
% By Steve Pilon (spilon@osisoft.com) August 2009
%
%
%Peker endret cmd = actxserver('ADODB.Command');
%               r = actxserver('ADODB.Recordset');
%               set(r,'CursorType',3);
%               set( r , 'LockType' , 'adLockBatchOptimistic' );

%open recordset and run query
cmd = actxserver('ADODB.Command');
r = actxserver('ADODB.Recordset');
set(r,'CursorType',3);
set( r , 'LockType' , 'adLockBatchOptimistic' );
invoke( r , 'Open' , sql , cn );
invoke(cn,'BeginTrans');

try
  r = invoke(cn,'Execute',sql);

  invoke(cn,'CommitTrans');
  sclSuccess = 1
catch
  invoke(cn,'RollbackTrans');
  sclSuccess = 0
end

%retrieve data from recordset
if r.recordcount>0
    x=invoke(r,'getrows');
    x=x';
else
    x=[];
end

%close recordset
invoke(r,'close');
```

Vedlegg 7.6 – Visual Basic Application script for å kjøre Matlab - Rainflow

```
Sub Matlab()
Dim X As String
Dim y As String
Dim z As String
Dim F As String
Dim dblInput As Double
'Dim oImage As PBSymLib.Bitmap
'Dim oSymbol_Image As PBObjLib.Symbol
Dim test As String
z = "Kører rainflow analyse fra,til"
X = ThisDisplay.textbox1.Text
y = ThisDisplay.TextBox2.Text
F = X + y
MsgBox (F), 0, (z)
Dim oMatlab As Object
Set oMatlab = New MLApp.MLApp
Visible = True
sResult = oMatlab.PutWorkspaceData("X", "base", X)
sResult = oMatlab.PutWorkspaceData("Y", "base", y)
sResult = oMatlab.Execute("cd D:\Matlabfiles\MATLAB\processbook_matlab")
sResult = oMatlab.Execute("lagre_bilde_test(X,Y)")
ThisDisplay.oImage.Load ("C:\Users\student\Desktop\test.bmp")
End Sub
```

Vedlegg 7.7 – Hovedprogram for Rainflow algoritme

```
function  lagre_bilde_test(X,Y)
    server = 'STUDENT-PC';

    % genererer connection string
      cnstr = strcat('Provider=PIOLEDB.1;Data Source=',server,';Persist
      Security Info=False');

    % Åpne tilkobling
    cn = adodbcn(cnstr);




    %hent data fra ønsket tidsinterval med 1 sekunds tidsstep,
    %fra tid X til tid Y, sendt til matlab av VBA script
    %i prockbook
    SampledData = PI_SampledData(server,'Wireforce',X,Y,'1s',cn);

    sig=(SampledData(:, 3));
    sig=cell2mat(sig);
    time=(SampledData(:,2));


    [ext, exttime] = sign2ext(sig, time)
    ext=double(ext);
    rf = rainflow(ext);
    bilde=figure; rfhist(rf,30,'ampl')
    hold

    saveas(bilde,'C:\Users\student\Desktop/test','bmp');
    %
    x='rainflow analyse ferdig';
```